

Algorithmen 1

Übung 2 Amortisierte Analyse, Sortieren



Datenstrukturen bisher in Algo 1

Array

Datenstrukturen bisher in Algo 1

Array

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

Datenstrukturen bisher in Algo 1

Array

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)

Datenstrukturen bisher in Algo 1

Array

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$

Datenstrukturen bisher in Algo 1

Array

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

Listen

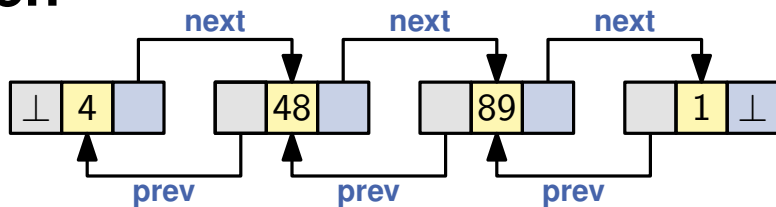
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

Listen



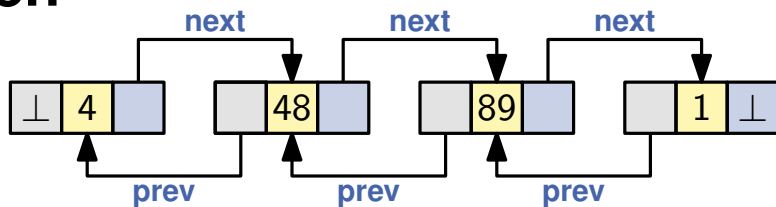
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

Listen



- Einfügen, Löschen, etc. in $O(1)$ Zeit
- **kein** Wahlfreier Zugriff

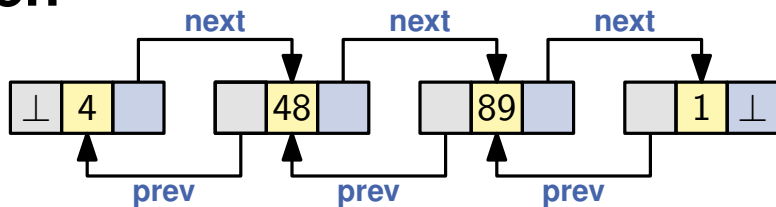
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

Listen



- Einfügen, Löschen, etc. in $O(1)$ Zeit
- **kein** Wahlfreier Zugriff

Weitere Datenstrukturen

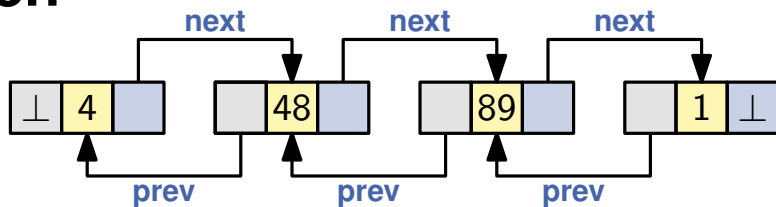
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

Listen



- Einfügen, Löschen, etc. in $O(1)$ Zeit
- **kein** Wahlfreier Zugriff

Weitere Datenstrukturen

- Stack

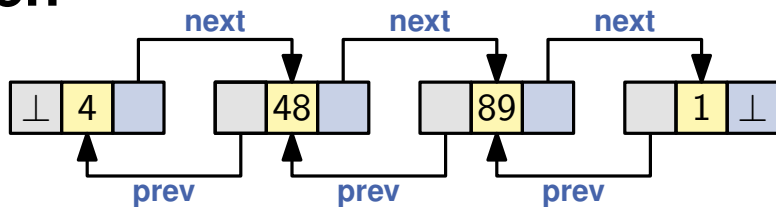
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

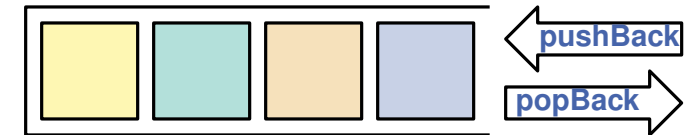
Listen



- Einfügen, Löschen, etc. in $O(1)$ Zeit
- **kein** Wahlfreier Zugriff

Weitere Datenstrukturen

- Stack



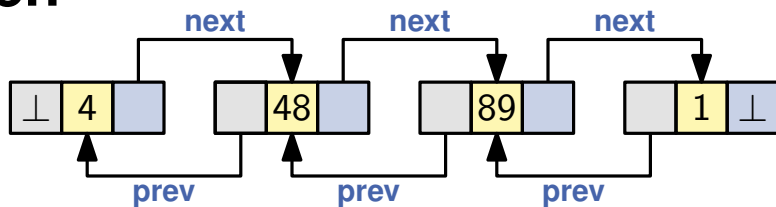
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

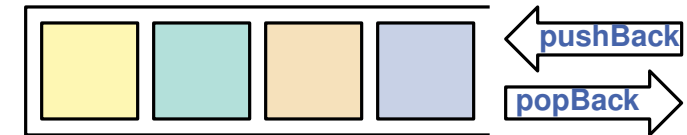
Listen



- Einfügen, Löschen, etc. in $O(1)$ Zeit
- **kein** Wahlfreier Zugriff

Weitere Datenstrukturen

- Stack



- Queue

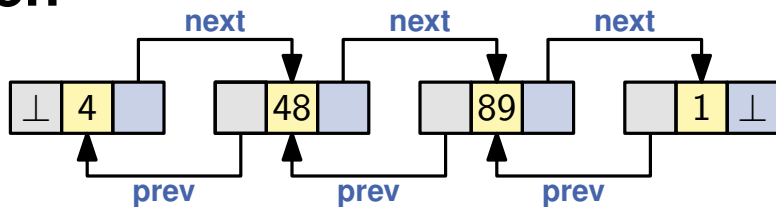
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

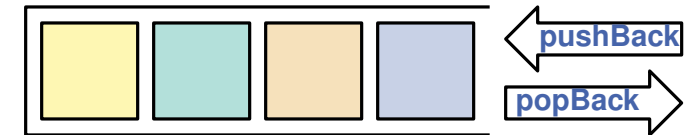
Listen



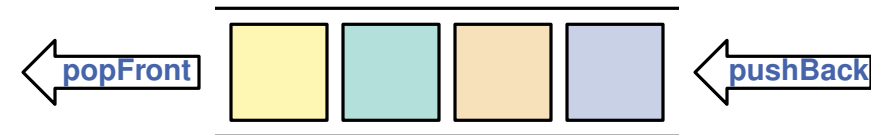
- Einfügen, Löschen, etc. in $O(1)$ Zeit
- **kein** Wahlfreier Zugriff

Weitere Datenstrukturen

- Stack



- Queue



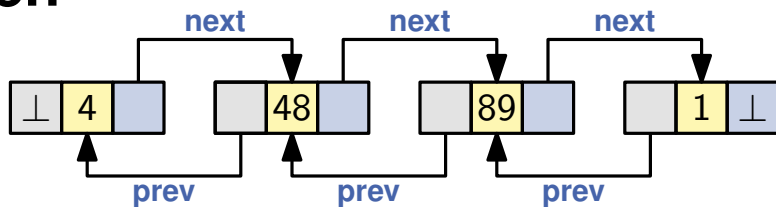
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

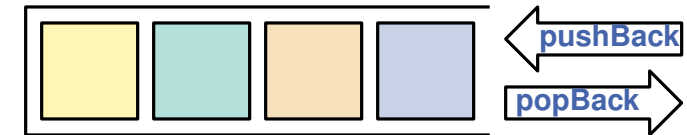
Listen



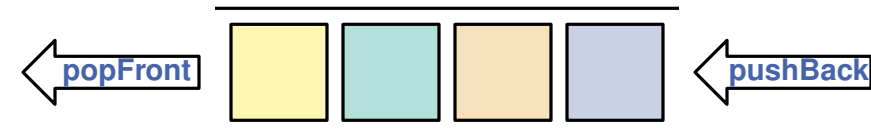
- Einfügen, Löschen, etc. in $O(1)$ Zeit
- **kein** Wahlfreier Zugriff

Weitere Datenstrukturen

- Stack



- Queue



- Double-ended queue

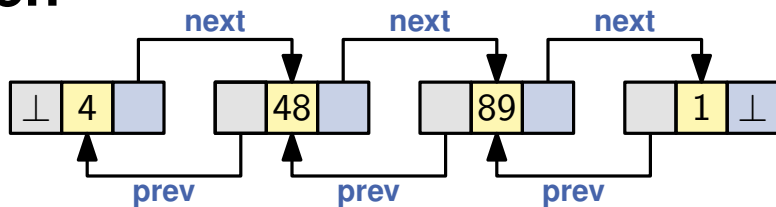
Datenstrukturen bisher in Algo 1

Array (dynamisch / unbeschränkt)

4	48	89	1	0	9	13	7	32	76	17	5
---	----	----	---	---	---	----	---	----	----	----	---

- *wahlfreier* Zugriff (*random access*)
 - Schreiben / Lesen von Wert an Stelle i in $O(1)$ Zeit, z.B. $A[i] := 5$
- **pushBack()** in $O(1)$ Zeit (amortisiert)

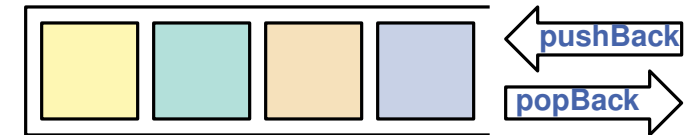
Listen



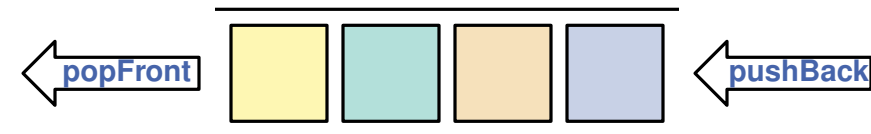
- Einfügen, Löschen, etc. in $O(1)$ Zeit
- **kein** Wahlfreier Zugriff

Weitere Datenstrukturen

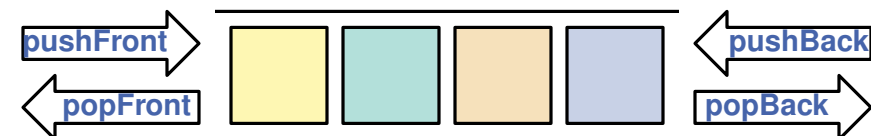
Stack



Queue



Double-ended queue



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

Motivation: Datenstrukturen

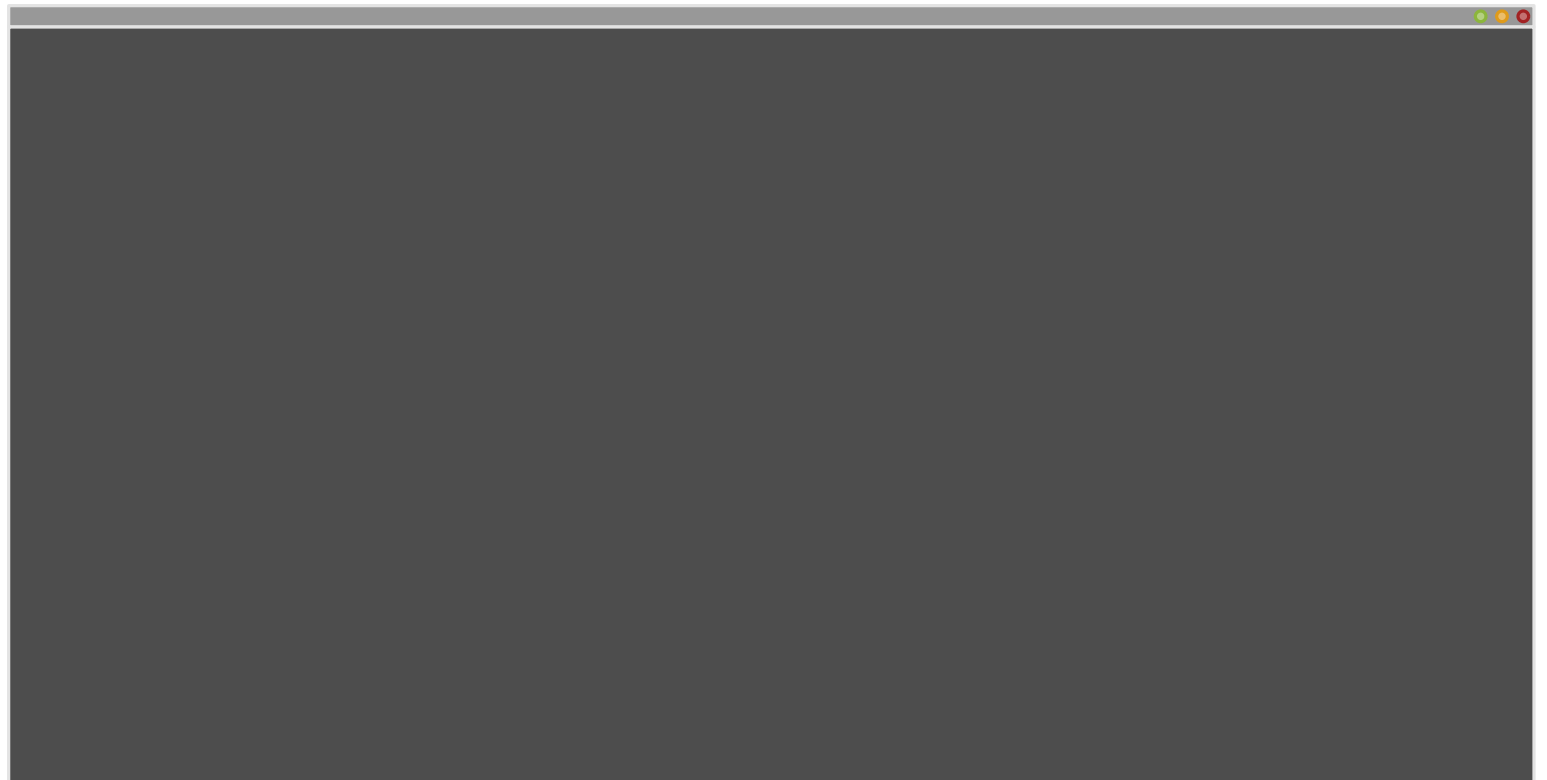
Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers

Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

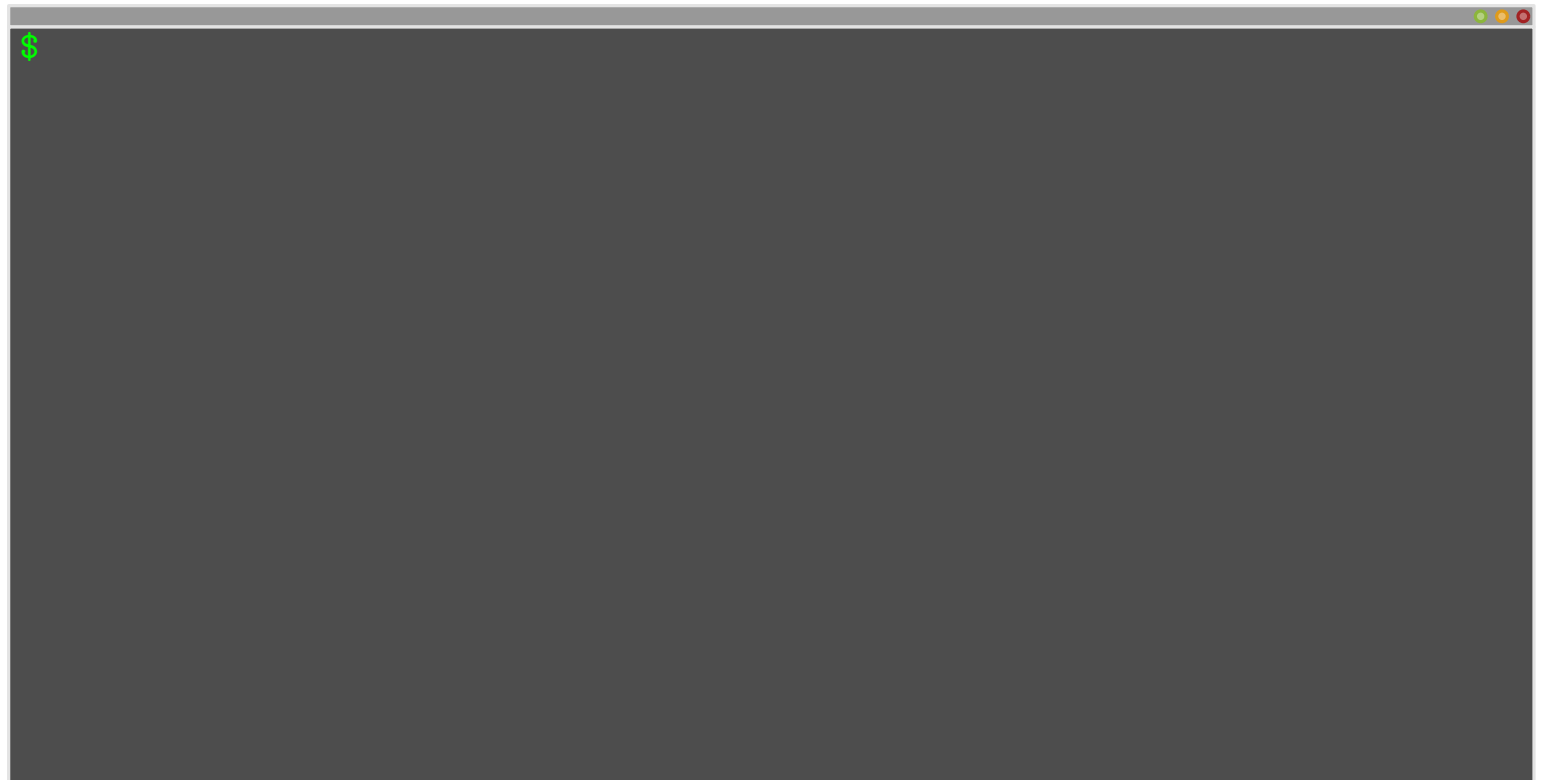
- Abstraktion des Speichers



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers

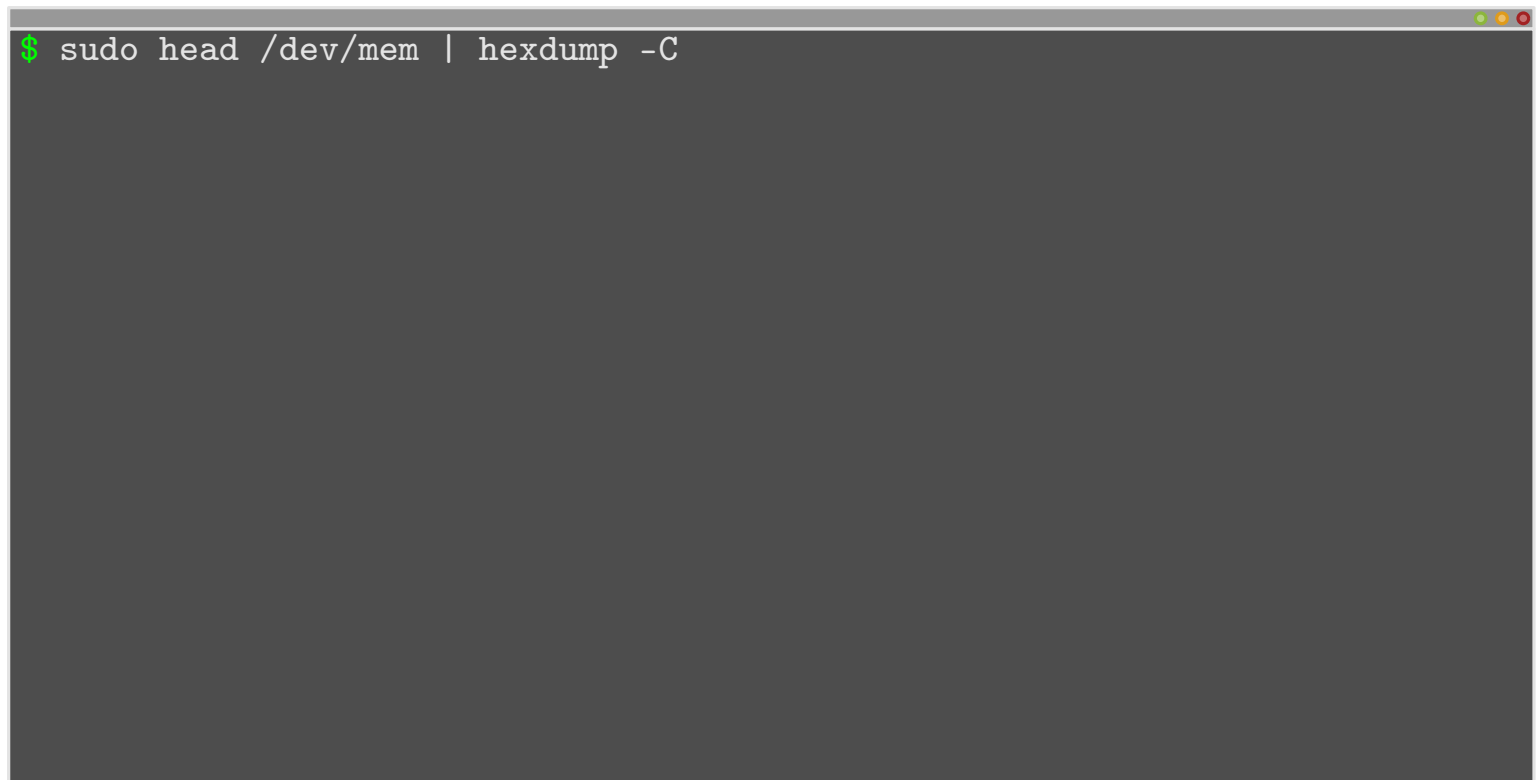
```
$ ls /dev/mem
/dev/mem
```

Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers

```
$ sudo head /dev/mem | hexdump -C
```



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers

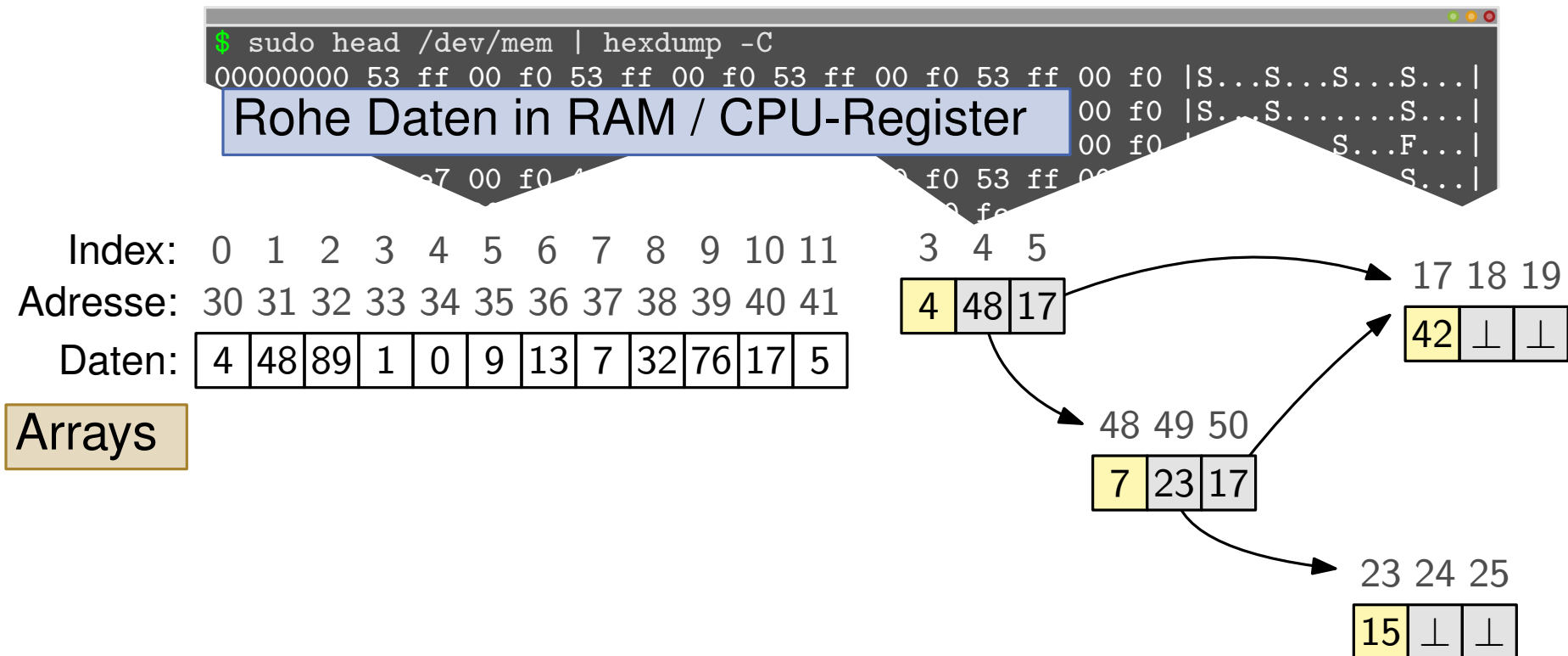
```

$ sudo head /dev/mem | hexdump -C
00000000 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
00000010 53 ff 00 f0 53 ff 00 f0 cc e9 00 f0 53 ff 00 f0 |S...S.....S...|
00000020 a5 fe 00 f0 87 e9 00 f0 53 ff 00 f0 46 e7 00 f0 |.....S...F...|
00000030 46 e7 00 f0 46 e7 00 f0 57 ef 00 f0 53 ff 00 f0 |F...F...W...S...|
00000040 22 00 00 c0 4d f8 00 f0 41 f8 00 f0 fe e3 00 f0 |"...M...A.....|
00000050 39 e7 00 f0 59 f8 00 f0 2e e8 00 f0 d4 ef 00 f0 |9...Y.....|
00000060 a4 f0 00 f0 f2 e6 00 f0 6e fe 00 f0 53 ff 00 f0 |.....n...S...|
00000070 ed ef 00 f0 53 ff 00 f0 c7 ef 00 f0 ed 57 00 c0 |...S.....W..|
00000080 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
*
00000100 59 ec 00 f0 3d 00 c0 9f 53 ff 00 f0 ed 69 00 c0 |Y...=...S....i..|
00000110 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
*
00000180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
000afea0 00 00 00 00 00 00 00 00 aa aa aa 00 aa aa aa 00 |.....|
000afeb0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
...
  
```


Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

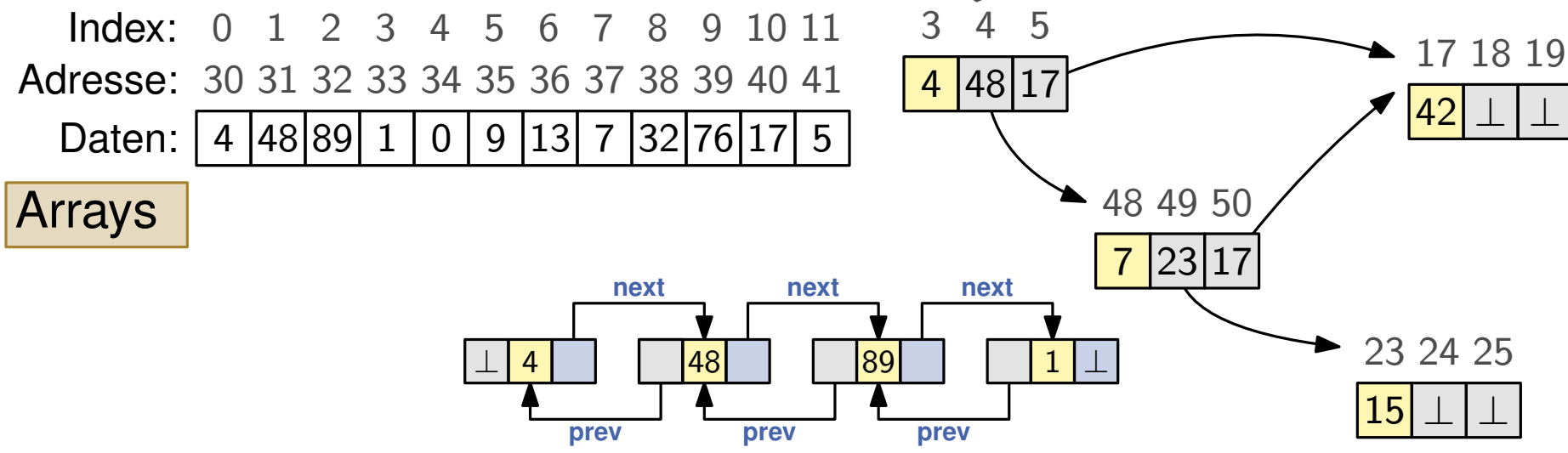
- Abstraktion des Speichers

```

$ sudo head /dev/mem | hexdump -C
00000000 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
00000004 00 f0 |S...S.....S...|
00000008 00 f0 |S...F...|
0000000c 00 f0 |S...|
00000010 00 f0 |S...|
00000014 00 f0 |S...|
00000018 00 f0 |S...|
0000001c 00 f0 |S...|
00000020 00 f0 |S...|
00000024 00 f0 |S...|
00000028 00 f0 |S...|
0000002c 00 f0 |S...|
00000030 00 f0 |S...|
00000034 00 f0 |S...|
00000038 00 f0 |S...|
0000003c 00 f0 |S...|
00000040 00 f0 |S...|
00000044 00 f0 |S...|
00000048 00 f0 |S...|
0000004c 00 f0 |S...|
00000050 00 f0 |S...|
00000054 00 f0 |S...|
00000058 00 f0 |S...|
0000005c 00 f0 |S...|
00000060 00 f0 |S...|
00000064 00 f0 |S...|
00000068 00 f0 |S...|
0000006c 00 f0 |S...|
00000070 00 f0 |S...|
00000074 00 f0 |S...|
00000078 00 f0 |S...|
0000007c 00 f0 |S...|
00000080 00 f0 |S...|
00000084 00 f0 |S...|
00000088 00 f0 |S...|
0000008c 00 f0 |S...|
00000090 00 f0 |S...|
00000094 00 f0 |S...|
00000098 00 f0 |S...|
0000009c 00 f0 |S...|
000000a0 00 f0 |S...|
000000a4 00 f0 |S...|
000000a8 00 f0 |S...|
000000ac 00 f0 |S...|
000000b0 00 f0 |S...|
000000b4 00 f0 |S...|
000000b8 00 f0 |S...|
000000bc 00 f0 |S...|
000000c0 00 f0 |S...|
000000c4 00 f0 |S...|
000000c8 00 f0 |S...|
000000cc 00 f0 |S...|
000000d0 00 f0 |S...|
000000d4 00 f0 |S...|
000000d8 00 f0 |S...|
000000dc 00 f0 |S...|
000000e0 00 f0 |S...|
000000e4 00 f0 |S...|
000000e8 00 f0 |S...|
000000ec 00 f0 |S...|
000000f0 00 f0 |S...|
000000f4 00 f0 |S...|
000000f8 00 f0 |S...|
000000fc 00 f0 |S...|

```

Rohe Daten in RAM / CPU-Register



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

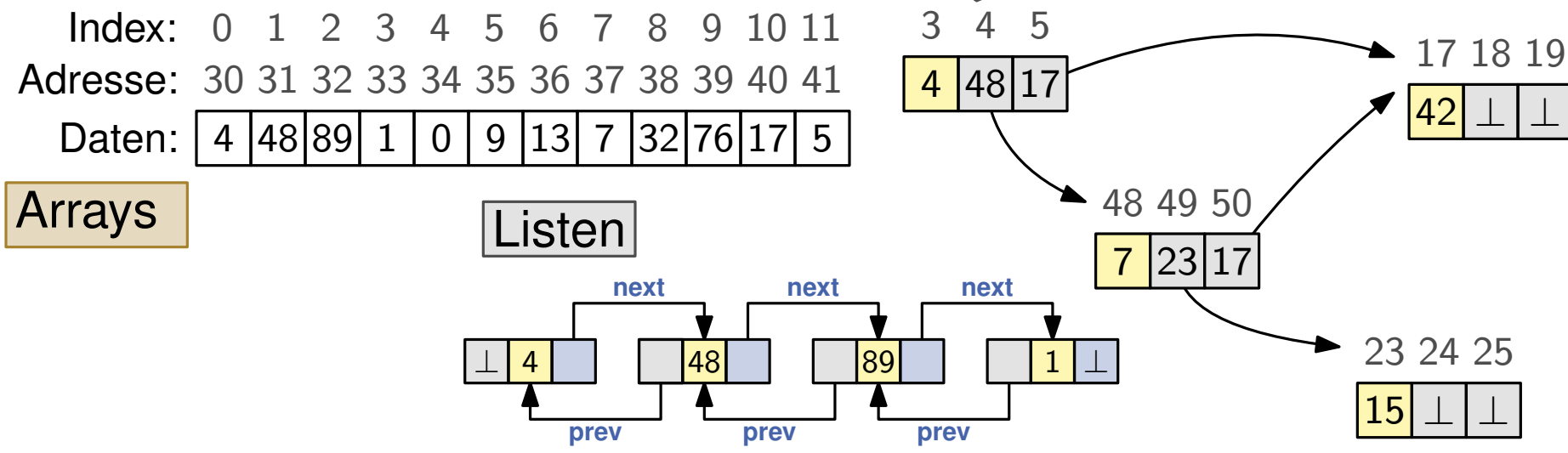
- Abstraktion des Speichers

```

$ sudo head /dev/mem | hexdump -C
00000000 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
00000004 00 f0 |S...S.....S...|
00000008 00 f0 |S...F...|
0000000c 00 f0 |S...|
00000010 00 f0 |S...|
00000014 00 f0 |S...|
00000018 00 f0 |S...|
0000001c 00 f0 |S...|
00000020 00 f0 |S...|
00000024 00 f0 |S...|
00000028 00 f0 |S...|
0000002c 00 f0 |S...|
00000030 00 f0 |S...|
00000034 00 f0 |S...|
00000038 00 f0 |S...|
0000003c 00 f0 |S...|
00000040 00 f0 |S...|
00000044 00 f0 |S...|
00000048 00 f0 |S...|
0000004c 00 f0 |S...|
00000050 00 f0 |S...|
00000054 00 f0 |S...|
00000058 00 f0 |S...|
0000005c 00 f0 |S...|
00000060 00 f0 |S...|
00000064 00 f0 |S...|
00000068 00 f0 |S...|
0000006c 00 f0 |S...|
00000070 00 f0 |S...|
00000074 00 f0 |S...|
00000078 00 f0 |S...|
0000007c 00 f0 |S...|
00000080 00 f0 |S...|
00000084 00 f0 |S...|
00000088 00 f0 |S...|
0000008c 00 f0 |S...|
00000090 00 f0 |S...|
00000094 00 f0 |S...|
00000098 00 f0 |S...|
0000009c 00 f0 |S...|
000000a0 00 f0 |S...|
000000a4 00 f0 |S...|
000000a8 00 f0 |S...|
000000ac 00 f0 |S...|
000000b0 00 f0 |S...|
000000b4 00 f0 |S...|
000000b8 00 f0 |S...|
000000bc 00 f0 |S...|
000000c0 00 f0 |S...|
000000c4 00 f0 |S...|
000000c8 00 f0 |S...|
000000cc 00 f0 |S...|
000000d0 00 f0 |S...|
000000d4 00 f0 |S...|
000000d8 00 f0 |S...|
000000dc 00 f0 |S...|
000000e0 00 f0 |S...|
000000e4 00 f0 |S...|
000000e8 00 f0 |S...|
000000ec 00 f0 |S...|
000000f0 00 f0 |S...|
000000f4 00 f0 |S...|
000000f8 00 f0 |S...|
000000fc 00 f0 |S...|

```

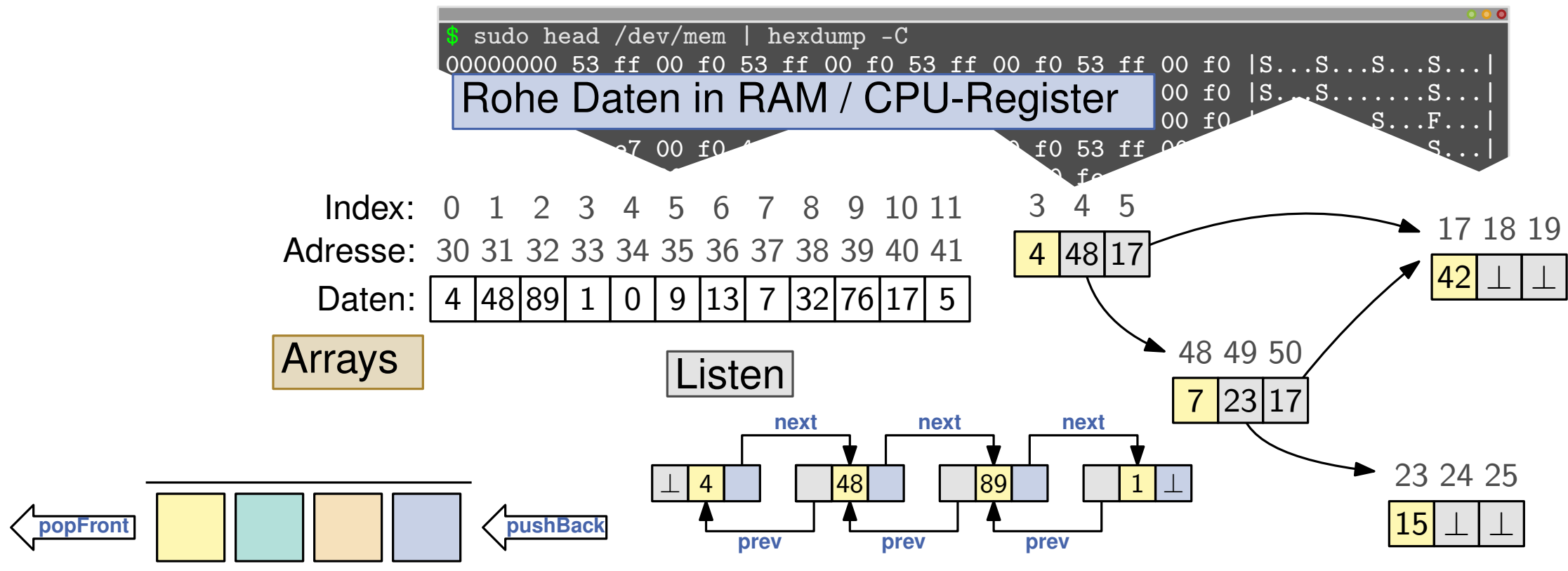
Rohe Daten in RAM / CPU-Register



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers
 - Rad nicht neu erfinden

```

$ sudo head /dev/mem | hexdump -C
00000000 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
00000004 00 f0 |S...S.....S...|
00000008 00 f0 |S...F...|
0000000c 00 f0 |S...|
00000010 00 f0 |S...|
00000014 00 f0 |S...|
00000018 00 f0 |S...|
0000001c 00 f0 |S...|
00000020 00 f0 |S...|
00000024 00 f0 |S...|
00000028 00 f0 |S...|
0000002c 00 f0 |S...|
00000030 00 f0 |S...|
00000034 00 f0 |S...|
00000038 00 f0 |S...|
0000003c 00 f0 |S...|
00000040 00 f0 |S...|
00000044 00 f0 |S...|
00000048 00 f0 |S...|
0000004c 00 f0 |S...|
00000050 00 f0 |S...|
00000054 00 f0 |S...|
00000058 00 f0 |S...|
0000005c 00 f0 |S...|
00000060 00 f0 |S...|
00000064 00 f0 |S...|
00000068 00 f0 |S...|
0000006c 00 f0 |S...|
00000070 00 f0 |S...|
00000074 00 f0 |S...|
00000078 00 f0 |S...|
0000007c 00 f0 |S...|
00000080 00 f0 |S...|
00000084 00 f0 |S...|
00000088 00 f0 |S...|
0000008c 00 f0 |S...|
00000090 00 f0 |S...|
00000094 00 f0 |S...|
00000098 00 f0 |S...|
0000009c 00 f0 |S...|
000000a0 00 f0 |S...|
000000a4 00 f0 |S...|
000000a8 00 f0 |S...|
000000ac 00 f0 |S...|
000000b0 00 f0 |S...|
000000b4 00 f0 |S...|
000000b8 00 f0 |S...|
000000bc 00 f0 |S...|
000000c0 00 f0 |S...|
000000c4 00 f0 |S...|
000000c8 00 f0 |S...|
000000cc 00 f0 |S...|
000000d0 00 f0 |S...|
000000d4 00 f0 |S...|
000000d8 00 f0 |S...|
000000dc 00 f0 |S...|
000000e0 00 f0 |S...|
000000e4 00 f0 |S...|
000000e8 00 f0 |S...|
000000ec 00 f0 |S...|
000000f0 00 f0 |S...|
000000f4 00 f0 |S...|
000000f8 00 f0 |S...|
000000fc 00 f0 |S...|

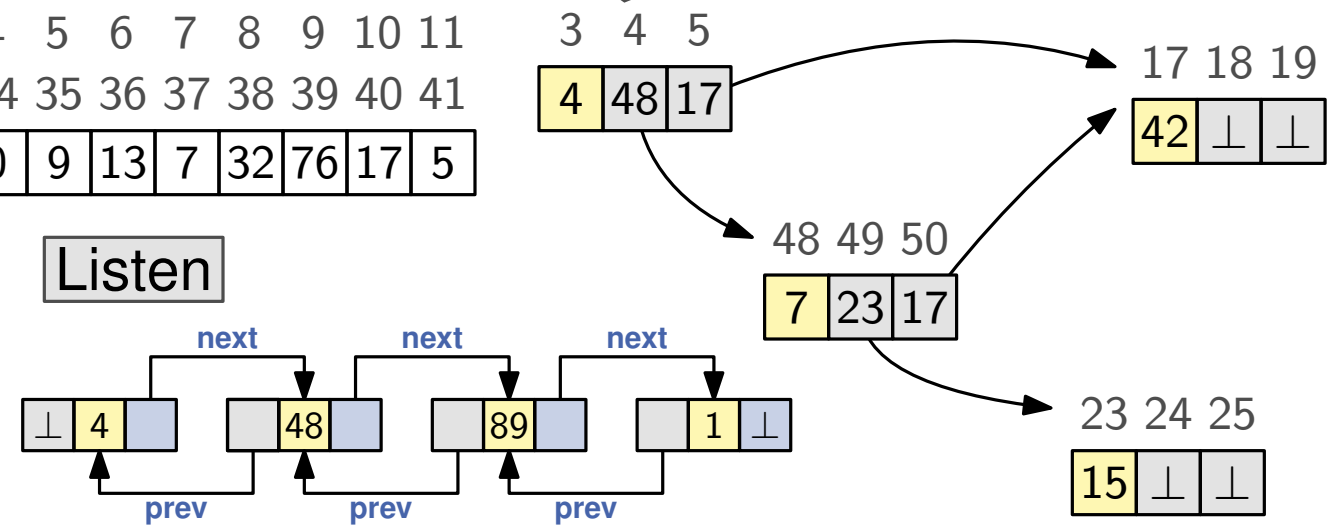
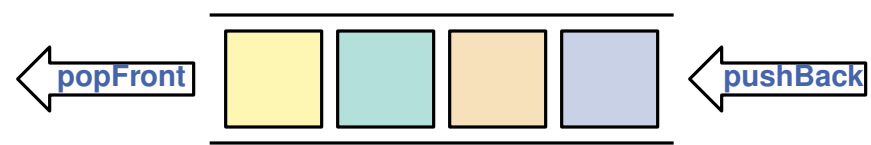
```

Rohe Daten in RAM / CPU-Register

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Adresse:	30	31	32	33	34	35	36	37	38	39	40	41
Daten:	4	48	89	1	0	9	13	7	32	76	17	5

Arrays

Listen



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers
 - Rad nicht neu erfinden
 - keine unnötigen Details bei Arbeit auf hohem Abstraktionslevel

```

$ sudo head /dev/mem | hexdump -C
00000000 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
00000010 00 f0 |S...S...S...S...|
00000020 00 f0 |S...F...S...|
00000030 00 f0 |S...|
00000040 00 f0 |S...|
00000050 00 f0 |S...|
00000060 00 f0 |S...|
00000070 00 f0 |S...|
00000080 00 f0 |S...|
00000090 00 f0 |S...|
000000a0 00 f0 |S...|
000000b0 00 f0 |S...|
000000c0 00 f0 |S...|
000000d0 00 f0 |S...|
000000e0 00 f0 |S...|
000000f0 00 f0 |S...|

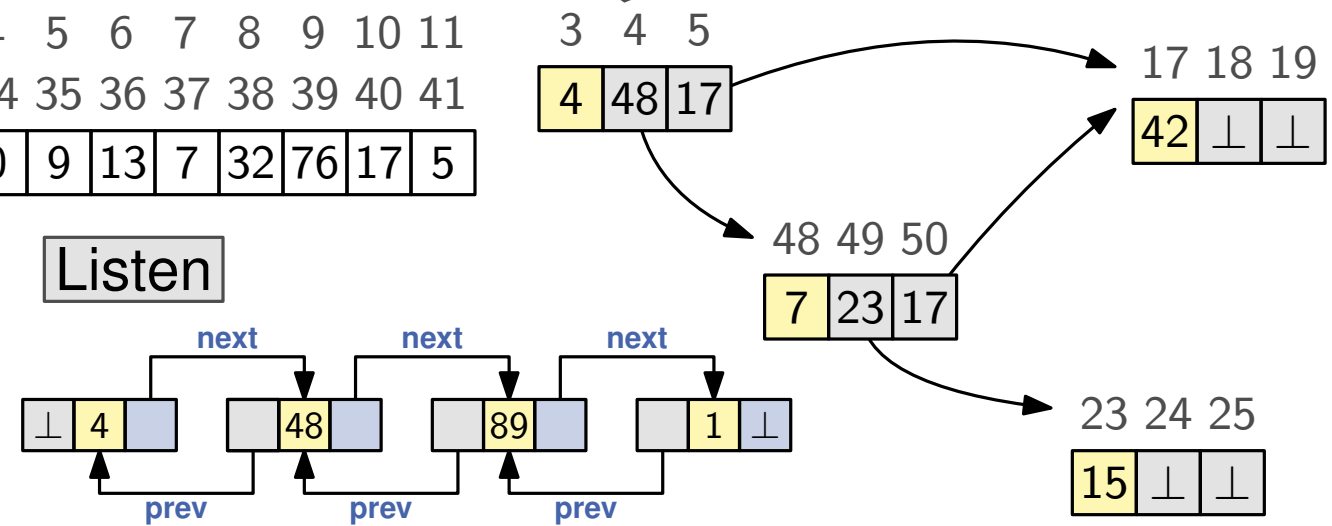
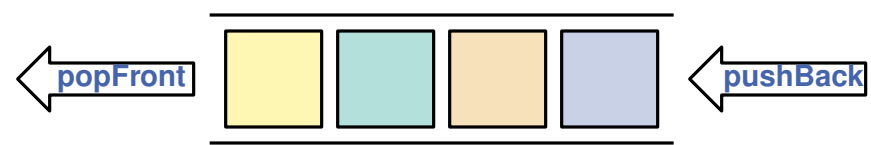
```

Rohe Daten in RAM / CPU-Register

Index:	0	1	2	3	4	5	6	7	8	9	10	11
Adresse:	30	31	32	33	34	35	36	37	38	39	40	41
Daten:	4	48	89	1	0	9	13	7	32	76	17	5

Arrays

Listen



Motivation: Datenstrukturen

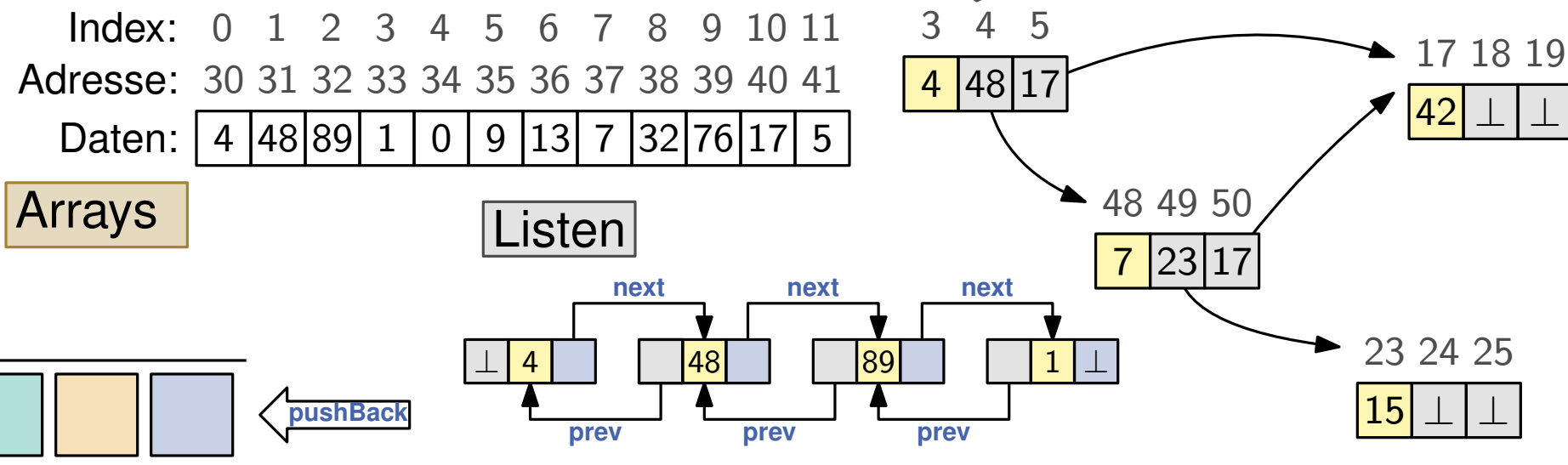
Was sind Datenstrukturen und wofür sind sie gut?

- Abstraktion des Speichers
 - Rad nicht neu erfinden
 - keine unnötigen Details bei Arbeit auf hohem Abstraktionslevel

```

$ sudo head /dev/mem | hexdump -C
00000000 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
00000010 00 f0 |S...S.....S...|
00000020 00 f0 |S...F...|
00000030 57 00 f0 40 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
    
```

Rohe Daten in RAM / CPU-Register



Laufzeit?



Motivation: Datenstrukturen

Was sind Datenstrukturen und wofür sind sie gut?

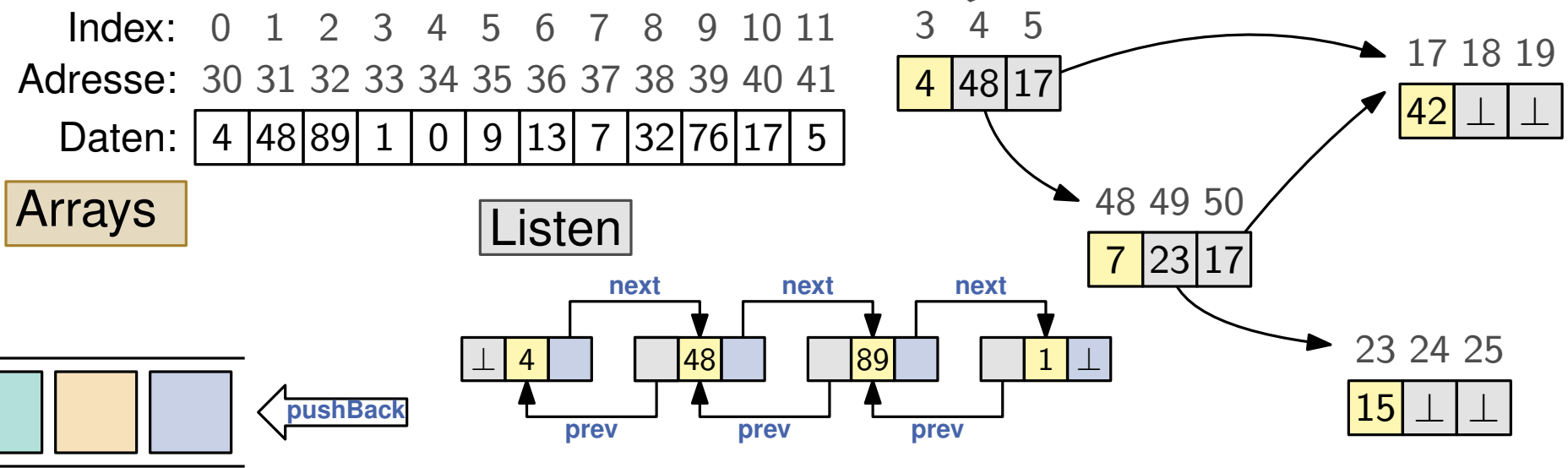
- Abstraktion des Speichers
 - Rad nicht neu erfinden
 - keine unnötigen Details bei Arbeit auf hohem Abstraktionslevel

```

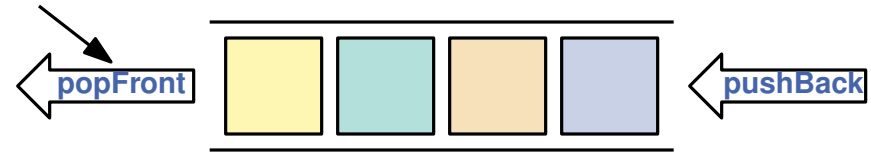
$ sudo head /dev/mem | hexdump -C
00000000 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 53 ff 00 f0 |S...S...S...S...|
00000004 00 f0 |S...S.....S...|
00000008 00 f0 |S...F...|
0000000c 00 f0 |S...|

```

Rohe Daten in RAM / CPU-Register



Laufzeit? $O(1)$



Motivation: Amortisierte Analyse

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

Beispiel: unbeschränktes Array

4	48		
---	----	--	--

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

Beispiel: unbeschränktes Array

4	48	89	
---	----	----	--

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

Beispiel: unbeschränktes Array

4	48	89	1
---	----	----	---

Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

Beispiel: unbeschränktes Array

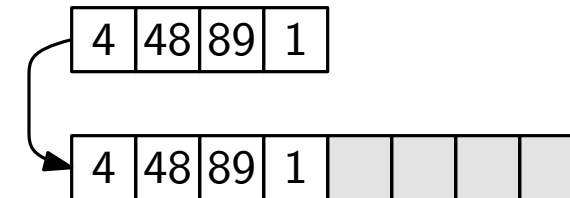
4	48	89	1
---	----	----	---



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

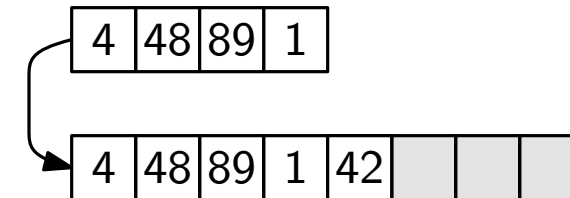
Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich

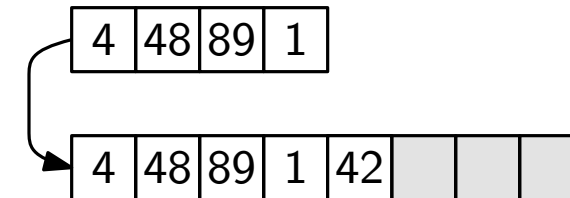
Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
 - z.B.: Operation meist günstig, seltener teurer

Beispiel: unbeschränktes Array



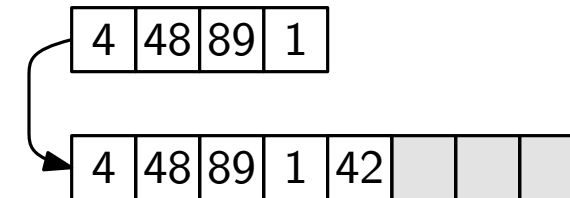
Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
 - z.B.: Operation meist günstig, seltener teurer

Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

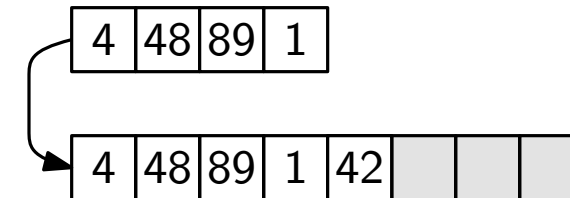
- Laufzeit von Operationen auf DS nicht immer gleich
 - z.B.: Operation meist günstig, seltener teurer

Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

Verschiedene Techniken zur Analyse

Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
 - z.B.: Operation meist günstig, seltener teurer

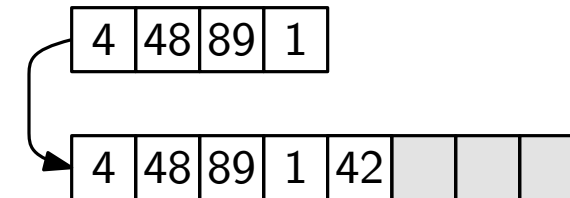
Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

Verschiedene Techniken zur Analyse

- Aggregation
- Charging
- Konto
- Potential

Beispiel: unbeschränktes Array



Motivation: Amortisierte Analyse

- Laufzeit von Operationen auf DS nicht immer gleich
 - z.B.: Operation meist günstig, seltener teurer

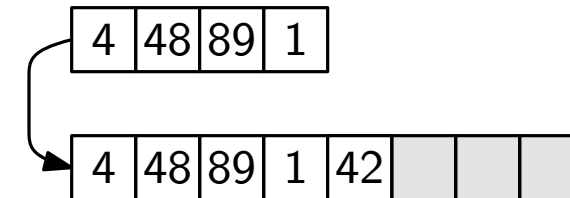
Idee

- **amortisierte Kosten**: durchschnittliche Kosten pro Operation in einer Folge von Operationen

Verschiedene Techniken zur Analyse

- Aggregation
- Charging
- Konto
- **Potential**

Beispiel: unbeschränktes Array



Beispiel: Binärzähler

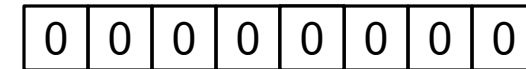
Idee

- Array A verwaltet Bits

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits



Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion `increment()` erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

Beispiel: Binärzähler

Idee

- Array *A* verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

Beispiel: Binärzähler

Idee

- Array *A* verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

Beispiel: Binärzähler

Idee

- Array *A* verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

Beispiel: Binärzähler

Idee

- Array *A* verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

Beispiel: Binärzähler

Idee

- Array *A* verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion `increment()` erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

Beispiel: Binärzähler

Idee

- Array *A* verwaltet Bits
- Funktion **increment()** erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion `increment()` erhöht Zähler

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion `increment()` erhöht Zähler

Algorithmus

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion `increment()` erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion `increment()` erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion `increment()` erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion `increment()` erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus

- suche Stelle i mit rechtester 0
- setze Stelle i auf 1
- setze Stellen rechts von i auf 0

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array *A* verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus in Pseudocode

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus in Pseudocode

increment()

```

i := 0
while  $A[i] = 1$  do
   $A[i] := 0$ 
   $i := i + 1$ 
 $A[i] := 1$ 

```

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Beispiel: Binärzähler

Idee

- Array A verwaltet Bits
- Funktion **increment()** erhöht Zähler

Algorithmus in Pseudocode

increment()

```

  i := 0
  while A[i] = 1 do
    A[i] := 0
    i := i + 1
  A[i] := 1

```

Laufzeit?

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Idee

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0	0
---	---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

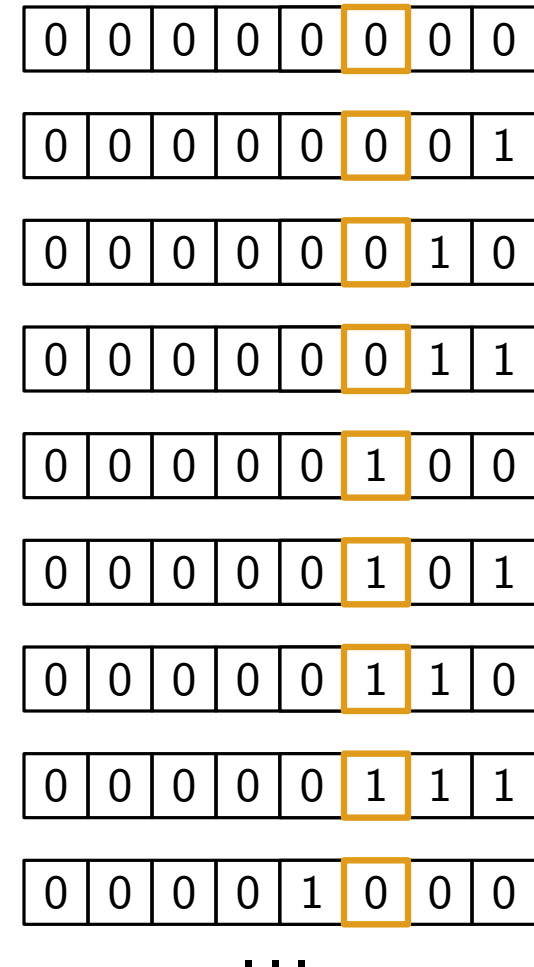
Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation



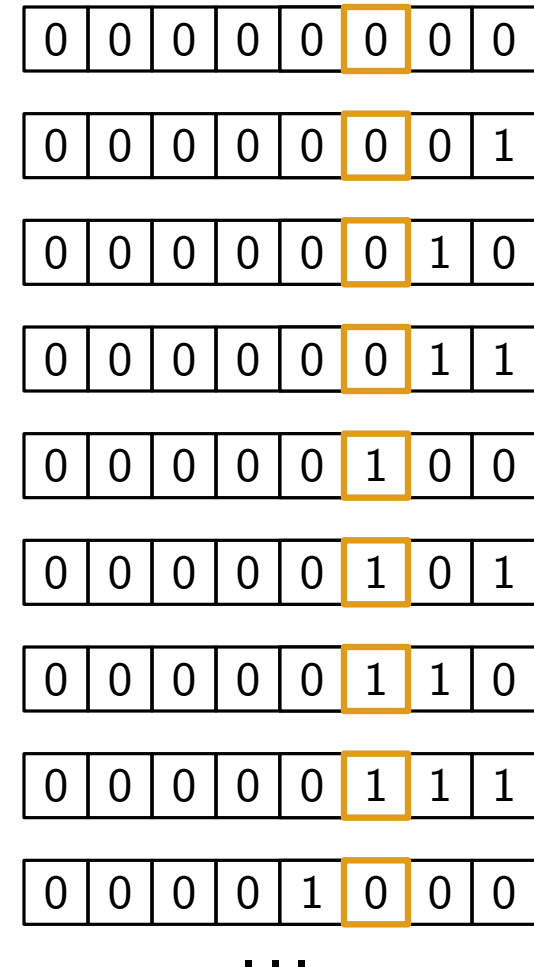
Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.



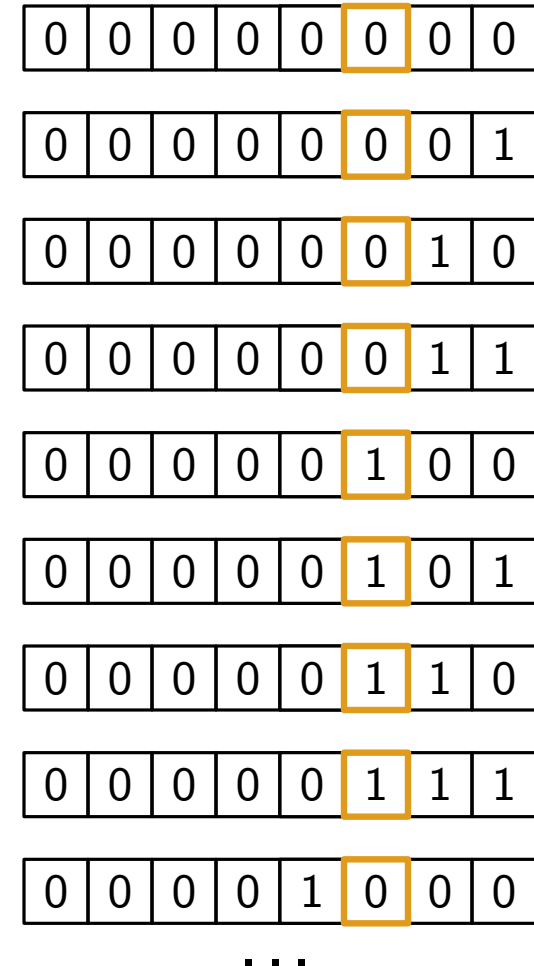
Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement



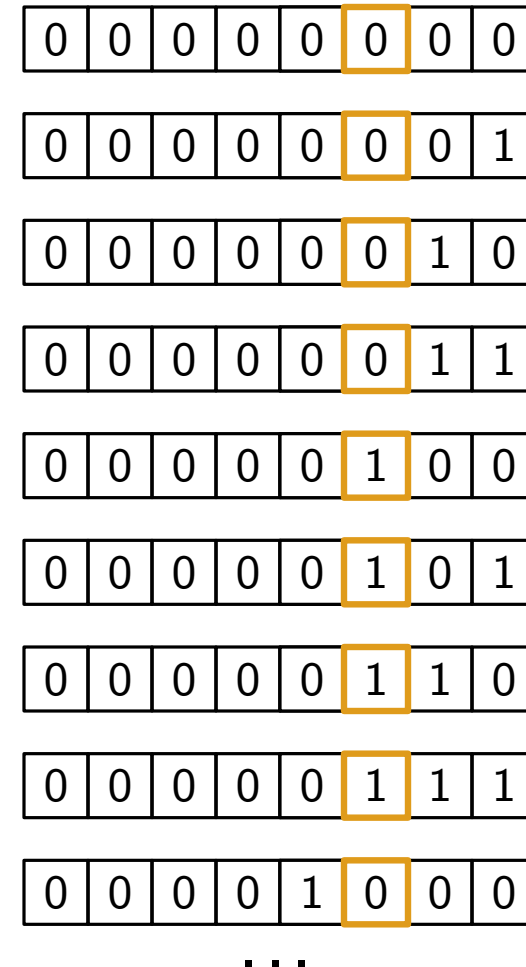
Analyse: Aggregatmethode

Idee

- berechne Gesamtkosten
- teile durch Anzahl von Operationen

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal



Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor$$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i}$$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \underbrace{\sum_{i=0}^{\infty} \frac{1}{2^i}} = 2n$$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \underbrace{\sum_{i=0}^{\infty} \frac{1}{2^i}}_{\square} = 2n$$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

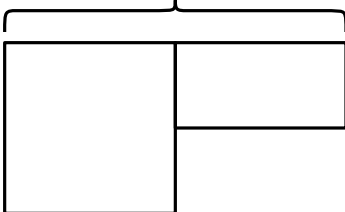
...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$


0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

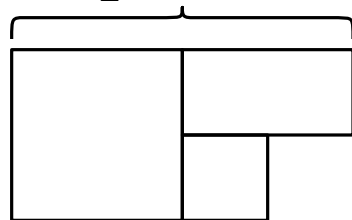
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

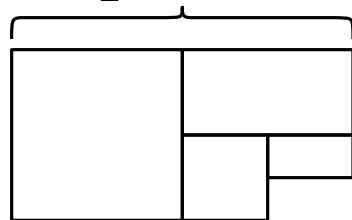
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

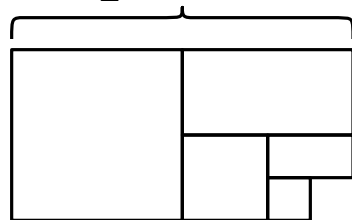
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

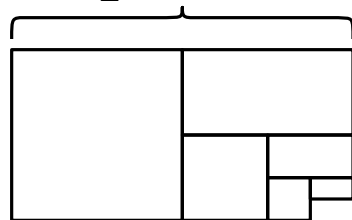
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

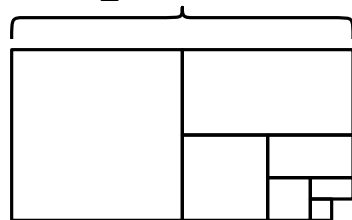
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

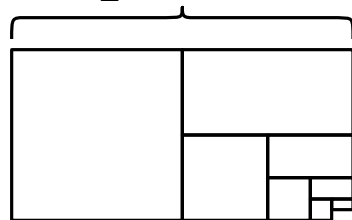
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

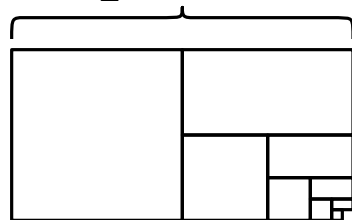
Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$



0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

D.h.: pro Operation nur 2 Bit-Flips

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Aggregatmethode

Beobachtung

- nicht jedes Bit flipt bei jeder Operation
- $A[0]$ flipt jedes mal, $A[1]$ jedes zweite, etc.
- $A[i]$ flipt bei jedem 2^i -ten Inkrement
- Bei n Aufrufen: $A[i]$ flipt $\lfloor \frac{n}{2^i} \rfloor$ mal

In Summe ergibt sich für k bits:

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

D.h.: pro Operation nur 2 Bit-Flips

Somit: **amortisierte Kosten** in $O(1)$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse mittels Charging

Analyse mittels Charging

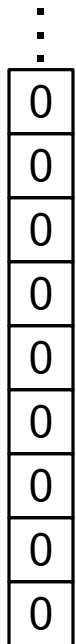
Idee

- teure Operationen legen Kosten um auf günstige Operationen

Analyse mittels Charging

Idee

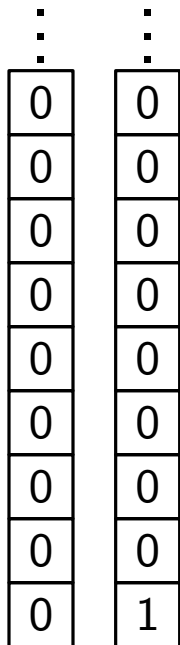
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

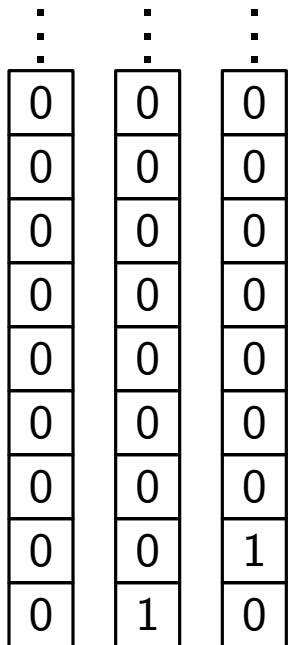
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

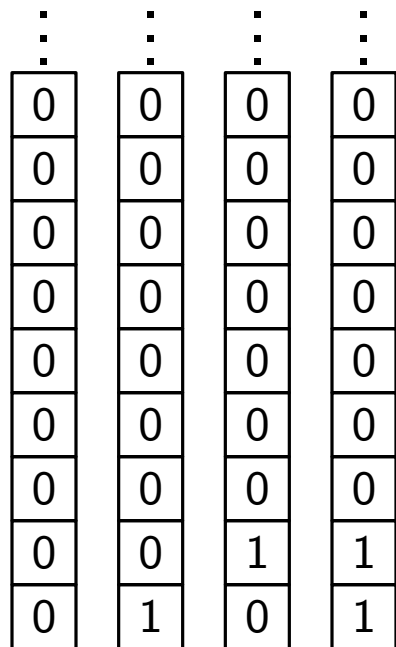
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

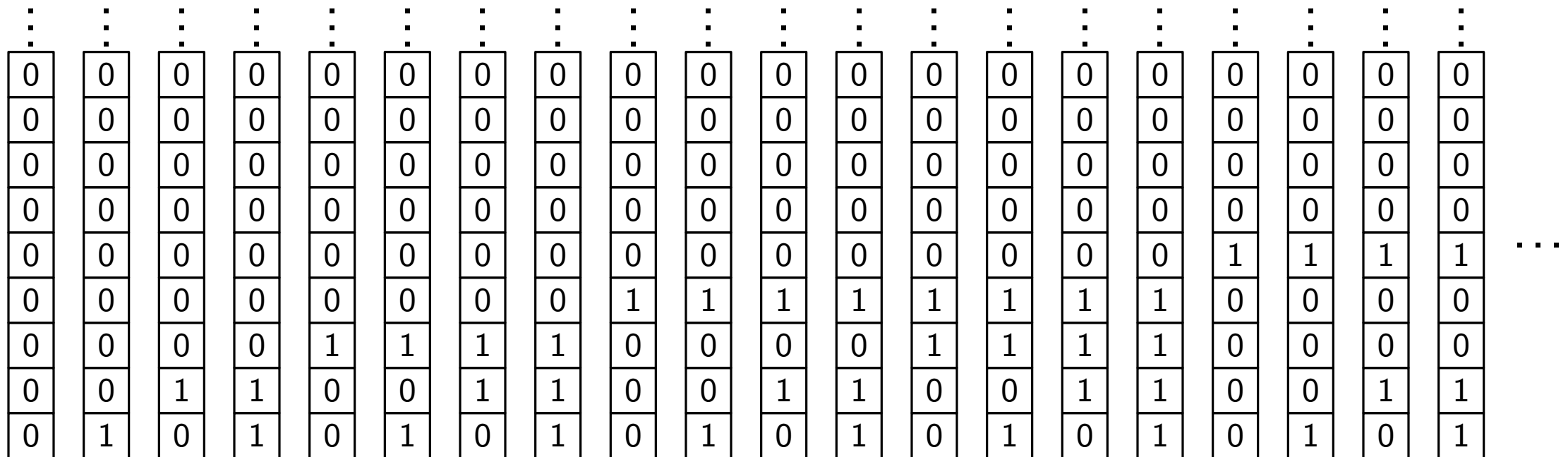
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

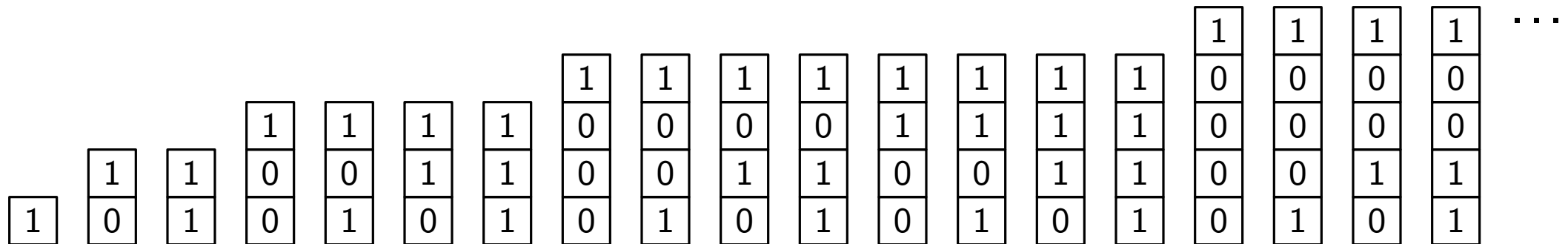
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

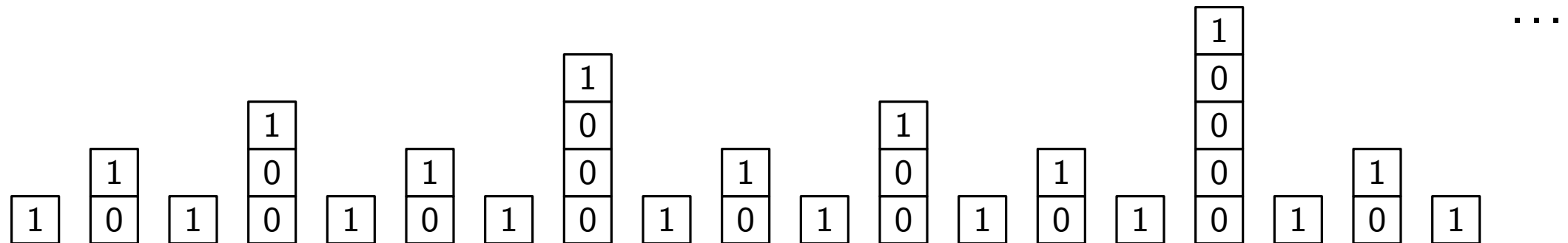
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

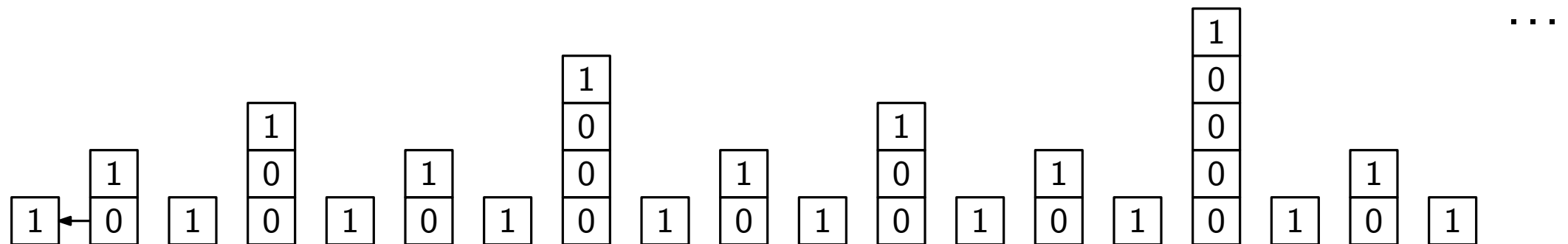
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

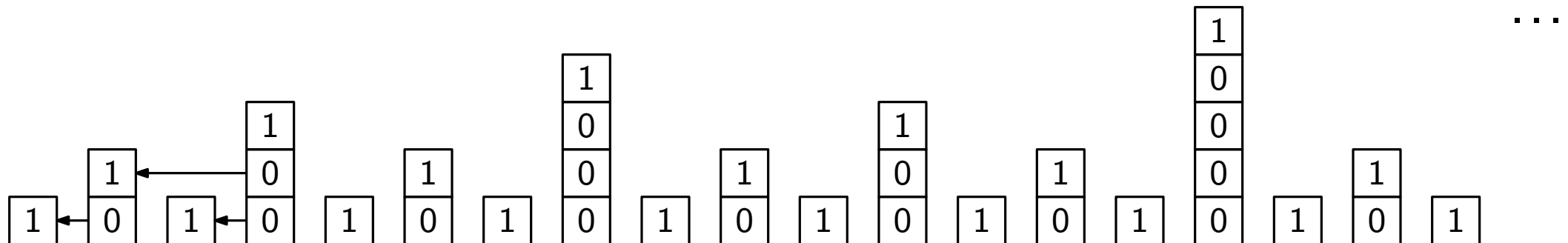
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

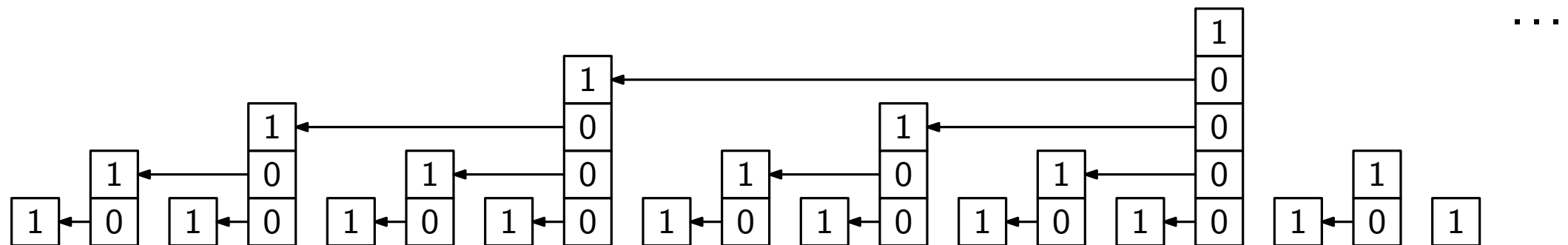
- teure Operationen legen Kosten um auf günstige Operationen



Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

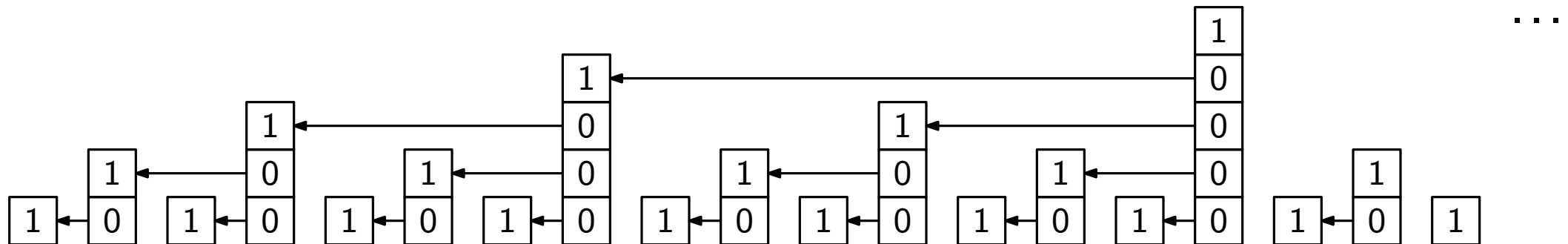


Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:



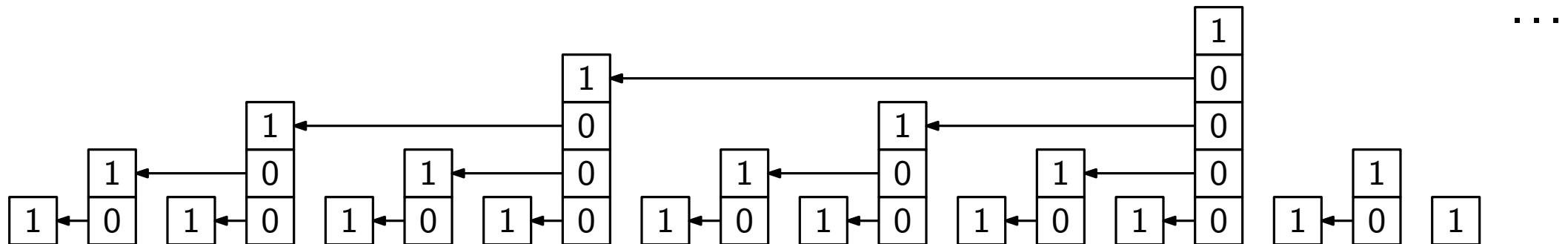
Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

- lege Kosten von 10-flip auf letzte Operation welche zuvor 01-flip ausgeführt hat



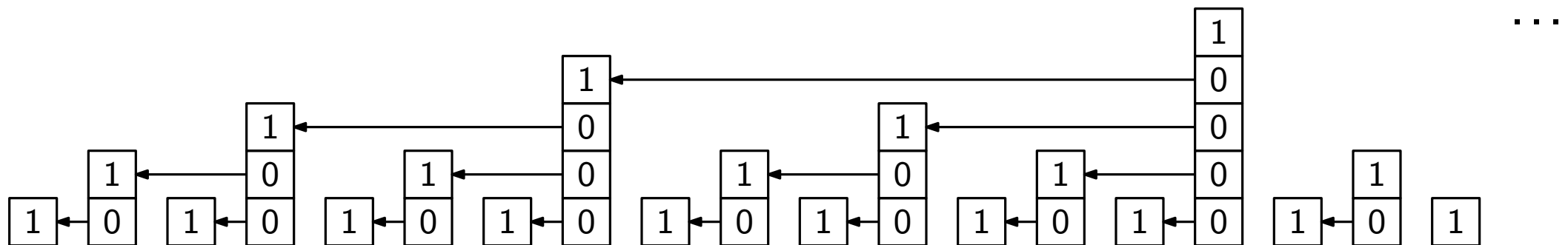
Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

- lege Kosten von 10-flip auf letzte Operation welche zuvor 01-flip ausgeführt hat
 - das geht für jeden 10-flip



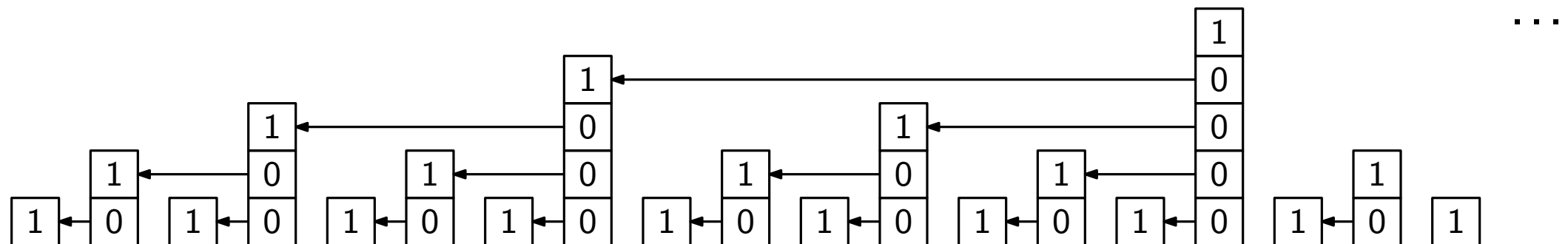
Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

- lege Kosten von 10-flip auf letzte Operation welche zuvor 01-flip ausgeführt hat
 - das geht für jeden 10-flip
- pro Operation übrig:



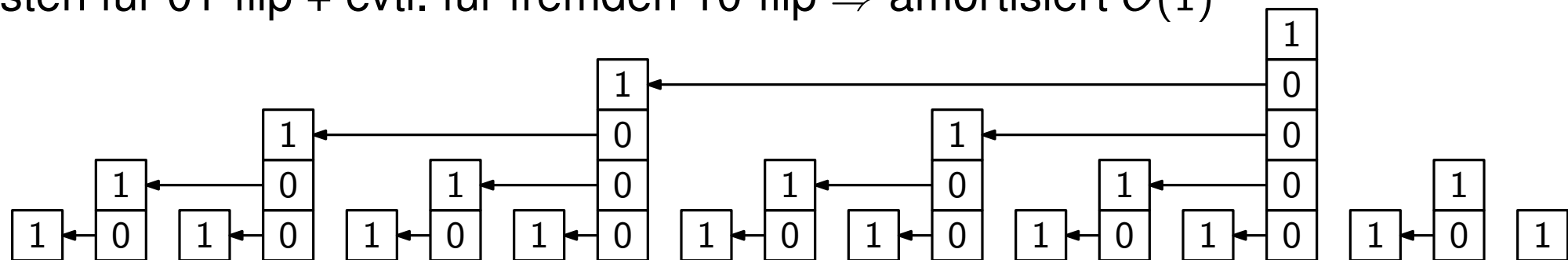
Analyse mittels Charging

Idee

- teure Operationen legen Kosten um auf günstige Operationen

Vorgehen hier:

- lege Kosten von 10-flip auf letzte Operation welche zuvor 01-flip ausgeführt hat
 - das geht für jeden 10-flip
- pro Operation übrig:
 - Kosten für 01-flip + evtl. für fremden 10-flip \Rightarrow amortisiert $O(1)$



Analyse: Potentialmethode

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

tatsächliche
Kosten



Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\text{Anstieg des Potenzials}}$

tatsächliche
Kosten

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\sum_{i=1}^n \hat{c}_i$$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n (c_i) + \sum_{i=1}^n (\Phi(D_i) - \Phi(D_{i-1}))$$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\begin{aligned}\sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n (c_i) + \sum_{i=1}^n (\Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^n (c_i) + \Phi(D_n) - \Phi(D_0)\end{aligned}$$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\begin{aligned}
 \sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n (c_i) + \sum_{i=1}^n (\Phi(D_i) - \Phi(D_{i-1})) \\
 &= \sum_{i=1}^n (c_i) + \Phi(D_n) - \Phi(D_0)
 \end{aligned}$$

Falls $\Phi(D_n) \geq \Phi(D_0)$: amortisierte Kosten obere Schranke für tatsächliche Kosten

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\begin{aligned}
 \sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n (c_i) + \sum_{i=1}^n (\Phi(D_i) - \Phi(D_{i-1})) \\
 &= \sum_{i=1}^n (c_i) + \Phi(D_n) - \Phi(D_0)
 \end{aligned}$$

Falls $\Phi(D_n) \geq \Phi(D_0)$: amortisierte Kosten obere Schranke für tatsächliche Kosten

\Rightarrow fordere $\Phi(D_i) \geq \Phi(D_0)$ f.a. $i > 0$

Analyse: Potentialmethode

Idee

- definiere Potentialfunktion $\Phi(D_i)$
 - Maß für Unaufgeräumtheit von D zum Zeitpunkt i
- definiere amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Definition sinnvoll?

$$\begin{aligned}
 \sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n (c_i) + \sum_{i=1}^n (\Phi(D_i) - \Phi(D_{i-1})) \\
 &= \sum_{i=1}^n (c_i) + \Phi(D_n) - \Phi(D_0)
 \end{aligned}$$

Falls $\Phi(D_n) \geq \Phi(D_0)$: amortisierte Kosten obere Schranke für tatsächliche Kosten

\Rightarrow fordere $\Phi(D_i) \geq \Phi(D_0)$ f.a. $i > 0$ oder: $\Phi(D_0) = 0$ und $\Phi(D_i) \geq 0$ f.a. $i > 0$

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$

gültige Pot.fun. :)

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tatsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tatsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tatsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$
- amort. Kosten: $\hat{c}_i =$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tatsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$
- amort. Kosten: $\hat{c}_i =$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tatsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$
- amort. Kosten: $\hat{c}_i = 2 \cdot \#01\text{-flips}$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Analyse: Potentialmethode

Potentialfunktion für Binärzähler:

- def. $\Phi(D_i) = \sum_{j=0}^{k-1} A[j]$ (Anzahl 1-bits zum Zeitpunkt i)

Dann ergibt sich:

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ *gültige Pot.fun. :)*
- tatsächliche Kosten: $c_i = \#01\text{-flips} + \#10\text{-flips}$
- Potentialänderung: $\#01\text{-flips} - \#10\text{-flips}$
- amort. Kosten: $\hat{c}_i = 2 \cdot \#01\text{-flips} \leq 2$

0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	0	1
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	0
---	---	---	---	---	---	---	---

0	0	0	0	0	1	1	1
---	---	---	---	---	---	---	---

0	0	0	0	1	0	0	0
---	---	---	---	---	---	---	---

...

Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Beispiel 2: Stacks und Queues

Beispiel 2: Stacks und Queues

Wiederholung

Beispiel 2: Stacks und Queues

Wiederholung

Queue

■ **push**(e) $O(1)$

■ **pop**() $O(1)$

■ **size**() $O(1)$

z.B. mittels verketteter Liste

Beispiel 2: Stacks und Queues

Wiederholung

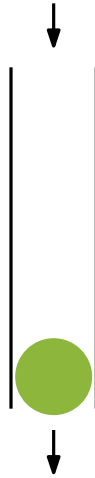
Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

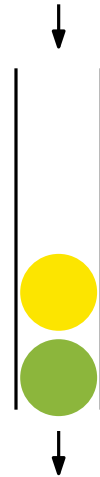
Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$

z.B. mittels verketteter Liste



Beispiel 2: Stacks und Queues

Wiederholung

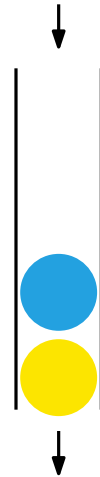
Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



Beispiel 2: Stacks und Queues

Wiederholung

Queue	
■ push (e)	$O(1)$
■ pop ()	$O(1)$
■ size ()	$O(1)$
<i>z.B. mittels verketteter Liste</i>	



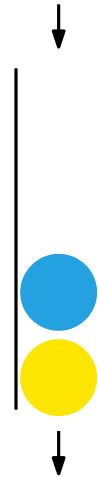
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste

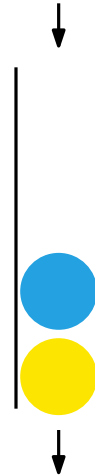
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

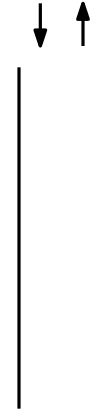
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



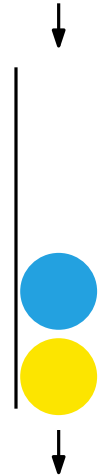
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

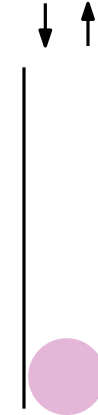
z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



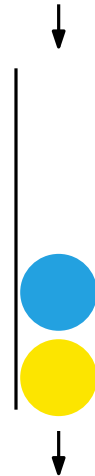
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



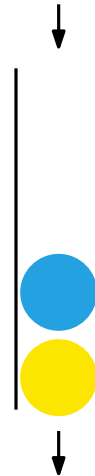
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



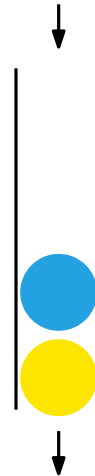
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



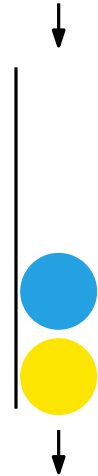
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



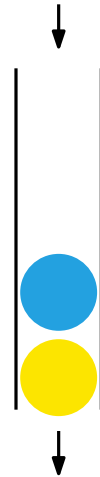
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



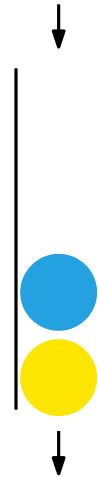
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Frage

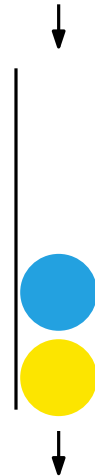
Beispiel 2: Stacks und Queues

Wiederholung

Queue

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste



Stack

- **push**(e) $O(1)$
- **pop**() $O(1)$
- **size**() $O(1)$

z.B. mittels verketteter Liste

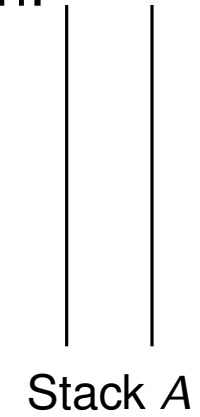


Frage

Angenommen wir können Stacks (als Blackbox) verwenden.
 (Wie) können wir daraus eine Queue bauen?

Eine Queue aus Stacks

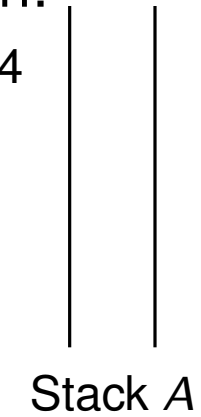
Operationen:



Eine Queue aus Stacks

Operationen:

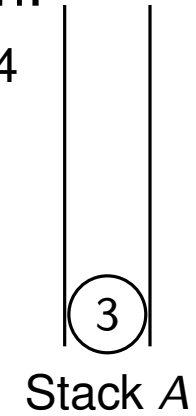
■ **push**: 3, 1, 4



Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4



Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

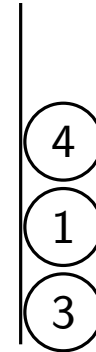


Stack A

Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4



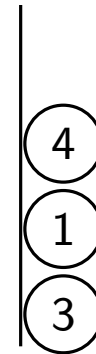
Stack A

Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

■ **pop**()



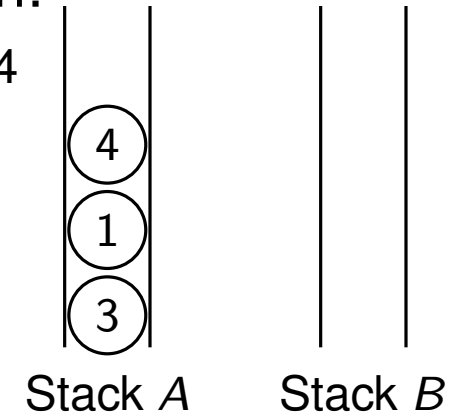
Stack A

Eine Queue aus Stacks

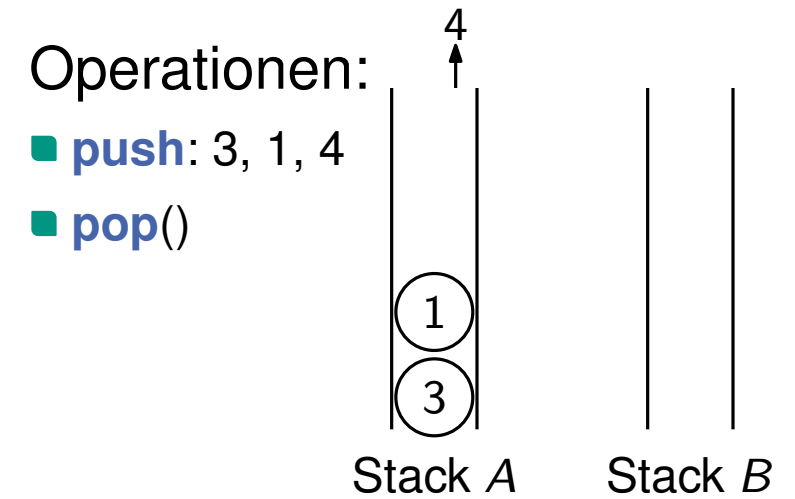
Operationen:

■ **push**: 3, 1, 4

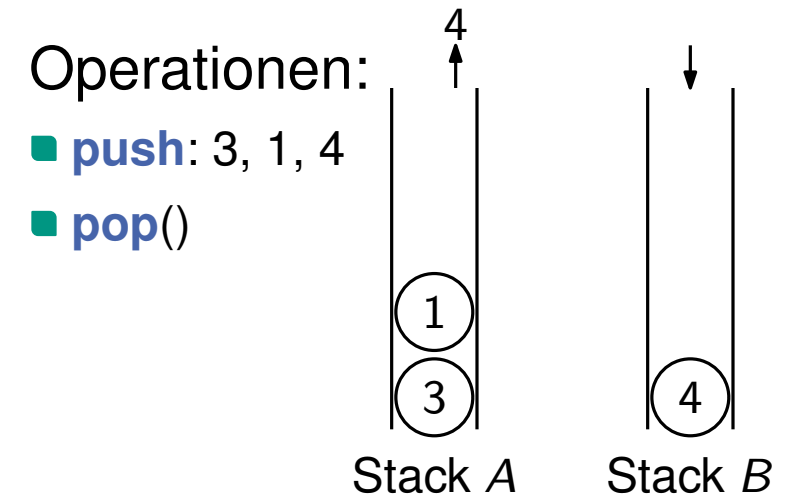
■ **pop**()



Eine Queue aus Stacks



Eine Queue aus Stacks

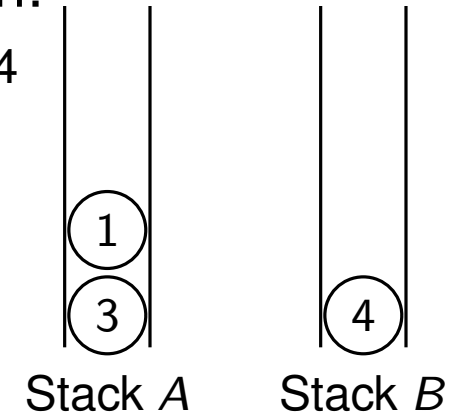


Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

■ **pop**()

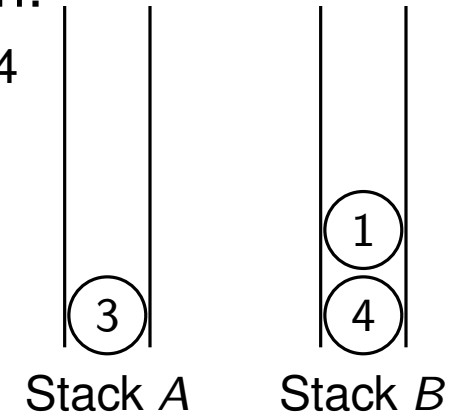


Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

■ **pop**()

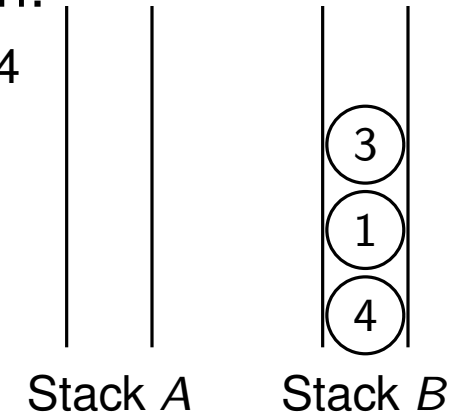


Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

■ **pop**()

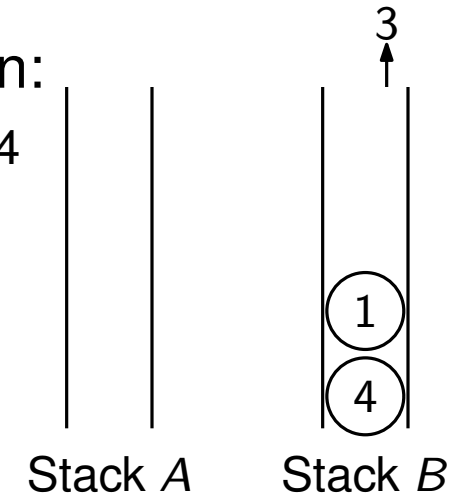


Eine Queue aus Stacks

Operationen:

■ **push**: 3, 1, 4

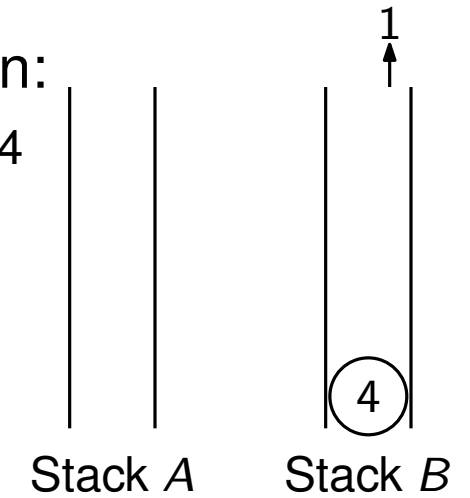
■ **pop**()



Eine Queue aus Stacks

Operationen:

- **push:** 3, 1, 4
- **pop()**
- **pop()**



Eine Queue aus Stacks

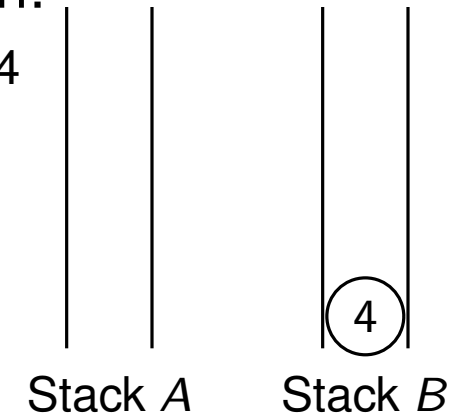
Operationen:

■ **push:** 3, 1, 4

■ **pop()**

■ **pop()**

■ **push:** 1,5



Eine Queue aus Stacks

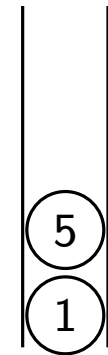
Operationen:

■ **push:** 3, 1, 4

■ **pop()**

■ **pop()**

■ **push:** 1,5



Stack A



Stack B

Eine Queue aus Stacks

Algorithmische Umsetzung

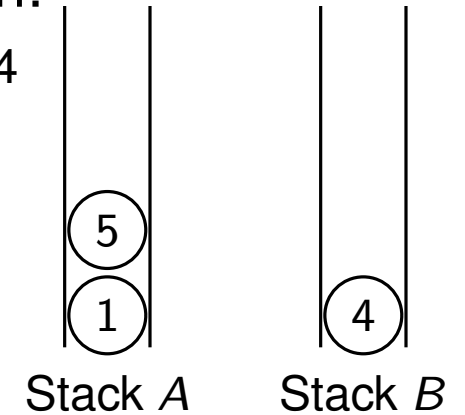
Operationen:

■ **push:** 3, 1, 4

■ **pop()**

■ **pop()**

■ **push:** 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen

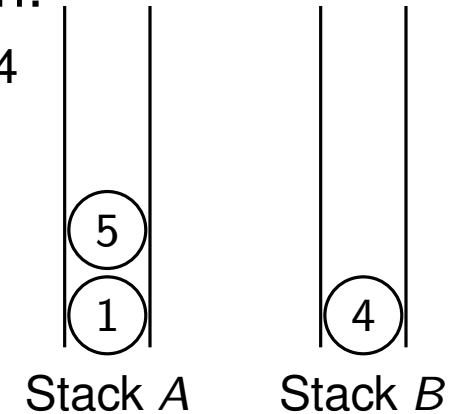
Operationen:

- **push**: 3, 1, 4

- **pop**()

- **pop**()

- **push**: 1,5



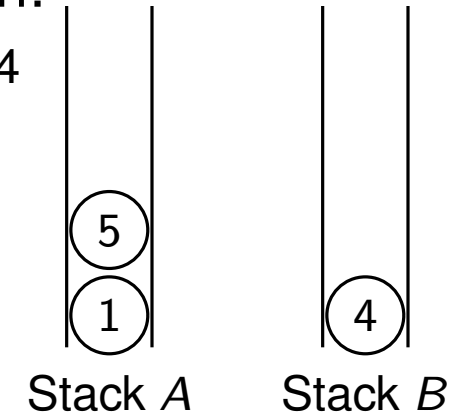
Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:

Operationen:

- **push**: 3, 1, 4
- **pop**()
- **pop**()
- **push**: 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:
 - falls B nicht leer: von B poppen

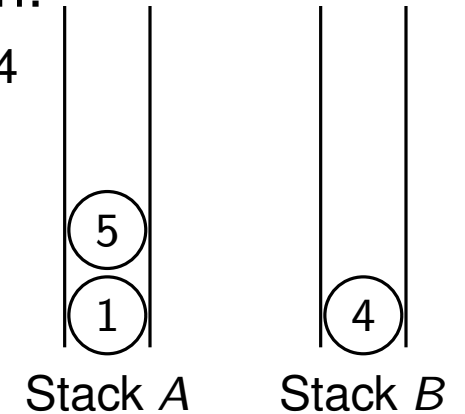
Operationen:

■ **push**: 3, 1, 4

■ **pop**()

■ **pop**()

■ **push**: 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:
 - falls B nicht leer: von B poppen
 - falls B leer: alles von A nach B verschieben

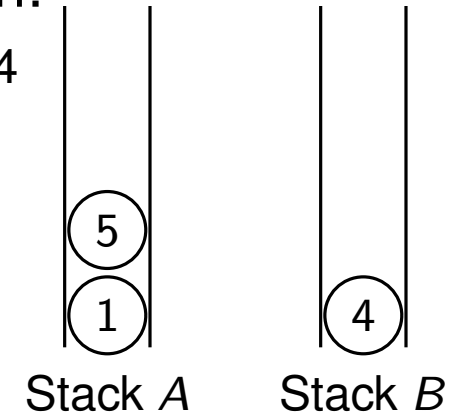
Operationen:

■ **push**: 3, 1, 4

■ **pop**()

■ **pop**()

■ **push**: 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:
 - falls B nicht leer: von B poppen
 - falls B leer: alles von A nach B verschieben

$O(1)$

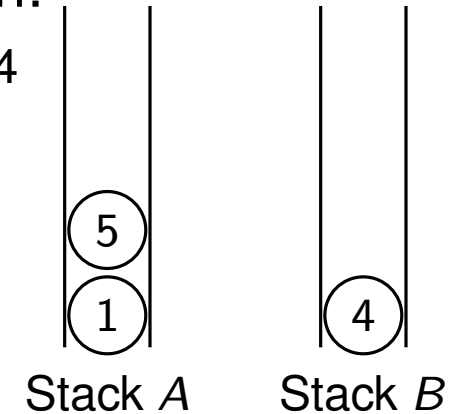
Operationen:

■ **push**: 3, 1, 4

■ **pop**()

■ **pop**()

■ **push**: 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:
 - falls B nicht leer: von B poppen
 - falls B leer: alles von A nach B verschieben

$O(1)$

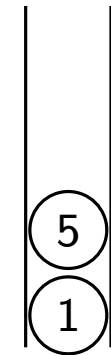
Operationen:

■ **push**: 3, 1, 4

■ **pop**()

■ **pop**()

■ **push**: 1,5



Stack A



Stack B

$O(1)$

Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:
 - falls B nicht leer: von B poppen
 - falls B leer: alles von A nach B verschieben

$O(1)$

$O(1)$

$O(|A|)$

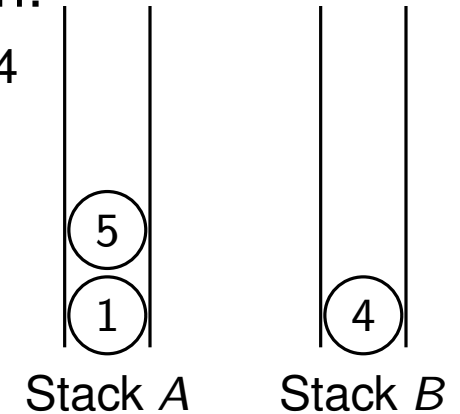
Operationen:

■ **push**: 3, 1, 4

■ **pop**()

■ **pop**()

■ **push**: 1,5



Eine Queue aus Stacks

Algorithmische Umsetzung

- **push**: auf A pushen
- **pop**:
 - falls B nicht leer: von B poppen
 - falls B leer: alles von A nach B verschieben

$O(1)$

$O(|A|)$

$O(1)$

$O(|A|)$

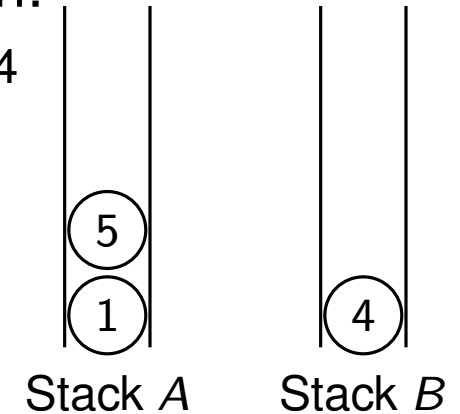
Operationen:

■ **push**: 3, 1, 4

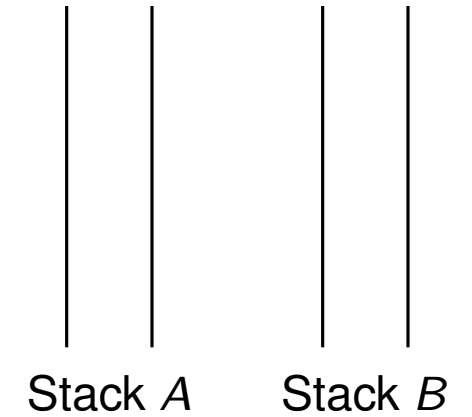
■ **pop**()

■ **pop**()

■ **push**: 1,5

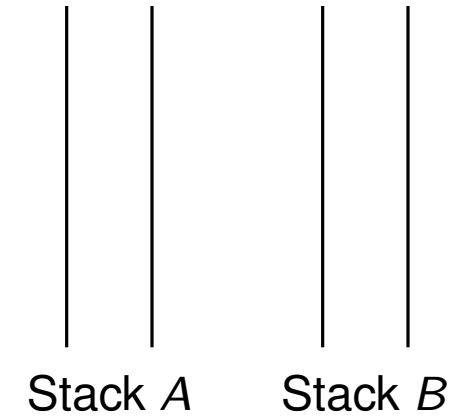


Analyse: Kontomethode



Analyse: Kontomethode

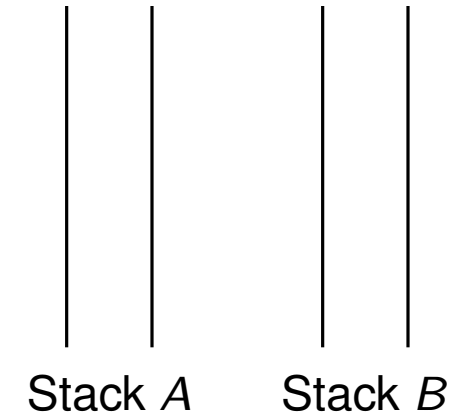
Idee



Analyse: Kontomethode

Idee

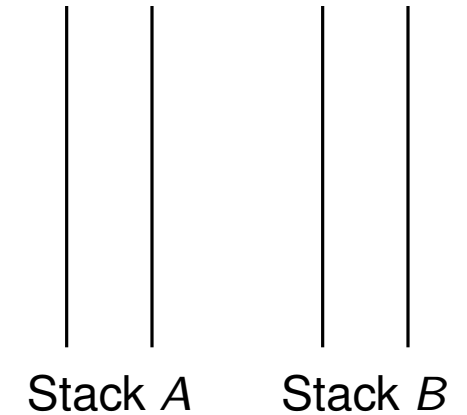
- günstige Operationen bauen Guthaben auf



Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

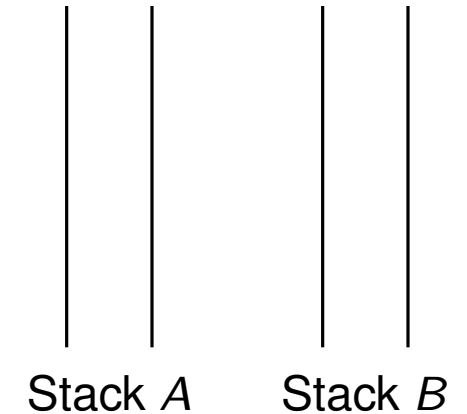


Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier



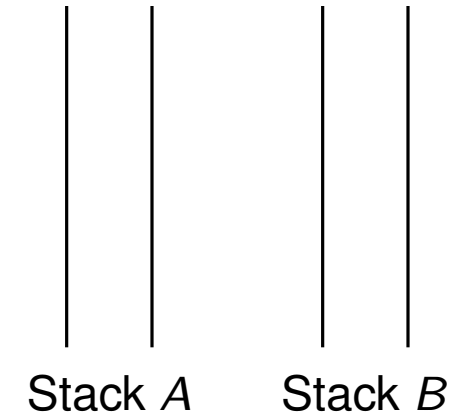
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen



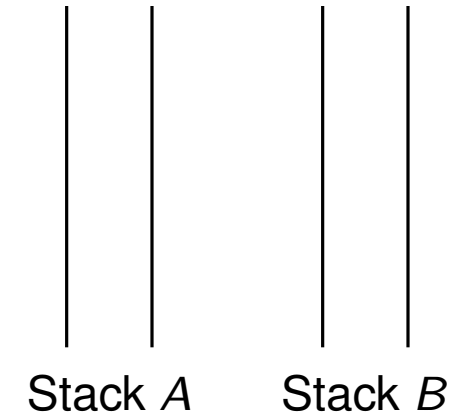
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



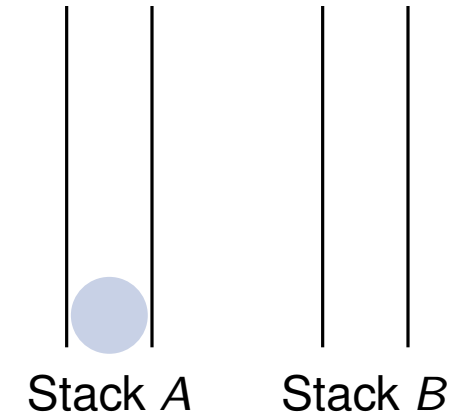
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



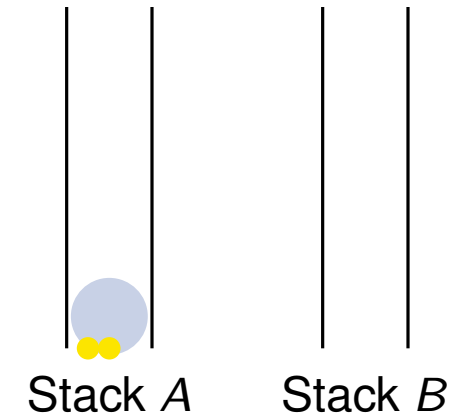
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



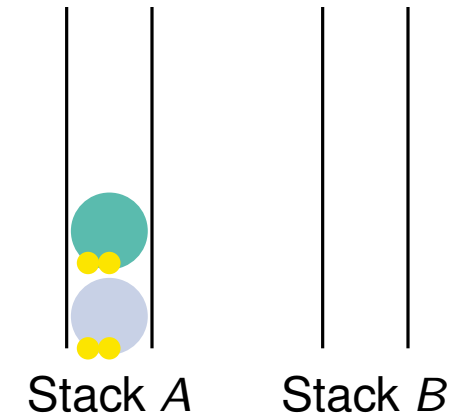
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



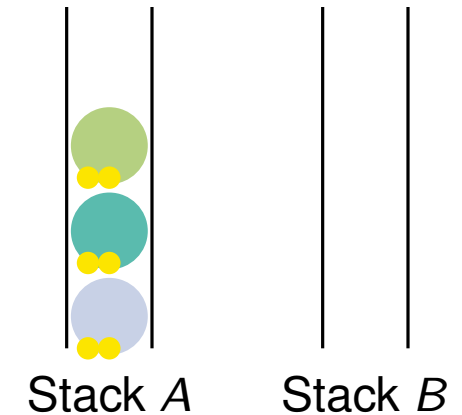
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3



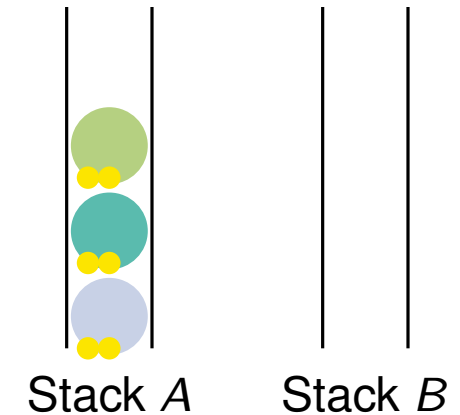
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1



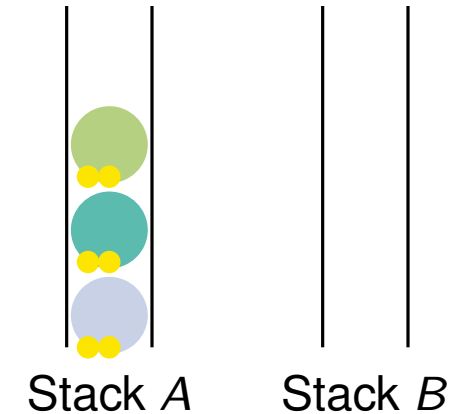
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls *B* leer: nutze 2 Guthaben für *A.pop()* und *B.push()*



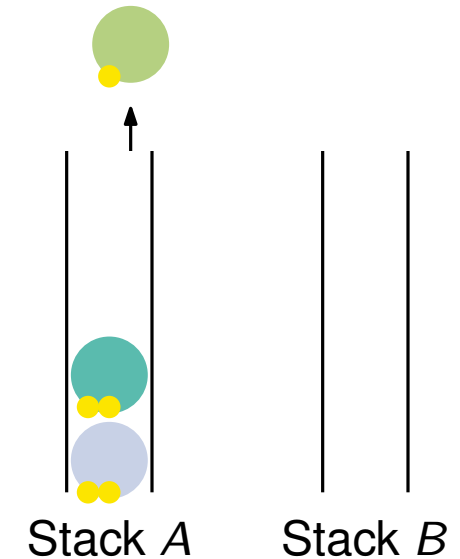
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls *B* leer: nutze 2 Guthaben für *A.pop()* und *B.push()*



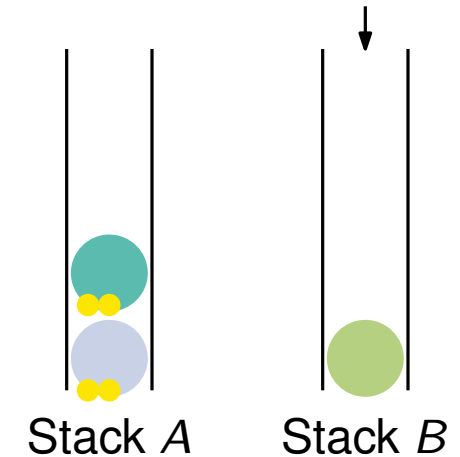
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls *B* leer: nutze 2 Guthaben für *A.pop()* und *B.push()*



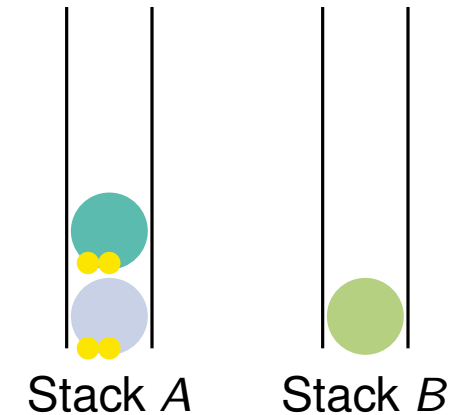
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls *B* leer: nutze 2 Guthaben für *A.pop()* und *B.push()*



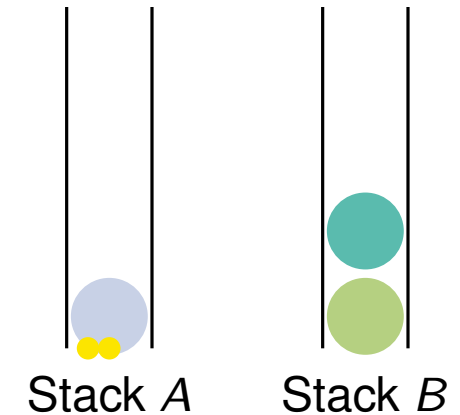
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls *B* leer: nutze 2 Guthaben für *A.pop()* und *B.push()*



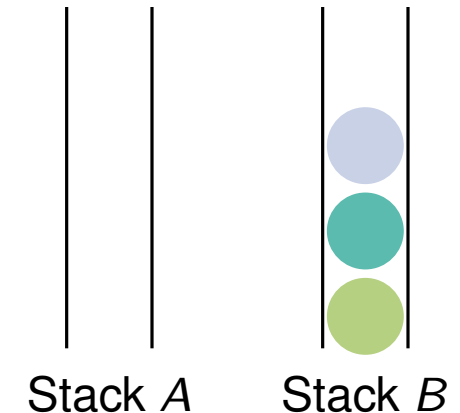
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls B leer: nutze 2 Guthaben für $A.\text{pop}()$ und $B.\text{push}()$



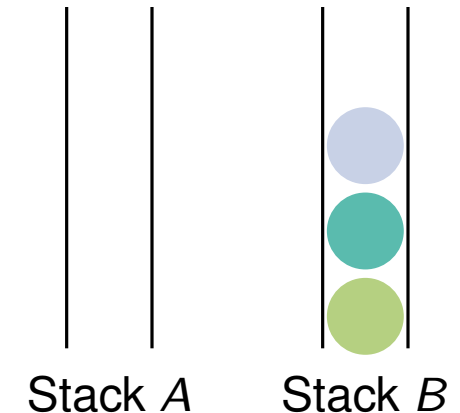
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls B leer: nutze 2 Guthaben für $A.\text{pop}()$ und $B.\text{push}()$ (für alle Elemente in A)



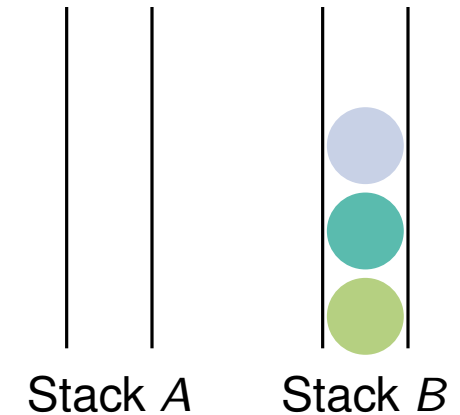
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls *B* leer: nutze 2 Guthaben für **A.pop()** und **B.push()** (für alle Elemente in *A*)
 - Falls *B* voll: **B.pop()** mit Kosten 1



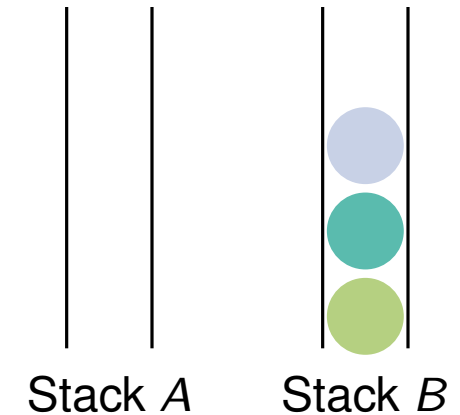
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls *B* leer: nutze 2 Guthaben für *A.pop()* und *B.push()* (für alle Elemente in *A*)
 - Falls *B* voll: *B.pop()* mit Kosten 1
- Konto wird nie negativ



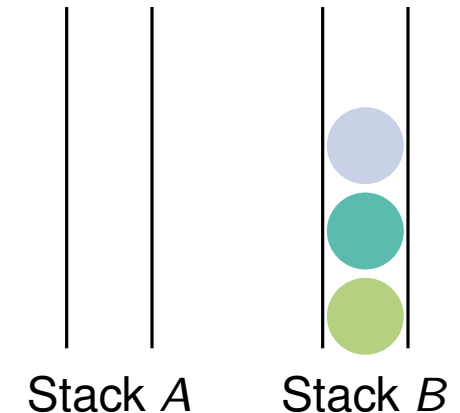
Analyse: Kontomethode

Idee

- günstige Operationen bauen Guthaben auf
- teure Operationen nutzen Guthaben

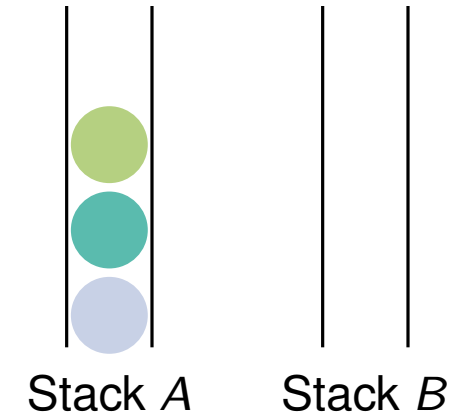
Umsetzung hier

- Gesamtkosten linear in **push** und **pop** Aufrufen
- Bei **push**: zahle 3
- Bei **pop**: zahle 1
 - Falls *B* leer: nutze 2 Guthaben für *A.pop()* und *B.push()* (für alle Elemente in *A*)
 - Falls *B* voll: *B.pop()* mit Kosten 1
- Konto wird nie negativ
- $3, 1 \in \Theta(1) \Rightarrow$ konstante amortisierte Kosten



Analyse: Potentialmethode

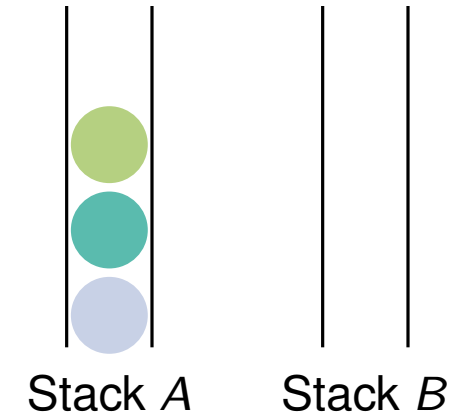
Potentialfunktion



Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

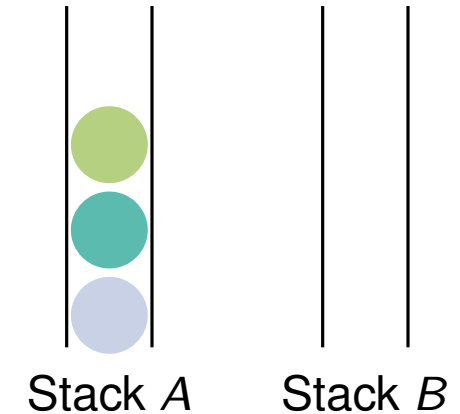


Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse



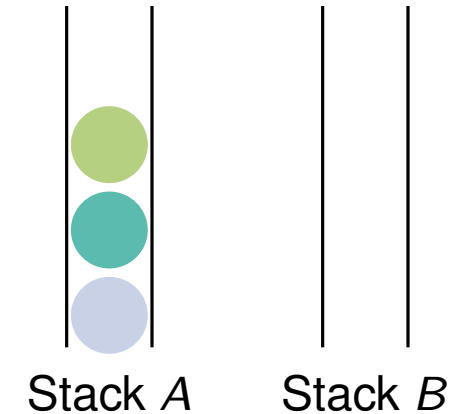
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$



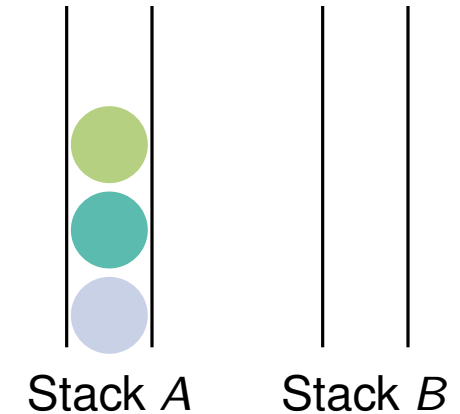
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0$, $\Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)



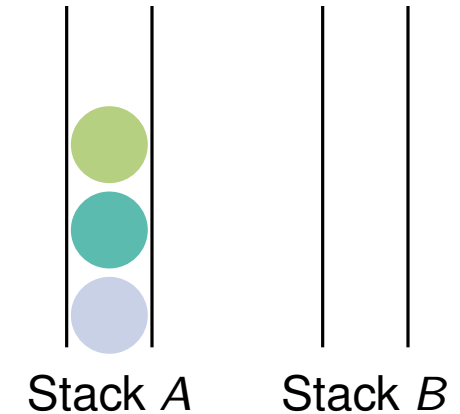
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:



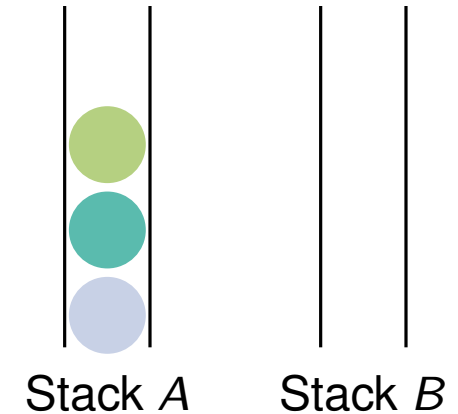
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

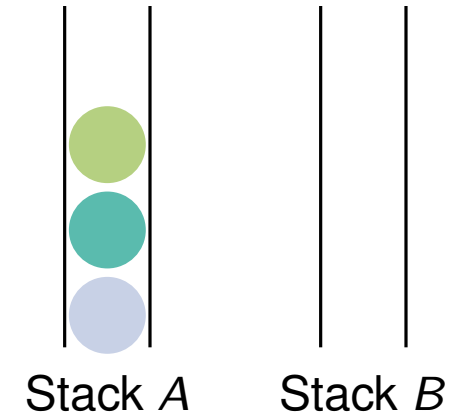
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

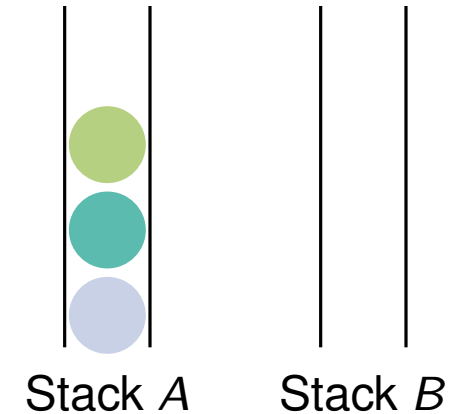
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

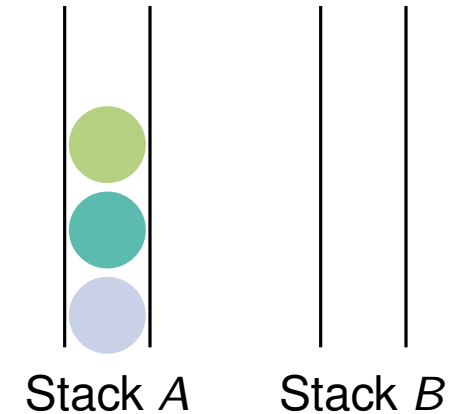
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

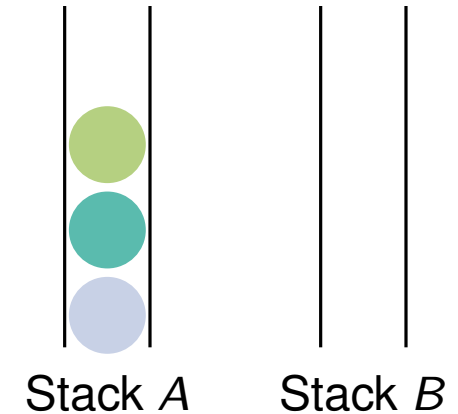
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
 - $c_i = 2 \cdot |A_{i-1}| + 1$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

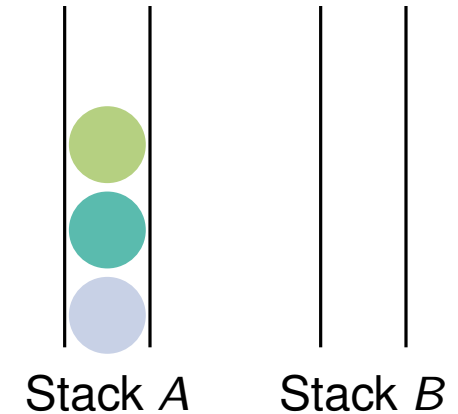
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
 - $c_i = 2 \cdot |A_{i-1}| + 1$
 - $\Phi(D_i) - \Phi(D_{i-1}) = -2|A_{i-1}|$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

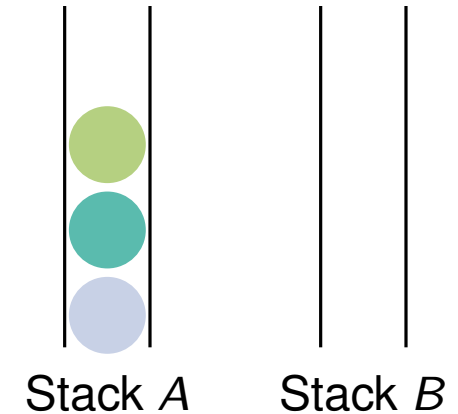
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1$
 - $c_i = 2 \cdot |A_{i-1}| + 1$
 - $\Phi(D_i) - \Phi(D_{i-1}) = -2|A_{i-1}|$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

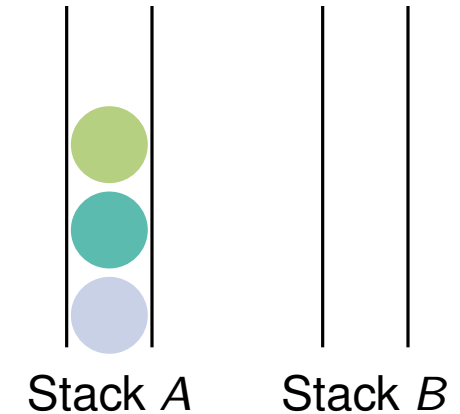
Analyse: Potentialmethode

Potentialfunktion

- Definiere $\Phi(D_i) = 2 \cdot |A_i|$ (Anzahl Elemente auf A zum Zeitpunkt i)

Analyse

- $\Phi(D_0) = 0, \Phi(D_i) \geq 0$ f.a. $i > 0$ (d.h. gültige Potentialfunktion)
- amortisierte Kosten **push**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 3$
- amortisierte Kosten **pop**:
 - $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1$
 - $c_i = 2 \cdot |A_{i-1}| + 1$
 - $\Phi(D_i) - \Phi(D_{i-1}) = -2|A_{i-1}|$



Definition: amortisierte Kosten $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

Sortieren

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
- ⟨ ⟩

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
- ⟨ ⟩

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 23 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 23 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 23 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 23 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 23, 38 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 23, 38 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 23, 38 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 23, 38 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

$\langle 23, 1, 38, 2, 7, 89 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

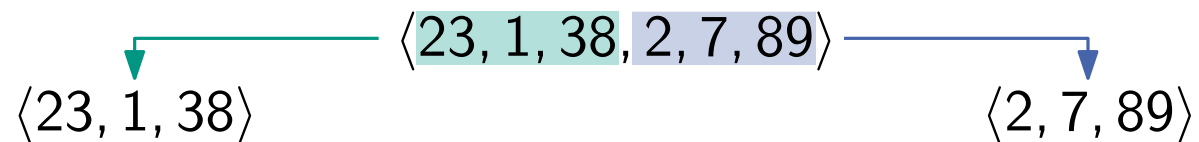
$\langle 23, 1, 38, 2, 7, 89 \rangle$

Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

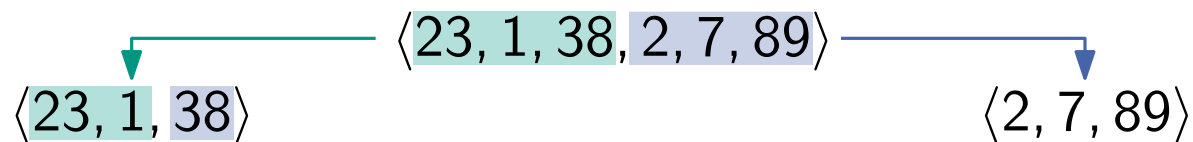


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

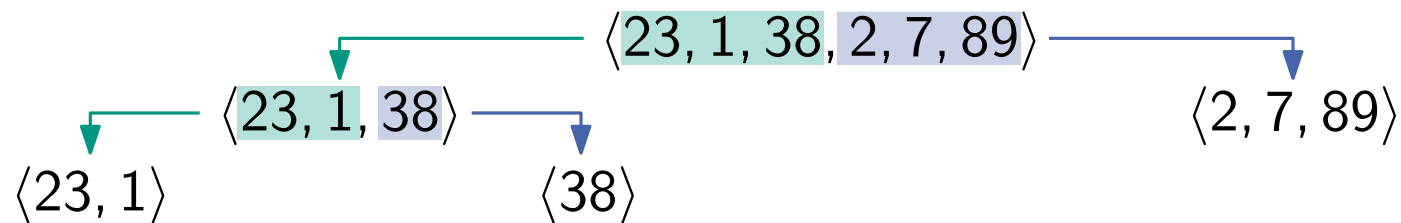


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

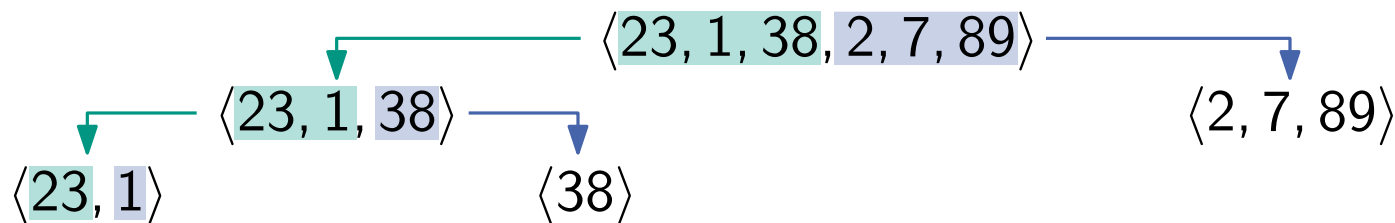


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

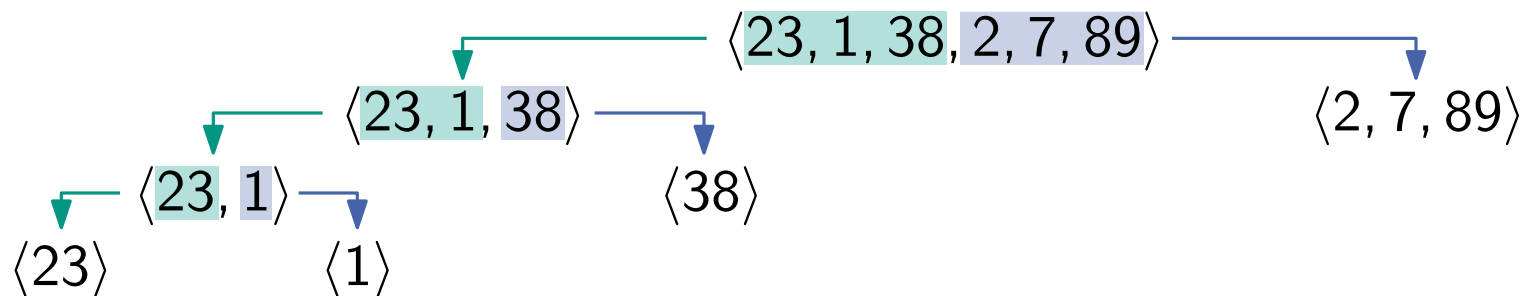


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

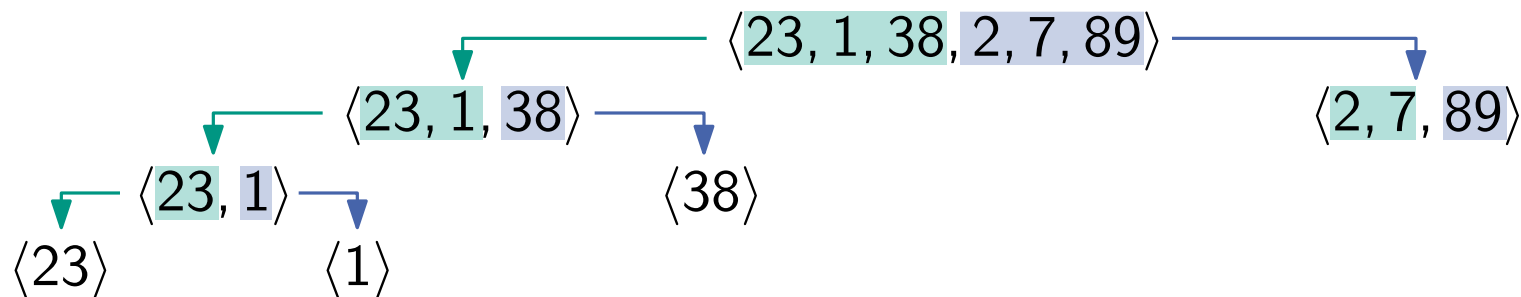


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

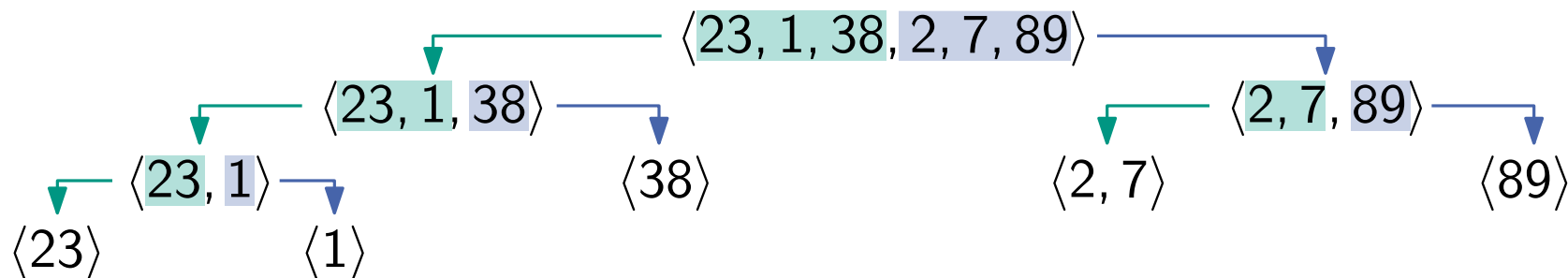


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

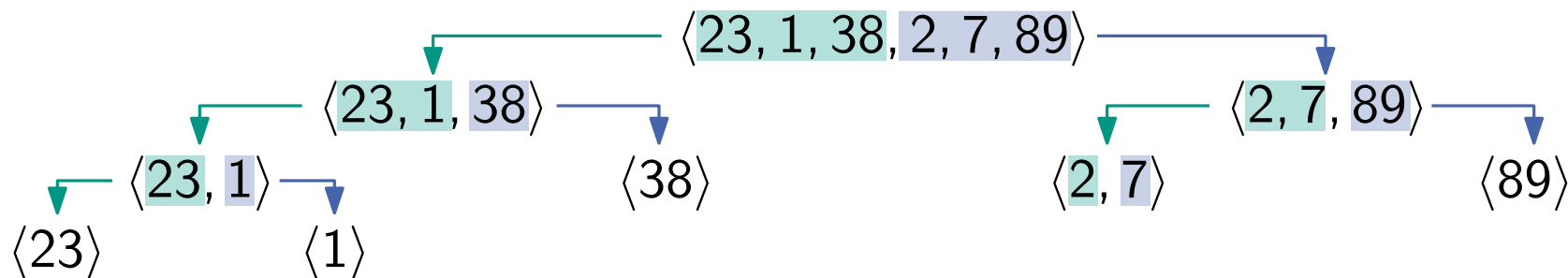


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

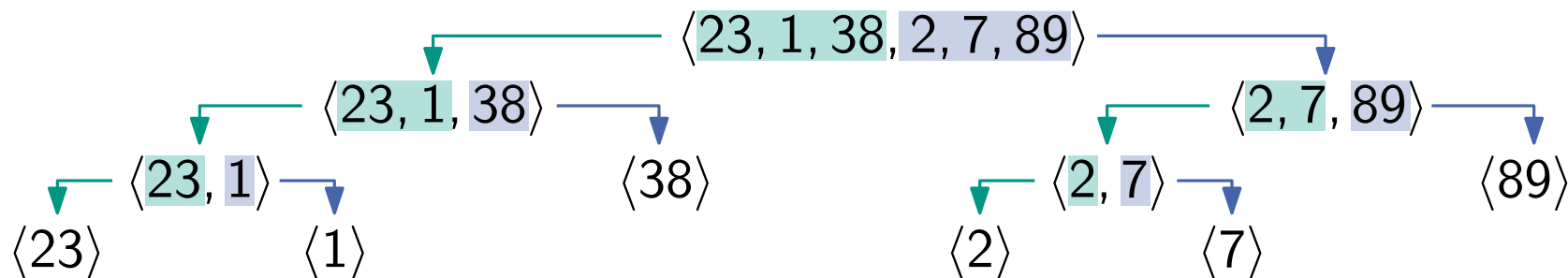


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

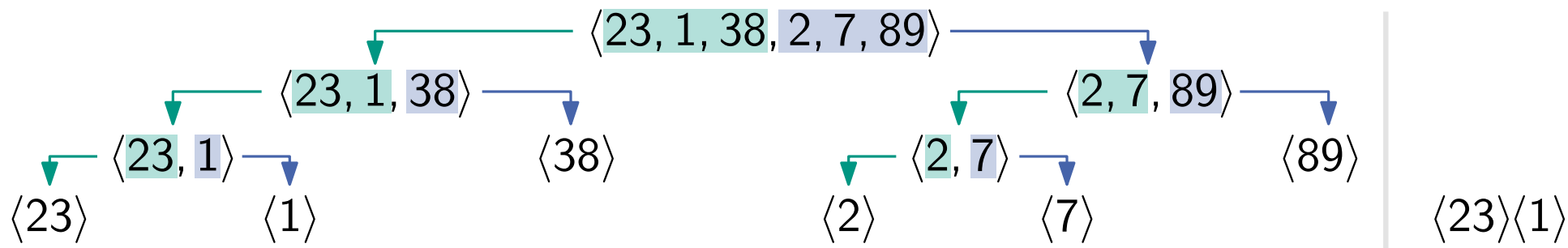


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

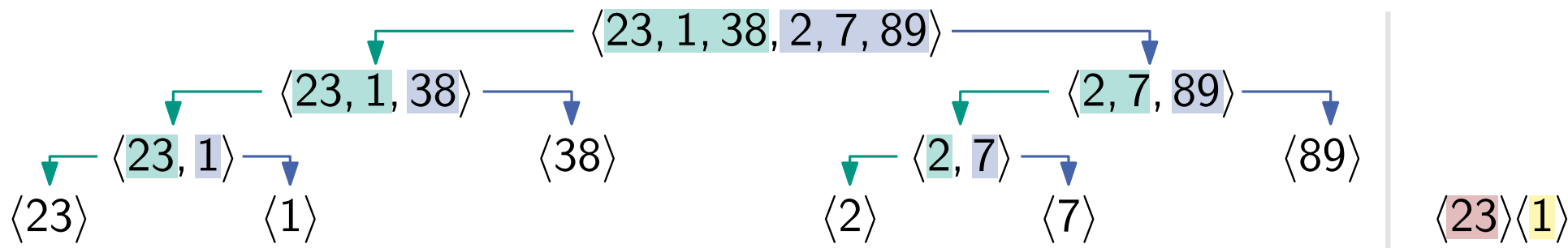


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

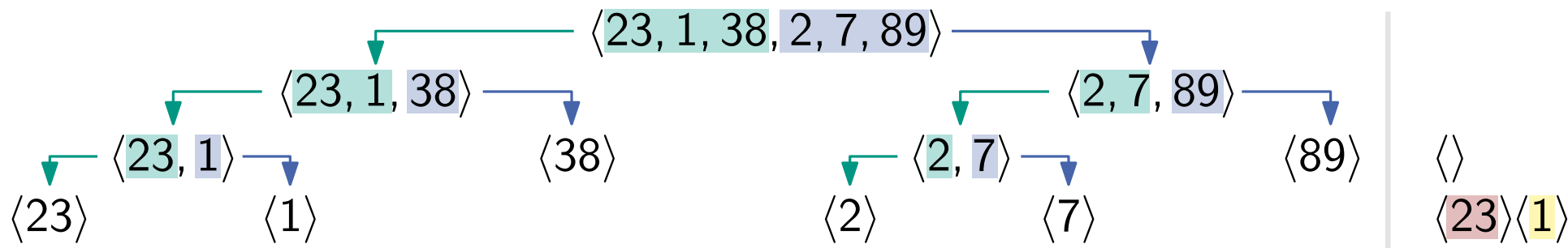


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

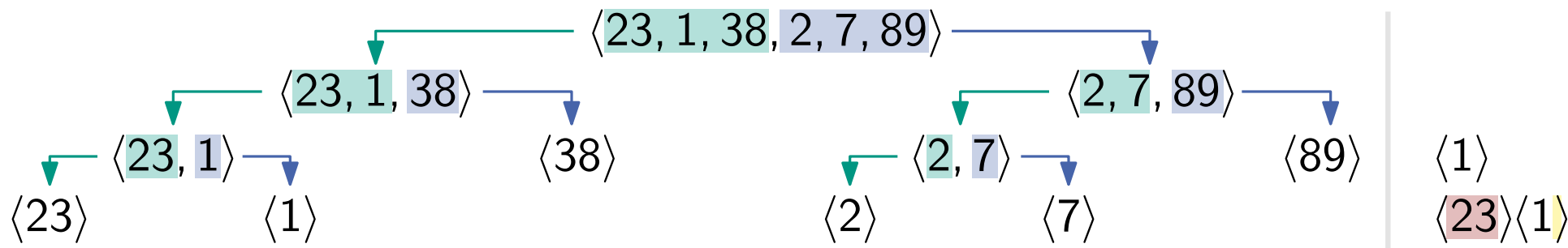


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

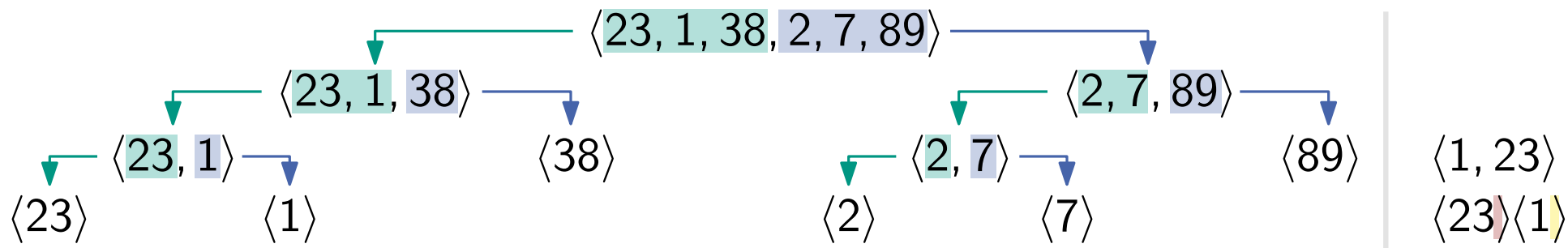


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

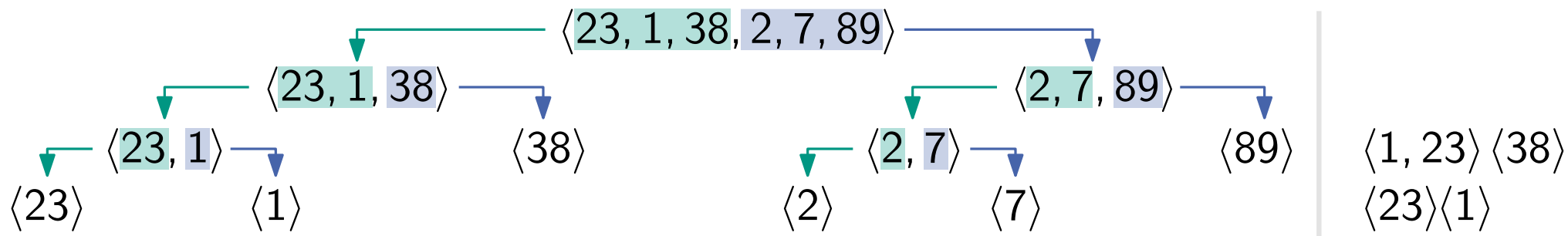


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

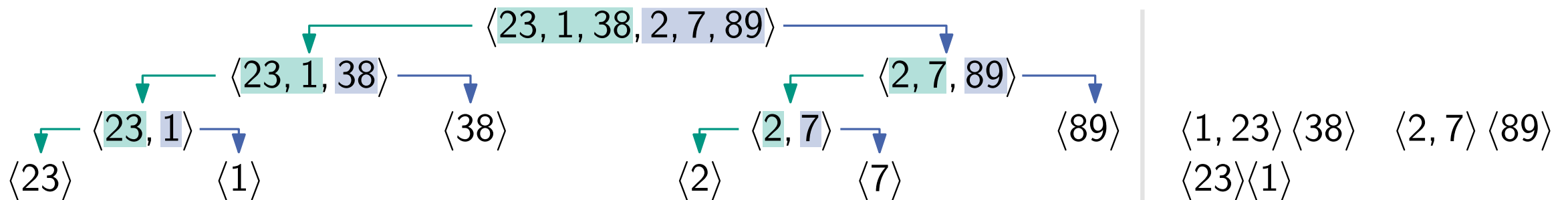


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

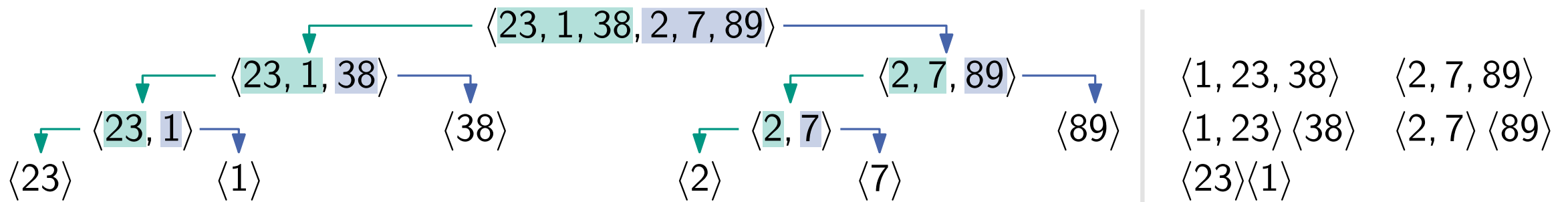


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

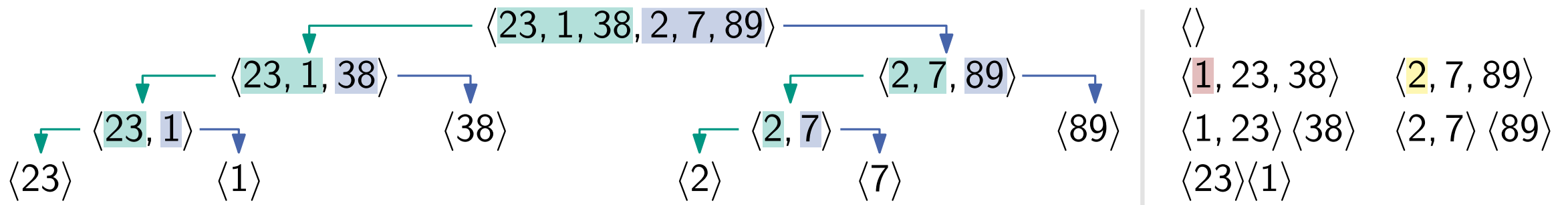


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

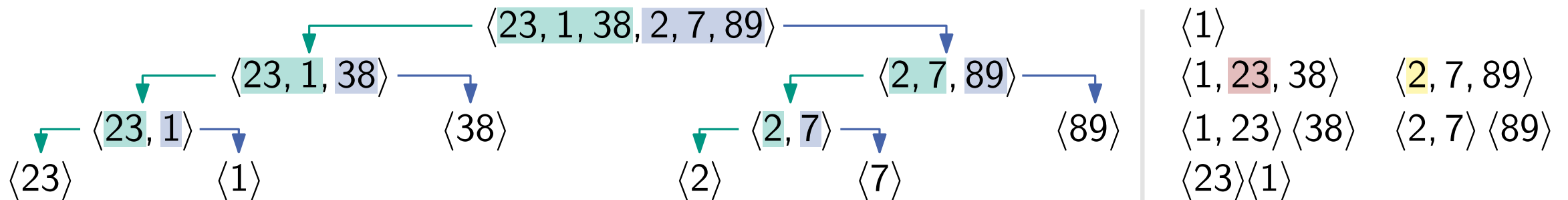


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

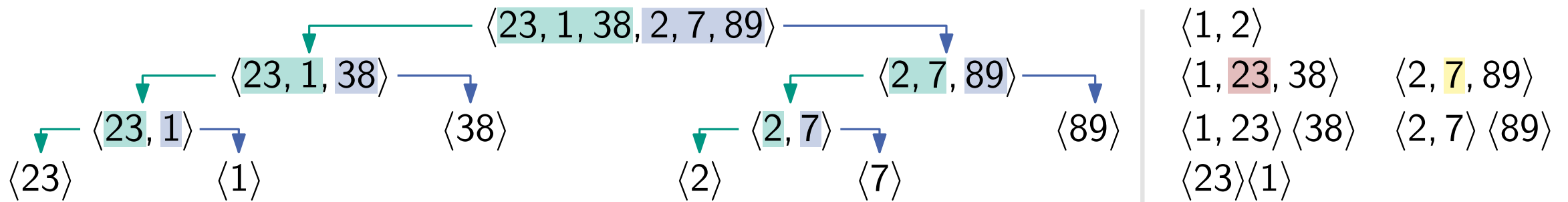


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

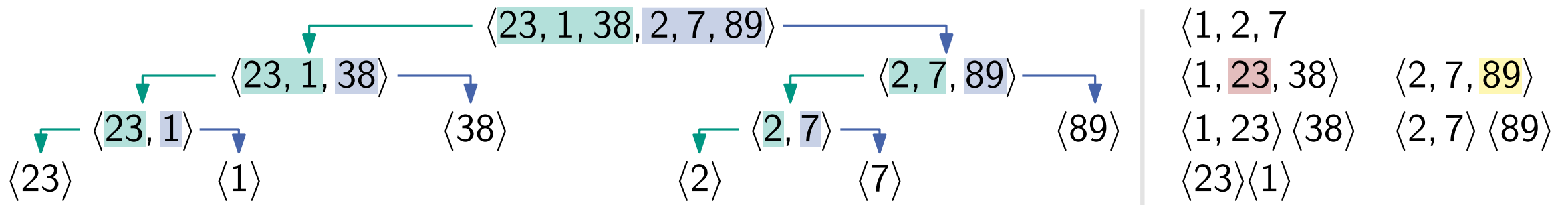


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

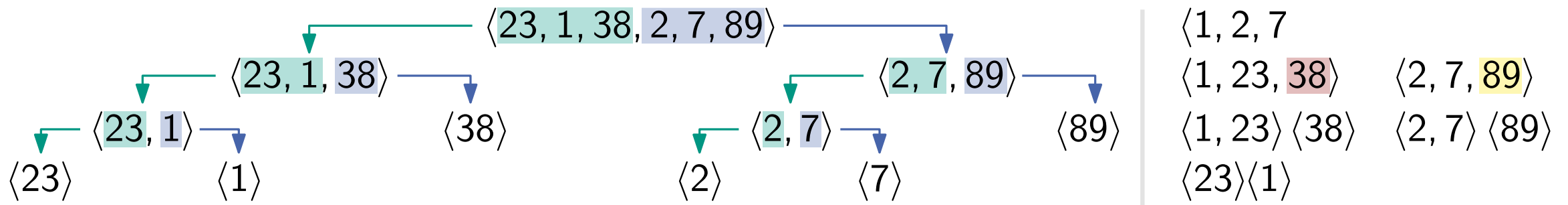


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

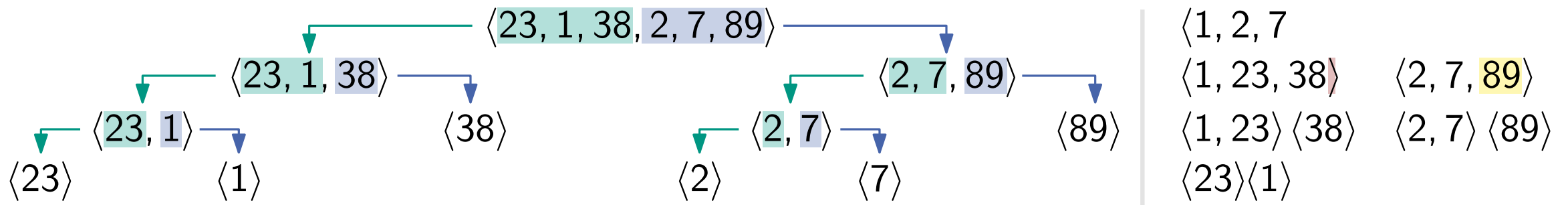


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden

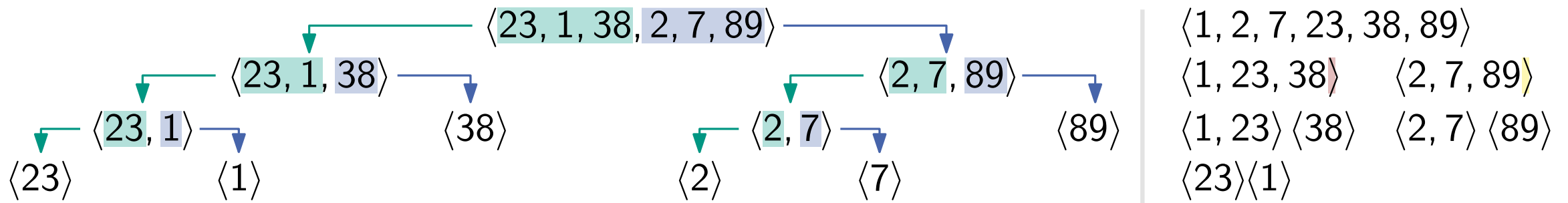


Sortieren

- Gegeben: n Elemente aus einer geordneten Menge (z.B.: Zahlen) $\langle 23, 1, 38, 2, 7, 89 \rangle$
- Gesucht: sortierte Folge dieser Elemente $\langle 1, 2, 7, 23, 38, 89 \rangle$

Algorithmen

- InsertionSort ($\Theta(n^2)$)
 - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln
 $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
 - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden



Sortieren

- QuickSort

- Teile ein Array rekursiv anhand eines Pivots in zwei Teile, welche jeweils alle Elemente kleiner bzw. größer dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

Sortieren

■ QuickSort

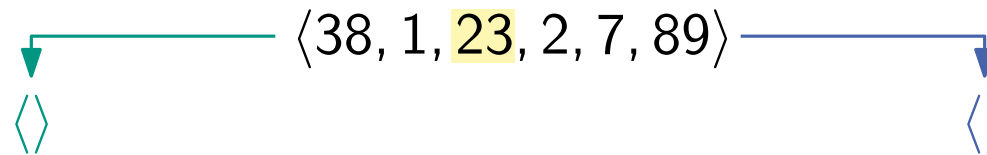
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente kleiner bzw. größer dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

Sortieren

■ QuickSort

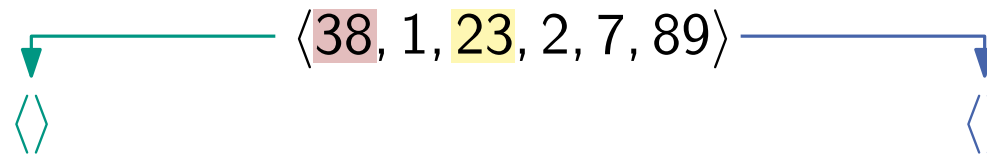
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

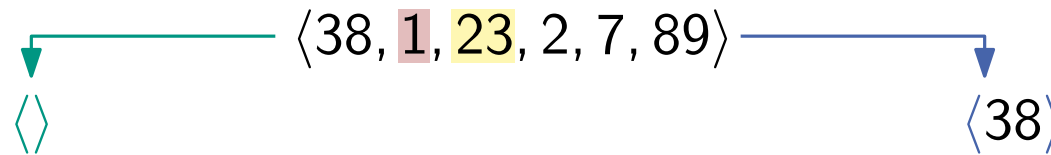
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

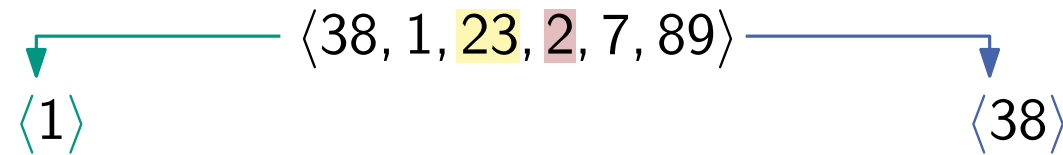
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

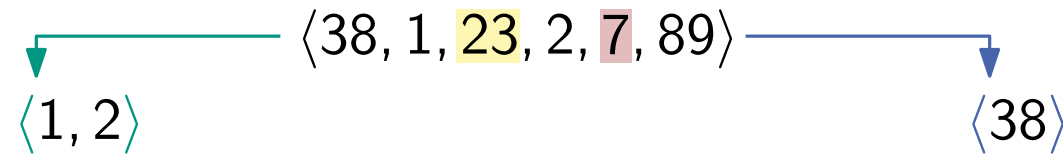
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

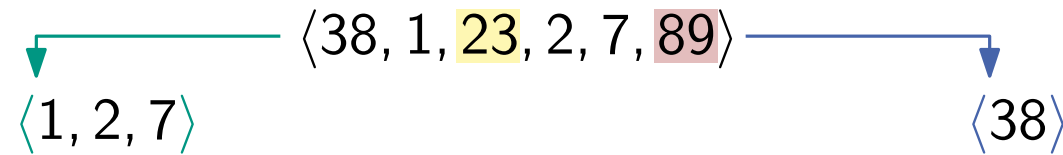
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

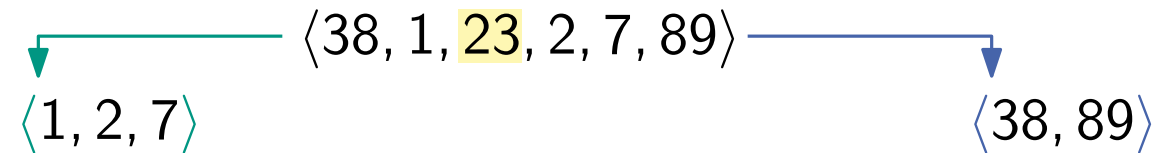
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

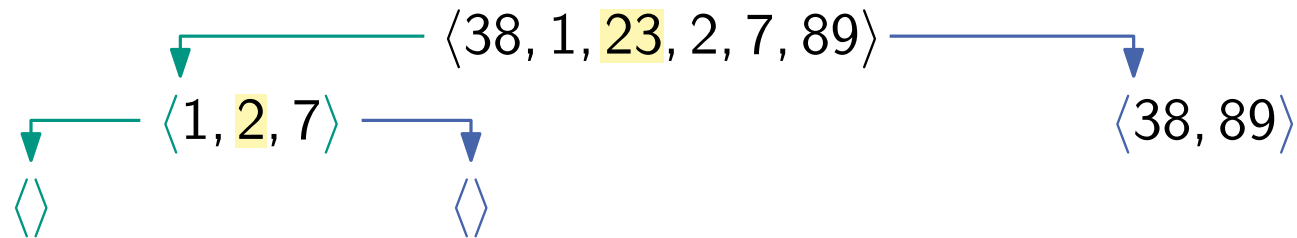
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

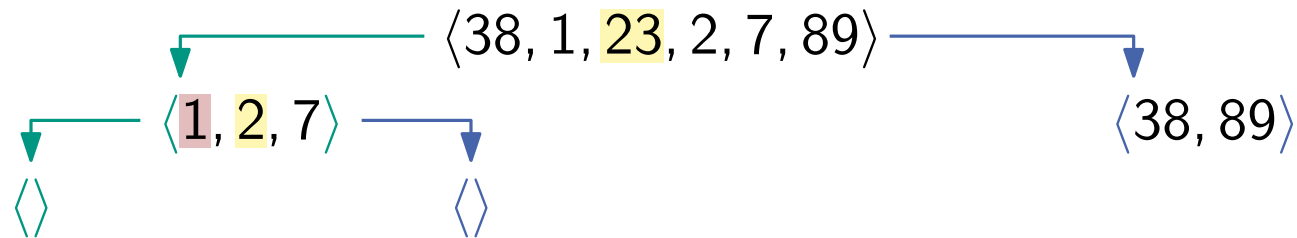
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

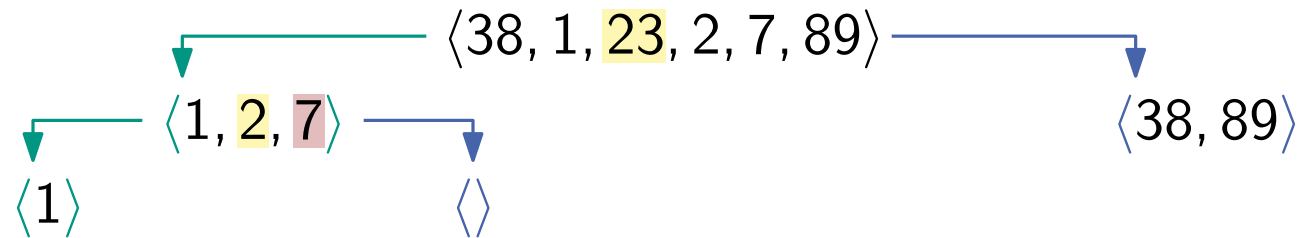
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

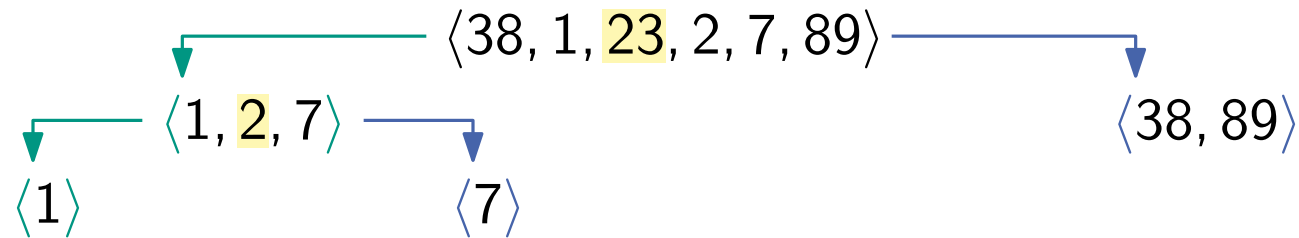
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

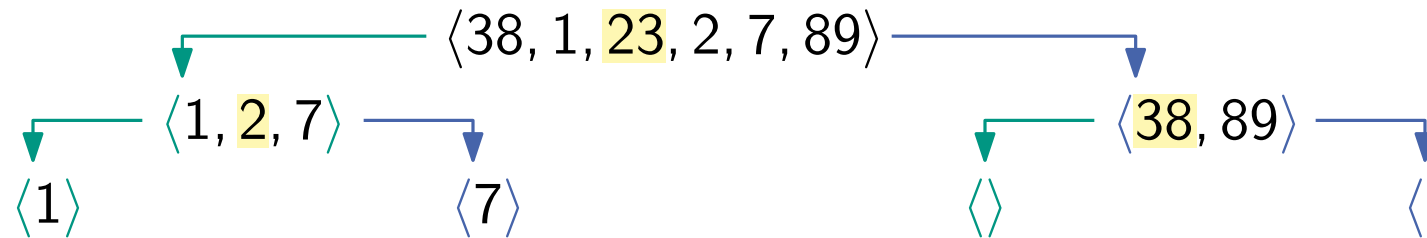
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

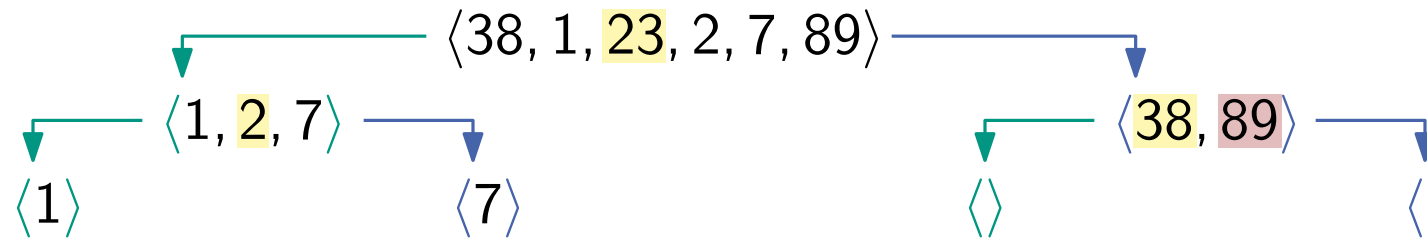
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

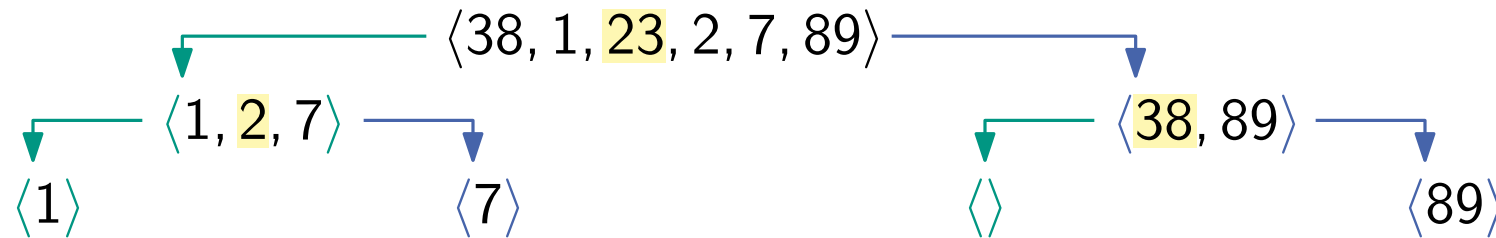
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

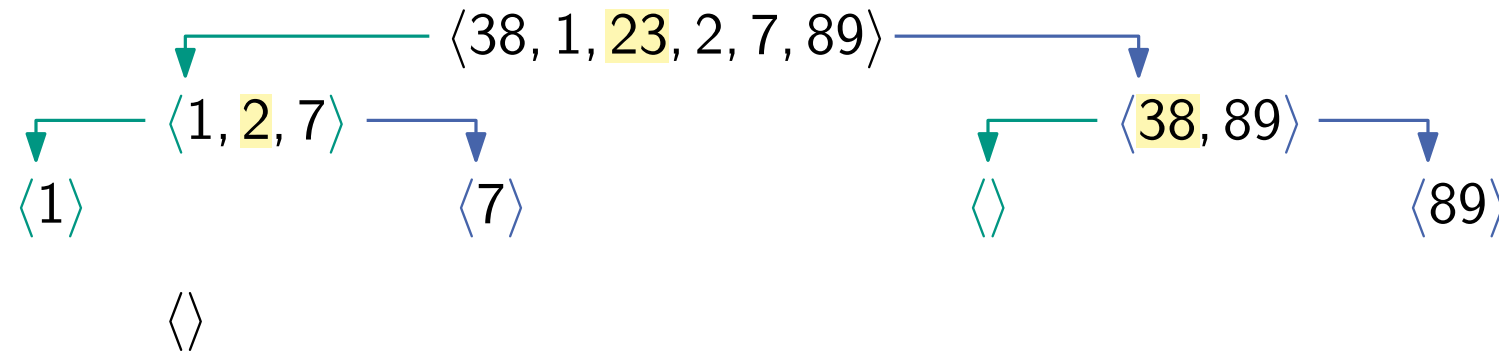
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

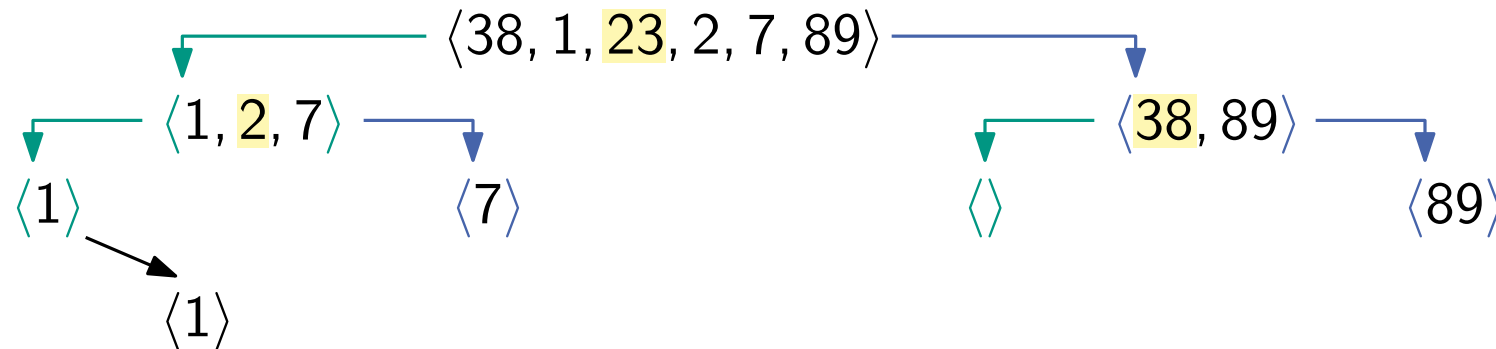
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

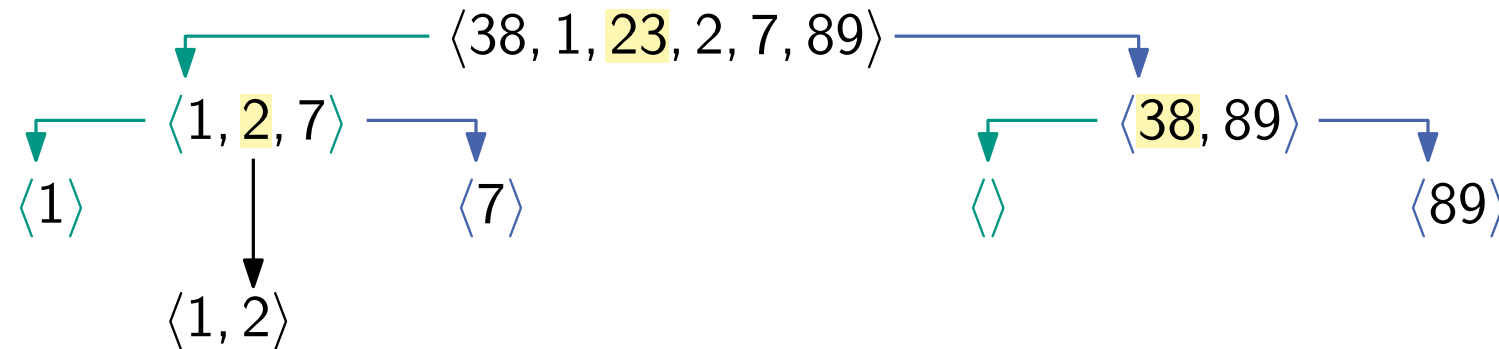
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

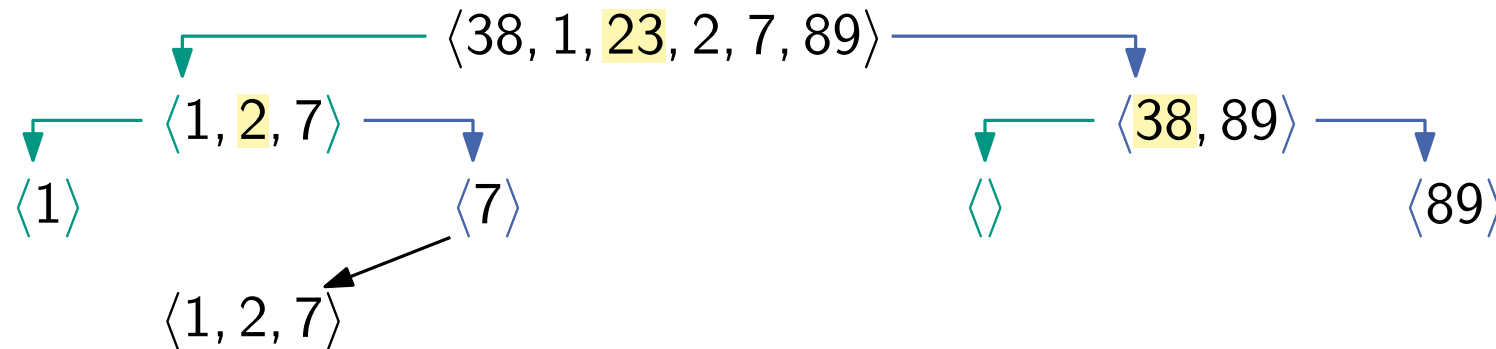
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

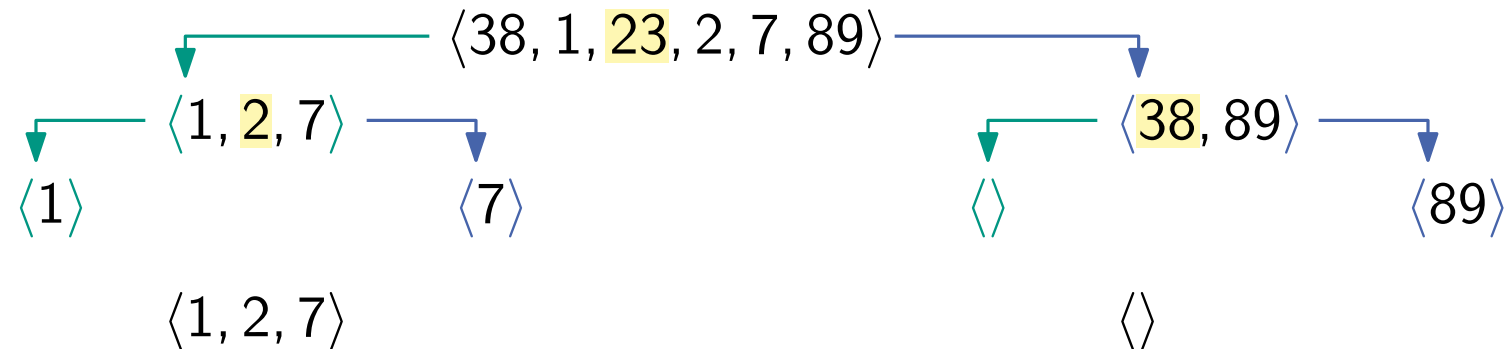
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

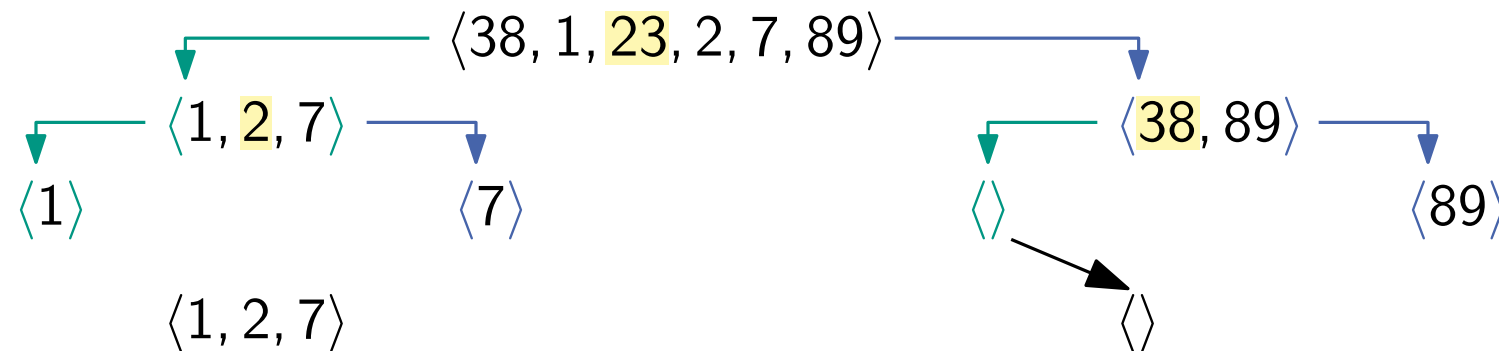
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

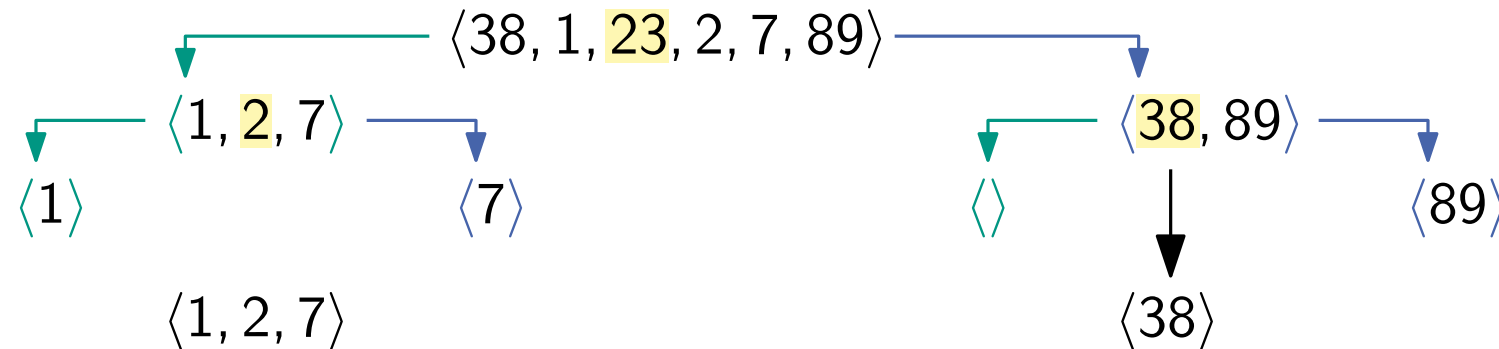
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

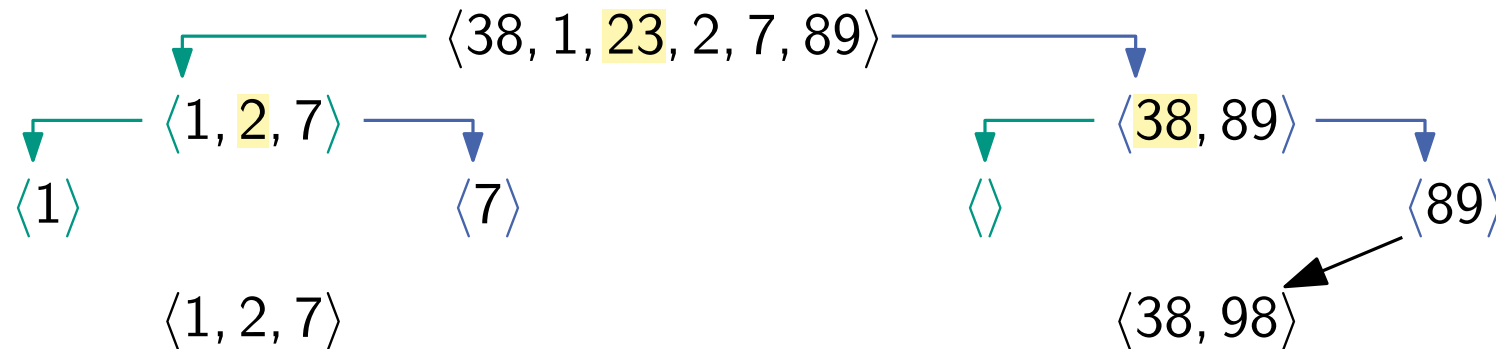
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

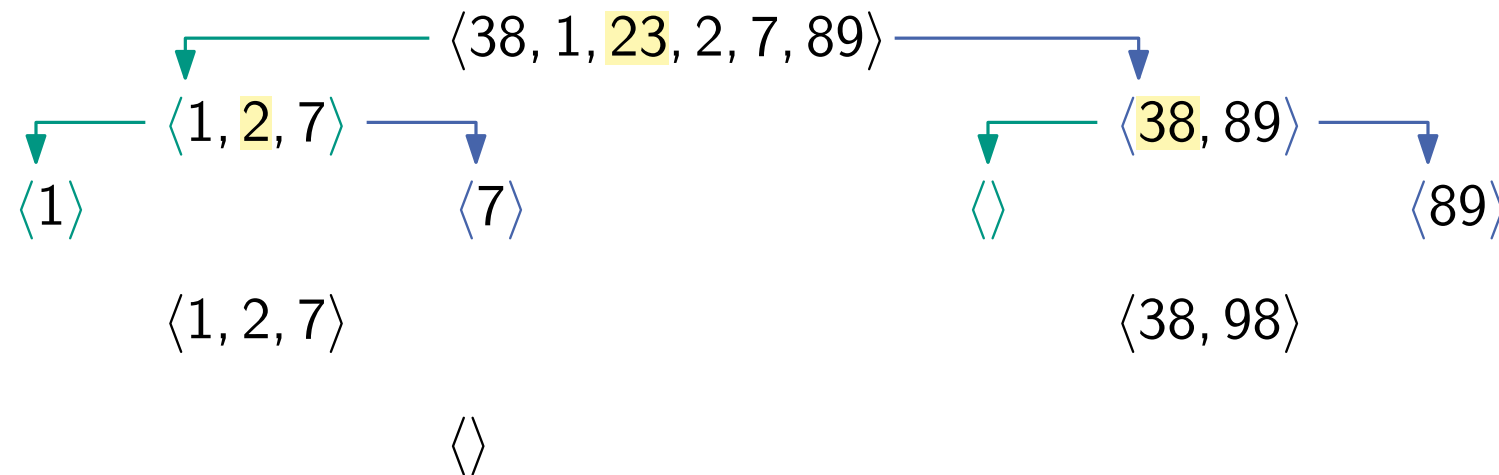
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

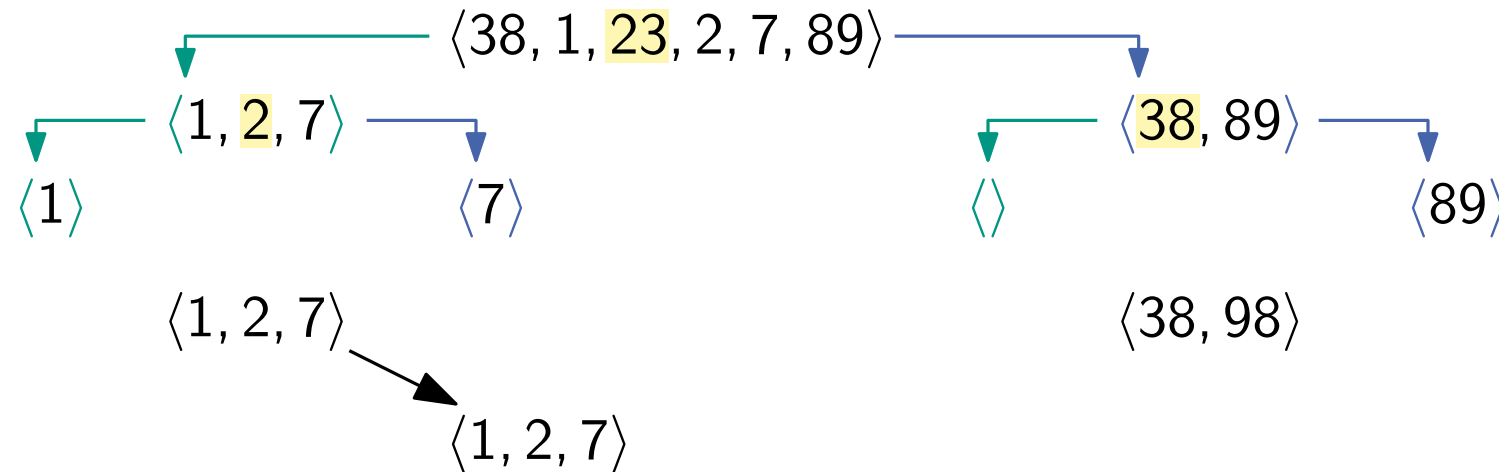
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

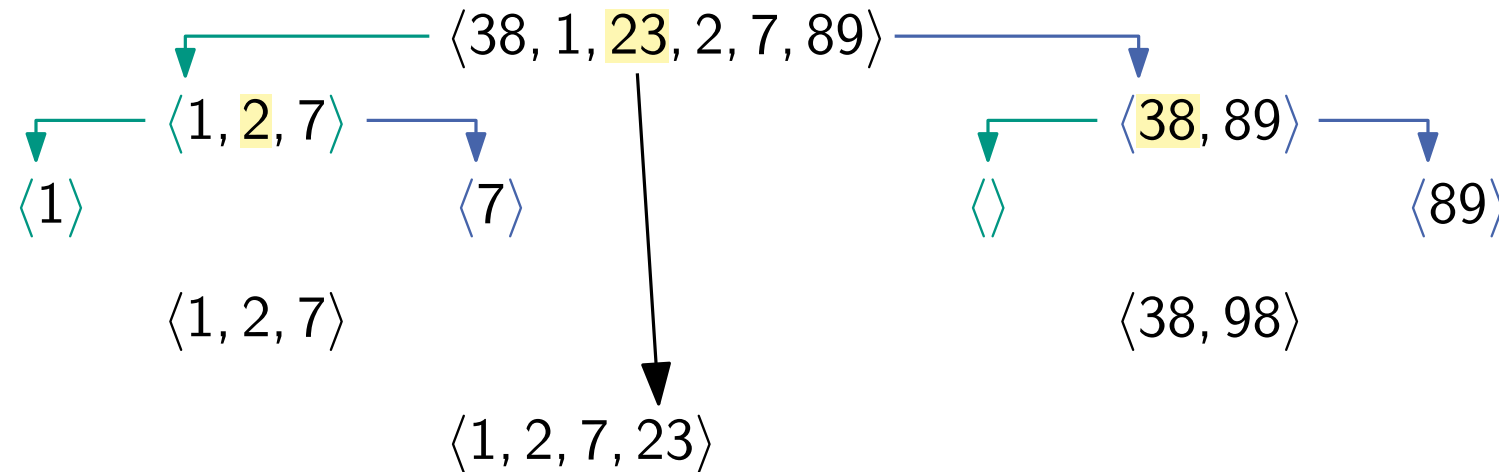
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

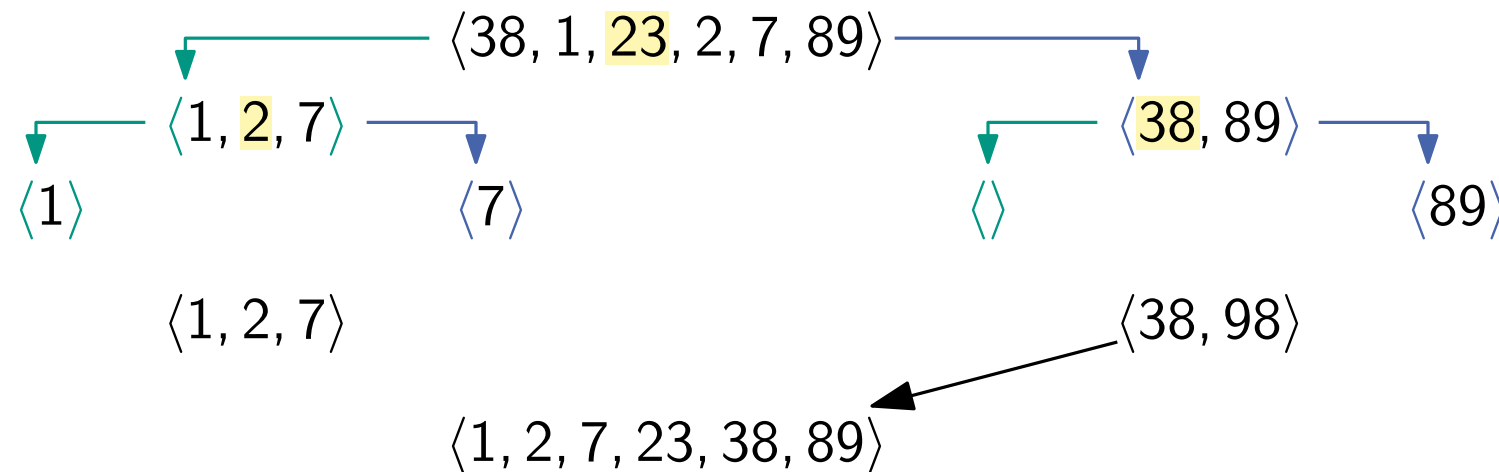
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

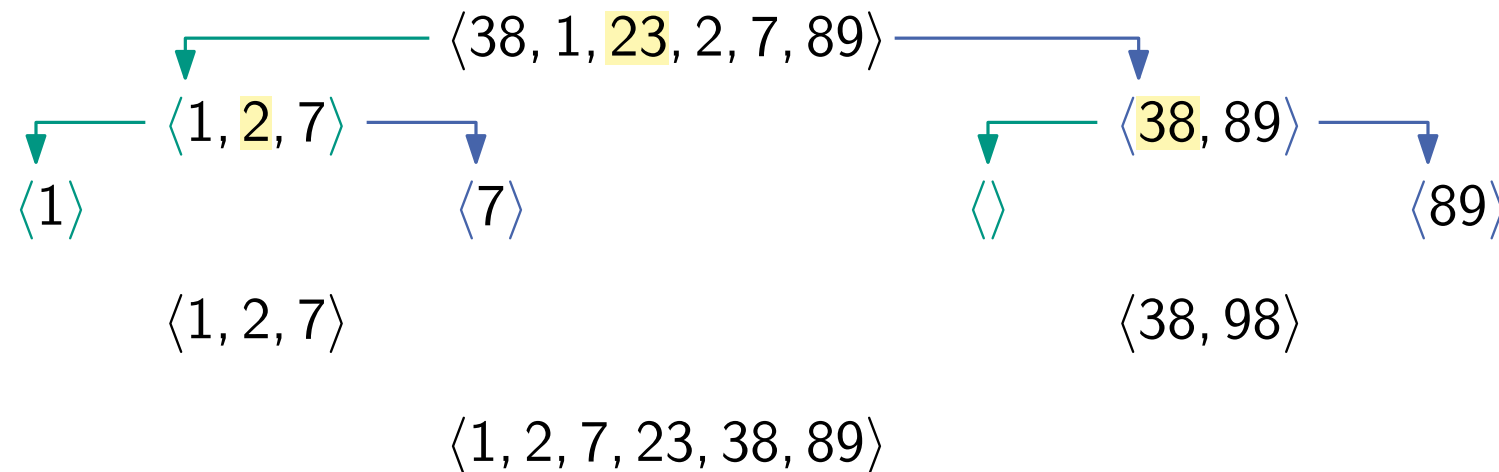
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



Sortieren

■ QuickSort

- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

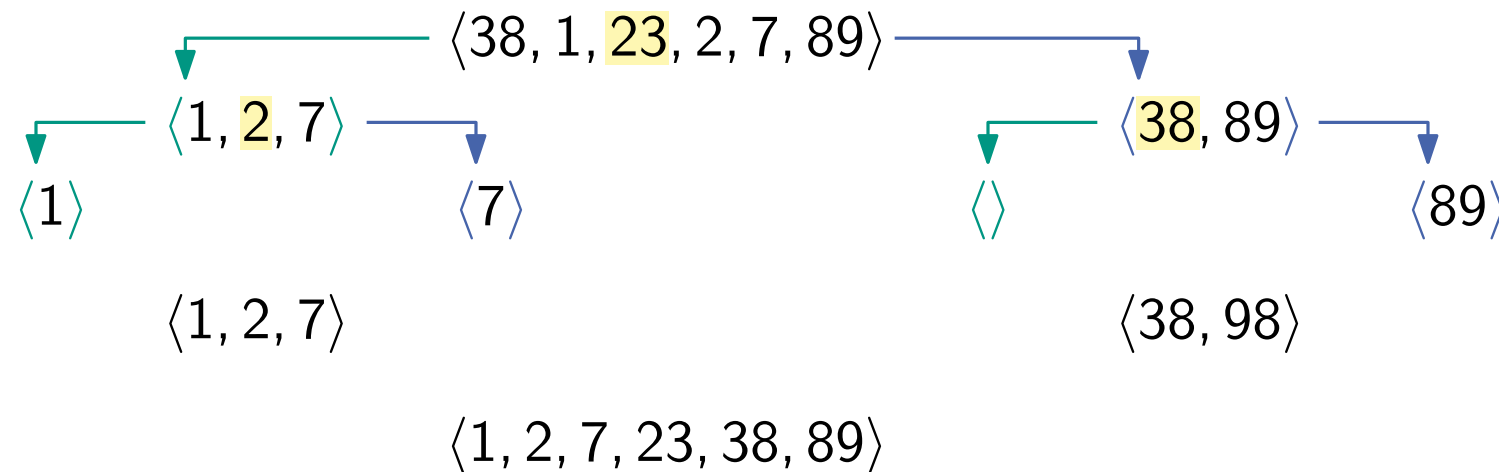


- Die Wahl des Pivots beeinflusst die Laufzeit

Sortieren

■ QuickSort

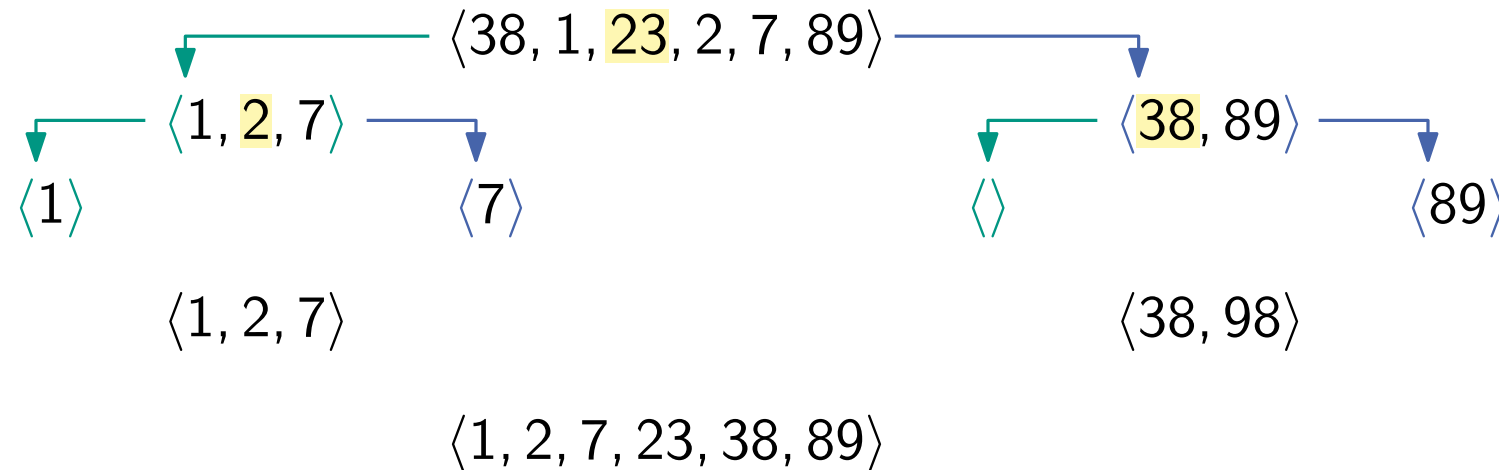
- Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



- Die Wahl des Pivots beeinflusst die Laufzeit
- Bereits gesehen
 - Worst Case \rightarrow Pivot ist immer kleinstes oder größtes Element

Sortieren

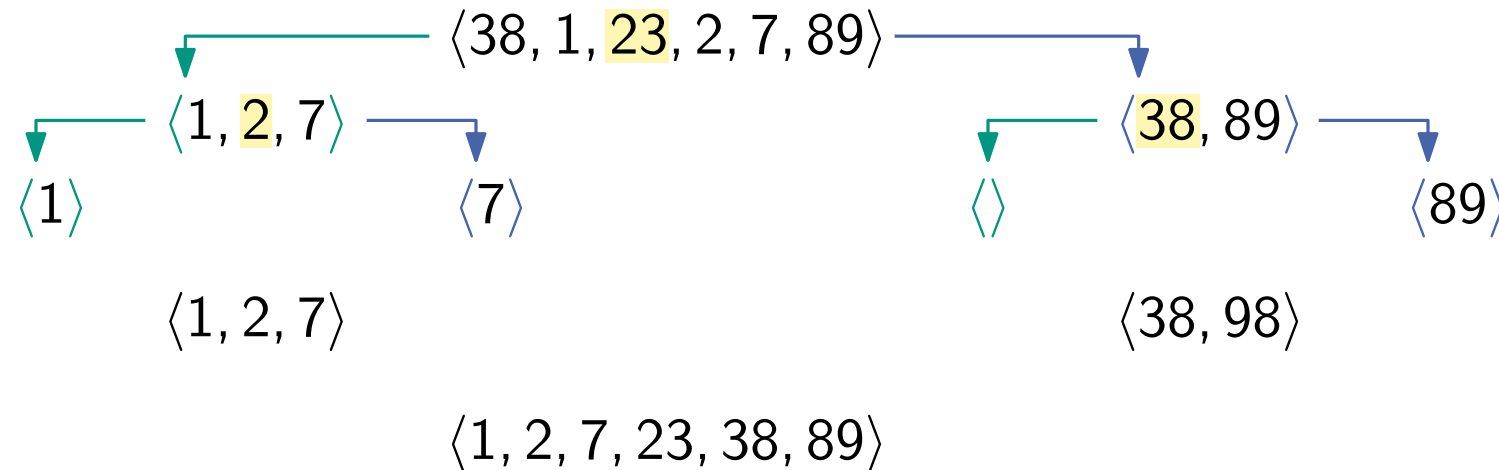
- QuickSort: Laufzeit erwartet $O(n \log n)$
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



- Die Wahl des Pivots beeinflusst die Laufzeit
- Bereits gesehen
 - Worst Case \rightarrow Pivot ist immer kleinstes oder größtes Element
 - Mittels zufällig gewähltem Pivot kommt man nah an den Best Case

Sortieren

- QuickSort: Laufzeit erwartet $O(n \log n)$
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten



- Die Wahl des Pivots beeinflusst die Laufzeit
- Bereits gesehen
 - Worst Case \rightarrow Pivot ist immer kleinstes oder größtes Element
 - Mittels zufällig gewähltem Pivot kommt man nah an den Best Case

Warum?

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

⟨38, 1, 23, 2, 7, 89⟩

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines Pivots in zwei Teile, welche jeweils alle Elemente kleiner bzw. größer dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines Pivots in zwei Teile, welche jeweils alle Elemente kleiner bzw. größer dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

⟨38, 1, 23, 2, 7, 89⟩

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?
- Auftritt: Adversary



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

⟨38, 1, 23, 2, 7, 89⟩

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?
- Auftritt: Adversary
 - kennt den Algorithmus



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

⟨38, 1, 23, 2, 7, 89⟩

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?
- Auftritt: Adversary
 - kennt den Algorithmus
 - baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?
- Auftritt: Adversary $\langle 1, 2, 3, 4, \dots, n \rangle$
 - kennt den Algorithmus
 - baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?
- Auftritt: Adversary $\langle 1, 2, 3, 4, \dots, n \rangle$
 - kennt den Algorithmus
 - baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

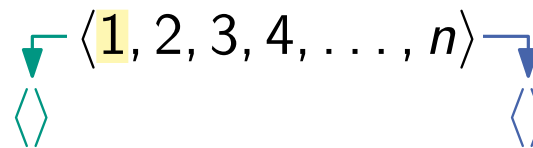
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

■ Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

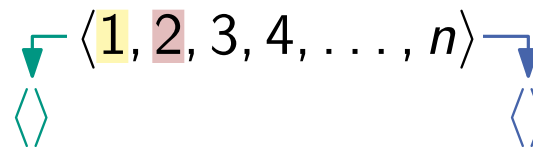
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

■ Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

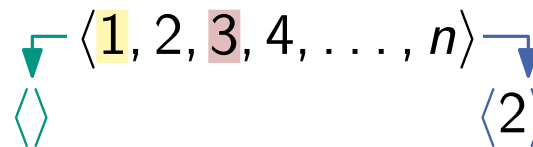
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

■ Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

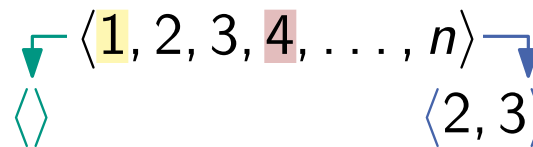
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

■ Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

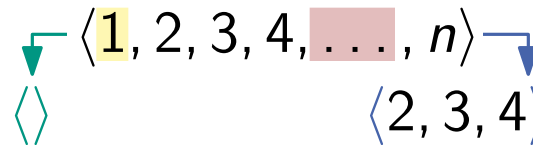
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

■ Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

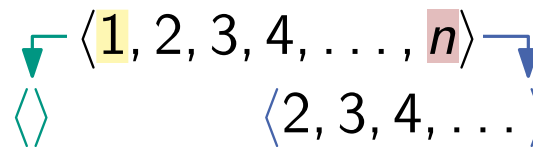
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

■ Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

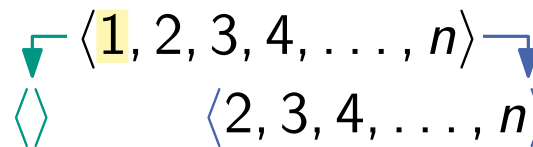
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

■ Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

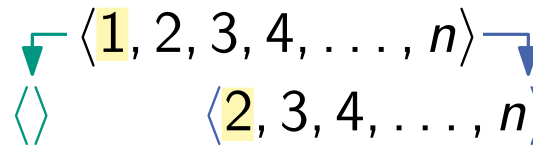
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

■ Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

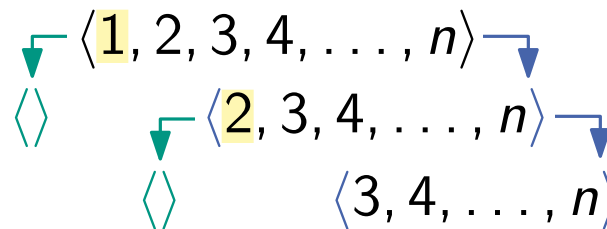
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

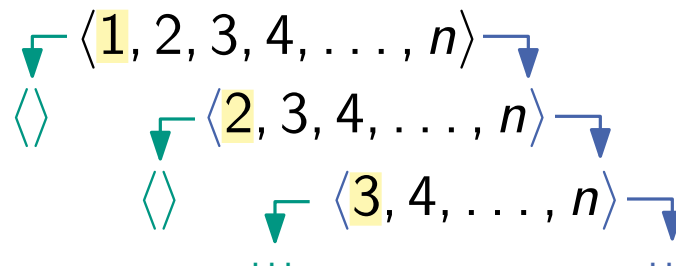
- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array
 - Was kann im schlimmsten Falls passieren?

Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>



QuickSort – Pivots

- Worst Case vs. Average Case – Adversary Sichtweise
 - Teile ein Array rekursiv anhand eines **Pivots** in zwei Teile, welche jeweils alle Elemente **kleiner** bzw. **größer** dem Pivot enthalten

$\langle 38, 1, 23, 2, 7, 89 \rangle$

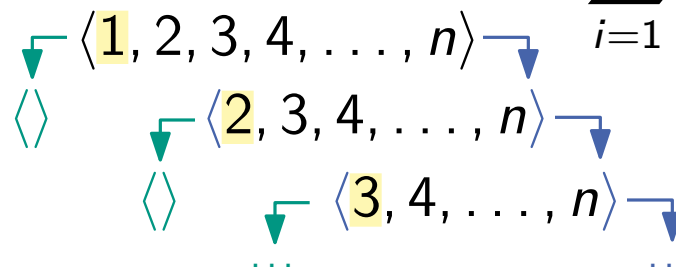
- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
 - wähle irgendein Element als Pivot → Beispiel: Pivot ist immer der erste Eintrag im Array

- Was kann im schlimmsten Falls passieren?

$$\sum_{i=1}^n i = \Theta(n^2)$$

- Auftritt: Adversary

- kennt den Algorithmus
- baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from <https://scryfall.com/card/mid/129/bloodthirsty-adversary>

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermässeln?

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!

Zufall!



Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.

Zufall!



Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir QuickSort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von QuickSort Adversary wird irrelevant!

Zufall!



Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von Quicksort Adversary wird irrelevant!

Zufall!

$\langle 1, 2, 3, 4, 5, 6 \rangle$



Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von Quicksort Adversary wird irrelevant!

Zufall!

Wähle unter allen Permutationen
zufällig gleich verteilt eine aus.

$\langle 1, 2, 3, 4, 5, 6 \rangle$
 $\langle 4, 3, 1, 6, 5, 2 \rangle$



Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von Quicksort Adversary wird irrelevant!

Zufall!

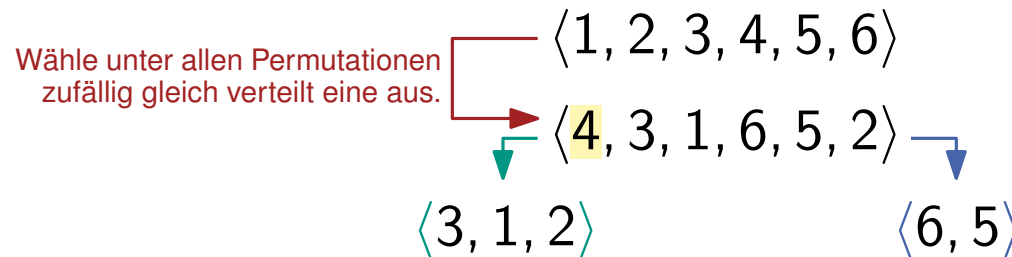


Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir QuickSort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von QuickSort Adversary wird irrelevant!

Zufall!

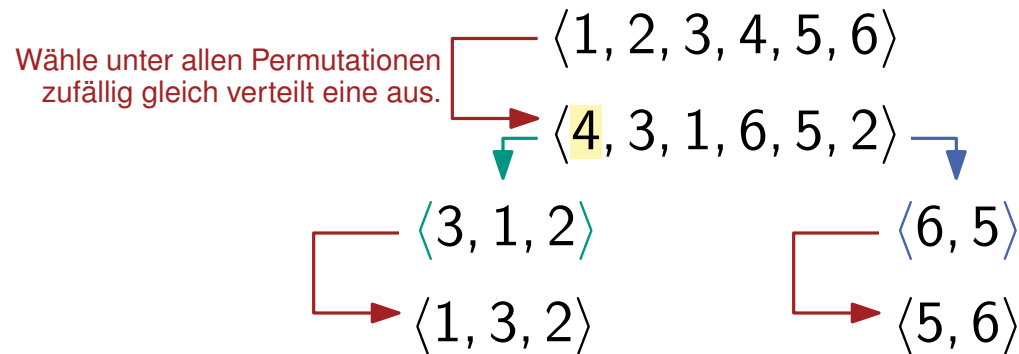


Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir QuickSort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von QuickSort Adversary wird irrelevant!

Zufall!

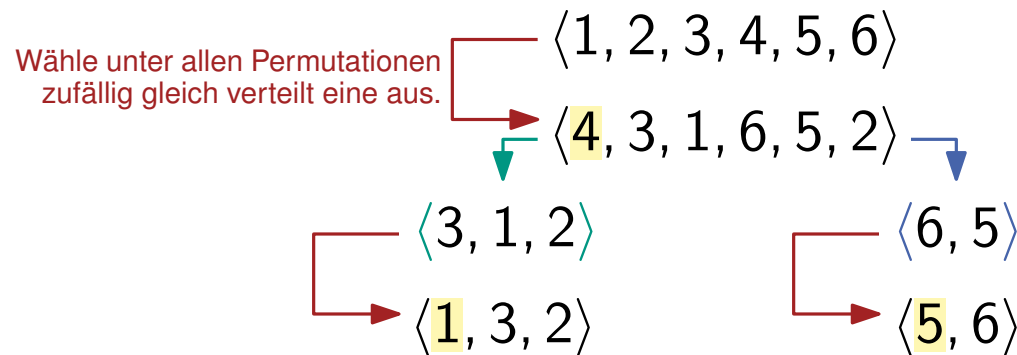


Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir QuickSort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von QuickSort Adversary wird irrelevant!

Zufall!

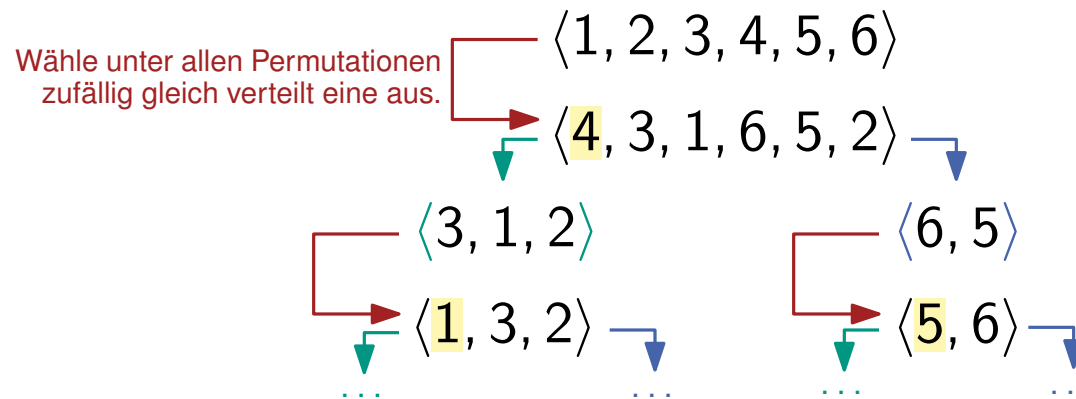


Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir QuickSort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von QuickSort Adversary wird irrelevant!

Zufall!

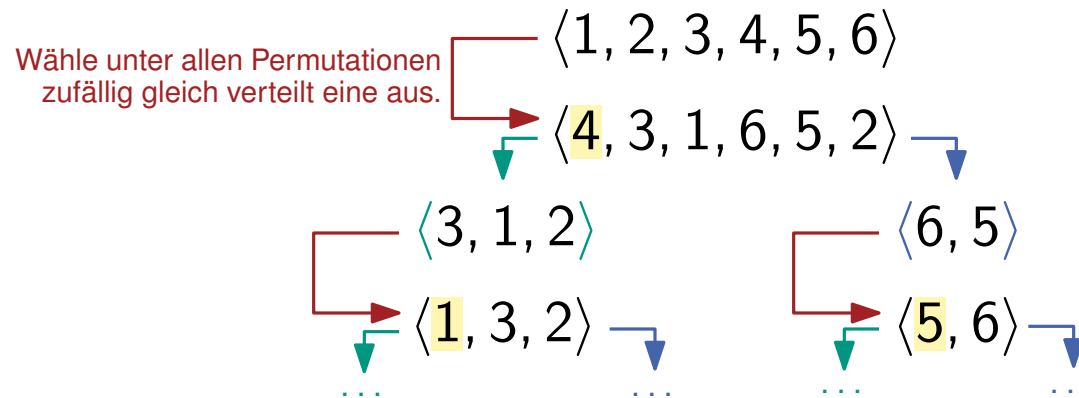


Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir QuickSort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von QuickSort Adversary wird irrelevant!

Zufall!



- Problem:
 - Extra Aufwand (um sortieren zu können, wird erstmal gemischt)

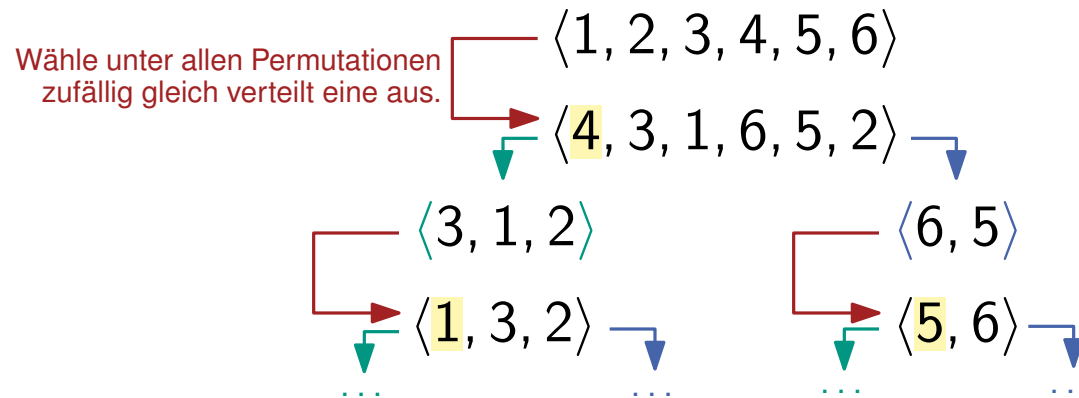


Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Wie können wir QuickSort Adversary die Tour vermässeln?
 - ... selbst wenn sie den Algorithmus kennt?
- Unser Algo braucht eine Komponente die sie nicht vorhersehen kann!
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
 - Die Wahl von QuickSort Adversary wird irrelevant!

Zufall!



- Problem:
 - Extra Aufwand (um sortieren zu können, wird erstmal gemischt)
 - Wird die Laufzeit dadurch wirklich besser?



Adapted from <https://scryfall.com/card/afr/79/tricksters-talisman>

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.
?

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.
- Wie kommt die zufällige [?]Permutation zustande?

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.
- Wie kommt die zufällige [?]Permutation zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle _, _, _, _, _, _, _, _, _, _ \rangle$

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.

- Wie kommt die zufällige [?]Permutation zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle$
→
 $\langle \square, _, _, _, _, _, _, _, _, _ \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.

- Wie kommt die zufällige [?]Permutation zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle \square, _, _, _, _, _, _, _, _, _ \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.
- Wie kommt die zufällige [?]Permutation zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, _, _, _, _, _, _, _, _, _ \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt **eins** aus

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.
- Wie kommt die zufällige [?]Permutation zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, _, _, _, _, _, _, _, _, _ \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
- Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.

- Wie kommt die zufällige [?]Permutation zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, _, _, _, _, _, _, _, _ \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
- Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.
- Wie kommt die zufällige [?] **Permutation** zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, _, _, _, _, _, _, _, _ \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
- Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus
- ...
- Für Index i : Wähle aus $n - i$ Elementen gleich verteilt eins aus

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.

- Wie kommt die zufällige [?] **Permutation** zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, a_7, a_6, a_3, a_9, a_4, a_2, a_0, a_8 \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
- Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus
- ...
- Für Index i : Wähle aus $n - i$ Elementen gleich verteilt eins aus

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.
- Wie kommt die zufällige [?] **Permutation** zustande?
 - $\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, a_7, a_6, a_3, a_9, a_4, a_2, a_0, a_8 \rangle$
 - Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
 - Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus
 - ...
 - Für Index i : Wähle aus $n - i$ Elementen gleich verteilt eins aus
- Welches Element landet beim Mischen an erster Stelle?

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.

- Wie kommt die zufällige [?] **Permutation** zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, a_7, a_6, a_3, a_9, a_4, a_2, a_0, a_8 \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
- Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus
- ...
- Für Index i : Wähle aus $n - i$ Elementen gleich verteilt eins aus
- Welches Element landet beim Mischen an erster Stelle?

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.

- Wie kommt die zufällige [?] **Permutation** zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, a_7, a_6, a_3, a_9, a_4, a_2, a_0, a_8 \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
- Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus
- ...
- Für Index i : Wähle aus $n - i$ Elementen gleich verteilt eins aus
- Welches Element landet beim Mischen an erster Stelle?
 - Das Pivot-Element wird nur von der ersten zufälligen Wahl beeinflusst

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir das erste Element als Pivot.

- Wie kommt die zufällige [?] **Permutation** zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, a_7, a_6, a_3, a_9, a_4, a_2, a_0, a_8 \rangle$

- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
- Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus
- ...
- Für Index i : Wähle aus $n - i$ Elementen gleich verteilt eins aus
- **Welches Element landet beim Mischen an erster Stelle?**
 - Das Pivot-Element wird nur von der ersten zufälligen Wahl beeinflusst
 - Die Sortierung der restlichen Elemente ist irrelevant

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir **das erste Element als Pivot.**

- Wie kommt die zufällige [?] **Permutation** zustande?

$\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, a_7, a_6, a_3, a_9, a_4, a_2, a_0, a_8 \rangle$

- **Für Index 0: Wähle aus n Elementen gleich verteilt eins aus**
 - Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus
 - ...
 - Für Index i : Wähle aus $n - i$ Elementen gleich verteilt eins aus
- **Welches Element landet beim Mischen an erster Stelle?**
 - Das Pivot-Element wird nur von der ersten zufälligen Wahl beeinflusst
 - Die Sortierung der restlichen Elemente ist irrelevant
- Demzufolge kann als Pivot auch einfach direkt ein *zufälliges* Element gewählt werden

QuickSort – Pivots

- Nach dem zufälligen Mischen des Arrays wählen wir **das erste Element als Pivot.**
- Wie kommt die zufällige Permutation zustande?
 $\langle a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9 \rangle \longrightarrow \langle a_5, a_1, a_7, a_6, a_3, a_9, a_4, a_2, a_0, a_8 \rangle$
- Für Index 0: Wähle aus n Elementen gleich verteilt eins aus
 - Für Index 1: Wähle aus $n - 1$ Elementen gleich verteilt eins aus
 - ...
 - Für Index i : Wähle aus $n - i$ Elementen gleich verteilt eins aus
- Welches Element landet beim Mischen an erster Stelle?
 - Das Pivot-Element wird nur von der ersten zufälligen Wahl beeinflusst
 - Die Sortierung der restlichen Elemente ist irrelevant
- Demzufolge kann als Pivot auch einfach direkt ein *zufälliges* Element gewählt werden

Bei einem beliebigen aber **festen Pivot** kann Quicksort Adversary eine schlechte Laufzeit erzwingen.
 Bei einem **zufällig gewählten Pivot** kann Quicksort Adversary die Laufzeit nicht beeinflussen.

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**
- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**
 - Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij}$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**
 - Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij}$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**
 - Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij}$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**
 - Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij}$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**
 - Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij}$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**
 - Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij}$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**
 - Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{E}[X_{ij}]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!

Zufall!

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{E}[X_{ij}]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

Dank der Linearität von \mathbb{E} können wir die X_{ij} einzeln betrachten, anstatt alle auf einmal!

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{E}[X_{ij}]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

Dank der Linearität von \mathbb{E} können wir die X_{ij} einzeln betrachten, anstatt alle auf einmal!

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{E}[X_{ij}]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

Indikatorvariable Y : $\mathbb{E}[Y] = 0 \cdot \Pr[Y = 0] + 1 \cdot \Pr[Y = 1]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

Dank der Linearität von \mathbb{E} können wir die X_{ij} einzeln betrachten, anstatt alle auf einmal!

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{E}[X_{ij}]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

Indikatorvariable Y : $\mathbb{E}[Y] = 0 \cdot \Pr[Y = 0] + 1 \cdot \Pr[Y = 1]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

Dank der Linearität von \mathbb{E} können wir die X_{ij} einzeln betrachten, anstatt alle auf einmal!

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{E}[X_{ij}] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

Indikatorvariable Y : $\mathbb{E}[Y] = 0 \cdot \Pr[Y = 0] + 1 \cdot \Pr[Y = 1]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten? $O(n \log(n))$ (siehe VL)
- Jetzt: Alternativer Beweis

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

- Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
- Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
- Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
- Das hängt davon ab, welche Elemente als Pivots gewählt werden! **Zufall!**

- Indikatorzufallsvariable $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

Dank der Linearität von \mathbb{E} können wir die X_{ij} einzeln betrachten, anstatt alle auf einmal!

$$X = \sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \rightarrow \mathbb{E}[X] = \mathbb{E} \left[\sum_{i=1}^n \sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \mathbb{E} \left[\sum_{j=i+1}^n X_{ij} \right] = \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{E}[X_{ij}] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

Linearität von \mathbb{E} : $\mathbb{E}[c \cdot Y + d \cdot Z] = c \cdot \mathbb{E}[Y] + d \cdot \mathbb{E}[Z]$

Indikatorvariable Y : $\mathbb{E}[Y] = 0 \cdot \Pr[Y = 0] + 1 \cdot \Pr[Y = 1]$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das **Pivot** ist links von a_i oder rechts von a_j

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das **Pivot** ist links von a_i oder rechts von a_j

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das **Pivot** ist links von a_i oder rechts von a_j

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das **Pivot** ist links von a_i oder rechts von a_j

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das **Pivot** ist links von a_i oder rechts von a_j
 - a_i und a_j kommen in das gleiche Teilarray

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das **Pivot** ist links von a_i oder rechts von a_j
 - a_i und a_j kommen in das gleiche Teilarray

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

Dieses Pivot hat keinen Einfluss darauf ob a_i und a_j verglichen werden!

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das Pivot ist links von a_i oder rechts von a_j
 - a_i und a_j kommen in das gleiche Teilarray
 - Fall 2: Das Pivot ist zwischen a_i und a_j

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

Dieses Pivot hat keinen Einfluss darauf ob a_i und a_j verglichen werden!

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das Pivot ist links von a_i oder rechts von a_j
 - a_i und a_j kommen in das gleiche Teilarray
 - Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

Dieses Pivot hat keinen Einfluss darauf ob a_i und a_j verglichen werden!

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das Pivot ist links von a_i oder rechts von a_j
 - a_i und a_j kommen in das gleiche Teilarray
 - Fall 2: Das **Pivot** ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

Dieses Pivot hat keinen Einfluss darauf ob a_i und a_j verglichen werden!

Mit diesem Pivot werden a_i und a_j niemals verglichen!

$$X_{ij} = 0$$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.
- Welche Laufzeit können wir erwarten?
- Jetzt: Alternativer Beweis
 - Laufzeit ist bestimmt durch die Anzahl der Vergleiche X
 - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
 - Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?
 - Das hängt davon ab, welche Elemente als Pivots gewählt werden!
 - Fall 1: Das Pivot ist links von a_i oder rechts von a_j
 - a_i und a_j kommen in das gleiche Teilarray
 - Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
 - Fall 3: Das Pivot ist a_i oder a_j

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

Dieses Pivot hat keinen Einfluss darauf ob a_i und a_j verglichen werden!

Mit diesem Pivot werden a_i und a_j niemals verglichen!

$X_{ij} = 0$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.

- Welche Laufzeit können wir erwarten?

- Jetzt: Alternativer Beweis

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

- Laufzeit ist bestimmt durch die Anzahl der Vergleiche X

- Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

- Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?

- Das hängt davon ab, welche Elemente als Pivots gewählt werden!

- Fall 1: Das Pivot ist links von a_i oder rechts von a_j

Dieses Pivot hat keinen Einfluss darauf ob a_i und a_j verglichen werden!

- a_i und a_j kommen in das gleiche Teilarray

- Fall 2: Das Pivot ist zwischen a_i und a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

- a_i und a_j kommen in verschiedene Teilarrays

- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Mit zufälliger Wahl des Pivots ist es unwahrscheinlicher, dass wir den Worst Case treffen.

- Welche Laufzeit können wir erwarten?

- Jetzt: Alternativer Beweis

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

- Laufzeit ist bestimmt durch die Anzahl der Vergleiche X

- Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

- Betrachte jedes Paar a_i, a_j von Elementen und frage: Werden a_i und a_j verglichen?

- Das hängt davon ab, welche Elemente als Pivots gewählt werden!

- Fall 1: Das Pivot ist links von a_i oder rechts von a_j

Dieses Pivot hat keinen Einfluss darauf ob a_i und a_j verglichen werden!

- a_i und a_j kommen in das gleiche Teilarray

- Fall 2: Das Pivot ist zwischen a_i und a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

- a_i und a_j kommen in verschiedene Teilarrays

- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt
- Dieser Bereich enthält $j - i + 1$ Elemente

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt
- Dieser Bereich enthält $j - i + 1$ Elemente

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$



- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

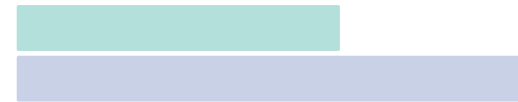
Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt
- Dieser Bereich enthält $j - i + 1$ Elemente

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$



- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

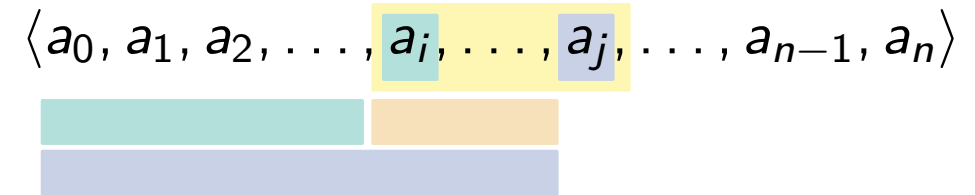
Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt
- Dieser Bereich enthält $j - i + 1$ Elemente

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$



- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

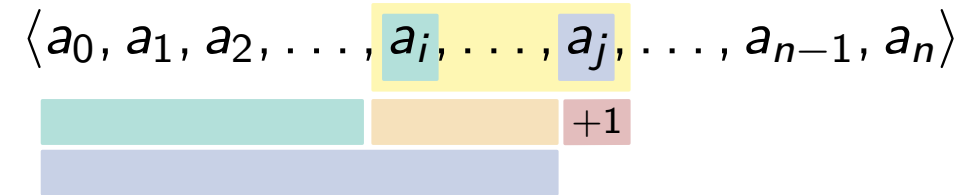
Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt
- Dieser Bereich enthält $j - i + 1$ Elemente

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$



- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt
- Dieser Bereich enthält $j - i + 1$ Elemente
- Wie wahrscheinlich ist es, dass $X_{ij} = 1$ gilt, wenn wir schon wissen, dass das Pivot in diesem Bereich ist?

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt
- Dieser Bereich enthält $j - i + 1$ Elemente $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
- Wie wahrscheinlich ist es, dass $X_{ij} = 1$ gilt, wenn wir schon wissen, dass das Pivot in diesem Bereich ist?

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

- Betrachte die Pivots die zufällig gewählt werden
- Pivots außerhalb von $\{a_i, \dots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \dots, a_j\}$ fällt
- Dieser Bereich enthält $j - i + 1$ Elemente $\langle a_0, a_1, a_2, \dots, a_i, \dots, a_j, \dots, a_{n-1}, a_n \rangle$
- Wie wahrscheinlich ist es, dass $X_{ij} = 1$ gilt, wenn wir schon wissen, dass das Pivot in diesem Bereich ist?

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

- Fall 2: Das Pivot ist zwischen a_i und a_j
 - a_i und a_j kommen in verschiedene Teilarrays
- Fall 3: Das Pivot ist a_i oder a_j

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$

Mit diesem Pivot werden a_i und a_j auf jeden Fall verglichen! $X_{ij} = 1$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

QuickSort – Pivots

$$\begin{aligned}\mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\ &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}\end{aligned}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}$$

$$j = i+1 \rightarrow j-i+1 = i+1-i+1 = 2$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}
 \end{aligned}$$

$j = i+2 \rightarrow j-i+1 = i+2-i+1 = 3$
 $j = i+1 \rightarrow j-i+1 = i+1-i+1 = 2$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}$$

$$\begin{array}{lll}
 j = n & \rightarrow j-i+1 = & n-i+1 \\
 \dots & \dots & \dots \\
 j = i+2 & \rightarrow j-i+1 = i+2-i+1 = & 3 \\
 j = i+1 & \rightarrow j-i+1 = i+1-i+1 = & 2
 \end{array}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}$$

$$\begin{array}{lll}
 j = n & \rightarrow j-i+1 = & n-i+1 \\
 \dots & \dots & \dots \\
 j = i+2 & \rightarrow j-i+1 = i+2-i+1 = & 3 \\
 j = i+1 & \rightarrow j-i+1 = i+1-i+1 = & 2
 \end{array}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}$$

$$\begin{array}{l}
 j = n \quad \rightarrow j-i+1 = \quad \quad \quad n-i+1 \\
 \dots \quad \quad \quad \dots \quad \quad \quad \dots \\
 j = i+2 \rightarrow j-i+1 = i+2-i+1 = 3 \\
 j = i+1 \rightarrow j-i+1 = i+1-i+1 = 2
 \end{array}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}$$

$$\begin{array}{l}
 j = n \rightarrow j - i + 1 = n - i + 1 \\
 \dots \quad \dots \\
 j = i + 2 \rightarrow j - i + 1 = i + 2 - i + 1 = 3 \\
 j = i + 1 \rightarrow j - i + 1 = i + 1 - i + 1 = 2
 \end{array}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}$$

$$= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k}$$

$$\begin{array}{l}
 j = n \rightarrow j - i + 1 = \dots \quad n - i + 1 \\
 \dots \quad \dots \\
 j = i + 2 \rightarrow j - i + 1 = i + 2 - i + 1 = 3 \\
 j = i + 1 \rightarrow j - i + 1 = i + 1 - i + 1 = 2
 \end{array}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}$$

$$= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k}$$

$$\begin{array}{l}
 j = n \rightarrow j - i + 1 = n - i + 1 \\
 \dots \quad \dots \\
 j = i + 2 \rightarrow j - i + 1 = i + 2 - i + 1 = 3 \\
 j = i + 1 \rightarrow j - i + 1 = i + 1 - i + 1 = 2
 \end{array}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1}$$

$$= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k}$$

$$\begin{array}{l}
 j = n \rightarrow j - i + 1 = n - i + 1 \\
 \dots \quad \dots \\
 j = i + 2 \rightarrow j - i + 1 = i + 2 - i + 1 = 3 \\
 j = i + 1 \rightarrow j - i + 1 = i + 1 - i + 1 = 2
 \end{array}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k}
 \end{aligned}$$

$$\begin{aligned}
 j = n &\rightarrow j-i+1 = n-i+1 \\
 \dots &\dots \\
 j = i+2 &\rightarrow j-i+1 = i+2-i+1 = 3 \\
 j = i+1 &\rightarrow j-i+1 = i+1-i+1 = 2
 \end{aligned}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k}
 \end{aligned}$$

$\frac{2}{k} > 0$, mehr Summanden machen die Summe größer

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

- Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

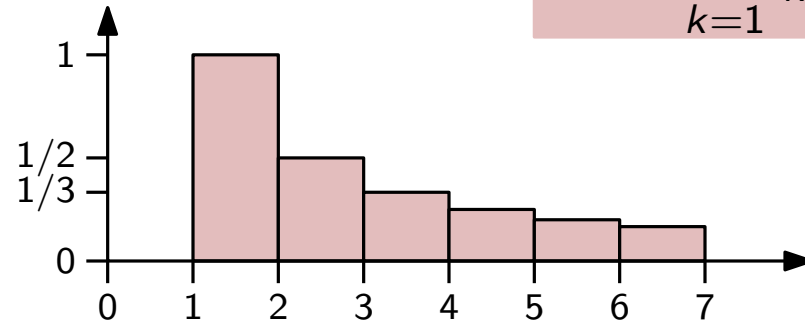
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$



$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

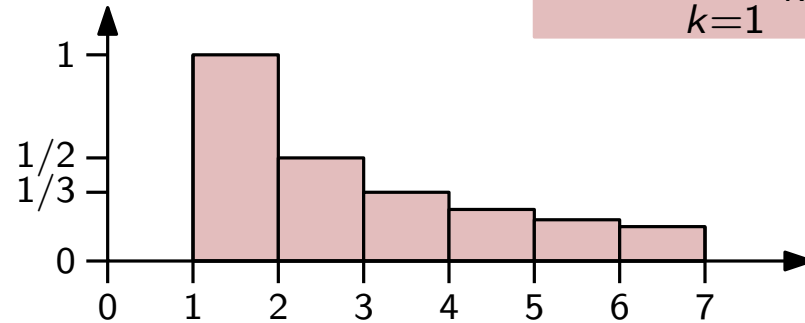
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$



$$\int_1^n \frac{1}{x} dx \leq \sum_{k=1}^n \frac{1}{k} = H_n$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

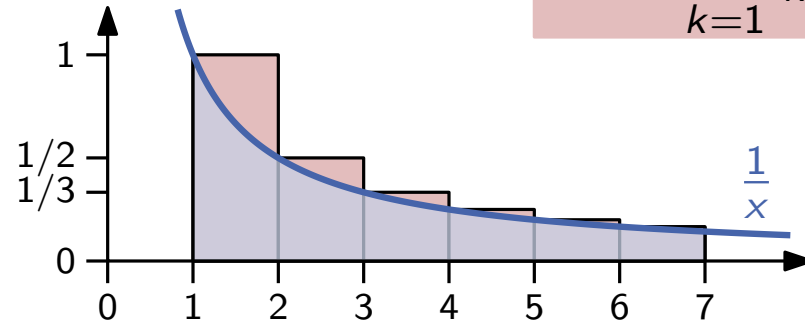
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$



$$\int_1^n \frac{1}{x} dx \leq \sum_{k=1}^n \frac{1}{k} = H_n$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

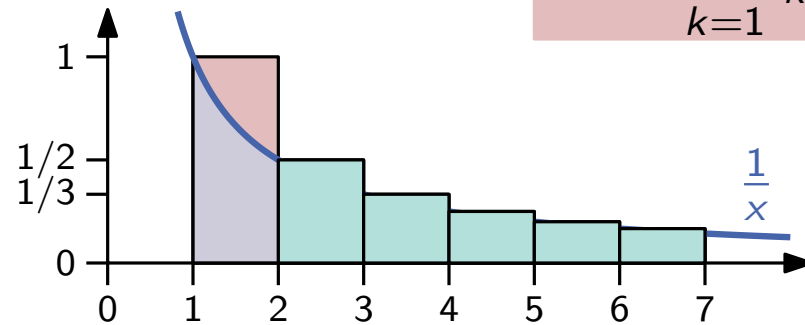
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$



$$\int_1^n \frac{1}{x} dx \leq \sum_{k=1}^n \frac{1}{k} = H_n$$

$= H_n - 1$

$$\sum_{k=2}^n \frac{1}{k}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

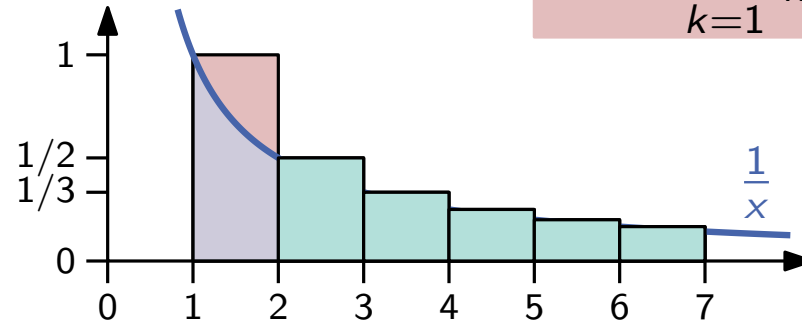
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$



$$\int_1^n \frac{1}{x} dx \leq \sum_{k=1}^n \frac{1}{k} = H_n$$

$$= H_n - 1$$

$$\sum_{k=2}^n \frac{1}{k} = \sum_{k=1}^{n-1} \frac{1}{k+1}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

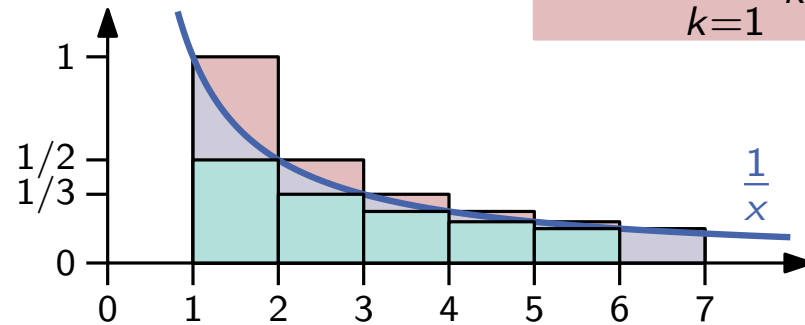
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$



$$\int_1^n \frac{1}{x} dx \leq \sum_{k=1}^n \frac{1}{k} = H_n$$

$$= H_n - 1$$

$$\sum_{k=2}^n \frac{1}{k} = \sum_{k=1}^{n-1} \frac{1}{k+1}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

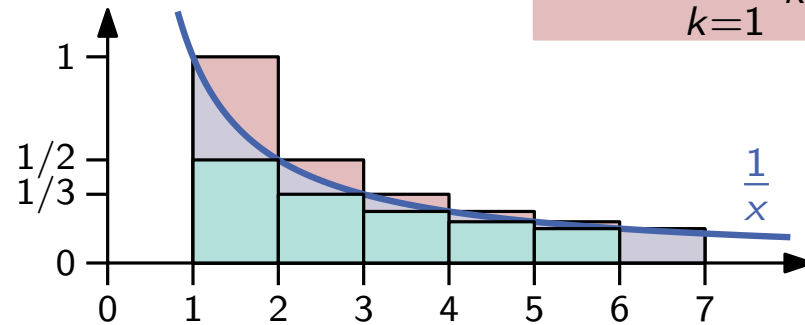
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$



$$\int_1^n \frac{1}{x} dx \leq \sum_{k=1}^n \frac{1}{k} = H_n$$

↓

$$\sum_{k=2}^n \frac{1}{k} \leq \int_1^n \frac{1}{x} dx$$

= $H_n - 1$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

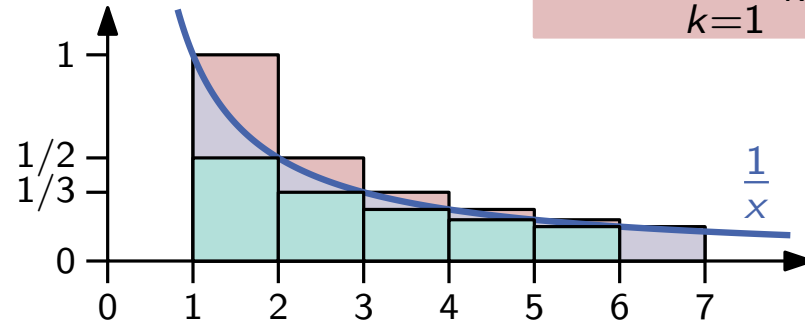
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k}$



$$\begin{aligned}
 \int_1^n \frac{1}{x} dx &\leq \sum_{k=1}^n \frac{1}{k} = H_n \\
 &= H_n - 1 \\
 \sum_{k=2}^n \frac{1}{k} &\leq \int_1^n \frac{1}{x} dx = [\ln(x)]_1^n = \ln(n)
 \end{aligned}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

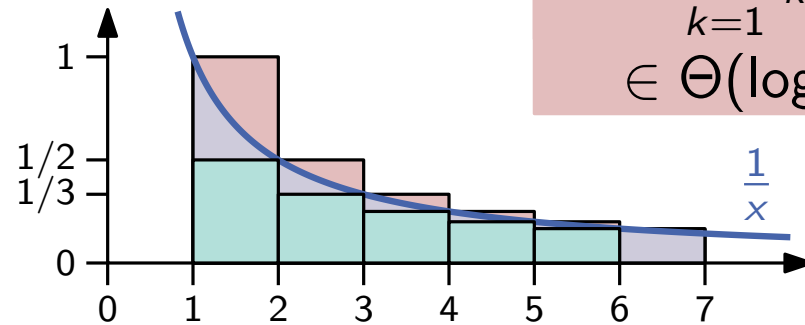
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k} \in \Theta(\log(n))$



$$\begin{aligned}
 \int_1^n \frac{1}{x} dx &\leq \sum_{k=1}^n \frac{1}{k} = H_n \\
 &= H_n - 1 \\
 \sum_{k=2}^n \frac{1}{k} &\leq \int_1^n \frac{1}{x} dx = [\ln(x)]_1^n = \ln(n)
 \end{aligned}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

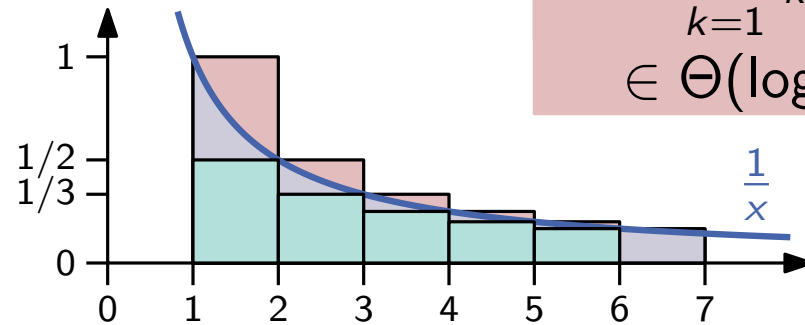
$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\begin{aligned}
 \mathbb{E}[X] &= \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1] \\
 &= \sum_{i=1}^n \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^n \sum_{k=2}^{n-i+1} \frac{2}{k} \\
 &\leq \sum_{i=1}^n \sum_{k=2}^n \frac{2}{k} \\
 &= n \cdot 2 \cdot \sum_{k=2}^n \frac{1}{k}
 \end{aligned}$$

Harmonische Summe

■ Harmonische Zahl $H_n = \sum_{k=1}^n \frac{1}{k} \in \Theta(\log(n))$



$$\begin{aligned}
 \int_1^n \frac{1}{x} dx &\leq \sum_{k=1}^n \frac{1}{k} = H_n \\
 &= H_n - 1 \\
 \sum_{k=2}^n \frac{1}{k} &\leq \int_1^n \frac{1}{x} dx = [\ln(x)]_1^n = \ln(n)
 \end{aligned}$$

$$\mathbb{E}[X] = \sum_{i=1}^n \sum_{j=i+1}^n \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

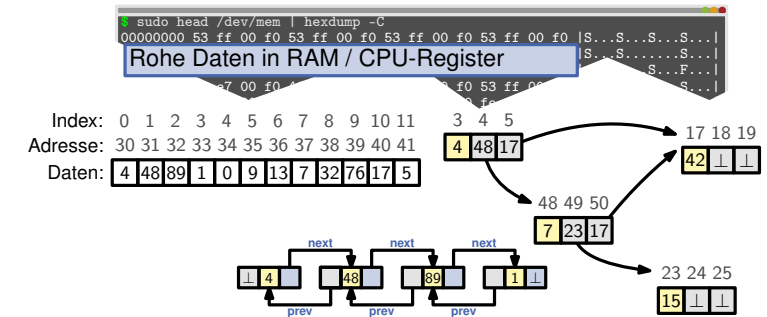
Theorem

Auf jeder Eingabe der Länge n benötigt QuickSort mit zufälligen Pivots erwartet $\leq 2n \ln(n)$ Vergleiche.

Zusammenfassung

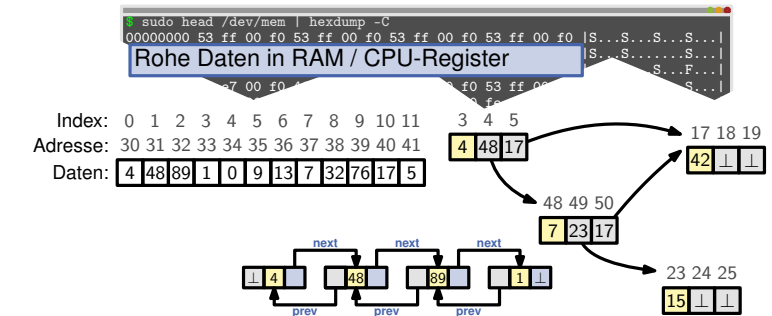
Zusammenfassung

- Datenstrukturen
 - Was sind Datenstrukturen?
 - Nützlichkeit des Konzepts Datenstruktur



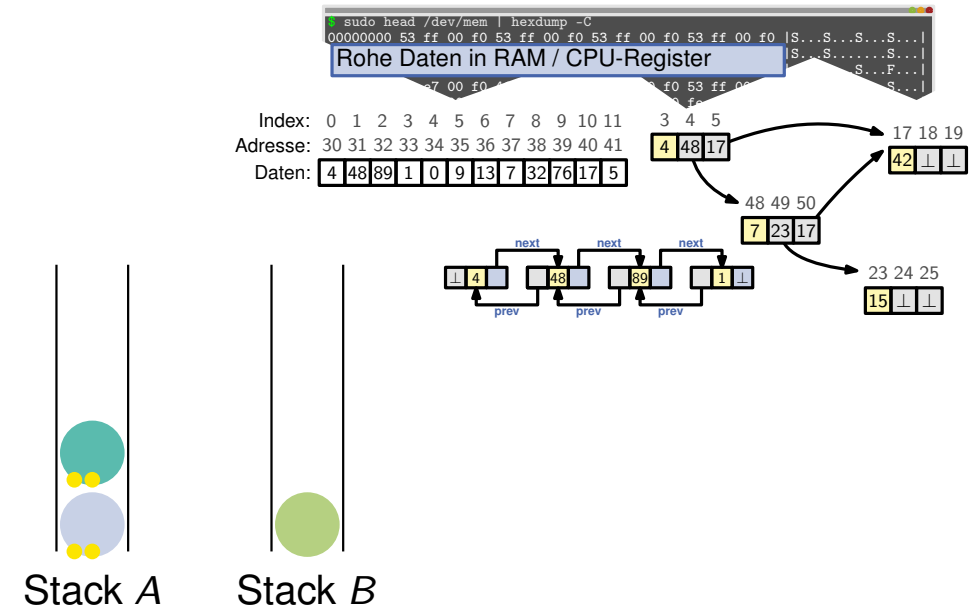
Zusammenfassung

- Datenstrukturen
 - Was sind Datenstrukturen?
 - Nützlichkeit des Konzepts Datenstruktur
- Amortisierte Analyse
 - Beispiele, Korrektheit Potentialmethode
 - Binärzähler
 - Queue aus Stacks



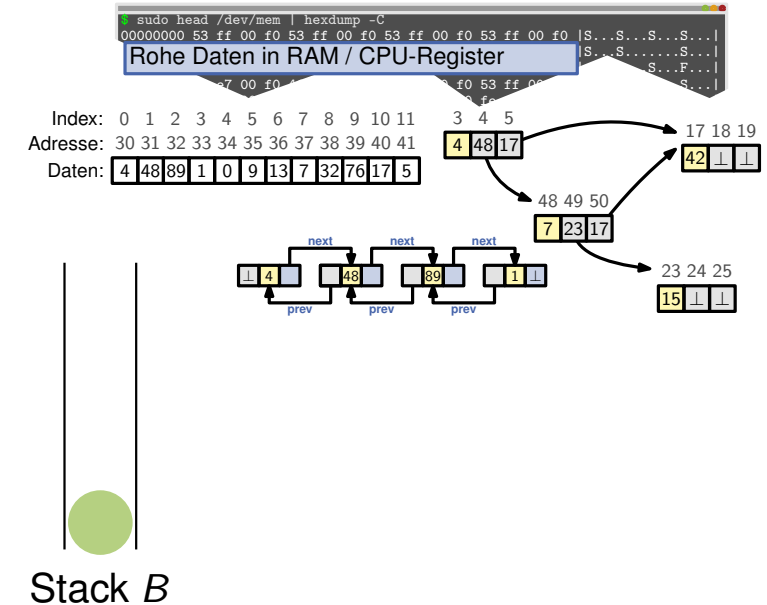
Zusammenfassung

- Datenstrukturen
 - Was sind Datenstrukturen?
 - Nützlichkeit des Konzepts Datenstruktur
- Amortisierte Analyse
 - Beispiele, Korrektheit Potentialmethode
 - Binärzähler
 - Queue aus Stacks



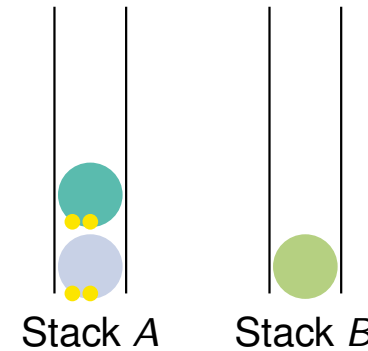
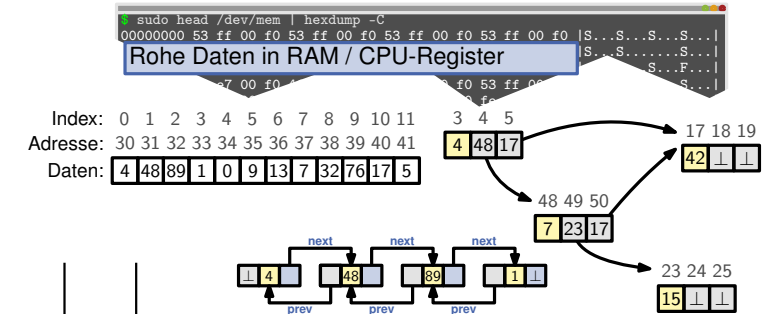
Zusammenfassung

- Datenstrukturen
 - Was sind Datenstrukturen?
 - Nützlichkeit des Konzepts Datenstruktur
- Amortisierte Analyse
 - Beispiele, Korrektheit Potentialmethode
 - Binärzähler
 - Queue aus Stacks
- Sortieren
 - Wiederholung: Insertion-Sort, Merge-Sort, Quick-Sort
 - Adversary Sichtweise und zufälliges Pivot-Element
 - Analyse der erwarteten Laufzeit von Quick-Sort



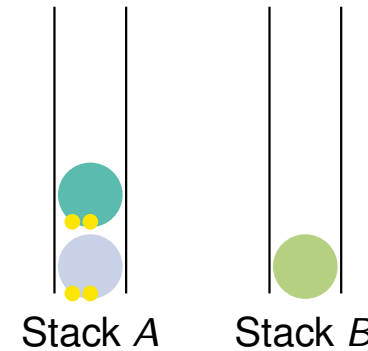
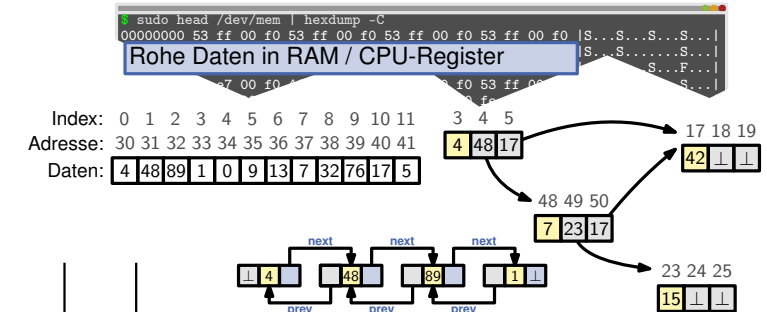
Zusammenfassung

- Datenstrukturen
 - Was sind Datenstrukturen?
 - Nützlichkeit des Konzepts Datenstruktur
- Amortisierte Analyse
 - Beispiele, Korrektheit Potentialmethode
 - Binärzähler
 - Queue aus Stacks
- Sortieren
 - Wiederholung: Insertion-Sort, Merge-Sort, Quick-Sort
 - Adversary Sichtweise und zufälliges Pivot-Element
 - Analyse der erwarteten Laufzeit von Quick-Sort



Zusammenfassung

- Datenstrukturen
 - Was sind Datenstrukturen?
 - Nützlichkeit des Konzepts Datenstruktur
- Amortisierte Analyse
 - Beispiele, Korrektheit Potentialmethode
 - Binärzähler
 - Queue aus Stacks
- Sortieren
 - Wiederholung: Insertion-Sort, Merge-Sort, Quick-Sort
 - Adversary Sichtweise und zufälliges Pivot-Element
 - Analyse der erwarteten Laufzeit von Quick-Sort



Zusammenfassung

- Datenstrukturen
 - Was sind Datenstrukturen?
 - Nützlichkeit des Konzepts Datenstruktur
- Amortisierte Analyse
 - Beispiele, Korrektheit Potentialmethode
 - Binärzähler
 - Queue aus Stacks
- Sortieren
 - Wiederholung: Insertion-Sort, Merge-Sort, Quick-Sort
 - Adversary Sichtweise und zufälliges Pivot-Element
 - Analyse der erwarteten Laufzeit von Quick-Sort
 - $E[X + Y] = E[X] + E[Y]$, Integral um Summen abzuschätzen

