TU Wien
Institut für Logic and Computation
Algorithms and Complexity Group

186.866 Algorithmen und Datenstrukturen VU 8.0

1. Test, 2024S

17. Mai 2024

Gruppe A

Nachfolgende Angaben gut leserlich in Blockschrift machen.

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Sie dürfen die Lösungen nur auf diesen Prüfungsbogen schreiben. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie dokumentenechte Stifte (keine Bleistifte, etc.).

Die Verwendung von Taschenrechnern, Mobiltelefonen, Smartwatches, Tablets, Digitalkameras, Skripten, Büchern, Mitschriften, Ausarbeitungen oder vergleichbaren Hilfsmitteln ist unzulässig.

Kennzeichnen Sie bei Ankreuzfragen eindeutig, welche Kästchen Sie kreuzen. Streichen Sie Passagen, die nicht gewertet werden sollen, deutlich durch. Schreiben Sie leserlich, unleserliche Antworten werden nicht gewertet.

	A1	A2	A3	A4	A5	Summe
Erreichbare Punkte:	20	20	20	20	20	100
Erreichte Punkte:						

Viel Erfolg!

Aufgabe A1: Algorithmenanalyse **(20 Punkte)**

a) (6 Punkte) Gegeben ist der nachfolgende Algorithmus. Dieser erhält folgende zwei Eingabeparameter:

- Das Array A welches mit **positiven ganzen Zahlen ohne** 0 befüllt ist.
- Die Länge n des Arrays A.

BC: wenn $A[i] > n$

WC: $A[i] = 1$ → $\underbrace{1, 2, 4, 8, 16 \ldots \geq n}_{\lfloor \log_2(n) \rfloor}$

Function $f(A, n)$**:**
  for $i = 0, \ldots, n-1$
    while $A[i] < n$
      $A[i] \leftarrow A[i] + A[i]$

Geben Sie die Best-Case und die Worst-Case Laufzeiten des Algorithmus in Abhängigkeit von n in Θ-Notation an. Verwenden Sie möglichst einfache Terme.

Best-Case:	$\Theta(n)$
Worst-Case:	$\Theta(n \log n)$

b) (10 Punkte) Bestimmen Sie die Laufzeiten und Rückgabewerte der beiden unten angegebenen Algorithmen in Abhängigkeit von $n \in \mathbb{N}$ in Θ-Notation. Verwenden Sie möglichst einfache Terme.

$b = \sum_{i=1}^{n^2} \sum_{j=1}^{i} \frac{12 \log(i)}{i} = \sum_{i=1}^{n^2} 12 \log(i)$

$= \log((n^2)!)$

$= \Theta(n^2 \log(n))$

(i)

$i \leftarrow n^4 \cdot 3^{(n^2)}$
$k \leftarrow 0$
while $i > 0$
  $k \leftarrow k + 3i$
  $i \leftarrow \lfloor \frac{i}{7} \rfloor$
return k

(ii)

$a \leftarrow 0$
$b \leftarrow 0$
for $i = 1, \ldots, n^2$
  for $j = 1, \ldots, i$
    if $i \mod 2 = 0$ **then**
      $a \leftarrow a + b$
    $b \leftarrow b + 12 \frac{\log(i)}{i}$
return $\min(a, b)$

$a = \Theta(n^2 \log n \cdot \frac{n^2}{2})$

$= \Theta(n^4 \log n)$

	(i)	(ii)
Laufzeit:	$\Theta(n^2)$	$\Theta(n^4)$
Rückgabewert:	$\Theta(n^4 \cdot 3^{n^2})$	$\Theta(n^2 \log n)$

$i = n^4 \cdot 3^{(n^2)}$ jede Iteration /7

$\Rightarrow \log_7(n^4 \cdot 3^{n^2}) = \log_7(n^4) + \log_7(3^{n^2}) = 4 \cdot \log_7(n) + n^2 \log_7(3)$

$k + 3i = k + 3 \cdot \left(\frac{n^4 3^{n^2}}{7} \right) \approx 3 \cdot n^4 \cdot 3^{n^2} \cdot \left(\frac{1}{1 - \frac{1}{7}} \right) = 3 \cdot n^4 \cdot 3^{n^2} \cdot \frac{7}{6}$

$= \Theta(n^4 \cdot 3^{n^2})$

c) (4 Punkte) Seien f und g Funktionen. Beweisen Sie die folgende Aussage:

$$\text{Wenn } f(n) = O(g(n)), \text{ dann } f(\sqrt{n}) = O(g(\sqrt{n})).$$

Der Anfang des Beweises ist bereits gegeben.

Beweis. Wir nehmen an, es gilt $f(n) = O(g(n))$. Daher gibt es $c > 0$ und $n_0 > 0$, sodass ...

$f(n) = O(g(n)) \Rightarrow f(n) \leq c \cdot g(n) \quad \forall\, n \geq n_0$

z.z. $f(\sqrt{n}) = O(g(\sqrt{n})) \Rightarrow f(\sqrt{n}) \leq \tilde{c} \cdot g(\sqrt{n}) \quad \forall\, n \geq \tilde{n_0}$

man setze $\tilde{n_0} = n_0^2 \Rightarrow n \geq n_0^2 \Rightarrow \sqrt{n} \geq n_0$

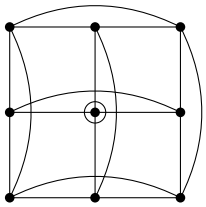
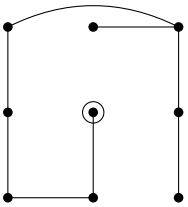
in diesem Fall sogar $c = \tilde{c} \Rightarrow f(\sqrt{n}) \leq c \cdot g(\sqrt{n}) \quad \forall \sqrt{n} \geq n_0$

somit gilt: $f(n) = O(g(n))$ dann auch $f(\sqrt{n}) = O(g(\sqrt{n}))$

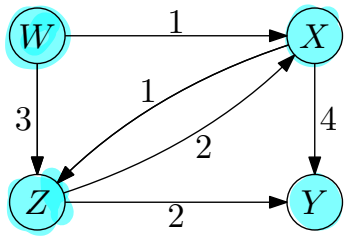
Aufgabe A2: Graphen **(20 Punkte)**

a) (4 Punkte) Betrachten Sie den linken Graphen und kreuzen Sie für jeden der beiden rechten Bäume an, ob es ein Breitensuchbaum (BFS) oder ein Tiefensuchbaum (DFS) des linken Graphen ist. Hierbei nehmen Sie an, dass der Startknoten der hervorgehobene Knoten in der Mitte ist.

(+1 Punkt für jede richtige, -1 Punkt für jede falsche und 0 Punkte für keine Antwort, keine negativen Gesamtpunkte auf diese Unteraufgabe)

		
DFS	☒ Ja ☐ Nein	☐ Ja ☒ Nein
BFS	☐ Ja ☒ Nein	☐ Ja ☒ Nein

b) (6 Punkte) Führen Sie den Algorithmus von Dijkstra auf dem nachfolgenden Graphen mit W als Startknoten aus. Ergänzen Sie dabei für jeden Durchlauf der äußeren Schleife das Distanzarray `d` und den Knoten, der neu als `Discovered` markiert wurde in der unten angeführten Tabelle.

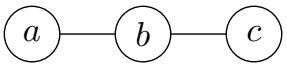


`Discovered`	`d`(W)	`d`(X)	`d`(Y)	`d`(Z)
W	0	1	∞	3
X	0	1	4	2
Z	0	1	4	2
Y	0	1	4	2

c) (4 Punkte) Betrachten Sie folgende Operationen auf einem Min-Heap und auf einer einfach verketteten Liste. Kreuzen Sie in jeder Zeile an, auf welcher Datenstruktur die Operation die asymptotisch kleinere Worst-Case Lauftzeit hat.

	Liste schneller	Heap schneller	gleich schnell	
Einfügen	☐	☒	☐	$\log n$
Maximum löschen	☐	☐	☒	n
Suchen	☐	☐	☒	n
Minimum löschen	☐	☒	☐	$\log n$

d) (6 Punkte) Sei G ein ungerichteter **zusammenhängender** Graph. Wir sagen ein Knoten v ist ein *Engpass* von G, wenn es zwei verschiedene Knoten u, w mit $u \neq v \neq w$ gibt, sodass jeder Pfad von u nach w durch v geht. Zum Beispiel ist im folgenden Graph b ein Engpass, aber a nicht.



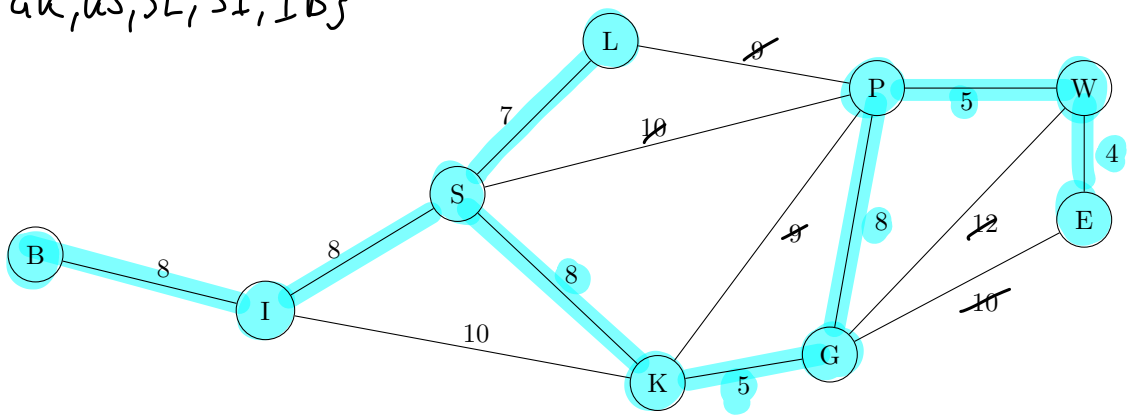
Vervollständigen Sie folgenden Pseudocode eines Algorithmus, sodass dieser alle Engpässe von G berechnet. Nehmen Sie dafür an, dass die Methode `Komponenten`(H) eine Liste aller Zusammenhangskomponenten eines Graphen H liefert.

```
Engpässe(G = (V, E)):
S ← ∅
for each v ∈ V
    H ← [ G \ v ]
    L ← Komponenten(H)
    if [ |L| > 1 ]
        S ← S ∪ {v}
return S
```

Aufgabe A3: Greedy **(20 Punkte)**

a) (6 Punkte) Berechnen und markieren Sie einen minimalen Spannbaum des untenstehenden Graphen. Geben Sie eine mögliche Reihenfolge an, in der der Algorithmus von **Prim** die Kanten des Graphen für einen Spannbaum auswählt. Nehmen Sie an, dass der Algorithmus **mit dem Knoten W startet**. Geben Sie die Reihenfolge als Liste von Knotenpaaren an (z.B.: WG, ...). Bei mehreren gleichwertigen Möglichkeiten, nehmen Sie die lexikographisch alphabetisch kleinste Kante, wobei der schon im Spannbaum enthaltene Knoten als erster betrachtet wird. Wenn also z.B. die Knoten V und Z schon im Spannbaum sind und zwischen den Kanten VY und ZX entschieden wird, nehmen Sie VY).

{WE, WP, PG, GK, KS, SL, SI, IB}



b) (8 Punkte) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

(+2 Punkte für jede richtige, -2 Punkte für jede falsche und 0 Punkte für keine Antwort, keine negativen Gesamtpunkte auf diese Unteraufgabe)

(Q1) Seien $G_1 = (V, T_1)$ und $G_2 = (V, T_2)$ zwei MSTs eines gewichteten Graphen $G = (V, E)$. Dann ist $T_1 \cap T_2 \neq \emptyset$.
☒ Wahr ☐ Falsch

(Q2) Sei G ein gewichteter schlichter Graph, in dem alle Kantengewichte unterschiedlich sind, außer zwei, welche gleich sind. Dann enthält jeder MST beide jene Kanten, die gleiche Gewichte haben.
☐ Wahr ☒ Falsch

(Q3) Sei G ein gewichteter Graph, in dem Kantengewichte möglicherweise gleich sind. Der Algorithmus von Prim liefert nicht immer den gleichen MST als Kruskal, aber beide liefern immer einen korrekten MST.
☒ Wahr ☐ Falsch

(Q4) Sei G ein gewichteter Graph, in dem die Kante e das größte Gewicht hat und alle anderen Kanten strikt kleinere Gewichten haben. Dann ist e in keinem MST enthalten.
☐ Wahr ☒ Falsch

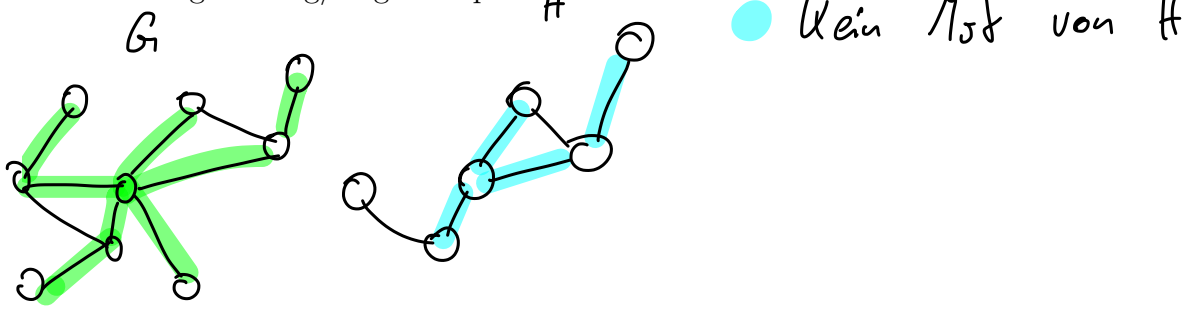
c) (6 Punkte) Ein Graph $H = (V_H, E_H)$ ist ein induzierter Teilgraph eines Graphen $G = (V_G, E_G)$, wenn $V_H \subseteq V_G$ und $E_H = \{e \in E_G \mid e \subseteq V_H\}$. D.h., H ist eindeutig durch eine Teilmenge der Knoten von G spezifiziert.

Sei $H = (V_H, E_H)$ ein zusammenhängender induzierter Teilgraph eines zusammenhängenden gewichteten Graphen $G = (V, E)$ und sei $G_T = (V, T)$ ein MST von G.

Ist $H_T = (V_H, T \cap E_H)$ ein MST von H? Begründen Sie Ihre Antwort mit einem Beweis/Gegenbeispiel.

☐ Ja ☒ Nein

Begründung/Gegenbeispiel:



Aufgabe A4: Divide-and-Conquer **(20 Punkte)**

In der folgenden Aufgabe ist $A = [a_1, \ldots, a_n]$ stets ein **aufsteigend sortiertes** Array von n ganzen Zahlen und x eine ganze Zahl. Nehmen Sie an, wir haben einen **Algorithmus** $\texttt{T}(A, x)$ zur Verfügung, der den kleinstmöglichen Index $i \in \{1, \ldots, n\}$ zurückgibt, für den $a_i = x$ gilt bzw. -1 wenn $x \notin A$.

a) Man bestimme, ob das Element x in A **mehr als** $(n/2)$-mal vorkommt. Nehmen Sie an, dass n **ungerade** ist. Nutzen Sie den Algorithmus $\texttt{T}(A, x)$ für Ihre Lösung.

(i) (8 Punkte) Vervollständigen Sie den folgenden Pseudocode so, dass er das obige Problem löst. Verwenden Sie **keine rekursiven Aufrufe**. Es soll `true` zurückgegeben werden, falls A und x die Eigenschaft des Problems erfüllen, andernfalls `false`.

1 2 3 4 5

x, x, x, 1, 2

1, 2, x, x, x
1 1

procedure `MajorityAppearance`(A, x)
`// A ist ein aufsteigend sortiertes Array ungerader Länge`
$n \leftarrow$ `length`(A)

$i \leftarrow$ T(A, x)

if $i >$ $\lceil \frac{n}{2} \rceil$ **then**
return `false`

if i = −1 **then**
return `false`

if A[n] ≠ x and A[1] ≠ x **then**
return false
else
return true

(ii) (2 Punkte) Kreuzen Sie an, ob die folgende Aussage wahr oder falsch ist.

Ihr Algorithmus `MajorityAppearance` hat eine Worst-Case Laufzeit, die asymptotisch mit jener von `T` übereinstimmt.

☒ Wahr ☐ Falsch

b) (4 Punkte) Wir geben nun den Algorithmus `T` explizit an (der also den kleinstmöglichen Index $i \in \{1, \ldots, n\}$ zurückgibt, für den $a_i = x$ gilt; oder -1 zurückgibt, wenn $x \notin A$):

```
procedure T(A, x)
    // A ist ein Array aus aufsteigend sortierten Zahlen
    n ← length(A)  // (diese Zuweisung erfolgt in konstanter Laufzeit)
    low ← 1
    high ← n
    result ← −1

    while low ≤ high
        mid ← ⌈(low + high)/2⌉
        if A[mid] == x then
            result ← mid
            high ← mid − 1
        else if x < A[mid] then
            high ← mid − 1
        else
            low ← mid + 1
    return result
```

Vervollständigen Sie die folgenden Beobachtungen durch Angabe von **möglichst engen** oberen bzw. unteren Schranken:

In O-Notation liegt die Worst-Case Laufzeit von `T` in $O(\log(n))$.

In Ω-Notation liegt die Worst-Case Laufzeit von `T` in $\Omega(\log(n))$.

c) (6 Punkte) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

(+2 Punkte für jede richtige, -2 Punkte für jede falsche und 0 Punkte für keine Antwort, keine negativen Gesamtpunkte auf diese Unteraufgabe)

(Q1) Gegeben sei ein allgemeines (vergleichsbasiertes) Sortierverfahren. Müssen mit diesem n verschiedene Schlüssel sortiert werden, dann kann der Worst-Case Aufwand dafür bestenfalls in $\Omega(n \log(n))$ liegen.
☒ Wahr ☐ Falsch

(Q2) Für den Algorithmus *Insertionsort* ist $\Theta(n^2)$ seine Worst-Case Laufzeit und $\Theta(n \log(n))$ seine Average-Case Laufzeit.
☐ Wahr ☒ Falsch

(Q3) Sei $T(2^k)$, $k \in \mathbb{N}$, die Worst-Case Laufzeit von *Mergesort* bei Ausführung auf einer Liste der Länge 2^k. Dann gilt folgender Zusammenhang:

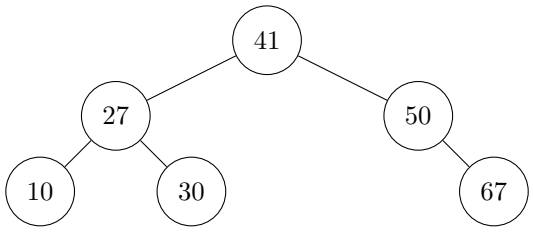
$$\sqrt{T(2^{k+1})} \leq T(2^k) + T(2^k) + O(2^{k+1}).$$

$T(n) = 2T(n/2) + O(n)$

☒ Wahr ☐ Falsch

Aufgabe A5: Suchbäume und Hashing **(20 Punkte)**

a) (4 Punkte) Geben Sie die *Preorder*- und *Postorder*-Durchmusterungsreihenfolge des folgenden binären Suchbaumes an.



root-left-right

Preorder: 41, 27, 10, 30, 50, 67

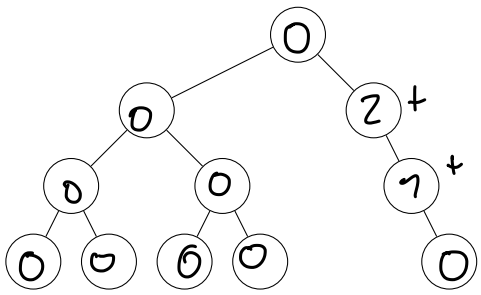
left-right-root

Postorder: 10, 30, 27, 67, 50, 41

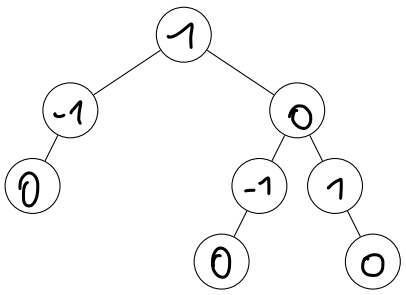
b) (4 Punkte) Nehmen Sie an, die folgenden Bäume sind durch Einfügen eines Elementes in einen AVL-Baum entstanden. Kreuzen Sie bei jedem Baum an, ob keine, eine einfache, oder eine doppelte Rotation notwendig ist, um die AVL-Eigenschaft wiederherzustellen. Die Richtung der Rotation muss nicht angegeben werden.

(+1 Punkt für jede richtige, -1 Punkt für jede falsche und 0 Punkte für keine Antwort, keine negativen Gesamtpunkte auf diese Unteraufgabe)

++ = left

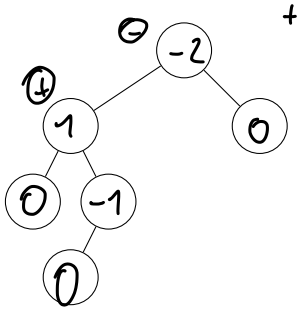


(i) ☐ keine ☒ einfach ☐ doppelt



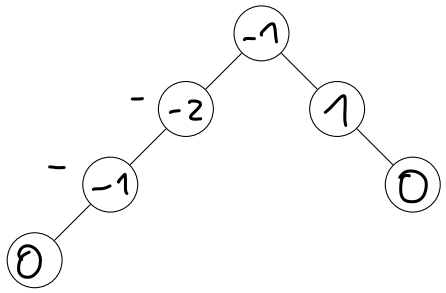
(ii) ☒ keine ☐ einfach ☐ doppelt

+- = left right



(iii) ☐ keine ☐ einfach ☒ doppelt

-- = right



(iv) ☐ keine ☒ einfach ☐ doppelt

c) (6 Punkte) Bestimmen Sie die Worst-Case Laufzeit der Operationen Suchen und Einfügen in natürlichen binären Suchbäumen, AVL-Bäumen und B-Bäumen abhängig von der Knotenzahl n in Θ-Notation. Füllen Sie dazu die folgende Tabelle aus.

	Suchen	Einfügen
natürliche Suchbäume	$\Theta(n)$	$\Theta(n)$
AVL-Bäume	$\Theta(\log n)$	$\Theta(\log n)$
B-Bäume	$\Theta(\log n)$	$\Theta(\log n)$

d) (6 Punkte) Gegeben ist eine Hashtabelle sowie die Hashfunktion $h_1(k) = k \bmod 11$.

Position	0	1	2	3	4	5	6	7	8	9	10
Schlüssel	22		101	14	15		61	40			87

Fügen Sie den Schlüssel 3 als neues Element mittels der jeweiligen Sondierungsvariante in die obige Hashtabelle ein. Geben Sie die Position an, an der der Schlüssel eingefügt wird. Geben Sie außerdem alle Positionen an, bei denen es zu einer Kollision kommt.

(i) Double Hashing mit $h(k,i) = (h_1(k) + ih_2(k)) \bmod 11$, wobei h_2 definiert ist als $h_2(k) = 1 + (k \bmod 5)$.

$h_1 = 3$
$h_2 = 4$

Kollisionen: 3, 7, 0, 4, (0 1 2 3 4) Eingefügt an Position: 8

(ii) Lineares Sondieren mit $h(k,i) = (h_1(k) + i) \bmod 11$.

Kollisionen: 3, 4 (0 1) Eingefügt an Position: 5

(iii) Quadratisches Sondieren mit $h(k,i) = (h_1(k) + \frac{1}{2}i + \frac{1}{2}i^2) \bmod 11$.

Kollisionen: 3, 4, 6, . (0 1 2 3) Eingefügt an Position: 9