Algorithmen 1 **1. Klausur** **Sommersemester 2023**

31.08.2023

Bitte klebe hier den Aufkleber mit deiner Matrikelnummer auf.

- Bringe den Aufkleber mit deiner Matrikelnummer oben auf diesem Deckblatt an und beschrifte jedes Aufgabenblatt mit deiner Matrikelnummer.
- Diese Klausur umfasst 16 Seiten (diese Titelseite eingeschlossen) und 6 Aufgaben. Es können maximal 60 Punkte erreicht werden.
- Schreibe die Lösungen auf die Aufgabenblätter und Rückseiten. Am Ende der Klausur sind zusätzliche Leerseiten. Fordere zusätzliches Papier bitte nur an, falls du den gesamten Platz aufgebraucht hast.
- Es werden nur Lösungen gewertet, die mit dokumentenechten Stiften geschrieben sind.
- Als Hilfsmittel ist ein A4-Papier mit beliebigem Inhalt erlaubt.
- Die Tackernadel darf nicht gelöst werden.
- Die Bearbeitungszeit beträgt 2 Stunden.
- Schreibe nicht in die Tabelle auf dieser Seite.

Aufgabe	Mögliche Punkte	Erreichte Punkte
1	10	
2	10	
3	10	
4	10	
5	10	
6	10	
Gesamt	60	

1. **Kleinaufgaben** [10 Punkte]

Hinweis: $\log(n) = \log_2(n)$ und $\log^2(n) = \log(n) \cdot \log(n)$.

(a) Ordne die folgenden Funktionen so an, dass genau dann $f \in O(g)$ gilt, wenn f links von g steht.(3 Punkte)

$$n \cdot \sqrt{n}, \qquad 2^n, \qquad 3 \cdot \log^4(n), \qquad n^2 + \sqrt{n}, \qquad n \cdot \log(n), \qquad 4 \cdot n^3, \qquad n!$$

(b) Die Funktion FOO(n) ist durch den unten gegebenen Pseudocode definiert. Bestimme eine Funktion f in Abhängigkeit von n, sodass FOO(n) Laufzeit $\Theta(f(n))$ benötigt. Vereinfache f so weit wie möglich. Gehe davon aus, dass DOSOMETHING() $\Theta(1)$ Zeit benötigt. (1 Punkt)

```
1: function FOO(n : ℕ)
2:     for i from 1 to n do
3:         j := 1
4:         while j · j < n do
5:             DOSOMETHING()
6:             j := j + 1
```

Laufzeit:

(c) Gib für die folgenden Fragestellungen jeweils die Laufzeit eines möglichst effizienten Algorithmus, der das Problem löst, in O-Notation an. (3 Punkte)

0. Beispiel: Sortiere n vergleichbare Objekte. $\rightarrow O(n \log(n))$
1. Bestimme die zweitkleinste Zahl in einem unsortierten Array mit n Zahlen.

2. In einem Text der Länge n, bei dem an jeder Stelle einer von 26 möglichen Buchstaben steht, soll der Buchstabe bestimmt werden, der am häufigsten vorkommt.

3. Addiere zwei quadratische Matrizen mit n Zeilen.

(d) Bestimme für folgende Situationen, welche möglichst einfache und effiziente Datenstruktur jeweils geeignet ist und gib an, welche Operationen der von dir gewählten Datenstruktur dabei relevant sind. (3 Punkte)
Hinweis: In der Vorlesung haben wir folgende Datenstrukturen kennengelernt: Listen, (dynamische) Arrays, Hashtabellen, binäre Heaps, $(2,3)$-Bäume, Union-Find, Adjazenzliste.

1. Eine Magierin hat viele Beutel mit Murmeln. Für jede Murmel weiß sie, in welchem Beutel sie sich befindet. Ein Zuschauer darf immer wieder einen Beutel aussuchen und alle Murmeln aus diesem Beutel in einen anderen Beutel seiner Wahl umfüllen. Die Magierin soll danach für jede Murmel sagen können, in welchem Beutel sie sich befindet.

   Datenstruktur:

   Operationen:

2. Der Mensakoch bekommt jeden Tag Paprikas geliefert, die verschiedene Ablaufdaten haben. Beim Kochen möchte er immer eine Paprika verwenden, die als nächstes abläuft.

   Datenstruktur:

   Operationen:

3. Alan arbeitet in einer großen Bibliothek mit vielen Büchern. Manchmal werden alte Bücher aussortiert und neue Bücher eingekauft. Er fragt sich häufig, wie viele Bücher in der Bibliothek von einer gegebenen Person verfasst wurden.

   Datenstruktur:

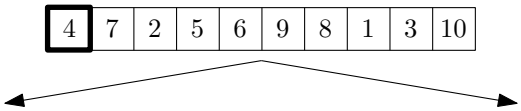
   Operationen:

2. **Sortieren** [10 Punkte]

(a) Das folgende Array soll mit dem Quicksort-Algorithmus aus der Vorlesung sortiert werden, wobei als Pivotelement immer das *erste* Element eines Arrays gewählt wird.

4	7	2	5	6	9	8	1	3	10

Vervollständige den zugehörigen Rekursionsbaum, wobei jeder Knoten des Rekursionsbaums einem rekursiven Aufruf entspricht. Gib für jeden Knoten das zu sortierende Teilarray und das gewählte Pivotelement an. (3 Punkte)



(b) Sortiere folgendes Array mit dem LSD-Radix-Sort-Algorithmus mit Basis 10 aus der Vorlesung. Gib dabei für jede Stelle i der Dezimaldarstellung den Zustand der Buckets an. (4 Punkte)

521	423	125	755	645	741	789	332	123	842

Hinweis: Falls mehrere Zahlen in das gleiche Bucket gehören, schreibe sie untereinander in das richtige Bucket.

	0	1	2	3	4	5	6	7	8	9
$i = 0$										
$i = 1$										
$i = 2$										

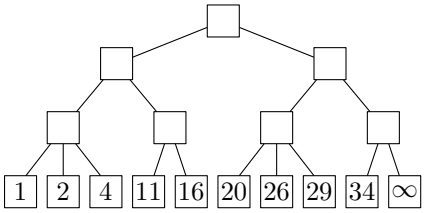
(c) Wir betrachten nun Mergesort und die Methode `merge`(A_1, A_2). Wir nennen einen Merge von zwei Arrays A_1 und A_2 *verzahnt*, wenn niemals zwei aufeinander folgende Zahlen aus dem gleichen Array genommen werden. Beispielsweise ist der Merge von $[1, 4, 8]$ mit $[2, 6, 9]$ verzahnt, der Merge von $[1, 4, 8]$ mit $[2, 3, 9]$ aber nicht. Gib ein Array A an mit den folgenden Eigenschaften:

- Es enthält die Zahlen $1, 2, 3, 4, 5, 6, 7, 8$ jeweils genau einmal.
- Bei der Ausführung von Mergesort auf dem Array A ist jeder Merge verzahnt.

(3 Punkte)

3. **(2, 3)-Bäume** [10 Punkte]

In dieser Aufgabe beschäftigen wir uns mit $(2, 3)$-Bäumen, bei denen der ∞-Trick angewendet wird. Gegeben sei folgender $(2, 3)$-Baum:



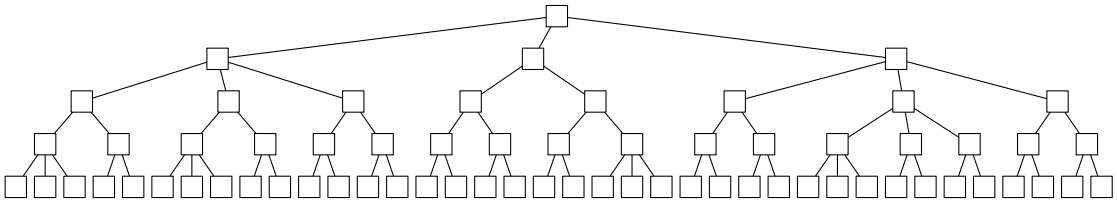
(a) Es sollen nun nacheinander die Operationen `insert(17)` und `remove(34)` ausgeführt werden. Gib die Subroutinen an, die jeweils durchgeführt werden müssen, um diese Operationen umzusetzen und anschließend die $(2, 3)$-Baum-Eigenschaft wiederherzustellen. Zeichne den Baum nach jeder Anwendung einer Subroutine.
Hinweis: Mögliche Subroutinen sind *Aufspalten*, *Verschmelzen*, *Blatt Löschen*, *Blatt Anhängen*, *Ausbalancieren*. Du brauchst keine Schlüssel in den inneren Knoten einzutragen. (4 Punkte)

`insert(17):`

Zeichne den Zustand nach `remove(34)` oben auf der nächsten Seite.

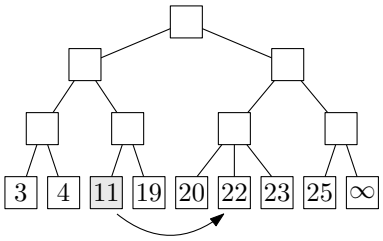
`remove(34)` (nach `insert(17)`):

(b) Wir betrachten nun einen großen $(2, 3)$-Baum. Markiere alle Blätter v im Baum, sodass ein Einfügen vor v alle Knoten auf dem Pfad zur Wurzel aufspaltet. Markiere außerdem die Knoten, die dabei aufgespaltet werden. Gib zusätzlich an, unter welchen Bedingungen ein Einfügen vor einem Blatt w eines beliebigen $(2, 3)$-Baums alle Knoten auf dem Pfad zur Wurzel aufspaltet. (3 Punkte)



Bedingungen:

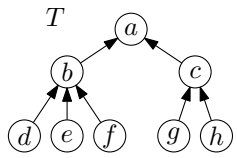
(c) Nun sollen $(2,3)$-Bäume etwas ausgedünnt werden. Wir sagen, dass ein Element x im Baum *verdoppelt vorkommt*, wenn es im Baum ein weiteres Element $y = 2 \cdot x$ gibt. Im folgenden Baum kommt Element 11 verdoppelt vor.



Beschreibe einen Algorithmus in Worten, der als Eingabe einen $(2,3)$-Baum erhält und unter allen verdoppelt vorkommenden Elementen das *kleinste* löscht. Nenne und begründe außerdem das asymptotische Laufzeitverhalten des Algorithmus. (3 Punkte)

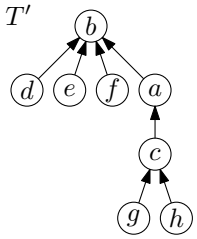
4. **Umwurzelung** [10 Punkte]

Wir sagen, ein Baum T ist an Knoten $r \in V(T)$ *gewurzelt*, wenn alle Kanten zu r hin zeigen (das heißt, es gibt von jedem Knoten einen gerichteten Pfad zur Wurzel). Für jeden Knoten $v \in V$ sei $|T_v|$ die *Teilbaumgröße von* v, also die Anzahl Knoten, von denen es einen Pfad zu v gibt (inklusive v). Im folgenden Beispiel ist für jeden Knoten v der Wert von $|T_v|$ angegeben.



Knoten	a	b	c	d	e	f	g	h
Teilbaumgröße	8	4	3	1	1	1	1	1

Eine *Umwurzelung* von der aktuellen Wurzel r zu einem Knoten v macht v zur neuen Wurzel und richtet die Kanten entsprechend. Wenn wir beispielsweise den oben gezeichneten Baum T von a zu b umwurzeln, erhalten wir folgenden gewurzelten Baum T'.



(a) Sei T'' der Baum, den man erhält, wenn man T stattdessen von a zu g umwurzelt. Gib die Teilbaumgrößen von T'' an. (1 Punkt)

Knoten	a	b	c	d	e	f	g	h
Teilbaumgröße								

(b) Sei T ein Baum mit Wurzel r, sei c ein Kind der Wurzel und sei T' der Baum, den wir erhalten, wenn wir T von r zu c umwurzeln. Angenommen, du kennst für jeden Knoten v die Teilbaumgröße $|T_v|$ in T. Gib für jeden Knoten v in T' die Teilbaumgröße $|T'_v|$ in Abhängigkeit von den Teilbaumgrößen in T an.(3 Punkte)

(c) Zeige die folgende Aussage: Für jedes $n > 2$ gibt es einen Baum mit n Knoten sowie zwei Knoten r_1 und r_2, sodass die Umwurzelung von r_1 zu r_2 die Teilbaumgröße eines Knotens um $\Theta(n)$ ändert. (2 Punkte)

Wir betrachten nun eine Datenstruktur, die einen Baum T mit Wurzel r verwaltet, dessen Knoten von 0 bis $n-1$ durchnummeriert sind. Diese Datenstruktur besteht aus

- einem Array `parent`, das den Baum repräsentiert, indem es an Stelle v den Elter von v speichert. Für die Wurzel r gilt: `parent[`r`]` $= r$.
- einem Array `tbg`, das an Stelle v die Teilbaumgröße von v speichert.

Wir gehen davon aus, dass diese Arrays zu Beginn korrekt initialisiert werden. Die Datenstruktur verfügt außerdem über eine Operation `swapRootToChild`(c), die den Baum in Zeit $\Theta(1)$ von der aktuellen Wurzel zum Kind c der Wurzel umwurzelt, indem sie das `parent`-Array und das `tbg`-Array entsprechend anpasst.

(d) Es soll nun eine neue Operation `swapRootToNode`(v) implementiert werden, die einen *beliebigen* Knoten v aus T entgegen nimmt und T von r zu v umwurzelt. Gib einen Algorithmus in Pseudocode an, der `swapRootToNode` umsetzt. Verwende folgende Signatur: (4 Punkte)

Hinweis: Fällt dir ein Algorithmus ein, dessen asymptotische Laufzeit nicht größer ist als die Distanz von v zu r?

`swapRootToNode`$(v\colon \mathbb{N})$:

---

5. **Fête de la musique** [10 Punkte]

In einer Straße, in der nur Bands wohnen, wird ein Fest stattfinden. Aus den unterschiedlich beliebten Bands soll dazu eine gute Auswahl getroffen werden, um sie vor ihrer jeweiligen Haustür spielen zu lassen. Damit sich die Bands nicht gegenseitig übertönen, muss dabei aber darauf geachtet werden, dass zwei ausgewählte Bands nicht zu nah beieinander wohnen.

Unsere Eingabe besteht also aus den Bands $B = \{1, \ldots, n\}$, wobei Band i die Beliebtheit $p_i \in \mathbb{N}$ hat. Zudem ist ein Mindestabstand $d \in \mathbb{N}$ gegeben. Die *Beliebtheit* einer Teilmenge $S \subseteq B$ von Bands definieren wir als die Summe der Beliebtheiten der einzelnen Bands $p(S) = \sum_{i \in S} p_i$. Wir nennen S *gültig*, falls für alle Paare ausgewählter Bands $i, j \in S$ mit $i \neq j$ der Mindestabstand eingehalten wird, also wenn $|i - j| \geq d$.

Beim Problem d-BeliebtesMusikFest geht es darum, eine gültige Teilmenge mit möglichst großer Beliebtheit zu finden, wobei d der Mindestabstand ist. Eine solche Teilmenge nennen wir *optimal*.

Als Beispiel betrachten wir die folgende Instanz von 2-BeliebtesMusikFest zusammen mit einer gültigen, aber *nicht-optimalen* Teilmenge $S_1 = \{1, 3, 5, 7, 9\}$ mit Beliebtheit $p(S_1) = 2 + 6 + 1 + 8 + 2 = 19$.

Band i	1	2	3	4	5	6	7	8	9	
Beliebtheit p_i	2	1	6	8	1	1	8	1	2	
Enthalten in S_1	⊗	○	⊗	○	⊗	○	⊗	○	⊗	⟶ $p(S_1) = 19$

(a) Zeige, dass die Teilmenge S_1 im obigen Beispiel nicht optimal ist, indem du eine optimale Teilmenge S_2 angibst (bspw. durch Markieren in der unteren Tabelle).
Gib außerdem die Beliebtheit von S_2 an. (2 Punkte)

Band i	1	2	3	4	5	6	7	8	9	
Beliebtheit p_i	2	1	6	8	1	1	8	1	2	
Enthalten in S_2	○	○	○	○	○	○	○	○	○	⟶ $p(S_2) =$

Für beliebiges $d > 0$ wollen wir nun ein dynamisches Programm für d-BeliebtesMusikFest entwickeln, wobei wir zuerst nur die Beliebtheit einer optimalen Teilmenge ermitteln wollen. Das DP soll für jeden Index $i \in B = \{1, \ldots, n\}$ eine Teillösung $X[i]$ in einem Array X verwalten.

(b) Gib an, welche Bedeutung eine Teillösung $X[i]$ hat und stelle darauf aufbauend die Rekurrenz auf, mit deren Hilfe das Array X korrekt ausgefüllt werden kann. (5 Punkte)

Hinweis: Achte darauf, auch den Basisfall der Rekurrenz anzugeben. Für volle Punktzahl sollte die Berechnung des DPs in $O(n)$ Zeit möglich sein.

(c) Wir wollen nun nicht nur die Beliebtheit einer optimalen Teilmenge ermitteln, sondern auch die Menge selbst. Gib an,

- welche Informationen dafür während der Berechnung des dynamischen Programms gespeichert werden,
- wie die Lösung dann rekonstruiert werden kann und
- was die asymptotische Laufzeit dieser Rekonstruktion ist. (3 Punkte)

6. **Straßenausbesserung** [10 Punkte]

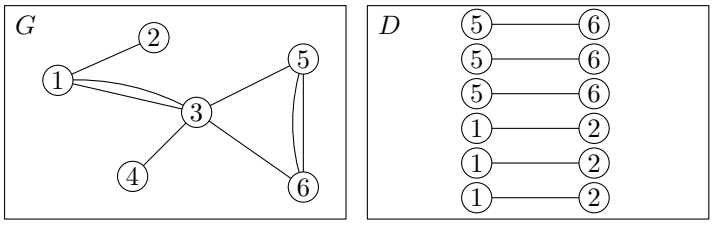
In einem Straßennetzwerk fallen häufig Reparaturen an, um Schlaglöcher auszubessern. Dazu wird auf einer Straße einfach eine komplett neue Schicht Asphalt aufgetragen. Sobald eine Straße jedoch aus mehr als k Schichten besteht (wobei $k \in \mathbb{N}$ mit $k > 0$), ist sie zu schwer. Um Platz für eine neue Schicht zu machen, werden die alten Schichten abgetragen und zur Schicht-Deponie gebracht. Nun sollen diese Ausbesserungsarbeiten mit einer Datenstruktur verwaltet werden.

Die Datenstruktur basiert auf einem *Multi*-Graphen G, der das Straßennetzwerk abbildet. Das heißt, in G kann es zwischen zwei Knoten *mehrere* Kanten geben. Jede Kante repräsentiert dabei eine Schicht. Zusätzlich wird die Deponie mithilfe eines Stacks D abgebildet. Die Datenstruktur unterstützt die folgenden Operationen:

- `dispose`(u, v): Falls es zwischen u und v in G mindestens eine Kante gibt, wird eine beliebige Kante zwischen u und v aus G entfernt und auf den Stack D gelegt, wofür $\Theta(1)$ Zeit benötigt wird.
- `coat`(u, v): Falls u und v in G bereits mit k Kanten verbunden sind, wird k-mal `dispose`(u, v) verwendet, um diese in Zeit $\Theta(k)$ auf den Stack D zu verschieben. In jedem Fall wird in Zeit $\Theta(1)$ zwischen u und v eine neue Kante in G eingefügt.
- `clear`(): Entfernt alle m_D Kanten, die aktuell auf dem Stack D liegen. Dafür wird $\Theta(m_D)$ Zeit benötigt.

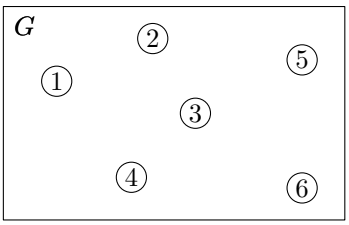
Im Folgenden gehen wir immer davon aus, dass G und D initial *keine* Kanten enthalten. Die Abbildung zeigt für $V = \{1, 2, 3, 4, 5\}$ und $k = 3$ den Zustand der Datenstruktur, den man erhält, wenn man die Operationen `coat`$(1, 2)$ 4-mal, `coat`$(1, 3)$ 2-mal, `coat`$(5, 6)$ 5-mal und `coat`$(3, 4)$, `coat`$(3, 5)$ und `coat`$(3, 6)$ jeweils 1-mal ausführt.

Zustand 1:



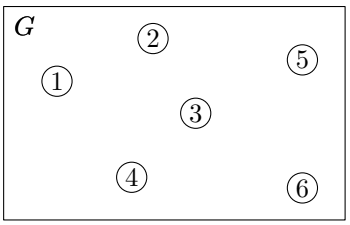
(a) Gib den Zustand der Datenstruktur an, den man erhält, wenn man ausgehend von *Zustand 1* die Operationen `coat`$(1, 3)$, `coat`$(1, 3)$, `dispose`$(1, 2)$, `coat`$(2, 5)$ und `coat`$(5, 6)$ ausführt. Zeichne dazu den Graphen G (oder gib die Kantenmenge an) und gib die Anzahl m_D von Kanten an, die in der Deponie D sind.(2 Punkte)

Zustand 2:



$m_D =$ ☐

Zustand 2:



(Kopie desselben Graphen, falls du dich beim ersten vertan hast. Markiere deutlich, welche Lösung zu bewerten ist.)

(b) Im Folgenden gehen wir von einem Zustand aus, in dem die Datenstruktur keine Kanten enthält. Gib eine Knotenmenge V an und für jedes $n \in \mathbb{N}$ eine Sequenz von Operationen, sodass

- die Sequenz aus $\Theta(n)$ Operationen besteht,
- es eine `clear`-Operation gibt, die Laufzeit $\Theta(n)$ hat und
- die Sequenz $\Theta(n)$ viele `clear`-Operationen enthält.

Deine Konstruktion soll für jede Konstante k funktionieren. Begründe deine Antwort. (3 Punkte)

Hinweis: Achte darauf, dass n *nicht* die Anzahl der Knoten in V beschreibt.

(c) Zeige, dass in jeder beliebigen Abfolge von `coat`-, `dispose`- und `clear`-Operationen jede Operation amortisiert konstante Kosten hat. (5 Punkte)