TU Wien
Institut für Logic and Computation
Algorithms and Complexity Group

186.866 Algorithmen und Datenstrukturen VU 8.0

1. Test, 2022S

20. Mai 2022

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) (4 Punkte) Ordnen Sie folgende Funktionen nach Dominanz ($\ll$), beginnend mit der asymptotisch am schwächsten wachsenden. Es genügt die Funktionen zu reihen, ein Beweis der Gültigkeit der Relationen ist nicht erforderlich.

$$\frac{4n^5 + 5n^4}{7n^3}, \quad 3^{-n}, \quad \log(n), \quad 2 \cdot \sqrt{5n^6}, \quad (3n)!, \quad 1.1^n, \quad \frac{n}{50}$$

b) (4 Punkte) Bestimmen Sie die Laufzeit des angegebenen Algorithmus in Abhängigkeit des Eingabeparameters $n \in \mathbb{N}$ in Θ-Notation. Verwenden Sie hierfür möglichst einfache Terme.

$k \leftarrow 0$
for $i = 1, \ldots, n$
    for $j = 1, \ldots, \lceil \log(i) \rceil$
        $k \leftarrow k + 1$
    for $j = i + n, \ldots, 3n$
        $k \leftarrow k - 1$
return k

Laufzeit (in Θ-Notation):

c) (4 Punkte) Bestimmen Sie die Best-Case und Worst-Case Laufzeit des angegebenen Algorithmus in Abhängigkeit des Eingabeparameters $n \in \mathbb{N}$ in Θ-Notation. Das Integer-Array A der Länge n ist Teil der Eingabe. Verwenden Sie hierfür möglichst einfache Terme.

if $n < 1000$ **then**
    $a \leftarrow 1$
    for $j = 1, \ldots, 2^n$
        $a \leftarrow 2a$
    return a
else
    for $i = 1, \ldots, n$
        if $A[i] \bmod 2 = 0$ **then**
            return i
    return 0

Best-Case Laufzeit (in Θ-Notation):

Worst-Case Laufzeit (in Θ-Notation):

d) (6 Punkte) Bestimmen Sie die Laufzeit und den Rückgabewert des angegebenen Algorithmus in Abhängigkeit vom Eingabeparameter $n \in \mathbb{N}$ in Θ-Notation. Verwenden Sie hierfür möglichst einfache Terme.

$a \leftarrow n$
$b \leftarrow 0$
for $i = 1, \ldots, n$
  for $j = 1, \ldots, n$
    $a \leftarrow a/2$
    if $a < 1$ **then**
      return b
    $b \leftarrow b + 1$
return 0

Laufzeit (in Θ-Notation):	
Rückgabewert (in Θ-Notation):	

e) (2 Punkte) Sei $f : \mathbb{N}^+ \to \mathbb{N}^+$ eine Funktion. Nehmen Sie an, es gilt:

$$\lim_{n\to\infty} \frac{f(n)}{n^2} = 0$$

Was folgt daraus? Kreuzen Sie alle zutreffenden Aussagen an:

(MC1) ☐ $f(n)$ ist in $O(n^2)$

(MC2) ☐ $f(n)$ ist in $\Theta(n^2)$

(MC3) ☐ $f(n) \ll n^2$

(MC4) ☐ keine der zuvor genannten Aussagen

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

a) (5 Punkte) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

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

(Q1) Sei G ein gerichteter Graph für den genau eine topologische Sortierung existiert, dann ist G schwach zusammenhängend.

☐ Wahr ☐ Falsch

(Q2) Sei G ein gerichteter Graph für den genau eine topologische Sortierung existiert, dann ist G stark zusammenhängend.

☐ Wahr ☐ Falsch

(Q3) Jeder gerichtete Graph mit n Knoten, der einen gerichteten Kreis der Länge n enthält, ist stark zusammenhängend.

☐ Wahr ☐ Falsch

(Q4) Jeder azyklische gerichtete Graph mit n Knoten besitzt genau n starke Zusammenhangskomponenten.

☐ Wahr ☐ Falsch

(Q5) Jeder gerichtete Graph mit k schwachen Zusammenhangskomponenten besitzt höchstens k starke Zusammenhangskomponenten.

☐ Wahr ☐ Falsch

b) (6 Punkte) Gegeben ist folgendes Array, das zur Repräsentation eines Heaps verwendet wird (Kodierung wie aus der Vorlesung bekannt).

0	5	4	2	9	3	1	8

Führen Sie den Algorithmus zur Erstellung eines Min-Heaps aus der Vorlesung aus. Geben Sie den initialen Baum und den Baum nach jeder Iteration an (in Baumdarstellung, nicht als Array).

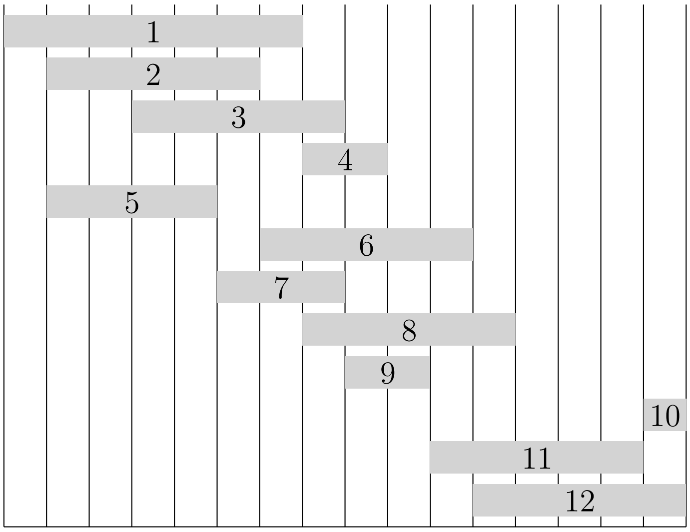
c) (9 Punkte) Ein ungerichteter Graph $G = (V, E)$ wird **bipartit** genannt genau dann wenn die Knoten in zwei disjunkte Mengen A, B partitioniert werden können, sodass für jede Kante $(u, v) \in E$ gilt: $u \in A, v \in B$ oder $v \in A, u \in B$. Es existiert also keine Kante, die beide Endknoten in der gleichen Menge hat.

Geben Sie an, wie die Breitensuche verwendet werden kann, um für einen gegebenen Graphen mit n Knoten und m Kanten in Zeit $O(n + m)$ zu testen, ob er bipartit ist oder nicht.

Eine präzise Beschreibung reicht als Antwort. Ein detaillierter Pseudocode oder Korrektheitsbeweis ist nicht erforderlich.

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

a) (4 Punkte) Geben Sie eine optimale Lösung für folgende Interval Scheduling Instanz an:



b) (3 Punkte) Vervollständigen Sie durch Ankreuzen den folgenden Text, sodass er einen Algorithmus beschreibt, der das Interval Scheduling Problem in Polynomialzeit löst:

Sortiere die Intervalle aufsteigend nach

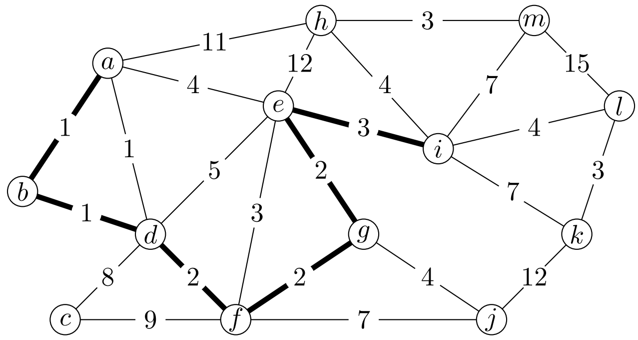
☐ Länge ☐ Startpunkt ☐ Anzahl ihrer Konflikte.

Solange noch Intervalle da sind, füge

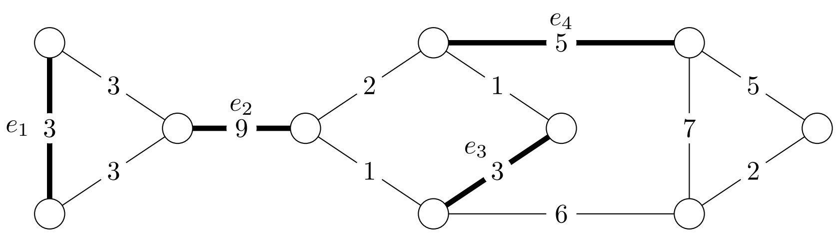
☐ das erste Intervall ☐ das letzte Intervall ☐ ein beliebiges Intervall

zur Lösung hinzu und lösche alle Intervalle die mit diesem Intervall in Konflikt stehen.

c) (6 Punkte) Betrachten Sie folgenden durch fette Kanten markierten partiellen Spannbaum, wie der Algorithmus von Kruskal oder Prim ihn im Laufe seiner Ausführung bisher berechnet haben könnte. Geben Sie für jeden der beiden Algorithmen an, welche Kante als nächstes zum partiellen Spannbaum hinzugefügt wird. Wenn es mehrere Möglichkeiten gibt, geben Sie alle an.



d) (4 Punkte) Geben Sie für die hervorgehobenen vier Kanten an, ob sie in keinem minimalen Spannbaum, in mindestens einem aber nicht allen minimalen Spannbäumen oder in allen minimalen Spannbäumen enthalten sind.



(Q1) e_1: ☐ in keinem ☐ in mind. einem, aber nicht allen ☐ in allen

(Q2) e_2: ☐ in keinem ☐ in mind. einem, aber nicht allen ☐ in allen

(Q3) e_3: ☐ in keinem ☐ in mind. einem, aber nicht allen ☐ in allen

(Q4) e_4: ☐ in keinem ☐ in mind. einem, aber nicht allen ☐ in allen

e) (3 Punkte) Begründen Sie Ihre Entscheidung für e_4 entweder mithilfe des Kreislemmas oder des Kantenschnittlemmas.

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

a) (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) Der Stack von Runs mit den Längen (Länge des obersten Elements zuerst) $(7, 11, 19, 26)$ erfüllt die Invarianten von TimSort.
☐ Wahr ☐ Falsch

(Q2) SelectionSort wie er in der Vorlesung besprochen wurde ist ein stabiles Sortierverfahren.
☐ Wahr ☐ Falsch

(Q3) Die Anzahl der Vergleiche, die ein allgemeines (d.h. vergleichsbasiertes) Sortierverfahren im **Best-Case** durchführt, liegt immer in $\Omega(n/\log n)$.
☐ Wahr ☐ Falsch

(Q4) Zum Verschmelzen (Mergen) zweier sortierter Arrays A und B mit jeweils n Elementen und $\forall i, j \in \{1, \ldots, n\} : A[i] < B[j]$ werden in TimSort $\Theta(\log n)$ Vergleiche durchgeführt.
☐ Wahr ☐ Falsch

b) (12 Punkte) Gegeben sind zwei natürliche Zahlen x und y mit jeweils $n = 2^k$ Bit. Man betrachte nun die Bitstrings der binären Repräsentation von x und y und teile sie in die Hälfte der höherwertigen und die Hälfte der niederwertigen Bits. Ist beispielsweise $x = 01101010_2$, dann ergibt sich die höherwertige Hälfte zu 0110_2 und die niederwertige Hälfte zu 1010_2.

Sei `Split(`x`)` die Funktion, die diese Aufteilung vornimmt. D.h. $x_1, x_0 \leftarrow$ `Split(`01101010_2`)` resultiert in $x_1 = 0110_2$ und $x_0 = 1010_2$. Mit $x_1, x_0 \leftarrow$ `Split(`x`)` und $y_1, y_0 \leftarrow$ `Split(`y`)` ergibt sich das Produkt $x \cdot y$ dann zu

$$z = x \cdot y = \underbrace{x_1 \cdot y_1}_{z_2} \cdot 2^n + \underbrace{(x_1 \cdot y_0 + x_0 \cdot y_1)}_{z_1} \cdot 2^{n/2} + \underbrace{x_0 \cdot y_0}_{z_0}.$$

Beachten Sie, dass folgende Gleichung gilt:

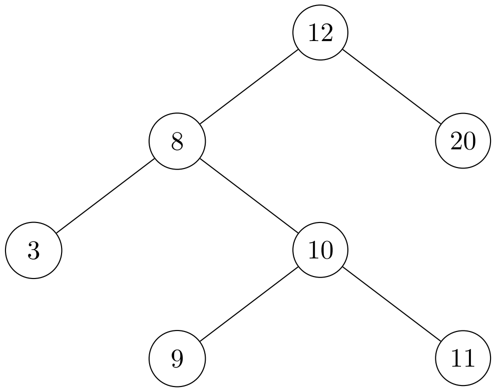
$$z_1 = z_2 + z_0 - (x_1 - x_0) \cdot (y_1 - y_0)$$

Der Divide-and-Conquer-Algorithmus auf der nächsten Seite soll die beiden Übergabeparameter x und y miteinander multiplizieren. Vervollständigen Sie die fehlenden Teile. Verwenden Sie dazu nur Additionen, Subtraktionen, die Betragsfunktion $|\cdot|$ und rekursive Aufrufe von `Multiply`. Achten Sie darauf, dass sich die Methode `Multiply` nur **drei** mal selbst aufruft.

```
Multiply(x, y, k):
// x und y sind ganze, nichtnegative Zahlen mit jeweils n = 2^k Bit

if k = 0 then
    return x · y

x_1, x_0 ← Split(x)
y_1, y_0 ← Split(y)

A ← [          ]

C ← [          ]
if (x_1 > x_0 ∧ y_1 > y_0) ∨ (x_1 < x_0 ∧ y_1 < y_0) then
    B ← [          ]
else
    B ← [          ]

return A · 2^(2^k) + B · 2^(2^(k-1)) + C
```

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

a) (12 Punkte) Gegeben ist folgender binärer Suchbaum T:



(i) Geben Sie die *Inorder*- und *Postorder*-Durchmusterungsreihenfolge von T an.

(ii) Wie viele Schlüsselvergleiche benötigt eine erfolgreiche Suche in T durchschnittlich?

Hinweis: Bei der Suche nach dem Wurzelknoten wird genau ein Schlüsselvergleich benötigt.

(iii) Bestimmen Sie die Balance jedes Knoten in T, wie aus der Vorlesung zu AVL-Bäumen bekannt.

(iv) Rebalancieren Sie T indem Sie die benötigte(n) Rotation(en) durchführen, wie aus der Vorlesung zu AVL-Bäumen bekannt. Geben Sie den resultierenden Baum nach jedem Zwischenschritt an, also nach jeder Rotation.

b) (4 Punkte) Gegeben ist folgende Hashtabelle sowie die Hashfunktion $h'(k) = k \bmod 7$.

Position	0	1	2	3	4	5	6
Schlüssel		8		17	4		20

Fügen Sie den Schlüssel 10 als neues Element mittels der jeweiligen Sondierungsvariante in die Hashtabelle ein. Geben Sie dabei alle Zwischenschritte an, also alle Kollisionen zu denen es beim Einfügen kommt.

(i) Lineares Sondieren mit $h(k,i) = (h'(k) + i) \bmod 7$.

(ii) Quadratisches Sondieren mit $h(k,i) = (h'(k) + \frac{1}{2}i + \frac{1}{2}i^2) \bmod 7$.

c) (4 Punkte) Geben Sie bei folgenden Aussagen an, ob diese wahr oder falsch sind.

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

(Q1) Beim Einfügen eines Schlüssels k in eine Hashtabelle mittels *Verbesserung nach Brent* kann es vorkommen, dass ein sich bereits in der Tabelle befindliches Element k' verschoben wird, um für k Platz zu machen.

☐ Wahr ☐ Falsch

(Q2) Beim Uniform Hashing in einer Tabelle mit Größe m erhält jeder Schlüssel mit gleicher Wahrscheinlichkeit eine bestimmte der $m!$ Permutationen von $0, 1, \ldots, m-1$ als Sondierungsreihenfolge zugeordnet.

☐ Wahr ☐ Falsch

(Q3) Beim Hashing mit Verkettung der Überläufer gilt für den Belegungsfaktor $\alpha = \frac{n}{m}$ immer $\alpha \leq 1$, wobei m die Größe der Hashtabelle ist und n die Anzahl der in der Tabelle gespeicherten Schlüssel.

☐ Wahr ☐ Falsch

(Q4) Bei offenen Hashverfahren wird das Flag *„wieder frei"* gesetzt, um sicherzustellen, dass weiterhin alle Elemente in der Hashtabelle gefunden werden können.

☐ Wahr ☐ Falsch