Karlsruher Institut für Technologie

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

Übungsblatt 04

Algorithmen I - Sommersemester 2022

Abgabe im ILIAS bis 18.05.2022, 14:00 Uhr

Bitte beschrifte Deine Abgabe gut sichtbar mit Deinem Namen und Deiner Matrikelnummer. Achte insbesondere bei handschriftlichen Abgaben auf Lesbarkeit und genügend Platz für Korrektur-Anmerkungen. Die Abgabe erfolgt über das Übungsmodul in der Gruppe Deines Tutoriums im ILIAS. Gib Deine Ausarbeitungen in *einer* PDF-Datei ab.

Aufgabe 1 - Quo vadis? (8 Punkte)

Betrachte die folgenden drei Algorithmen:

```
1:  PRIMUS(A : [ℕ, n], k : ℕ): ℕ
2:      c : ℕ = 0
3:      for i ∈ {0, ..., n − 1} do
4:          for j ∈ {i + 1, ..., n − 1} do
5:              if A[i] + A[j] = k then
6:                  c := c + 1
7:              end
8:          end
9:      end
10:     return c
```

```
1:  SECUNDUS(A : [ℕ; n], ℓ : ℕ = 0, r : ℕ = n − 1)
2:      if ℓ < r then
3:          key : ℕ = A[ℓ]
4:          A[ℓ] = A[r]
5:          A[r] = key
6:          SECUNDUS(A, ℓ + 1, r − 1)
7:      end
```

```
1:  TERTIUS(A : [ℕ; n])
2:      for i ∈ {0, ..., n − 1} do
3:          if A[i] ≠ 0 then
4:              a : ℕ = A[i]
5:              A[i] ≔ 0
6:              b : ℕ = TERTIUS(A)
7:              if a > b ∧ i < n − 1 then
8:                  return b
9:              end
10:             return a
11:         end
12:     end
13:     return 0
```

1. Gegeben sei das folgende Array: $A = \langle 6, 42, 3, 35, 7, 14, 5, 10, 4, 5\rangle$
   Gib für jeden der drei obigen Algorithmen den Inhalt von A sowie falls vorhanden den Rückgabewert an, nachdem der Algorithmus darauf angewendet wurde. Verwende für PRIMUS den Parameter $k = 20$. (3 Punkte)

2. Beschreibe für jeden der drei Algorithmen, was er leistet. Bestimme und begründe seine asymptotische Laufzeit. (5 Punkte)

Lösung 1

1. - PRIMUS:
     $$A = \langle 6, 42, 3, 35, 7, 14, 5, 10, 4, 5\rangle$$
     Rückgabewert: 1
   - SECUNDUS:
     $$A = \langle 5, 4, 10, 5, 14, 7, 35, 3, 42, 6\rangle$$
   - TERTIUS:
     $$A = \langle 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\rangle$$
     Rückgabewert: 3

2. - PRIMUS gibt die Anzahl an Paaren i, j mit $i \neq j$ an, für die gilt, dass $A[i] + A[j] = k$.
     Für den Eintrag an Index $k \in \{0, \ldots, n-1\}$ müssen $n - 1 - k$ Einträge überprüft werden. Damit ergibt sich eine Gesamtlaufzeit in
     $$\Theta\Big(\sum_{i=1}^{n-1} i\Big) = \Theta\Big(\frac{n^2 - n}{2}\Big) = \Theta(n^2)$$
   - SECUNDUS spiegelt sein Eingabearray.
     Die Einträge werden paarweise vertauscht und an jedem Index $i \in \{0, \ldots, n-$

1} wird genau ein Mal geschrieben. Damit liegt die Laufzeit von SECUNDUS in $\Theta(n)$.

- TERTIUS macht jeden Eintrag seines Eingabearrays zu 0 und gibt den minimalen Wert zurück, der zuvor im Array enthalten war
  Im worst-case enthält das Array keine 0. Dann ruft sich TERTIUS n-Mal rekursiv auf. Im i-ten rekursiven Aufruf wird die Schleife i-Mal durchlaufen, bis an Index $i-1$ der erste Eintrag ungleich 0 gefunden wird. Dann wird TERTIUS rekursiv aufgerufen und anschießend die for-Schleife $n-i$ weitere Male durchlaufen. Damit erhalten wir eine Laufzeit in $\Theta(n^2)$.

Aufgabe 2 - SummitSort (6 Punkte)

Wir bezeichnen ein Array $A : [\mathbb{N};\ n]$ als *summit-sortiert*, wenn für jedes Paar von Indizes $i, j \in \{0, \ldots, n-1\}$ folgendes gilt:

- Wenn $i \leq \lfloor \frac{n}{2} \rfloor \wedge j \leq \lfloor \frac{n}{2} \rfloor$, dann ist $A[i] \geq A[j]$, wenn $i \geq j$
- Wenn $i > \lfloor \frac{n}{2} \rfloor \wedge j > \lfloor \frac{n}{2} \rfloor$, dann ist $A[i] \geq A[j]$, wenn $i \leq j$

1. Füge die beiden folgenden summit-sortierten Arrays A_1, A_2 zu einem summit-sortierten Array zusammen und gib das Resultat an. (2 Punkte)

$$A_1 = \langle 1, 9, 24, 12, 9 \rangle, \quad A_2 = \langle 3, 12, 18, 18, 21, 5 \rangle$$

   Entscheide und begründe, ob dein Resultat die einzige gültige Lösung ist.

2. Wir wollen SUMMITSORT mit Hilfe von MERGESORT aus der Vorlesung entwickeln. Der Algorithmus SUMMITSORT soll statt MERGE die Hilfsroutine SUMMITMERGE aufrufen, sonst jedoch identisch zu MERGESORT sein und die gleiche asymptotische Laufzeit haben. Beschreibe SUMMITMERGE und beweise dessen Korrektheit.
   Hinweis: In der Vorlesung wurde die Korrektheit von MERGE mit Hilfe einer Invariante gezeigt. (3 Punkte)

3. Begründe, warum SUMMITSORT mit deinem SUMMITMERGE aus Teilaufgabe 2 die gleiche asymptotische Laufzeit hat wie MERGESORT. (1 Punkt)

Lösung 2

1. Eine mögliche Lösung ist

$$\langle 1, 5, 9, 12, 18, 21, 24, 18, 12, 9, 3 \rangle.$$

   Eine weitere gültige Lösung ist

$$\langle 1, 5, 9, 12, 18, 24, 21, 18, 12, 9, 3 \rangle,$$

   hier sind 21 und 24 vertauscht. Somit ist die Lösung nicht eindeutig.

2. SUMMITMERGE erhält zwei summit-sortierte Arrays $B : [\mathbb{N};\ l], C : [\mathbb{N};\ m]$ und fügt sie zu einem summit-sortierten Array $A : [\mathbb{N};\ l + m]$ wie folgt zusammen: Zunächst stellen wir eine aufsteigende Sortierung in B her. Dazu invertieren wir das Teilarray $\langle B[\lfloor \frac{l}{2} \rfloor + 1], \ldots, B[l-1] \rangle$ (Warum das Ergebnis aufsteigend sortiert ist, wird unten gezeigt). Mit C verfahren wir analog. Nun wenden wir MERGE aus der Vorlesung auf B und C an und erhalten das aufsteigend sortierte Array A' der Länge $l+m$. A entsteht aus A', indem das Teilarray $\langle A'[\lfloor \frac{l+m}{2} \rfloor + 1], \ldots, A'[l+m-1] \rangle$ invertiert wird.
   Wir müssen beweisen, dass A tatsächlich summit-sortiert ist. Dazu können wir nutzen, dass B und C (wenn SUMMITMERGE korrekt ist) bereits summit-sortiert sind. Also wissen wir:
   - $\langle B[0], \ldots, B[\lfloor \frac{l}{2} \rfloor] \rangle$ und $\langle C[0], \ldots, C[\lfloor \frac{m}{2} \rfloor] \rangle$ sind aufsteigend sortiert
   - $\langle B[\lfloor \frac{l}{2} \rfloor + 1], \ldots, B[l-1] \rangle$ und $\langle C[\lfloor \frac{m}{2} \rfloor + 1], \ldots, C[m-1] \rangle$ sind absteigend sortiert

   Darüber hinaus wissen wir, dass B und C ebenfalls durch SUMMITMERGE sortiert wurden und damit $B[\lfloor \frac{l}{2} \rfloor] \leq B[l-1]$ und $C[\lfloor \frac{m}{2} \rfloor] \leq C[m-1]$. Da dies für Arrays der Länge 1 (welche auch immer summit-sortiert sind) gilt, wissen wir auch, dass dies auf der tiefsten Rekursionsebene sichergestellt wird. Also sind B und C nach dem Vertauschen ihrer Elemente tatsächlich aufsteigend sortiert, weswegen auch A' sortiert ist. Damit folgt direkt, dass A summit-sortiert ist.

3. Das Invertieren eines Teilarrays von benötigt für B $\Theta(l)$, für C $\Theta(m)$ Zeit, denn an jeder Eintrag in B bzw. C wird maximal ein Mal neu geschrieben. Wir wissen, dass die Laufzeit von MERGE in $\Theta(A'.\text{size}) = \Theta(m+l)$ liegt. Das Invertieren eines Teilarrays von A' hat ebenfalls einen Zeitbedarf in $\Theta(m+l)$. Somit liegt sich die Gesamtlaufzeit von SUMMITMERGE in $\Theta(m+n)$, ebenso wie die Laufzeit von MERGE. Damit entspricht die asymptotische Laufzeit von SUMMITSORT der von MERGESORT.

Aufgabe 3 - Gesucht, gefunden (5+4 Punkte)

Gegeben sei ein unsortiertes Array $A : [\mathbb{N};\ n]$. Wir wollen das k-kleinste Element in A bestimmen.

1. Beschreibe einen Algorithmus, der für ein ein gegebenes Element $A[i]$ mit $i \in \{0, n-1\}$ den Index bestimmt, an dem $A[i]$ stehen würde, wenn A sortiert wäre. Nenne und begründe die Laufzeit deines Algorithmus. (1 Punkt)

2. Entscheide und begründe, welche Werte die Konstante $b \in \mathbb{R}_+$ annehmen kann, sodass für die folgende Rekurrenz $T(n) \in O(n)$ gilt. (1 Punkt)

$$T(n) = \begin{cases} \Theta(1) & \mid n = 1 \\ T(\frac{n}{b}) + \Theta(n) & \mid \text{sonst} \end{cases}$$

3. Beschreibe einen Algorithmus, welcher in $O(n)$ das k-kleinste Element eines unsortierten Arrays A der Kapazität n bestimmt. Dein Algorithmus darf dabei die Reihenfolge der Einträge in A vertauschen. Du darfst annehmen, dass du den Median eines unsortierten Arrays in $\Theta(n)$ bestimmen kannst. Begründe, warum dein Algorithmus die geforderte Laufzeit hat. (3 Punkte)

∗ Wir nehmen nun nicht mehr an, den Median eines unsortierten Arrays in $\Theta(n)$ bestimmen zu können. Gib einen randomisierten Algorithmus an, welcher das k-kleinste Element von A in erwartet $O(n)$ findet. Begründe die Korrektheit des Algorithmus und der Laufzeit. (4 Punkte)

Hinweis: Teilaufgaben, die nicht mit einer Nummer, sondern mit ∗ versehen sind, sind Zusatzaufgaben. Hier kannst du dir Bonuspunkte verdienen.

Lösung 3

1. Wir können die Anzahl k an Elementen in A bestimmen, die kleiner als $A[i]$ sind, indem wir alle anderen Elemente in A mit $A[i]$ vergleichen. Damit wissen wir, dass $A[i]$ am Index k stehen würde, wäre A sortiert.
   Dieser Algorithmus führt einen Vergleich pro Element in A durch und hat damit eine Laufzeit in $\Theta(n)$.

2. Wir verwenden das Mastertheorem und die in der Vorlesung vorgestellte Notation. Es ist $a = 1$ und $c = 1$. Wir betrachten die drei Fälle des Mastertheorems:
   a) Damit $a < b^c$ gilt muss $b > 1$ sein. Dann gilt $T(n) \in O(n)$.
   b) Gilt $a = b^c = b$, dann wäre $T(n) \in O(n \cdot \log(n))$ und damit nicht in $O(n)$.
   c) Damit $a > b^c$ muss $b < 1$ gelten. Dann würde sich allerdings die Instanzgröße mit jedem rekursiven Aufruf größer werden und damit würde $n = 1$ i.d.R. nicht erreicht. Also kann b nicht kleiner 1 sein.

   Also kann b alle Werte echt größer 1 annehmen.

3. Der Algorithmus erhält als Eingabeparameter das Array A selbst und die Zahl k. Zunächst wird der Median m von A bestimmt und anschließend A mit m als Pivotelement partitioniert. Dazu wird die gleiche Methode verwendet wir bei QUICKSORT. Wurde m durch die Partitionierung an den Index $k - 1$ geschrieben, so ist m das gesuchte Element. Ansonsten vergleichen wir den Index i, an dem m in A eingeordnet wurde, mit k. Gilt $i < k - 1$, so ruft sich der Algorithmus rekursiv auf dem Teilarray rechts von m auf, ansonsten auf dem Teilarray links von m.
   Die Laufzeit dieses Algorithmus liegt in $O(n)$: Wir nehmen an, dass wir den Median eines (Teil-)Arrays in $\Theta(n)$ bestimmen können. Anschließend partitionieren wir ein Array, was ebenfalls in Linearzeit abläuft, da jedes Element des Arrays genau ein Mail mit dem Pivotelement verglichen und maximal ein Mal mit einem anderen vertauscht wird. Somit ergibt sich pro rekursivem Aufruf zusätzliche Arbeit mit linearem Zeitbedarf. Da wir anhand des Medians partitionieren, erfolgt

der nächste Aufruf des Algorithmus mit einem Array von ungefähr halber Größe. Damit ergibt sich die folgende Rekurrenz:

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

Nach Teilaufgabe 2 liegt die Gesamtlaufzeit des Algorithmus damit in $O(n)$.

∗ Wir betrachten den in Teilaufgabe 3 beschriebenen Algorithmus mit der folgenden Anpassung: Statt den Median des übergebenen Arrays zu bestimmen, wählt er ein Pivotelement zufällig aus und partitioniert anhand dessen das Array.
Es bleibt zu zeigen, dass die Laufzeit dieses Algorithmus in erwartet $O(n)$ liegt. Dazu schätzen wir die erwartete Laufzeit in Abhängigkeit von den gewählten Pivots ab. Wir betrachten hierbei immer ein Teilarray $A[k], A[k+1], \ldots, A[k+m-1]$ mit m Elementen, welches partitioniert werden soll. Wie schon bei der Analyse von QUICKSORT unterscheiden wir zwischen guten und schlechten Pivots. Bei ersteren liegen in beiden Teilarrays nach dem Partitionieren maximal $\lfloor \frac{2m}{3} \rfloor$ Elemente. Das bedeutet, dass das Pivot durch das Partitionieren an einem Index i mit $k + \lceil \frac{m}{3} \rceil \leq i \leq k + \lfloor \frac{2m}{3} \rfloor$ stehen muss. Die Wahrscheinlichkeit dafür ist $\frac{1}{3}$. Wurde kein gutes Pivotelement gewählt, so kann im schlechtesten Fall ein Teilarray $m-1$ Elemente erhalten. Zusätzlich werden pro rekursivem Aufruf alle Elemente mit dem Pivot verglichen, was zusätzlichen linearen Zeitaufwand bedeutet. Also gibt es eine Konstante $c \in \mathbb{R}$ so, dass cn eine obere Schranke für den zusätzlichen Aufwand ist. Damit gilt für die erwartete Laufzeit $T(n)$ des Algorithmus:

$$T(n) \leq \frac{1}{3} \cdot T\Big(\Big\lfloor \frac{2n}{3} \Big\rfloor\Big) + \Big(1 - \frac{1}{3}\Big) \cdot T(n) + cn$$

also

$$\frac{1}{3} \cdot T(n) \leq \frac{1}{3} \cdot T\Big(\Big\lfloor \frac{2n}{3} \Big\rfloor\Big) + cn$$

und damit

$$\begin{aligned} T(n) &\leq T\Big(\Big\lfloor \frac{2n}{3} \Big\rfloor\Big) + 3cn \\ &\leq T\Big(\Big\lfloor \frac{4n}{9} \Big\rfloor\Big) + \frac{6cn}{3} + 3cn \\ &\leq T\Big(\Big\lfloor \frac{8n}{27} \Big\rfloor\Big) + \frac{12cn}{9} + \frac{6cn}{3} + 3cn \\ &\leq \ldots \\ &\leq 3cn \sum_{i=0}^{\infty} \Big(\frac{2}{3}\Big)^i \\ &\leq 3cn \cdot 3 = 9cn \in O(n) \end{aligned}$$

Also liegt die Laufzeit des randomisierten Algorithmus in erwartet $O(n)$.