Prof. Dr. Ir. Joost-Pieter Katoen Christian Dehnert, Jonathan Heinen, Thomas Ströder, Sabrina von Styp

Klausur Datenstrukturen und Algorithmen SoSe 2012

Vorname: _____

Nachname: _____

Matrikelnummer: _____

Studiengang (bitte **genau** einen markieren):

- ○ Informatik Bachelor
- ○ Informatik Lehramt
- ○ Mathematik Bachelor
- ○ Computational Engineering Science
- ○ Sonstiges: _____

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**20**	
Aufgabe 2	**20**	
Aufgabe 3	**20**	
Aufgabe 4	**20**	
Aufgabe 5	**20**	
Aufgabe 6	**20**	
Summe	**120**	

Allgemeine Hinweise:

- **Auf alle Blätter** (inklusive zusätzliche Blätter) müssen Sie **Ihren Vornamen, Ihren Nachnamen und Ihre Matrikelnummer** schreiben.
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- Schreiben Sie mit **dokumentenechten** Stiften, nicht mit roten oder grünen Stiften und nicht mit Bleistiften.
- Bitte beantworten Sie die Aufgaben auf den Aufgabenblättern (benutzen Sie auch die Rückseiten).
- Geben Sie für jede Aufgabe **maximal eine** Lösung an. Streichen Sie alles andere durch. Andernfalls werden alle Lösungen der Aufgabe mit **0 Punkten** bewertet.
- Werden **Täuschungsversuche** beobachtet, so wird die Klausur mit **0 Punkten** bewertet.
- Geben Sie am Ende der Klausur **alle Blätter zusammen mit den Aufgabenblättern ab**.
- Gehen Sie bei Codeanalysen davon aus, dass sämtliche Instruktionen wie arithmetische Operationen (+,-.*,/), Vergleiche usw. in konstanter Zeit $\mathcal{O}(1)$ ausgeführt werden.

Name: **Matrikelnummer:**

Aufgabe 1 ($\mathcal{O}$-Notation): (6 + 6 + 8 = 20 Punkte)

a) Sortieren Sie für die unten gegebenen Funktionen die $\mathcal{O}$-Klassen $\mathcal{O}(a(n)), \mathcal{O}(b(n)), \mathcal{O}(c(n)), \mathcal{O}(d(n))$ und $\mathcal{O}(e(n))$ bezüglich ihrer Teilmengenbeziehung. Nutzen Sie ausschließlich die echte Teilmenge $\subset$ sowie die Gleichheit $=$ für die Beziehungen zwischen den Mengen. Folgendes Beispiel illustriert diese Schreibweise für einige Funktionen f_1 bis f_5 (diese haben nichts mit den unten angegebenen Funktionen zu tun):

$$\mathcal{O}(f_4(n)) \subset \mathcal{O}(f_3(n)) = \mathcal{O}(f_5(n)) \subset \mathcal{O}(f_1(n)) = \mathcal{O}(f_2(n))$$

Die angebenen Beziehungen müssen weder bewiesen noch begründet werden.

$a(n) = n^2 \cdot log_2 n + 42$ $b(n) = 2^n + n^4$ $c(n) = 2^{2 \cdot n}$ $d(n) = 2^{n+3}$ $e(n) = \sqrt{n^5}$

b) Beweisen oder widerlegen Sie $(\log_2 n)^2 \in O(n)$.

c) Beweisen oder widerlegen Sie $n! \in \Omega(n^{n-2})$.

Name: **Matrikelnummer:**

Aufgabe 2 (Sortieren): (6 + 3 + 3 + 8 = 20 Punkte)

a) Sortieren Sie das folgende Array mittels des Heapsort-Algorithmus aus der Vorlesung.

5	9	13	7	2	3

Geben Sie das vollständige Array nach jeder Versickerung (Heapify-Operation) an, bei der sich der Arrayinhalt ändert.

Hinweis: Es könnte hilfreich sein, die jeweils noch unsortierten Arraybereiche zusätzlich als Heap darzustellen. Dies ist jedoch optional.

b) Beweisen oder widerlegen Sie die folgende Aussage:

Heapsort ist stabil.

c) Geben Sie die asymptotische Best-Case Laufzeit (Θ) von Mergesort an. Begründen Sie Ihre Antwort (ein Verweis darauf, dass dies in der Vorlesung gezeigt wurde, reicht nicht aus).

d) Bestimmen Sie die Best-Case Laufzeit (Θ) des untenstehenden `natMergeSort` Sortieralgorithmus in Abhängigkeit der Arraylänge n und geben Sie an, ob dieser Algorithmus ein stabiler Sortieralgorithmus ist. Nehmen Sie dazu an, dass die Initialisierung des `R` Arrays in konstanter Zeit zu bewerkstelligen ist. Begründen Sie Ihre Antwort.

```
void natMergeSort(int E[], int n) { // Eingabearray und seine Laenge
    int R[] = 0; // initialisiere ein neues Array mit 0 Werten
    int r = 1;   // Index fuer R; R[0] bleibt 0, da r mit 1 beginnt
    for (int i = 0; i < n - 1; i++) {
        if (E[i] > E[i + 1]) { // falsche Reihenfolge?
            R[r] = i + 1;      // neues sortiertes Teilstueck gefunden
            r++;
        }
    }
    R[r] = n;
    sort(E, R, 0, r - 1);
}

// nahezu Mergesort
void sort(int E[], int R[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2; // finde Mitte
        sort(E, R, left, mid);        // sortiere linke Haelfte
        sort(E, R, mid + 1, right);   // sortiere rechte Haelfte
        // Verschmelzen der sortierten Haelften
        // merge Operation ist aus Vorlesung bekannt
        // erwartet ein Array und drei Positionen
        // fuegt die beiden Arrayabschnitte zwischen den Positionen
        // sortiert zusammen und setzt sie in das Originalarray ein
        merge(E, R[left], R[mid + 1] - 1, R[right + 1] - 1);
    }
}
```

Hinweis: Sie dürfen Eigenschaften von aus der Vorlesung bekannten Sortierverfahren in Ihrer Begründung benutzen.

Name: **Matrikelnummer:**

Aufgabe 3 (Bäume): (3 + 2 + 8 + 7 = 20 Punkte)

Gegeben sei der folgende Algorithmus für nicht leere, binäre Bäume:

```
class Node{
    Node left, right;
    int value;
}

void do(Node node){
    Node m = max(node);
    if(m.value > node.value){
    // swap der Werte von m und node
        int tmp = m.value;
        m.value = node.value;
        node.value = tmp
    }
    if(node.left != null)
        do(node.left);
    if(node.right != null)
        do(node.right);
}
```

```
Node max(Node node){
  Node max = node;
    if(node.left != null){
        node tmp = max(node.left);
        if(tmp.value > max.value)
            max = tmp;
    }
    if(node.right != null){
        node tmp = max(node.right);
        if(tmp.value > max.value)
            max = tmp;
    }
    return max;
}
```

a) Beschreiben Sie in möglichst wenigen Worten die Auswirkung der Methode `do(tree)`.

b) Die Laufzeit der Methode `max(tree)` ist für sämtliche Eingaben linear in n, der Anzahl von Knoten im übergebenen Baum `tree`. Begründen Sie kurz, warum `max(tree)` eine lineare Laufzeit hat.

c) Geben Sie in Abhängigkeit von n, der Anzahl von Knoten im übergebenen Baum `tree`, jeweils eine Rekursionsgleichung für die asymptotische Best- ($B(n)$) und Worst-Case Laufzeit ($W(n)$) des Aufrufs `do(tree)` sowie die entsprechende Komplexitätsklasse (Θ) an. Begründen Sie Ihre Antwort.

Hinweis: Überlegen Sie, ob die Struktur des übergebenen Baumes Einfluss auf die Laufzeit hat. Die lineare Laufzeit von `max(tree)` darf vorausgesetzt werden.

d) Beweisen sie per Induktion die folgende Aussage: Ein Rot-Schwarz-Baum der Schwarzhöhe h hat maximal $rot(h) = \frac{2}{3} \cdot 4^h - \frac{2}{3}$ rote Knoten.

Name: **Matrikelnummer:**

Aufgabe 4 (Hashing): (4 + 4 + 5 + 5 + 2 = 20 Punkte)

Gegeben seien die Zahlen 43, 13, 61, 41, 53, 95 und 31 und die Hashfunktion $h(x) = x \mod 10$ sowie eine Hashtabelle mit 10 Plätzen. Füge Sie die Elemente mit Hilfe folgender Hashverfahren ein.

a) offene Adressierung mit linearem Sondieren,

0	1	2	3	4	5	6	7	8	9

b) offene Adressierung mit doppeltem Hashing und folgender zweiter Hashfunktion: $h'(x) = 7 - (x \mod 7)$.

0	1	2	3	4	5	6	7	8	9

c) Ein weiteres Hashverfahren ist Brent-Hashing. Im Gegensatz zum Hashing aus der Vorlesung wird bei einer Kollision im Brent-Hashing die nächste Sondierungsposition für beide Elemente berechnet und das alte Element genau dann verschoben, wenn dies einen Sondierungsschritt und das neue Element mehr als einen (weiteren) braucht.

Seien h_1 und h_2 Hashfunktionen, so ist $h(k, i) = (h_1(k) + i \cdot h_2(k)) \mod m$ die Hashfunktion, die man beim Brent-Hashing benutzt.

Brent-Hashing für ein Element k_1 im i-ten Sondierungsschritt funktioniert nun wie folgt.

- Ist $h(k_1, i)$ frei, so füge k_1 an dieser Position ein.
- Ist $h(k_1, i)$ belegt mit einem Element k_2, $h(k_1, i+1)$ ebenfalls belegt, k_2 wurde mit j Sondierungsschritten eingefügt (d. h. $h(k_1, i) = h(k_2, j)$) und $h(k_2, j+1)$ ist frei, so füge k_2 an der Position $h(k_2, j+1)$ ein und k_1 an der bisherigen Position von k_2.
- Ansonsten fahre mit der Sondierung für k_1 und $i+1$ fort.

Gegeben sei nun eine Hashtabelle mit 11 Plätzen und die Hashfunktion

$$h(k, i) = ((k \mod 11) + i \cdot (k \mod 7)) \mod 11.$$

Fügen Sie die folgenden Werte mittels Brent-Hashing ein: 24, 22, 48, 68 und 59.

0	1	2	3	4	5	6	7	8	9	10

d) Vervollständigen Sie den nachfolgenden Algorithmus `brentInsert`, der ein Element `k` nach Brent-Hashing in eine Hashtabelle `table` der Länge `m` einfügt. Die dabei verwendete Datenstruktur `Hash` hat zwei Felder: `key` vom Typ `int` und `state`, welches die Werte `free`, `used` und `deleted` annehmen kann. Gehen Sie davon aus, dass auf die Hashfunktionen h_1 und h_2 mit entsprechenden Funktionen `int h1(int value)` und `int h2(int value)` zugegriffen werden kann. Die zu vervollständigenden Teile sind durch Unterstreichungen markiert.

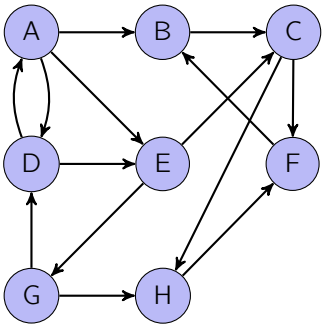
```
void brentInsert(Hash table[], int m, int k) {
    int insertPos = h1(k);
    while (table[insertPos].state == used) {

        int newNext = _____;

        int oldNext = _____;

        if (table[newNext].state == free ||
            table[oldNext].state == used) {
            insertPos = newNext;
        } else {

            table[oldNext].key = _____;

            table[oldNext].state = used;
            table[insertPos].state = deleted;
        }
    }

    table[insertPos].key = _____;

    table[insertPos].state = used;
}
```

e) Welche Änderungen müssen an der normalen Suche bei offener Adressierung (Algorithmus `hashSearch` aus der Vorlesung) vorgenommen werden, um für Brent-Hashing korrekt zu funktionieren?

Name: **Matrikelnummer:**

Aufgabe 5 (Graphen): (2 + 6 + 7 + 5 = 20 Punkte)

a) Betrachten Sie den folgenden gerichteten Graphen G_1:

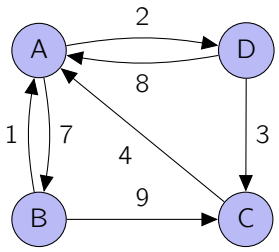


Geben Sie den Kondensationsgraphen $G_1 \downarrow$ an. Beschriften Sie die Knoten im Kondensationsgraphen mit den Namen aller Knoten, die zur jeweiligen starken Zusammenhangskomponente gehören. Bilden beispielsweise die Knoten 1 und 3 eine starke Zusammenhangskomponente, so sieht der zugehörige Knoten im Kondensationsgraphen wie folgt aus:

b) Beweisen oder widerlegen Sie die folgende Aussage:

Für jeden zusammenhängenden, ungerichteten, gewichteten Graphen $G = (V, E)$ mit nicht-negativen Kantengewichten gibt es einen Knoten $v \in V$, sodass der SSSP-Baum von v in G ein minimaler Spannbaum von G ist.

c) Bestimmen Sie für den folgenden Graphen G_2 die Werte der kürzesten Pfade zwischen allen Paaren von Knoten. Nutzen Sie hierzu den Algorithmus von Floyd. Geben Sie das Distanzarray vor dem Aufruf, sowie nach jeder Iteration an.
Die Knotenreihenfolge sei mit A, B, C, D fest vorgegeben.



①	A	B	C	D
A				
B				
C				
D				

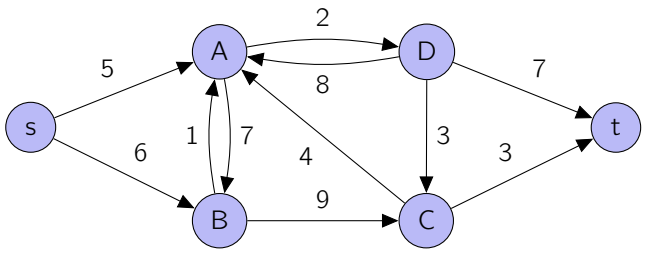
②	A	B	C	D
A				
B				
C				
D				

③	A	B	C	D
A				
B				
C				
D				

④	A	B	C	D
A				
B				
C				
D				

⑤	A	B	C	D
A				
B				
C				
D				

d) Geben Sie einen minimalen Schnitt und den Wert eines maximalen Flusses für folgendes Flussnetzwerk G_3 an.



Name: **Matrikelnummer:**

Aufgabe 6 (Dynamische Programmierung): (1 + 3 + 8 + 8 = 20 Punkte)

Gesucht ist ein Algorithmus zur Bestimmung der kleinsten Anzahl an Münzen, die nötig sind, um einen bestimmten Geldbetrag zu zahlen. Hierzu werden verschiedene Münzwerte $m_1, m_2, \ldots, m_k$ mit $k > 0$ und $m_i \geq 1$ sowie der zu zahlende Betrag B gegeben.

Beispiel:
Seien die Münzwerte 1, 2 und 5 sowie der Betrag 9 gegeben, so ist die kleinste Anzahl an Münzen 3, nämlich zwei Münzen des Werts 2 sowie eine mit dem Wert 5.

Der folgende Algorithmus soll die Aufgabe übernehmen:

```
int minimum(int m[], int k, int b) {
    sort(m); // sortiert die Werte absteigend
    int n = 0, c = 0;
    while (b > 0) {
        if (b < m[n]) {
            n++;
            if (n >= k) {
                return -1;
            }
        } else {
            b = b - m[n];
            c++;
        }
    }
    return c;
}
```

a) Auf welchem Prinzip beruht der gegebene Algorithmus?

b) Geben Sie ein Beispiel an, bei dem der obige Algorithmus nicht die optimale Lösung findet.

c) Geben Sie eine rekursive Gleichung für $C(i, b)$ an, wobei $C(i, b)$ die minimale Anzahl an Münzen ist, die benötigt wird, um den Betrag b mit den Münzwerten m_1 bis m_i zu bezahlen.

Beachten Sie, dass nicht unbedingt alle Beträge zahlbar sind (z. B. wenn es keinen Münzwert 1 gibt). Diese Fälle sollen durch den Wert ∞ repräsentiert werden.

d) Für zwei Buchstabensequenzen $a = a_1 a_2 \dots a_n$ und $b = b_1 b_2 \dots b_m$ lässt sich anhand der folgenden Rekursionsgleichung die Levenshtein-Distanz zwischen den Teilsequenzen $a_1 a_2 \dots a_i$ und $b_1 b_2 \dots b_j$ bestimmen:

$$D(i,j) = \begin{cases} j & \textit{falls } i = 0 \\ i & \textit{falls } j = 0 \\ D(i-1, j-1) & \textit{falls } a_i = b_j \\ \min(\ D(i-1,j-1),\ D(i-1,j),\ D(i,j-1)\ ) + 1 & \textit{sonst} \end{cases}$$

Nutzen Sie die oben gegebene Rekursionsgleichung, um gemäß dem Prinzip der dynamischen Programmierung die folgende Tabelle zu füllen, und geben Sie die resultierende Levenshtein-Distanz der Buchstabensequenzen AACHEN und ATHEN an.

		A	A	C	H	E	N
A							
T							
H							
E							
N							

Levenshtein-Distanz: