TU Wien
Institut für Logic and Computation
Algorithms and Complexity Group

186.866 Algorithmen und Datenstrukturen VU 8.0

1. Test, 2019S

14. Mai 2019

Gruppe A

Machen Sie die folgenden Angaben in deutlicher **Blockschrift**:

Nachname: | Vorname:

Matrikelnummer: | Unterschrift:

Sie dürfen die Lösungen nur auf die Angabeblätter schreiben, die Sie von der Aufsicht erhalten. Es ist nicht zulässig, eventuell mitgebrachtes eigenes Papier zu verwenden. Benutzen Sie dokumentenechte Schreibgeräte (keine Bleistifte!).

Die Verwendung von Taschenrechnern, Mobiltelefonen, 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. Unleserliche Antworten werden nicht gewertet.

	A1	A2	A3	A4	A5	Summe
Erreichbare Punkte:	22	18	20	20	20	100
Erreichte Punkte:						

Viel Erfolg!

Aufgabe A1: Algorithmenanalyse **(22 Punkte)**

a) (12 Punkte) Tragen Sie für die Codestücke `FunktionA` und `FunktionB` die Laufzeit und den Rückgabewert (z) in Abhängigkeit von n, jeweils in Θ-Notation in die nachfolgende Tabelle ein.

```
FunktionA(n):
    x ← n^2
    z ← 1
    while x > 1
        x ← x/3
        z ← 3z
    return z
```

```
FunktionB(n):
    j ← 0
    z ← 1
    for i ← 1 bis 3n
        if i mod 3 = 0 then
            j ← j + i
    j ← (2/3)j − n
    while j > 0
        z ← nz
        j ← j − 1
    return z
```

	`FunktionA`	`FunktionB`
Laufzeit	$\Theta(\log(n))$	$\Theta(n)$
Rückgabewert (z)	$\Theta(n^2)$	$\Theta(n^{n^2})$

b) (4 Punkte) Sei $c = \frac{1}{2}$. Finden Sie das kleinste $n_0 \in \mathbb{N}^+$, sodass für alle $n \geq n_0$ gilt

$$2 \cdot \sqrt{\frac{n+1}{4}} \leq cn.$$

Kleinstes $n_0 =$ 5

Was haben Sie somit gezeigt? Kreuzen Sie Zutreffendes an:

- ☐ $2 \cdot \sqrt{\frac{n+1}{4}}$ ist in $\Omega(n)$
- ☒ $2 \cdot \sqrt{\frac{n+1}{4}}$ ist in $O(n)$
- ☐ $2 \cdot \sqrt{\frac{n+1}{4}}$ ist in $\Theta(n)$
- ☐ keine der zuvor genannten Aussagen

$2\sqrt{\frac{n+1}{4}} \leq cn$ | $\cdot 2, \hat{}2$

$\frac{n+1}{4} \leq \frac{c^2 n^2}{1}$ | $c = \frac{1}{2}$

$n+1 \leq \frac{1}{4} \cdot n^2$

$\frac{n+1}{n^2} \leq \frac{1}{4}$ | ab $n = 5$

$n=5$ $\quad \frac{6}{25} \leq \frac{1}{4}$ ✓

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

(korrektes Kreuz: +2 Punkte, inkorrektes Kreuz: -2 Punkte, kein Kreuz: 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)

(i) Wenn $f = O(g)$ und $g = \Omega(h)$ gilt, dann gilt auch $f = \Theta(h)$.

☐ Wahr ☒ Falsch

(ii) Wenn die Worst-Case Laufzeit eines Algorithmus über alle Eingaben der Eingabegröße n in $O(n^2)$ liegt, so kann seine asymptotische Laufzeit nicht in $\Omega(n^3)$ liegen.

☒ Wahr ☐ Falsch

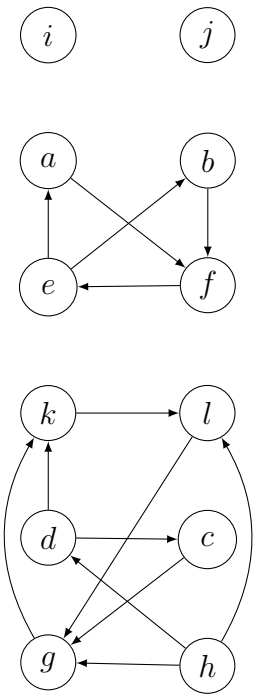
(iii) Wenn die Worst-Case Laufzeit eines Algorithmus über alle Eingaben der Eingabegröße n in $\Omega(n^2)$ liegt, so kann seine asymptotische Laufzeit nicht in $\Omega(n^3)$ liegen.

☐ Wahr ☒ Falsch

Aufgabe A2: Graphen **(18 Punkte)**

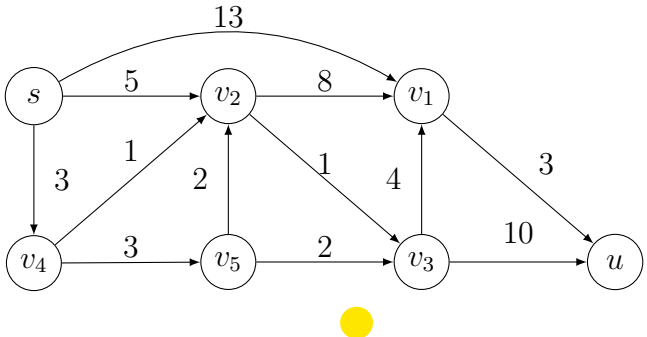
a) (5 Punkte) Gegeben ist der nachfolgende Graph $H = (V_H, E_H)$ mit 12 Knoten. Verwenden Sie die aus der Vorlesung bekannten Definitionen und kreuzen Sie Zutreffendes in der untenstehenden Tabelle an. Betrachten Sie für jede Zeile der Tabelle den gerichteten Teilgraphen $G = (V_G, E_G)$ von H mit der in der Tabelle gegebenen Knotenmenge V_G und $E_G = \{(a, b) \mid a, b \in V_G, (a, b) \in E_H\}$.

(korrekte Zeile: +1 Punkt, inkorrekte Zeile: -1 Punkt, keine Antwort (Zeile ohne Kreuz): 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)



V_G	G ist schwach zusammenhängend	G ist schwache Zusammenhangskomp. von H	G ist stark zusammenhängend	G ist starke Zusammenhangskomp. von H	keine der zuvor genannten Aussagen trifft zu
$\{c, l\}$	☒	☐	☐	☐	☐
$\{a, e, f\}$	☐	☐	☒	☐	☐
$\{c, d, g, h, k, l\}$	☒	☒	☐	☐	☐
$\{g, k, l\}$	☐	☐	☐	☐	☐
$\{i\}$	☐	☐	☐	☐	☐

b) (10 Punkte) Wenden Sie den aus der Vorlesung bekannten *Algorithmus von Dijkstra in der Implementierung mit einer Liste*, zum Finden eines kürzesten Pfades von s nach u auf dem gegebenen Graphen an. Dokumentieren Sie für jeden Schritt, bei dem sich die Menge Discovered ändert (diese enthält die bereits untersuchten Knoten), für jeden Knoten $d(\cdot)$ und die Mengen Discovered und L. Extrahieren Sie anschließend aus dem Ablauf einen kürzesten Pfad, sowie seine Länge.



1. Discovered = {s}
   L = {v4, v2, v1}

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$	0	13	5	∞	3	∞	∞

2. Discovered = {s, v4}
   L = {v2, v5, v1}

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$	0	13	4	∞	0	6	∞

3. Discovered = {s, v4, v2}
   L = {v3, v5, v1}

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$	0	12	0	5	0	6	∞

4. Discovered = {s, v4, v2, v3}
   L = {v5, v1, u}

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$	0	9	0	0	0	6	15

5. Discovered = {s, v4, v2, v3, v1}
   L = {v5, u}

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$	0	0	0	0	0	6	12

6. Discovered = {s, v4, v2, v3, v1, v5}
   L = {u}

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$	0	0	0	0	0	0	12

7. Discovered = {s, v4, v2, v3, v1, v5, u}
   L = {}

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$	0	0	0	0	0	0	0

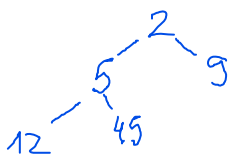
8. Discovered = {
   L = {

Knoten	s	v_1	v_2	v_3	v_4	v_5	u
$d(\cdot)$							

Pfad: 4, 2, 3, 1, u

Gewicht: 12

c) (3 Punkte) Zeichnen Sie einen Min-Heap zur Menge $\{12, 9, 2, 45, 5\}$.



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

Im Folgenden sind einige Fragen zu dem Themenbereich Greedy-Algorithmen gegeben. Beantworten Sie jeweils die gestellte Ja/Nein-Frage. Für jede Antwort ist zusätzlich eine (informelle) Begründung bzw. ein konkretes Gegenbeispiel notwendig.

a) Gegeben ist eine Währung deren Münzen eine Stückelung von $1, 3, 6$ und 13 haben. Liefert der Greedy-Algorithmus zum Geldwechseln aus der Vorlesung bei dieser Stückelung immer ein optimales Ergebnis?

☐ Ja ☒ Nein

Begründung/Gegenbeispiel:

	Greedy	Opt
18	13, 3, 1, 1	6, 6, 6

b) Sei G ein gewichter Graph, bei dem alle Kanten unterschiedliche Gewichte haben. Wenn Sie den Algorithmus von Kruskal und den Algorithmus von Prim verwenden um einen Minimum Spanning Tree (MST) von G zu berechnen, finden dann beide Algorithmen immer denselben MST?

☒ Ja ☐ Nein

Begründung/Gegenbeispiel:

Kruskal hat Optimalität. Da wenn alle Kantengewichte unterschiedlich sind GIBT es nur einen optimalen MST.

c) Sei T ein Spannbaum in einem gewichteten Graphen G. Wir nennen die Kanten in T mit dem höchsten Gewicht die Bottleneck-Kanten von T. Wir sagen T ist ein Minimum Bottleneck Spanning Tree (MBST) falls es keinen Spanning Tree von G gibt, dessen Bottleneck-Kanten ein geringeres Gewicht als die Bottleneck-Kanten von T haben. Ist jeder MBST auch ein minimaler Spannbaum?

☒ Ja ☐ Nein

Begründung/Gegenbeispiel:

Das Kreislemma zeigt, dass schwerste Kante nur mitgenommen wird, wenn diese nicht Teil eines Kreises ist.

Somit ist die Bottleneckkante eine unumgängliche Brücke für den Zusammenhang des Graphen.

Somit ist der MBST auch ein MST.

d) Sei G ein Graph, bei dem jedem Knoten v eine reele Zahl $w(v)$ als Gewicht zugeordnet ist. Für eine Menge von Knoten S schreiben wir $w(S)$ für die Summe $\Sigma_{v \in S} w(v)$ der Gewichte der Knoten in S. Eine *überdeckende Knotenmenge* von G ist eine Menge S von Knoten, so dass jede Kante von G zu einem Knoten in S inzident ist. Eine überdeckende Knotenmenge S heißt minimal, wenn für jede andere überdeckende Knotenmenge S^* stets $w(S) \leq w(S^*)$ gilt. Gegeben ist der folgende Greedy-Algorithmus:

`Greedy(`n`)`:
Input: Graph $G = (V, E)$ mit gewichteten Knoten
$S \leftarrow \emptyset$
$S_E \leftarrow \emptyset$
while $S_E \neq E$
Wähle einen Knoten $v \in V \setminus S$ mit minimalem Gewicht.
Füge v zu S hinzu.
Füge alle Kanten die indizent zu v sind zu S_E hinzu.

Findet dieser Algorithmus immer eine minimale überdeckende Knotenmenge?

☐ Ja ☐ Nein

Begründung/Gegenbeispiel:

Aufgabe A4: Hashing und Bäume **(20 Punkte)**

a) (12 Punkte) Im Folgenden geht es um die Best-Case- und Worst-Case-Laufzeiten unterschiedlicher Operationen für Hashing mit Verkettung von Überläufern, offenes Hashing und bei binären Suchbäumen in Abhängigkeit der n bereits eingefügten Elemente.

(korrekte Zeile: +1 Punkt, inkorrekte Zeile: -1 Punkt, leere bzw. unvollständige Zeile: 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)

(i) AVL-Bäume

Operation	**Best-Case**	**Worst-Case**
Erfolgreiches Einfügen	$\Theta(\log n)$	$\Theta(\log n)$
Erfolgreiches Suchen	$\Theta(1)$	$\Theta(\log n)$
Erfolgloses Suchen	$\Theta(\log n)$	$\Theta(\log n)$

(ii) Offenes Hashing mit linearem Sondieren

Operation	**Best-Case**	**Worst-Case**
Erfolgreiches Einfügen	$\Theta(1)$	$\Theta(n)$
Erfolgreiches Suchen	$\Theta(1)$	$\Theta(n)$
Erfolgloses Suchen	$\Theta(n)$	$\Theta(n)$

(iii) Hashing mit Verkettung der Überläufer

Operation	**Best-Case**	**Worst-Case**
Erfolgreiches Einfügen	$\Theta(1)$	$\Theta(n)$
Erfolgreiches Suchen	$\Theta(1)$	$\Theta(n)$
Erfolgloses Suchen	$\Theta(n)$	$\Theta(n)$

(iv) Binäre Suchbäume ohne Höhenbalancierung

Operation	**Best-Case**	**Worst-Case**
Erfolgreiches Einfügen	$\Theta(\log n)$	$\Theta(n)$
Erfolgreiches Suchen	$\Theta(1)$	$\Theta(n)$
Erfolgloses Suchen	$\Theta(\log n)$	$\Theta(n)$

b) (8 Punkte) Gegeben sind folgende natürliche Zahlen:

$$[16, 8, 17, 6, 1, 13, 15]$$

Fügen Sie diese in der vorgegebenen Reihenfolge jeweils in folgende Varianten von anfangs leeren Hashtabellen der Größe $m = 9$ ein. Berechnen Sie dann die durchschnittliche Anzahl von Schlüsselvergleichen bei einer erfolgreichen Suche eines Elements, wobei jedes dieser Elemente mit gleicher Wahrscheinlichkeit gesucht wird. Für das Sondieren gilt $i = 0, 1, \ldots, m - 1$.

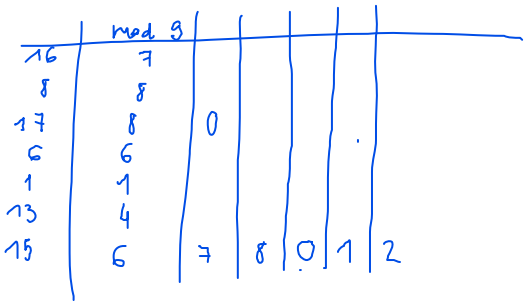
(je korrekt befüllter Hashtabelle: 2 Punkte, je korr. Anzahl Vergleiche: 2 Punkte)

(i) Verkettung der Überläufer mit $h(k) = k \bmod m$.

Durchschnittliche Anzahl von Vergleichen bei erfolgreicher Suche: $T =$ 13/7

Hashtabelle:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
17	1	15		13		6	16	8

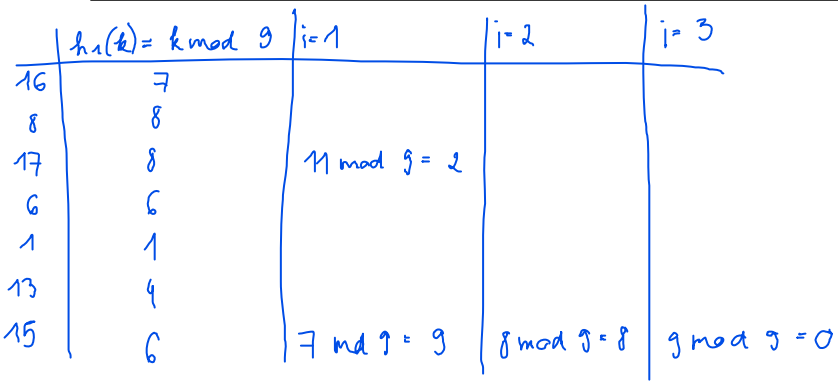


(ii) Double-Hashing mit $h(k) = (h_1(k) + ih_2(k)) \bmod m$, $h_1(k) = k \bmod m$ und $h_2(k) = 1 + (k \bmod 5)$.

Durchschnittliche Anzahl von Vergleichen bei erfolgreicher Suche: $T =$ 11/7

Hashtabelle:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
15	1	17		13		6	16	8



Aufgabe A5: Sortieralgorithmen **(20 Punkte)**

a) (4 Punkte) Können die folgenden Arrays als Run in einem Ablauf eines Timsort Algorithmus mit $Min\_Run = 6$ vorkommen? Kreuzen Sie Zutreffendes an.

	Ja	Nein
$[2, 2, 2, 2, 2, 2, 2]$	☐	☐
$[2, 4, 7, 6, 8, 11, 14]$	☐	☐
$[7, 7, 6, 6, 5, 5, 4]$	☐	☐
$[2, 6, 9, 11, 18, 15]$	☐	☐

(korrektes Kreuz: +1 Punkt, inkorrektes Kreuz: -1 Punkt, kein Kreuz: 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)

b) (16 Punkte) Dual Pivot Quicksort ist eine Variante von Quicksort, die in der Praxis effizienter ist als ein normaler Quicksort. Dual Pivot Quicksort funktioniert im Wesentlichen wie Quicksort, mit den folgenden Änderungen:

Bei Dual Pivot Quicksort werden zwei Elemente als Pivot gewählt. Wir nennen das kleinere Pivot Element linkes Pivot (LP) und das größere Pivot Element rechtes Pivot (RP). Das Array wird dann in drei Teile partitioniert: Die Elemente die kleiner sind als LP, die Elemente die größer oder gleich LP aber kleiner oder gleich RP sind, und die Elemente die größer als RP sind. Danach wird Dual Pivot Quicksort rekursiv auf jede der drei Partitionen aufgerufen.

Kreuzen Sie im Pseudocode auf der nächsten Seite die Codezeilen an, die ausgeführt werden müssen, um eine funktionierende Implementierung von Dual Pivot Quicksort zu erhalten. Je Block (grau hinterlegte Box) ist genau eine Auswahl korrekt.

(richtiges Kreuz +4 Punkte, falsches Kreuz/mehrere Kreuze im Block -4 Punkte, kein Kreuz 0 Punkte. Minimum für diese Unteraufgabe: 0 Punkte)

`DualPivotQuicksort`(A,left,right):

```
// Array A, wird von Element left bis Element right sortiert
if right − left ≥ 1 then
    // Erstes (=left) und letztes (=right) Element als Pivot
    p ← min{A[left], A[right]}    // p ist das kleinere Pivotelement
    q ← max{A[left], A[right]}    // q ist das größere Pivotelement
    l ← left + 1    // l begrenzt die linke Partition
    g ← right − 1    // g begrenzt die rechte Partition
    k ← l    // k ist die Laufvariable der Partitionierung
    ☒ while k ≤ g
    ☐ while l ≤ g
    ☐ while k ≤ q
        if A[k] < p then
            Tausche A[k] und A[l]
            l ← l + 1
        else if A[k] > q then
            while A[g] > q und k < g
                ☒ g ← g − 1
                ☐ k ← k + 1
                ☐ l ← l + 1
            Tausche A[k] und A[g]
            g ← g − 1
            if A[k] < p then
                Tausche A[k] und A[l]
                l ← l + 1
        ☐ g ← g − 1
        ☒ k ← k + 1
        ☐ l ← l + 1
    l ← l − 1
    g ← g + 1
    A[left] ← A[l]
    A[l] ← p // p wird an die finale Position gesetzt
    A[right] ← A[g]
    A[g] ← q // q wird an die finale Position gesetzt
    ☐ call DualPivotQuicksort(A, left + 1, l)
      call DualPivotQuicksort(A, l + 1, g)
      call DualPivotQuicksort(A, g + 1, right − 1)

    ☐ call DualPivotQuicksort(A, left, p − 1)
      call DualPivotQuicksort(A, p + 1, q − 1)
      call DualPivotQuicksort(A, q + 1, right)

    ☒ call DualPivotQuicksort(A, left, l − 1)
      call DualPivotQuicksort(A, l + 1, g − 1)
      call DualPivotQuicksort(A, g + 1, right)
```