Lehrstuhl für Informatik I
Algorithmen und Komplexität
Universität Würzburg

Würzburg, den 21. Januar 2021

Prof. Dr. Alexander Wolff
Felix Klesen, M. Sc.

2. Kurztest zur Vorlesung Algorithmen und Datenstrukturen (Wintersemester 2020/2021)

Bitte beachten Sie die folgenden Hinweise!

1. **Personalien:** Bitte tragen Sie Ihre Daten gut lesbar ein.

Nachname	
Vorname	
Matrikelnummer	
Übungsgruppe	

2. **Papier:** Tragen Sie Ihre Lösungen direkt unterhalb der Aufgaben ein. Verwenden Sie kein eigenes Papier.
3. Sie dürfen ein einseitig handgeschriebenes DIN-A4-Blatt mit Notizen verwenden. **Weitere Hilfsmittel sind nicht zugelassen.**
4. Schreiben Sie **nicht** mit Bleistift oder einem roten Stift.
5. Die Bearbeitungszeit beträgt 60 Minuten.

Wir wünschen Ihnen viel Erfolg!

Aufgabe	1	2	3	4	5	6	7	8	**Gesamt**
Punkte	6	9	9	6	5	10	10	5	60
Ergebnis									

Aufgabe 1 — Zufall

/ 6 P.

```
AlgoSum(int n)
    sum = 0
    for i = 1 to n do
        if Random(1, i) == 1 then
            sum = sum + i^2
    return sum
```

Die Methode `Random(1, i)` liefert zufällig gleichverteilt eine Zahl aus $\{1, 2, \ldots, i\}$.

Bestimmen Sie den Erwartungswert der Ausgabe von `AlgoSum(n)` in Abhängigkeit von n. Verwenden Sie hierzu geeignete Zufallsvariablen.

Aufgabe 2 — QuickSort

Im Folgenden sei n die Länge des zu sortierenden Eingabefeldes, wobei alle Einträge des Eingabefeldes paarweise verschieden sind.

(a) Was ist die asymptotische Best-Case-Laufzeit von `QuickSort` in Abhängigkeit von n? Begründen Sie kurz Ihre Antwort. / 2 P.

Die Best-Case-Laufzeit ist $\Theta($_____$)$, denn:

(b) Geben Sie ein Feld A der Länge 7 bestehend aus den Zahlen $1, \ldots, 7$ an, bei dem die Anzahl der Vergleiche, die `QuickSort` macht, so gering wie möglich ist. / 3 P.

$A =$ _____

Begründen Sie kurz, warum Ihr Feld A am wenigsten Vergleiche benötigt.

(c) Erklären Sie kurz, wie man die `Partition`-Methode modifizieren kann, sodass die Worst-Case-Laufzeit (nicht die erwartete Laufzeit) von `QuickSort` in $\Theta(n \log n)$ liegt? Begründen Sie Ihren Vorschlag. / 3 P.

(d) Warum wird die Modifikation aus Teilaufgabe (c) in der Praxis nicht angewandt, obwohl dadurch die Worst-Case-Laufzeit erheblich verbessert wird? / 1 P.

Aufgabe 3 — Hashing

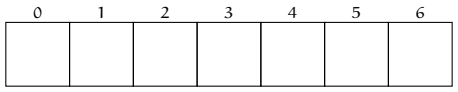
(a) Fügen Sie die Schlüssel in die gegebenen Hashtabellen ein und zeichnen Sie gegebenenfalls die Sondierfolge ein. Wenden Sie dabei doppeltes Hashing mit der Hashfunktion / 5 P.

- $h(k,i) = \left(h_1(k) + i \cdot h_2(k) \right) \mod 7$

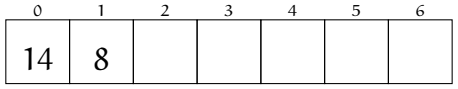
an, wobei

- $h_1(k) = k \mod 7$ und
- $h_2(k) = 2 + (k \mod 4)$.

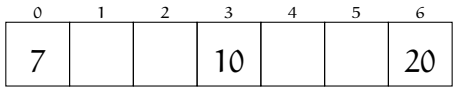
Fügen Sie rechts den Schlüssel 16 ein.



Fügen Sie rechts den Schlüssel 1 ein.



Fügen Sie rechts den Schlüssel 21 ein.



(b) In dieser Teilaufgabe wurde eine Folge von Zahlen (von links nach rechts) mit einer Hashfunktion in eine anfangs leere Hashtabelle eingefügt. Bestimmen Sie die fehlenden Parameter der Hashfunktionen, wobei Ihnen lediglich die Länge der Sondierfolge der Zahlen bekannt ist. Begründen Sie Ihre Antworten. Geben Sie auch die gefüllte Hashtabelle an. / 4 P.

Hashfunktion: $h(k,i) = (h_0(k) + i) \mod m \quad$ mit $\quad h_0(k) = k \mod m$

Auflösung von Kollisionen durch lineares Sondieren.

Zahlenfolge	5	3	16	6
Anzahl Kollisionen	0	0	1	0

$m =$ _____ und Hashtabelle $T[0..m-1] =$

Aufgabe 4 — CountingSort

/ 6 P.

Für ein Feld A von Zahlen wird `CountingSort`(A, B, k) ausgeführt. Nachdem die *ersten beiden* for-Schleifen durchlaufen wurden, enthält das Rechenfeld $C[0..k]$ folgende Werte:

$$C = \langle 0, 2, 3, 7, 7, 7, 8 \rangle \, .$$

Bestimmen Sie k, die Länge des Eingabefeldes $A.\text{length} = n$, den Inhalt des Rechenfeldes C nach der Ausführung der *ersten* for-Schleife und das Ausgabefeld $B[1..n]$ am Ende der Ausführung von `CountingSort`.

$k =$ _____ $n =$ _____

$C =$ _____

$B =$ _____

Aufgabe 5 — DeleteMulti

/ 5 P.

Erweitern Sie die Datenstruktur Liste um die Operation `DeleteMulti`(int k), die alle Elemente der Liste löscht, deren Schlüssel kleiner als der Eingabewert k ist. Wir nehmen dabei an, dass die Schlüssel vom Typ int sind.

Geben Sie die Operation in gut kommentiertem Pseudocode an. Die Laufzeit soll in $O(n)$ liegen, wobei n die aktuelle Länge der (unsortierten!) Liste ist.

Sie dürfen hierbei die aus der Vorlesung bekannten Operationen des abstrakten Datentyps Liste verwenden.

`DeleteMulti`(int k)

Aufgabe 6 — MaxStapel

Ein `MaxStapel` ist eine Datenstruktur, die eine Menge von Schlüsseln verwaltet und die folgenden Operationen besitzt:

- Die Operationen `Push`(key k) und `Pop`() verhalten sich wie bei einem Stapel.
- Die Operation `Maximum`() gibt den größten Schlüssel zurück, der sich in der Datenstruktur befindet. Dieser Schlüssel wird *nicht* aus der Datenstruktur entfernt.

(a) Wir implementieren einen `MaxStapel` durch zwei gewöhnliche Stapel, *welche die gleiche Höhe haben*: Der Stapel S_{keys} speichert die Schlüssel des `MaxStapel`. Der Stapel S_{max} verwaltet das Maximum; das Topelement von S_{max} ist also immer das augenblickliche Maximum des `MaxStapel`. / 6 P.

Implementieren Sie `Push`(key k), `Pop`() und `Maximum`() in Pseudocode, sodass alle drei Methoden eine Worst-Case-Laufzeit von $\Theta(1)$ haben.

`Push`(key k)

key `Pop`()

key `Maximum`()

(b) Seien nun alle Elemente, die in S_{keys} eingefügt werden, paarweise verschieden. Beschreiben Sie textuell, wie man die Implementierung ändern müsste, damit S_{max} in vielen Fällen weniger Schlüssel als S_{keys} enthält, die Operation `Maximum`() aber weiterhin wie gefordert funktioniert. / 4 P.

Aufgabe 7 — Binärbäume

(a) Zeigen Sie mittels vollständiger Induktion: / 5 P.

Jeder Binärbaum mit $n \geq 1$ *Knoten hat genau* $n - 1$ *Kanten.*

Induktionsanfang:

Induktionsvoraussetzung:

Induktionsschritt:

(b) Sei T ein Binärbaum, in dem jeder Knoten y das zusätzliche Attribut $x.size$ hat, das mit 0 initialisiert sei. / 5 P.

Geben Sie einen Algorithmus `ComputeSize` in Pseudocode an, nach dessen Ausführung für jeden Knoten x von T gilt, dass $x.size$ die Anzahl der Knoten ist, die der Teilbaum von T mit Wurzel x hat. `ComputeSize`(x) soll $x.size$ zurückgeben.

Die asymptotische Worst-Case-Laufzeit des Algorithmus soll $\Theta(n)$ sein, wobei n die Anzahl der Knoten von T ist.

int `ComputeSize`(Node $x = $ root)

Begründen Sie, warum Ihr Algorithmus die geforderte Laufzeit einhält.

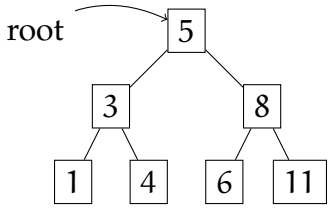
Aufgabe 8 — Ausgabe von Bäumen

Gegeben sei der folgende Algorithmus `AusgabeAlgo`, der auf einem Binärbaum arbeitet und dabei die Methode `BaumAlgo` aufruft.

```
AusgabeAlgo(Node x)
    S = new Schlange
    BaumAlgo(x, S)
    while not S.Empty() do
        k = S.Dequeue()
        gib k aus

BaumAlgo(Node x, Schlange S)
    S.Enqueue(x.key)
    if x.left ≠ nil then
        BaumAlgo(x.left, S)
    if x.right ≠ nil then
        BaumAlgo(x.right, S)
```

(a) Führen Sie `AusgabeAlgo`(T.root) für den folgenden Binärbaum T aus. / 2 P.



Ausgabe: _____

(b) Bestimmen Sie die Laufzeit von `AusgabeAlgo`(T.root) für einen Binärbaum T mit n Knoten. Begründen Sie Ihre Antwort. / 3 P.

Laufzeit: O(_____), denn: