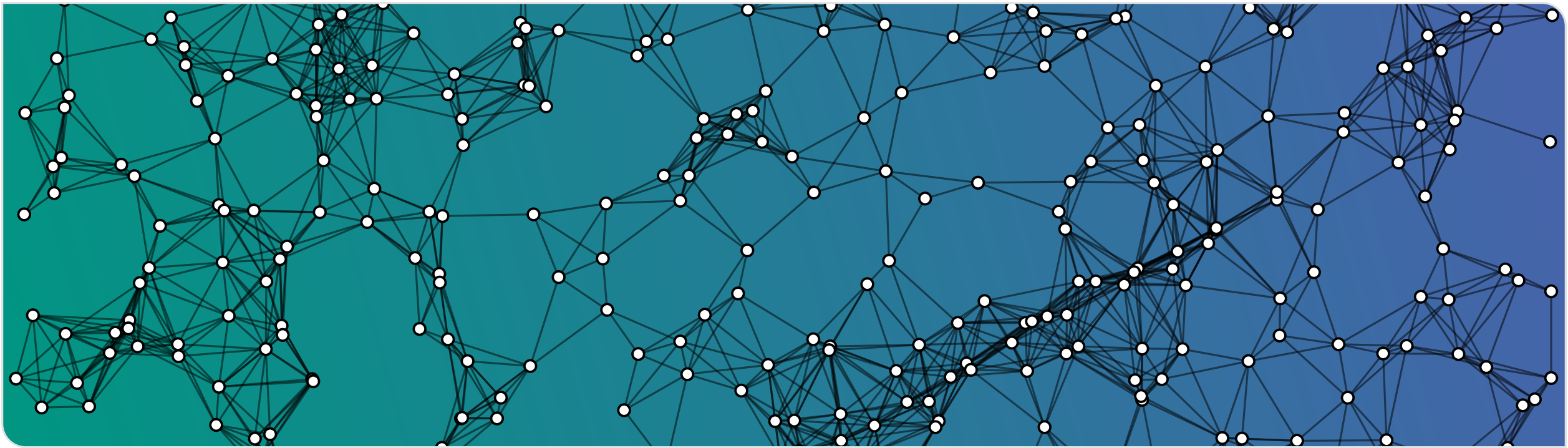
Karlsruher Institut für Technologie

Algorithmen 1

Übung 3
Sortieren, Hashing



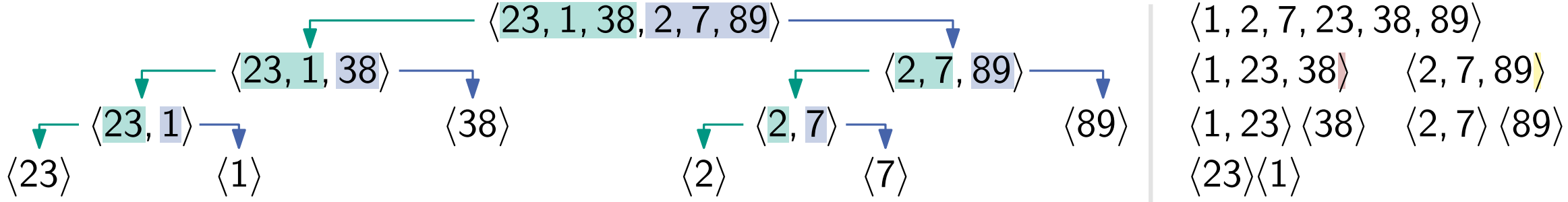
Sortieren

- 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

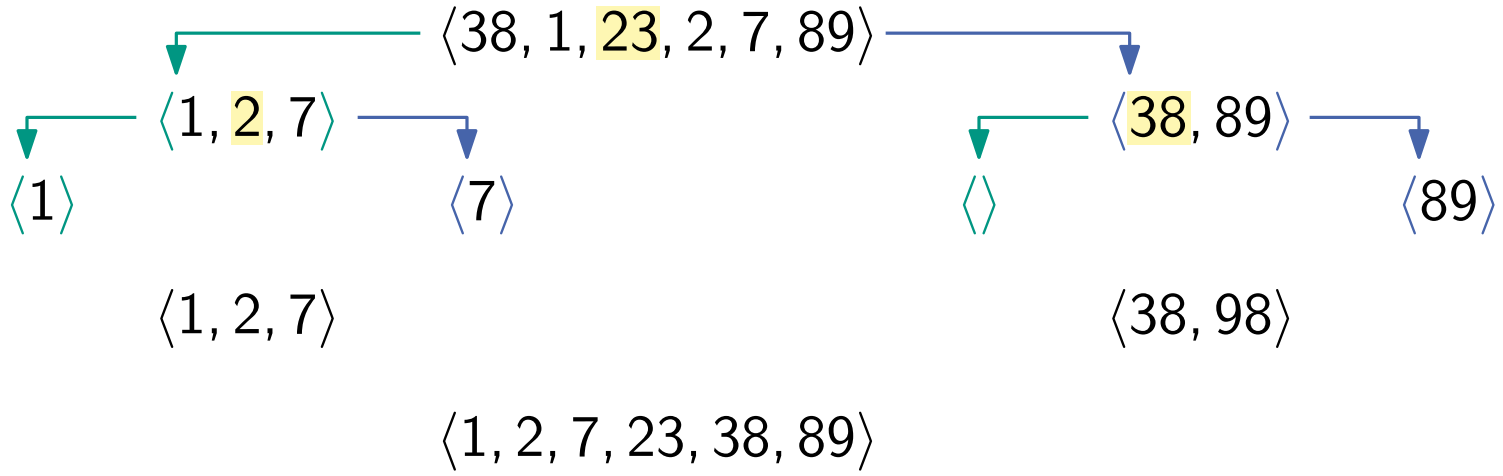
- InsertionSort ($\Theta(n^2)$)
  - Position jeden Elements durch binäre Suche im bisher sortierten Teil ermitteln

    $\langle 1, 2, 7, 23, 38, 89 \rangle$
- MergeSort ($\Theta(n \log(n))$)
  - Teile ein Array rekursiv in zwei Hälften, welche wieder zusammengeführt werden



Sortieren

- QuickSort ($\Theta(n \log(n))$)
  - 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

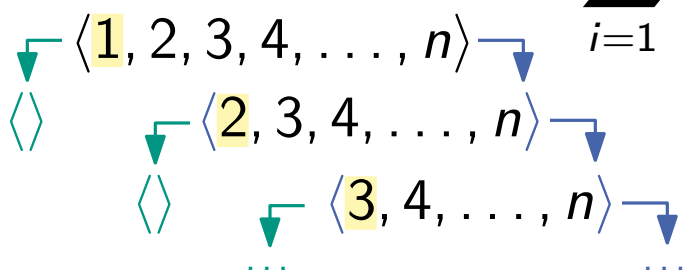
- Worst Case vs. Average Case – Adversary Sichtweise
  - Teile ein Array rekursiv anhand eines Pivots in zwei Teile, welche jeweils alle Elemente kleiner bzw. größer dem Pivot enthalten

$$\langle 38, 1, 23, 2, 7, 89 \rangle$$

- Wenn wir nichts über die Eingabe wissen, kann jeder Eintrag im Array ein gutes oder schlechtes Pivot-Element sein
  - wähle irgendein Element als Pivot $\rightarrow$ Beispiel: Pivot ist immer der erste Eintrag im Array
  - Was kann schlimmsten Falls passieren?

$$\sum_{i=1}^{n} i = \Theta\left(n^2\right)$$

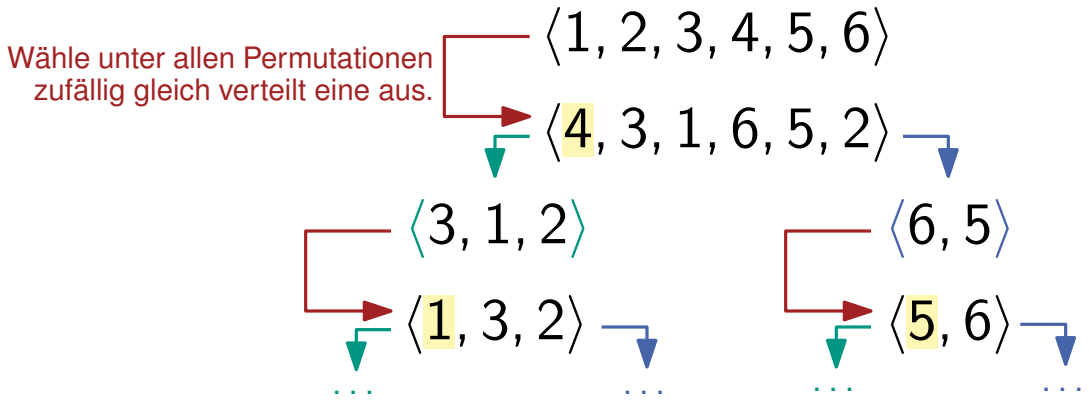
- Auftritt: Adversary
  - kennt den Algorithmus
  - baut eine Instanz die für schlechte Laufzeit sorgt



Adapted from https://scryfall.com/card/mid/129/bloodthirsty-adversary

QuickSort – Pivots

- Wie können wir Quicksort Adversary die Tour vermasseln?
  - ... selbst wenn sie den Algorithmus kennt?
  - ... unser Algo braucht eine Komponente die sie nicht vorhersehen kann **Zufall!**
- Idee: Bevor wir das Pivot-Element wählen, wird die Eingabe zufällig gemischt.
  - Die Wahl von Quicksort Adversary wird irrelevant!



- 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.
- Welches Element landet beim Mischen an erster Stelle?
  - 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
- 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.
- 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 $\langle a_0, a_1, a_2, \ldots, a_i, \ldots, a_j, \ldots, a_{n-1}, a_n \rangle$
  - 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 $X_{ij} = \begin{cases} 1, & \text{wenn } a_i \text{ und } a_j \text{ verglichen werden} \\ 0, & \text{sonst} \end{cases}$

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1]$$

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: $\mathbb{E}[Y] = \cancel{0 \cdot \Pr[Y = 0]} + 1 \cdot \Pr[Y = 1]$

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

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1]$$

  - Betrachte die Elemente in sortierter Reihenfolge $\langle a_0, a_1, a_2, \ldots, a_i, \ldots, a_j, \ldots, a_{n-1}, a_n \rangle$
  - 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

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! $X_{ij} = 0$

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 $\{a_i, \ldots, a_j\}$ interessieren uns nicht
- Das Schicksal von X_{ij} wird besiegelt wenn das erste Pivot in $\{a_i, \ldots, a_j\}$ 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?

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1]$$

$$\langle a_0, a_1, a_2, \ldots, a_i, \ldots, a_j, \ldots, a_{n-1}, a_n \rangle$$

$$\Pr[X_{ij} = 1] = \frac{2}{j - i + 1}$$

  - Fall 2: Das Pivot ist zwischen a_i und a_j
    - a_i und a_j kommen in verschiedene Teilarrays

    Mit diesem Pivot werden a_i und a_j niemals verglichen! $X_{ij} = 0$
  - 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

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

$$= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k}$$

$$\begin{array}{lll} j = n & \to j-i+1 = & n-i+1 \\ \cdots & \cdots & \cdots \\ j = i+2 & \to j-i+1 = i+2-i+1 = & 3 \\ j = i+1 & \to j-i+1 = i+1-i+1 = & 2 \end{array}$$

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

QuickSort – Pivots

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1}$$

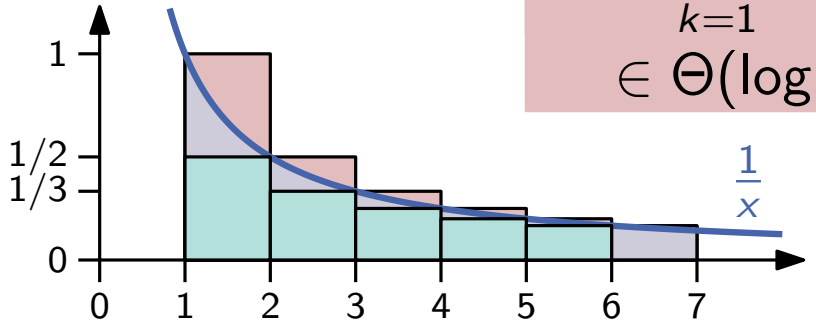
$$= \sum_{i=1}^{n} \sum_{k=2}^{n-i+1} \frac{2}{k}$$

$$\leq \sum_{i=1}^{n} \sum_{k=2}^{n} \frac{2}{k}$$

$$= n \cdot 2 \cdot \sum_{k=2}^{n} \frac{1}{k}$$

Harmonische Summe

- Harmonische Zahl $H_n = \sum_{k=1}^{n} \frac{1}{k} \in \Theta(\log(n))$



$$\int_1^n \frac{1}{x}\,dx \leq \sum_{k=1}^{n} \frac{1}{k} = H_n$$

$$\sum_{k=2}^{n} \frac{1}{k} \leq \int_1^n \frac{1}{x}\,dx = \left[\ln(x)\right]_1^n = \ln(n)$$

$= H_n - 1$

$$\mathbb{E}[X] = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \Pr[X_{ij} = 1]$$

$$\Pr[X_{ij} = 1] = \frac{2}{j-i+1}$$

Theorem
Auf jeder Eingabe der Länge n benötigt QuickSort mit zufälligen Pivots erwartet $\leq 2n\ln(n)$ Vergleiche.

Hashing

- Wir haben eine Menge von Paaren bestehend aus Schlüsseln und Werten
- Gegeben einen Schlüssel: Was ist der zugehörige Wert?
- Beispiel: Wenn sich ein Nutzer zum Online-Banking einloggen will, muss geprüft werden, ob das eingegebene Passwort auch zu dem Nutzer gehört.

Nutzername	Passwort
`PickleRick`	`dasNutella`
`Monkeee`	`b@n@n@`
`el0nMarsk`	`B1GR0CK3T`
`Stewdent23`	`hunter2`
`ProfB`	`passwort`
`12345678910`	

  - Nutzer `PickleRick` versucht sich mit dem Passwort `dieNutella` einzuloggen
- Datenstruktur: **Hashtabelle**
  - **Array** hält die Werte (die Einträge heißen **Buckets**)
  - **Hashfunktion** h bildet Schlüssel auf Arrayindex ab
- Idee: h bildet Nutzername auf Länge des Namens ab

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

0	
1	
⋮	
7	`b@n@n@`
8	
9	`B1GR0CK3T`
10	`dasNutella`
⋮	

Hash-Kollision!
$h(x) = h(y)$ obwohl $x \neq y$

Hash-Kollisionen

- Es existiert keine Hash-Funktion die im Worst Case gut ist
- Adversary: Baut Instanz wo $|U|/m$ Schlüssel auf den selben Bucket abgebildet werden
- Idee: Kollisionen in Kauf nehmen
  - **Chaining**: Bucket kann mehrere Werte halten (Liste oder Array)
  - **Simple Uniform Hashing Assumption** ← Aber bitte nicht alle!
    - Jeder Schlüssel landet in jedem der Buckets mit der gleichen Wahrscheinlichkeit
    - unabhängig von vorher eingefügten Schlüsseln

Ist die Anzahl eingefügter (Key, Value)-Paare linear in m, landen davon *erwartet* nur konstant viele in einem Bucket.

Was wenn mehr eingefügt werden?

  - **dynamisch wachsende Hashtabelle**: Wenn mehr als m Elemente eingefügt wurden, verdopple die Arraygröße
  - **Random Choice**: Bei Verdopplung wird zufällig eine neue Hashfunktion gewählt

Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Woher?

Adversary die Tour vermasseln

Universelle Hashfamilie

- Menge H von Hashfunktionen bei der für alle $k_1 \neq k_2 \in U$ und ein zufälliges $h \in H$ gilt: $\Pr[h(k_1) = h(k_2)] \leq 1/m$
- $\Rightarrow$ Mit zufällig gewähltem $h \in H$ kollidieren zwei Schlüssel höchstens mit Wahrscheinlichkeit $1/m$.

Theorem
Eine Hashtabelle kann die Operationen Einfügen, Suchen und Löschen in erwartet und amortisiert konstanter Zeit ausführen. (bezüglich Folge ausgeführter Operationen)

Key Takeaways

- **ERWARTET**
- funktioniert in der Praxis richtig gut
- selbst eine Hashfunktion zu entwerfen ist typischerweise keine gute Idee...

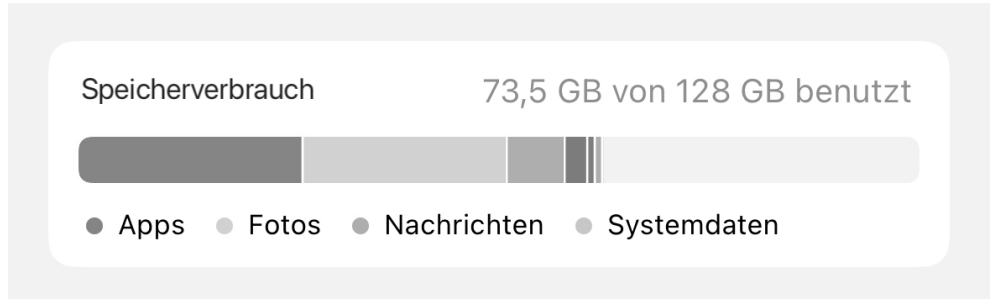
Notation
- U: Universum möglicher Schlüssel
- m: Größe des Arrays
- $h\colon U \to [0, m)$
- $|U| \gg m$

Platz sparen

- Problem: Smartphone-Speicher ist knapp
  - Fotos nehmen einen Großteil ein
- Idee: Platz sparen durch Löschen von Duplikaten
- Wie finden wir Duplikate?
- Annahme: Smartphone aus dem 18. Jhr.
  - Fotos in Graustufen
  - Wert eines Pixels in $\{0, \ldots, 255\}$
  - Ein Foto ist ein 2D Array
  - Duplikat: Arraywerte sind gleich
- Für zwei Fotos: Iteriere Pixel und überprüfe auf Gleicheit
- Nicht billig, aber nehmen wir in Kauf

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

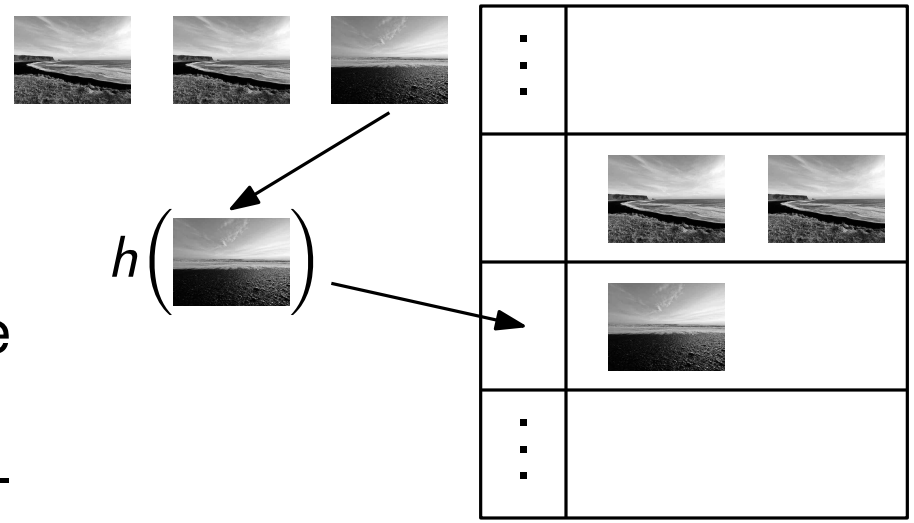


- Sei n die Anzahl der Bilder
- Vergleiche jedes Bild mit jedem anderen
- $\Theta(n^2)$ nicht-billige Vergleiche...

Das muss besser gehen...

Fotos hashen

- Definiere eine Hashfunktion h für Fotos
- Packe Fotos in Array von Buckets
- Vergleiche nur Fotos im selben Bucket
- Hash-Kollisionen?
  - Kein Problem, solange wir erwartet konstant viele Fotos pro Bucket haben
- Vergleiche jedes der n Fotos mit erwartet konstant vielen anderen Fotos
- Erwartete Laufzeit linear anstatt quadratisch!
- **Problem**: Wie hashen wir ein Foto?
- **Idee**: Summe der Pixelwerte
  - zwei Mal das gleiche Foto gibt den gleichen Wert
  - Duplikate landen im gleichen Bucket
  - h bildet auf große Wert ab $\rightarrow$ benutze zweite Hashfunktion um Indizes zu erhalten



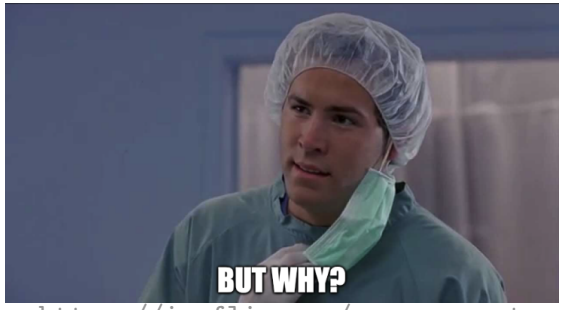
$h($  $) = 57 + 57 + 111 + 217 + \ldots$

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57

Fotos hashen

- Zwei Hashunfktionen
  - die erste bildet Fotos auf große Zahlen ab
  - die zweite bildet die großen Zahlen auf Buckets ab
- Das hat man tatsächlich gemacht ... und es ging so *richtig* schief!

https://youtu.be/r-TLSBdHe1A?t=2239



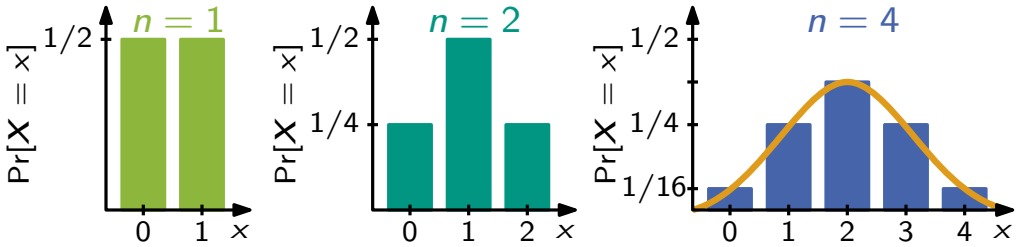
https://imgflip.com/memegenerator

Exkurs: Wahrscheinlichkeitstheorie

- Was passiert eigentlich, wenn ich viele Zufallsvariablen aufsummiere?
- Beispiel: Münzwürfe
  - n unabhängige Münzwürfe $X_1, X_2, \ldots, X_n$
  - Kopf: $X_i = 1$ mit $Pr[X_i = 1] = 1/2$, Zahl: $X_i = 0$ mit $Pr[X_i = 0] = 1/2$
  - Wie oft ist die Münze auf Kopf gelandet?

$$X = \sum_{i=1}^{n} X_i$$

Verteilung von X



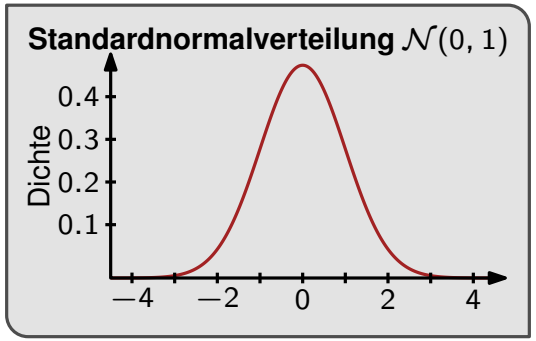
Wie viele Bit-Strings der Länge n gibt es die x mal eine 1 enthalten?

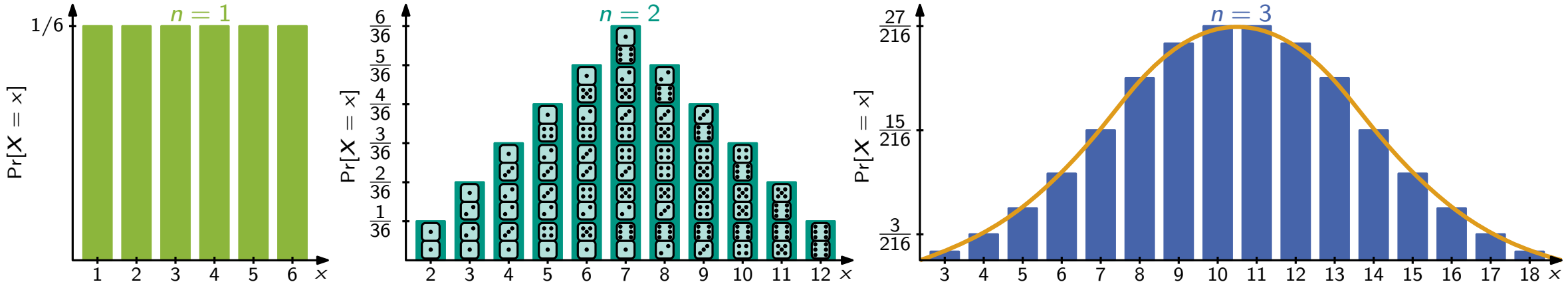
$$\Pr[X = x] = \frac{\binom{n}{x}}{2^n}$$

Wie viele Bit-Strings der Länge n gibt es?

Exkurs: Wahrscheinlichkeitstheorie

- Beispiel: 6-seitige Münzen aka Würfel
  - n unabhängige Würfelwürfe $X_1, X_2, \ldots, X_n$
  - Wurf: $X_i \in \{1, \ldots, 6\}$ mit $Pr[X_i = x] = 1/6$
  - Was ist die Summe der gewürfelten Zahlen?





Zentraler Grenzwertsatz

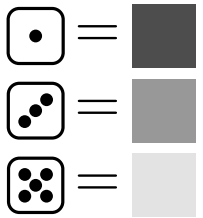
Die ursprüngliche Verteilung spielt keine Rolle!

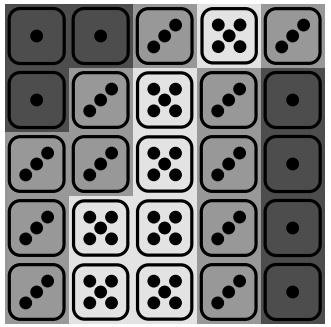
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

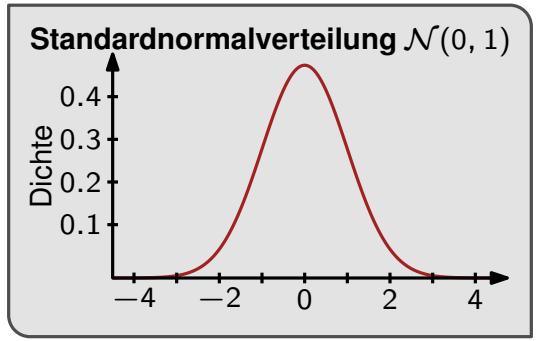
verschieben stauchen

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung





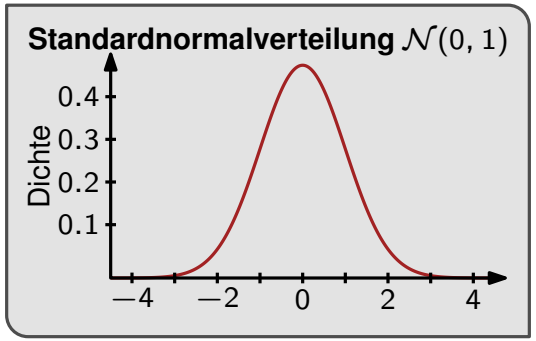


Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.

Fotos hashen

- Was hat das mit unseren Fotos zu tun?
- Wenn unsere Pixel Würfel sind, konvergiert die Verteilung der Summe der Pixelwerte gegen die Normalverteilung
- Unsere Pixel sind 256-seitige Würfel
  (wenn auch vermutlich nicht gleich verteilt)
- Unsere Hashunfktionen
  - die erste bildet Fotos auf ~~große~~ Zahlen ab
  - die zweite bildet nun die ~~großen~~ Zahlen auf Buckets ab
- Viele Fotos landen in den gleichen Buckets ... Laufzeitreduktion funktioniert nicht

57	57	111	217	111
57	111	217	111	57
111	111	217	111	57
111	217	217	111	57
111	217	217	111	57



viele — die gleichen — die gleichen

Hashfunktionen besser nicht selbst entwerfen!

Zentraler Grenzwertsatz
Sei $X_1, X_2, \ldots, X_n, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit Erwartungswert μ und Varianz σ^2 und sei $S_n = \sum_{i=1}^{n} X_i$ die n-te Teilsumme. Dann konvergiert die Zufallsvariable $Z_n = (S_n - n\mu)/(\sigma\sqrt{n})$ für $n \to \infty$ gegen die Verteilungsfunktion der Standardnormalverteilung $\mathcal{N}(0,1)$.