DAP2 – Heimübung 5

Ausgabedatum: 12. 5. 17 — Abgabedatum: Fr. 19. 5. 17 (Mo. 22. 5. für Gruppen 27–32) 12 Uhr

Schreiben Sie unbedingt immer Ihren **vollständigen Namen**, **Ihre Matrikelnummer** und **Ihre Gruppennummer** auf Ihre Abgaben!

Aufgabe 5.1 (5 Punkte): (Teile und Herrsche)

Eine endliche, natürliche Zahlenfolge $a_1, a_2 \ldots, a_n$ heißt *superwachsend*, falls jede Zahl größer als die Summe der vorherigen Zahlen ist, also

$$a_i > \sum_{j=1}^{i-1} a_j$$

für alle i, $2 \leq i \leq n$, gilt.

Wir betrachten nun das folgende Problem: Gegeben sei eine solche superwachsende endliche, natürliche Zahlenfolge und eine natürliche Zahl q, $a_1 \leq q \leq a_n$, für die es eine Teilmenge der Indizes $I \subseteq \{1, 2, \ldots, n\}$ gibt, sodass $\sum_{i \in I} a_i = q$ gilt. Gesucht ist eine Lösung I, für die diese Gleichung gilt. Im Folgenden untersuchen wir zuerst den grösten Index, der in einer Lösung enthalten ist.

Im folgenden Beispiel sei $n = 8$:

$$1, 3, 7, 12, 24, 51, 103, 221.$$

Für $q = 73$ gilt $q = 3 + 7 + 12 + 51$, also ist $I = \{2, 3, 4, 6\}$ eine Lösung und 6 der gröste enthaltene Index.

a) (1 Punkt) Entwerfen Sie einen Teile-und-Herrsche-Algorithmus, der für eine gültige Eingabe den größten Index einer Lösung bestimmt. Beschreiben Sie Ihren Algorithmus zunächst mit eigenen Worten. Die volle Punktzahl erreicht ein Algorithmus, der im schlechtesten Fall mit $\mathcal{O}(\log n)$ Rechenschritten auskommt und immer den richtigen Index ausgibt.

b) (1 Punkt) Geben Sie eine Implementierung Ihres Algorithmus in Pseudocode an.

c) (2 Punkte) Beweisen Sie die Korrektheit Ihres Algorithmus.

d) (1 Punkt) Um nun die gesamte Lösung für eine gültige Eingabe zu bestimmen, geht ein naiver Algorithmus für alle Indizes in absteigender Reihenfolge wie folgt vor. Er vergleicht den Wert a_i an der aktuellen Stelle i mit q: Falls er größer oder gleich ist, wird der aktuelle Index i in die Lösungsmenge aufgenommen und q auf $q - a_i$ gesetzt. Ist es hier möglich, die asymptotische Worst-Case-Laufzeit durch einen Teile-und-Herrsche-Ansatz zu verbessern? Begründen Sie Ihre Antwort.

Lösung:

a) Wir können beobachten, dass eine Lösung immer den größten Index i enthält, sodass $a_i \leq q$ gilt (d. h. $a_{i+1} > q$ oder $i = n$). Es gibt immer einen solchen Index, da $a_1 \leq q \leq a_n$.

Aus $a_{i+1} > q$ und $a_j > a_{i+1}$ für alle $j > i + 1$ folgt, dass alle Indizes in einer Lösungs kleiner oder gleich i ist.

Für alle Indizes kleiner als i gilt, dass

$$\sum_{j<i} a_j < a_i \leq q,$$

da die Folge superwachsend ist. Also reichen diese nicht aus, um die gewünschte Summe zu bilden. D. h., Index i ist selbst immer in einer Lösung enthalten.

Unser Algorithmus sucht also den größten Index, sodass $a_i \leq q$ gilt. Dies können wir mit einer binären Suche umsetzen, die in Laufzeit $O(\log n)$ arbeitet:

Initial wird ein Array der Länge n betrachtet, das an Stelle i den Wert a_i abgespeichert hat. Wir halbieren in jedem rekursiven Aufruf die Länge des betrachteten Teilarrays, indem wir den Wert in der Mitte des Teilarrays mit q vergleichen und, falls er größer ist, in der linken Teilhälfte (exklusive Mitte) weitersuchen, und sonst in der rechten Teilhälfte (inklusive Mitte) weitersuchen. Die Rekursion bricht ab, wenn die Länge 1 erreicht ist, da wir uns dann an der gesuchten Position befinden.

b) In Pseudocode erhalten wir etwa folgenden Algorithmus, wobei der erste Aufruf

$$\texttt{FindeIndex}(A, q, 1, n)$$

ist.

```
FindeIndex(A, q, ℓ, r):
1. if ℓ = r then
2.     return l
3. else
4.     i ← ⌊(ℓ + r + 1)/2⌋
5.     if A[i] > q then
6.         return FindeIndex(A, q, ℓ, i − 1)
7.     else
8.         return FindeIndex(A, q, i, r)
```

c) **Behauptung:** Der Algorithmus `FindeIndex` findet den gesuchten Index korrekt für alle gültigen Eingabearrays der Länge $n \in \mathbb{N}$ und ein gültiges $q \in \mathbb{N}$.

Beweis: Nach den Vorüberlegungen in Aufgabenteil a) reicht es zu zeigen, dass der Algorithmus immer den größten Index i findet mit $a_i \leq q$. Dies zeigen wir mittels vollständiger Induktion über $m = \lceil \log n \rceil$.

Induktionsanfang: Für $m = 0$, also $n = 1$, wird der Algorithmus mit den Parametern $\ell = r = 1$ aufgerufen. Also ist die Bedingung in Zeile 1 erfüllt und es wird in Zeile 2 Index 1 zurückgegeben. Das ist korrekt, da $q = a_1$ nach der Voraussetzung, dass eine Lösung existiert, sein muss.

Induktionsvoraussetzung: Für ein beliebiges $m \in \mathbb{N}$ gelte, dass der Algorithmus ein ein Eingabearray einer Länge n mit $m = \lceil \log n \rceil$ und ein gültiges $q \in \mathbb{N}$ den größten Index i mit $a_i \leq q$ korrekt findet.

Induktionsschritt $m \rightsquigarrow m+1$**:** Für ein n mit $\lceil \log n \rceil = m+1$, also $2^m < n \leq 2^{m+1}$ ist, ist die Bedingung in Zeile 1 nicht erfüllt. Es sei also $i = \lfloor (\ell + r + 1)/2 \rfloor$ der Index des mittleren Feldes im aktuell betrachteten Teilarray der Länge $n = r - \ell + 1$. Falls $A[i] = a_i > q$ ist, muss der Wert an gesuchter Position kleiner als a_i sein, also links von i liegen. In Zeile 6 wird korrekt der Aufruf für diesen Abschnitt durchgeführt. Da die neue Länge $i - 1 - \ell + 1 = \lfloor (\ell + r + 1)/2 \rfloor - \ell = \lfloor n/2 \rfloor$ ist, greift die Induktionsvoraussetzung und der Algorithmus gibt den korrekten Index zurück.

Falls $A[i] = a_i \leq q$ gilt, kann entweder i selbst der gesuchte größte Index sein, oder es gibt noch einen größeren Index mit $a_i \leq q$. Also muss im rechten Teilarray (inklusive i) weitergesucht werden. In Zeile 8 wird auf den korrekten Teilbereich zugegriffen. Dieser hat eine Länge von $r - i + 1 \lceil n/2 \rceil$, also arbeitet der Algorithmus hier ebenfalls korrekt.

d) Um nun eine gesamte Lösung, also alle Indizes in I zu finden, benötigen wir für jeden Index in I mindestens einen Rechenschritt, um ihn korrekt zu identifizieren und zurückzugeben / abzuspeichern. Im Worst Case gibt ist $q = \sum_{i=1}^{n-1} a_i$. Hier benötigt man also immer linear viele Rechenschritte. Die Technik aus den vorherigen Aufgabenteilen hilft uns also nicht weiter.

Bemerkung: Die Entscheidung, ob es für ein beliebiges $q \in \mathbb{N}$ eine Indexmenge wie oben beschrieben gibt, ist ein Spezialfall des Problems SUBSETSUM. Dieses ist für eine nicht notwendigerweise superwachsende Zahlenfolge ein NP-schwer.

Aufgabe 5.2 (5 Punkte): (Teile und Herrsche)

In einem Eiscafé gibt es zwei unterschiedliche Karten für Eis und für Getränke. Alice und Bob wollen die Preise der beiden Karten vergleichen und schreiben dafür zunächst jeder die Preise einer Karte aufsteigend sortiert in zwei Arrays $A[1..n]$ und $B[1..n]$ auf je eine Serviette. Wir nehmen an, dass alle Preise unterschiedlich sind. Nun möchten sie herausfinden, welches das n-günstigste Produkt ist, das man in dem Café bestellen kann (n ist die Länge der Arrays). Zum Beispiel ist für die Arrays $A = [1, 2, 8, 15]$ und $B = [5, 9, 10, 16]$ der Länge $n = 4$ der viertkleinste Wert 8. Sie haben außer ihrer Serviette nichts zu schreiben dabei, sodass sie weder neue Arrays erzeugen noch A und B zusammenfassen können. Im Kopf können sie nur je zwei Elemente $A[i]$ und $B[j]$, $1 \leq i, j \leq n$, vergleichen und keine anderen Operationen ausführen.

a) (2 Punkte) Entwerfen Sie einen Teile-und-Herrsche-Algorithmus, der in Zeit $\mathcal{O}(\log n)$ den Wert des n-kleinsten Elements findet, und beschreiben Sie ihn mit eigenen Worten. Geben Sie eine Implementierung Ihres Algorithmus in Pseudocode an. Achten Sie dabei auf die oben genannten Einschränkungen. Der Vergleich zweier Elemente kostet einen Rechenschritt.

b) (1 Punkte) Analysieren Sie die Laufzeit Ihres Algorithmus. Stellen Sie hierzu eine Rekursionsgleichung für die Laufzeit Ihres Algorithmus auf und lösen Sie diese.

c) (2 Punkte) Zeigen Sie die Korrektheit Ihres Algorithmus.

Hinweis: Sie dürfen sich in dieser Aufgabe auf den Fall beschränken, dass n eine Zweierpotenz ist. Eine allgemeine Lösung wird in der Übung besprochen.

Lösung:

a) Vorüberlegungen: Betrachte zwei Arrays A und B der Länge n. Liegt der n-kleinste Wert beispielsweise in $A[i]$ für einen Index i, $1 \leq i \leq n$, bedeutet das, dass $i - 1$ Werte in A kleiner sind (nämlich $A[1]$ bis $A[i-1]$) und die restlichen $n - i$ kleineren Werte in B liegen. Es gibt also einen Index $j = n - i$, so dass $B[j] < A[i]$, aber $B[j+1] > A[i]$ gelten. Analog gilt die gleiche Überlegung natürlich mit vertauschten Rollen von A und B.

Entsprechend gilt, dass, wenn man einen Index i' gefunden hat, für den $B[n - i'] < A[i']$ gilt, dass der gesuchte n-kleinste Wert entweder in $A[1..i']$ oder in $B[n - i' + 1..n]$ liegt. Wir können uns also das Teile-und-Herrsche-Prinzip zu Nutze machen und auf beiden Arrays parallel eine binäre Suche durchführen.

Um eine logarithmische Laufzeit in der Länge n der beiden Arrays zu garantieren, können wir i im ersten Schritt auf $\lfloor \frac{n}{2} \rfloor$ setzen und entsprechend $j = n - i = \lceil \frac{n}{2} \rceil$, so dass in jedem Schritt die gemeinsame Länge des betrachteten Arrays halbiert wird und jeweils der Teilarray aufgerufen wird, in dem der für entweder A oder B der gesuchte Wert liegt. Die Rekursion bricht ab, sobald beider Arrays den Wert 1 haben. Dann wird der kleinere der beiden Arrays ausgegeben. Warum dies in jedem Fall funktioniert, wird in Aufgabenteil (c) untersucht.

In Pseudocode erhalten wir folgenden Algorithmus.

`nkleinstesElem`(A, B):

1. $n \leftarrow$ `length`(A) ➢ $=$ `length`(B) `voraus.`
2. **if** $n = 1$ **then**
3. **if** $A[1] < B[1]$ **then**
4. **return** $A[1]$
5. **else**
6. **return** $B[1]$
7. **else**
8. $i \leftarrow \lfloor n/2 \rfloor$
9. $j = n - i$
10. **if** $B[j] < A[i]$ **then**
11. **return** `nkleinstesElem`$(A[1..i], B[j+1..n])$
12. **else**
13. **return** `nkleinstesElem`$(A[i+1..n], B[1..j])$

b) Die Worst-Case-Laufzeit $T(n)$ des Algorithmus `nkleinstesElem` bei Eingabelänge n beider Arrays ergibt sich durch folgende Rekursionsgleichung für zwei Konstanten c_1 und c_2.

$$T(n) = \begin{cases} c_1 & n = 1 \\ T(\lceil \frac{n}{2} \rceil) + c_2 & n > 1 \end{cases}$$

Falls $n = 1$ ist, ist die erste If-Bedingung (Zeile 2) wahr, so dass neben der Zuweisung in Zeile 1 ein Vergleich ausgeführt wird. Falls $n > 1$ ist, wird der Else-Fall (Zeile 7) betrachtet, so dass neben der Zuweisung in Zeile 1 zwei Zuweisungen und ein Vergleich durchgeführt werden sowie ein rekursiver Aufruf der Funktion `nkleinstesElem` mit zwei Arrays, deren Länge im schlechtesten Fall aufgerundet halbiert sind.

Für alle Eingaben der Dimension $n \in \mathbb{N}$ mit $k = \lceil \log n \rceil$ hat der Algorithmus eine Laufzeit von $T(n) = c_2 \cdot k + c_1$, also asyptotisch in $\mathcal{O}(\log n)$. Dies kann mit den aus der Vorlesung bekannten Methoden hergeleitet werden.

c) Die Korrektheit des Algorithmus zeigen wir ebenfalls mittels vollständiger Induktion über den aufgerundeten Logarithmus $k = \lceil \log n \rceil$ der gemeinsamen Länge n der beiden Arrays A und B.

Behauptung: Für alle $k \in \mathbb{N}$ gibt der Algorithmus `nkleinstesElem` für alle $n \in \mathbb{N}$ mit $k = \lceil \log n \rceil$ korrekt die n-kleinste Zahl der beiden Arrays aus.

Induktionsanfang: Für $k = 0$, also $n = 1$, bestehen die beiden Arrays A und B jeweils nur aus einem Element. Da in Zeile 2 die Bedingung erfüllt ist, wird in Zeile 4 oder Zeile 6 der kleinere der beiden Werte $A[1]$ und $B[1]$ ausgegeben, also korrekterweise der 1-kleinste Wert.

Induktionsvoraussetzung: Sei $k \in \mathbb{N}$ beliebig. Für alle $n \in \mathbb{N}$ mit $k = \lceil \log n \rceil$ gebe der Algorithmus bei Eingabe zweier Arrays mit gleicher Länge n das n-kleinste Element aus.

Induktionsschritt $k \rightsquigarrow k + 1$**:** Für ein n mit $\lceil \log(n) \rceil = k + 1$, also $2^k < n \leq 2^{k+1}$ ist die Bedingung in Zeile 2 falsch, es werden also in Zeile 8 $i = \lfloor \frac{n}{2} \rfloor$ und in Zeile 9 $j = n - i = \lceil \frac{n}{2} \rceil$ gesetzt.

Falls $B[j] < A[i]$ (Bedingung in Zeile 10), wird in Zeile 11 der Algorithmus rekursiv für $A[1..i]$ und $B[j+1..n]$ aufgerufen. Diese Arrays haben beide eine Länge von $i = \lfloor \frac{n}{2} \rfloor$. Wegen $\lceil \log(\lfloor \frac{n}{2} \rfloor) \rceil = k$, greift die Induktionsvoraussetzung und der Aufruf gibt korrekt die $(\lfloor \frac{n}{2} \rfloor)$-kleinste Zahl zurück.

Falls dieser Wert $A[i']$ für ein $i' \in \{1, \ldots, i\}$ ist, heißt das, dass in $A[1..i]$ also $i' - 1$ kleinere Werte liegen und in $B[j+1..n]$ entsprechend $\lfloor \frac{n}{2} \rfloor - i'$ kleinere Werte als $A[i']$ liegen. Für $A[1..n]$ und $B[1..n]$ bedeutet das, dass es $i' - 1$ und $j + \lfloor \frac{n}{2} \rfloor - i'$ kleinere Werte gibt, also ist $A[i']$ das x-kleinste Element für

$$x = i' - 1 + j + \left\lfloor \frac{n}{2} \right\rfloor - i' + 1 = j + i = n.$$

Das heißt, das gesuchte n-kleinste Element wird zurückgegeben.

Falls umgekehrt, $B[j']$ für ein $j \in \{j+1, \ldots, n\}$ im Rekursionsaufruf zurückgegeben wird, gilt nach Induktionsvoraussetzung, dass es $j' - 1 - j$ kleinere Elemente in $B[j+1..n]$ gibt und $\lfloor \frac{n}{2} \rfloor - (j' - j)$ kleinere Elemente in $A[1..n]$. Also ist $B[j']$ das x-kleinste Element in $A[1..n]$ und $B[1..n]$ für

$$x = j' - 1 - j + j + \left\lfloor \frac{n}{2} \right\rfloor - (j' - j) + 1 = i + j = n,$$

d. h., ebenfalls das gesuchte n-kleinste Element.

Der Else-Fall (Zeile 12) kann durch Vertauschung von A und B und der Länge $\lceil \frac{n}{2} \rceil$ analog argumentiert werden.

Insgesamt funktioniert der Algorithmus also korrekt. □