Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

Zusatzaufgaben 01

Algorithmen I – Sommersemester 2022

Gesamtpunkte: 34

Allgemeiner Hinweis:
Für $0 \leq c < 1$ gilt sowohl $0 < \sum_{i=0}^{\infty} c^i < \infty$ als auch $0 < \sum_{i=0}^{\infty} i \cdot c^i < \infty$. Dies darfst du in allen Aufgaben verwenden.

Aufgabe 1 - Warum einfach, wenn es auch so geht? (9 Punkte)

Gegeben sei der folgende Algorithmus:

```
ARCANE(A: [ℕ; n], x: ℕ): ℕ
    INSERTIONSORT(A)
    if n = 1 then
        if A[0] < x then
            return 1
        end
        return 0
    end
    left := A[0 ... ⌊n/2⌋]
    right := A[⌊n/2⌋ + 1 ... n]
    return ARCANE(left, x) + ARCANE(right, x)
```

1. Sei $A = \langle 4, 8, 2, 3, 19, 6 \rangle$. Gib jeweils die Ausgabe von ARCANE für die Eingaben $(A, 2), (A, 6), (A, 19)$ an. (1 Punkt)
2. Sei A ein Array und a ein Element in A. Was berechnet $\text{ARCANE}(A, a)$? (2 Punkte)
3. Stelle eine Rekurrenzgleichung für ARCANE auf und bestimme anhand dessen die asymptotische Laufzeit von ARCANE. (3 Punkte)

4. Ist die Laufzeit von ARCANE optimal? Wenn ja, dann begründe, warum das so ist. Ansonsten gib einen Algorithmus in Pseudocode an, der für jede Eingabe die gleiche Ausgabe liefert wie ARCANE, aber eine geringere asymptotische Laufzeit hat. Nenne in diesem Fall auch die Laufzeit dieses Algorithmus. (3 Punkte)

Lösung 1

1. $\text{ARCANE}(A, 2) = 0, \quad \text{ARCANE}(A, 6) = 3, \quad \text{ARCANE}(A, 19) = 5$

2. ARCANE liefert den Rang von a in A, also den Index, an dem a in A stehen würde, wäre A sortiert.

3. $$T(n) = \begin{cases} 1 & \mid n = 1 \\ 2 \cdot T(\frac{n}{2}) + n^2 & \mid \text{sonst} \end{cases}$$

   Wir lösen diese Rekurrenz:

   $$\begin{aligned} \sum_{i=0}^{\log_2(n)} 2^i \cdot \left(\frac{n}{2^i}\right)^2 &= \sum_{i=0}^{\log_2(n)} 2^i \cdot \frac{n^2}{2^{2i}} \\ &= \sum_{i=0}^{\log_2(n)} \frac{n^2}{2^i} \\ &= n^2 \sum_{i=0}^{\log_2 n} \frac{1}{2^i} \end{aligned}$$

   Nach obigem Hinweis ist $\sum_{i=0}^{\log_2 n} \frac{1}{2^i} \in \Theta(1)$. Damit ist $T(n) \in \Theta(n^2)$.

4. Die Laufzeit von ARCANE ist nicht optimal. Der folgende Algorithmus bestimmt den Rang eines Elements in $\Theta(n)$:

```
RANK(A: [ℕ; n], x: ℕ): ℕ
    rank: ℕ = 0
    for i ∈ {0, ..., n − 1} do
        if A[i] < x then
            rank := rank + 1
        end
    end
    return rank
```

Aufgabe 2 - Fleißige Heinzelmännchen (3 Punkte)

Gegeben sei der folgende Sortieralgorithmus:

```
GNOMESORT(A: [ℕ; n])
    pos: ℕ = 0
    while pos < n do
        if pos = 0 ∨ A[pos] ≥ A[pos − 1] then
            pos := pos + 1
        else
            SWAP(A[pos], A[pos − 1])
            pos := pos − 1
        end
    end
```

1. Führe GNOMESORT für die Eingabe $A = \langle 4, 2, 3, 7, 6 \rangle$. Gib dabei den Inhalt von A nach jeder SWAP-Operation an. (1 Punkt)
2. Gib die asymptotische Laufzeit von GNOMESORT an und begründe deine Antwort. (2 Punkte)

Lösung 2

1.

i	A nach der i-ten SWAP-Operation
0	$\langle 4, 2, 3, 7, 6 \rangle$
1	$\langle 2, 4, 3, 7, 6 \rangle$
2	$\langle 2, 3, 4, 7, 6 \rangle$
3	$\langle 2, 3, 4, 6, 7 \rangle$

2. Im ungünstigsten Fall ist das Eingabearray absteigend sortiert. Dann werden alle Einträge nacheinander an den Index 0 geswapped. Da SWAP konstante Laufzeit hat, ergibt sich eine Laufzeit von

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2} \in \Theta(n^2)$$

Aufgabe 3 - Verbindungen knüpfen (5 Punkte)

Im folgenden Graphen sollen Kanten eingefügt werden. Diese kannst du entweder einzeichnen oder als Tupel angeben. Deine Kantenmenge soll die folgenden Eigenschaften erfüllen:

1. Zu jedem Knoten auf der linken Seite soll es genau dann eine gerichtete Kante zu einem Knoten auf der rechten Seite geben, wenn die zugehörige Funktion in der jeweiligen Menge enthalten ist.

2. Zu jedem Knoten v auf der rechten Seite soll es genau dann eine gerichtete Kante zu einem anderen Knoten u auf der rechten Seite geben, wenn die Menge zu v eine Untermenge der Menge zu u ist.

A	$\log \log n$	$O(\sqrt{n})$	1
B	$\log_2(4^{2n})$	$\Theta(\log_5(n))$	2
C	$2^{\log \log(n^3)}$	$\Theta(n \log(n))$	3
D	$7^{\log_7(n) \cdot \log_7(n)}$	$\omega(n)$	4
E	$\frac{6n}{\log_n(2)}$	$o(n^2)$	5
F	$\frac{n^3}{2^{6 \log(n)}}$	$\Omega(n)$	6

Lösung 3

Korrekte Kanten:

- (A, 1) (A, 5)
- (B, 5) (B, 6)
- (C, 1) (C, 2) (C, 5)
- (D, 4) (D, 6)
- (E, 3) (E, 4) (E, 5) (E, 6)
- (F, 1) (F, 5)
- (1, 5) (2, 1) (2, 5) (3, 5) (4, 6) (3, 4) (3, 6)

Aufgabe 4 - Master-Theorem (3 Punkte)

Nenne drei Algorithmen, die wir in der Vorlesung kennengelernt haben und deren Laufzeiten sich mit Hilfe des Master-Theorems bestimmen lassen. Nenne für jeden Algorithmus außerdem seine Laufzeit und die zugehörigen Werte von a, b, c aus der Definition des Master-Theorems.

Lösung 4

1. Binäre Suche in $\Theta(\log(n))$: $a = 1, b = 2, c = 0$
2. MERGESORT in $\Theta(n \log(n))$: $a = 2, b = 2, c = 1$
3. Karatsubas Algorithmus in $\Theta(n^{\log_2(3)})$: $a = 3, b = 2, c = 1$

Aufgabe 5 - Rekurrenzen Lösen (6 Punkte)

Löse die folgenden Rekurrenzen, indem du die Kosten der zu ihnen korrespondierenden Rekurrenzbäume bestimmst. Beantworte dazu, wenn möglich, die folgenden Fragen

- Wie viele Knoten sind auf Lage i?
- Wie groß ist das n auf Lage i?
- Wie viel Zeit kostet ein Knoten in Lage i?

um somit die Gesamtkosten für Lage i und hieraus die Gesamtkosten abzuleiten.

1. $$T_1(n) = \begin{cases} 1 & \mid n = 1 \\ 2 \cdot T_1(\frac{n}{4}) + \sqrt{16n} & \mid \text{sonst} \end{cases}$$
2. $$T_2(n) = \begin{cases} 1 & \mid n = 1 \\ 3 \cdot T_2(\frac{n}{3}) + 2n & \mid \text{sonst} \end{cases}$$

3.

$$T_3(n) = \begin{cases} 1 & | \ n = 1 \\ 2 \cdot T_3(\frac{n}{3}) + n\log(n) & | \ \text{sonst} \end{cases}$$

4.

$$T_4(n) = \begin{cases} 1 & | \ n = 1 \\ T_4(\frac{n}{3}) + T_4(\frac{2n}{3}) + n & | \ \text{sonst} \end{cases}$$

Lösung 5

1. Jeder Knoten auf Lage $i-1$ wird in 2 Knoten aufgeteilt, welche je ein Viertel der Problemgröße erhalten. Also:
   - Anzahl Knoten auf Lage i: 2^i
   - Problemgröße auf Lage i: $\frac{n}{4^i}$
   - Kosten pro Knoten mit Problemgröße k auf Lage i: $\sqrt{16k}$

   Damit ergibt sich ein Gesamtaufwand von

$$\begin{aligned} \sum_{i=0}^{\log_4(n)} 2^i \cdot (\sqrt{\frac{16n}{4^i}}) &= \sum_{i=0}^{\log_4(n)} \sqrt{\frac{4^i \cdot 16n}{4^i}} \\ &= \sum_{i=0}^{\log_4(n)} \sqrt{16n} \\ &= 4\sqrt{n}\log_4(n) \end{aligned}$$

   Also ist $T_1(n) \in \Theta(\sqrt{n}\log(n))$.

2. Jeder Knoten auf Lage $i-1$ wird in 3 Knoten aufgeteilt, welche je ein Drittel der Problemgröße erhalten. Also:
   - Anzahl Knoten auf Lage i: 3^i
   - Problemgröße auf Lage i: $\frac{n}{3^i}$
   - Kosten pro Knoten mit Problemgröße k auf Lage i: $2k$

Damit ergibt sich ein Gesamtaufwand von

$$\sum_{i=0}^{\log_3(n)} 3^i \cdot \frac{2n}{3^i} = \sum_{i=0}^{\log_3(n)} 2n$$
$$= 2n \log_3(n)$$

Also ist $T_2(n) \in \Theta(n \log(n))$.

3. Jeder Knoten auf Lage $i-1$ wird in 2 Knoten aufgeteilt, welche je ein Drittel der Problemgröße erhalten. Also:
   - Anzahl Knoten auf Lage i: 2^i
   - Problemgröße auf Lage i: $\frac{n}{3^i}$
   - Kosten pro Knoten mit Problemgröße k auf Lage i: $k \log(k)$

   Damit ergibt sich ein Gesamtaufwand von

$$\sum_{i=0}^{\log_3(n)} 2^i \cdot \frac{n}{3^i} \cdot \log\left(\frac{n}{3^i}\right) = \sum_{i=0}^{\log_3(n)} \left(\frac{2}{9}\right)^i \cdot n \cdot \log\left(\frac{n}{3^i}\right)$$
$$= n \cdot \sum_{i=0}^{\log_3(n)} \left(\frac{2}{9}\right)^i \cdot (\log(n) - \log(3^i))$$
$$= n \cdot \sum_{i=0}^{\log_3(n)} \left(\frac{2}{9}\right)^i \cdot \log(n) - n \cdot \sum_{i=0}^{\log_3(n)} \left(\frac{2}{9}\right)^i \cdot i \cdot \log(3)$$
$$= n \log(n) \cdot \sum_{i=0}^{\log_3 n} \left(\frac{2}{9}\right)^i - \log(3) \cdot n \cdot \sum_{i=0}^{\log_3(n)} i \cdot \left(\frac{2}{9}\right)^i$$

   Da $\sum_{i=0}^{\log_3(n)} \left(\frac{2}{9}\right)^i \in \Theta(1)$ und $\sum_{i=0}^{\log_3 n} i \cdot \left(\frac{2}{9}\right)^i \in \Theta(1)$ nach obigem Hinweis, ist $T_4(n) \in \Theta(n \log(n))$.

4. Jeder Knoten auf Lage $i-1$ wird in 2 Knoten aufgeteilt, welche jedoch unterschiedliche Problemgrößen erhalten. Wir stellen allerdings fest, dass die Problemgrößen aller Knoten auf einer Lage i sich zu maximal n aufsummieren.
   Zudem hat der gesamte Rekurrenzbaum eine Tiefe von $\log_{3/2}(n)$ (Beach-

te: dabei haben nur die Knoten der höheren $\log_3(n)$ Lagen zwei Kinder). In jedem Knoten fallen zusätzliche lineare Kosten an, die Kosten auf einer beliebigen Lage liegen damit in $\Theta(n)$. Also ist $T_4(n) \in \Theta(n \log(n))$.

Aufgabe 6 - Hot Stuff (8 Punkte)

Wir wollen uns in dieser Aufgabe mit Hotlists befassen. Eine Hotlist besteht aus einem sortierten Array $A : [\mathbb{N};\ n]$ und einer unsortierten Liste `hot` : List$\langle\mathbb{N}\rangle$. Diese Datenstruktur bietet die folgenden Operationen an:

- `insert`$(x\colon \mathbb{N})$
  Enthält `hot` weniger als $\sqrt{n}$ Einträge, fügt `insert` x an `hot` an.
  Ansonsten wird `hot` sortiert. Anschließend werden A und `hot` zu einem neuen sortierten Array A' zusammengefügt, A wird verworfen und `hot` $= \langle x \rangle$ gesetzt.

- `has`$(x\colon \mathbb{N})\colon$ Bool
  Diese Operation sucht nach x mittels binärer Suche in A und mittels linearer Suche in `hot`

- `delete`$(x\colon \mathbb{N})$
  Die Anzahl d von `delete`-Aufrufen seit der letzten Zusammenführung von A und `hot` wird gezählt. Ist $d < \sqrt{n}$, so wird x wie bei `has` gesucht und als veraltet markiert. War x nicht in der Hotlist enthalten, geschieht nichts und dieser `delete`-Aufruf wird nicht gezählt.
  Ansonsten wird `hot` sortiert und A und `hot` bei bei `insert` beschrieben zu einem Array zusammengefügt, `hot` wird dabei geleert.

1. Gib die Wort-Case-Laufzeit für jede der drei Operationen an. (2 Punkte)
2. Welche Laufzeiten haben `insert` und `delete` im günstigen Fall? (1 Punkt)
3. Zeige mit einer Methode deiner Wahl, dass alle Operationen eine amortisierte Laufzeit in $O(\sqrt{n})$ haben. (5 Punkte)

Hinweis: Überlege dir, wie oft der günstige Fall einer Operation eintreten kann, bevor der ungünstige eintritt.

Lösung 6

1.
   - `insert`: $\Theta(\sqrt{n}\log(\sqrt{n}) + n + \sqrt{n}) = \Theta(n)$
   - `has`: $\Theta(\log(n) + \sqrt{n}) = \Theta(\sqrt{n})$
   - `delete`: $\Theta(\sqrt{n}\log(\sqrt{n}) + n + \sqrt{n}) = \Theta(n)$
2.
   - `insert`: $\Theta(1)$ (Einfügen in eine Liste)
   - `delete`: $\Theta(\sqrt{n})$ (wie `has`)
3. Wir werden die Operationen getrennt voneinander betrachten. Die Anzahl an Elementen kann sich nur durch `insert`-Operationen erhöhen und nur durch `delete`-Operationen verringern. Außerdem wissen wir, dass vor jeder „teuren“ `delete`-Operation $\sqrt{n}$ „günstige“ erfolgen müssen. Tatsächlich müssen auch vor jeder „teuren“ `insert`-Operation mindestens $\sqrt{n}$ „günstige“ `insert`-Operationen erfolgen, damit `hot` gefüllt wird.
   Für `has` ist nichts zu zeigen. Wir nutzen für `insert` und `delete` die Kontomethode:
   Eine günstige `insert`-Operation hebt ein Token vom Konto ab. Zusätzlich zahlt sie $\sqrt{n} + 1$ Tokens ein. Eine teure `insert`-Operation hebt n Tokens vom Konto ab. Da wie oben beschrieben von einer teuren `insert`-Operation $\sqrt{n}$ günstige erfolgen mussten, sind dann auf dem Konto n Tokens. Also wird der Kontostand durch das Abheben nicht negativ. Insgesamt haben alle `insert`-Operationen amortisierte Kosten in $O(\sqrt{n})$.
   Bei der Analyse von `delete` gehen wir analog vor. Eine günstigen `delete`-Operation zahlt $2\sqrt{n}$ Tokens auf das Konto ein und hebt $\sqrt{n}$ Tokens ab. Durch jede günstige `delete`-Operation erhöht sich also der Kontostand um $\sqrt{n}$. Eine teure `delete`-Operation hebt n Tokens ab. Wie in der Betrachtung von `insert` beschrieben wird der Kontostand dabei nicht negativ. Insgesamt haben alle `delete`-Operationen amortisierte Kosten in $O(\sqrt{n})$.