Max Wisniewski , Alexander Steen

Tutor: Lena Schlipf

Aufgabe 1

Sei $S = \{s_1, s_2, ..., s_n\}$ eine Menge von n paarweise verschiedenen Elementen aus einem totalgeordneten Universum. Seien $w_1, w_2, ..., w_n$ positive Gewichte, so dass das Element s_i Gewicht w_i hat. Es gelte $\sum_{i=1}^{n} w_i = 1$. Gesucht ist der *gewichtete Median* von S.

(a) Angenommen, Sie haben eine Funktion, welche den gewichteten Median in Zeit $T(n)$ bestimmt. Zeigen Sie, wie man den (normalen) Median in Zeit $O(n) + T(n)$ berechnen kann.

Beweis: Als Eingabe für unseren Algorithmus (Median) bekommen wir eine Liste von Werten $(s_1, ..., s_n)$. Diese erweitern wir nun auf die Form für den gewichteten Median. Dazu legen wir eine 2. Liste mit den Werten an (oder erweitern die andere Liste auf Tupel (s_i, w_i), was beides auf das selbe hinaus läuft). Für die Gewichte wählen wir $\forall 1 \leq i \leq n : w_i = \frac{1}{n}$. Auf die neue Strucktur können wir den Algorithmus für den *gewichteten Median* anwenden.

Laufzeit: Wir konstruieren zunächst die neue Liste. Dafür gehen wir einmal drüber und erzeugen Tupel. Das Gewicht können wir in $O(1)$ erstellen und den Tupel auch. Dies tun wir für n Elemente. Je nachdem, ob wir n besitzen oder noch suchen müssen, haben wir als Laufzeit n oder $2n$. Wir benötigen aber für das Erstellen der neuen Eingabe $O(n)$ Schritte. Danach führen wir den gegebenen Algorithmus aus.
$\Rightarrow T_{\text{Median}}(n) = T_{\text{gewichteter Median}}(n) + O(n)$

Median: Bleibt zu zeigen, dass das Ergebnis s_g der Median ist:
Kleinere Elemente, sei n_k die Anzahl der Elemente, die kleiner als s_g sind:

$$\sum_{s_i < s_g} w_i \overset{\text{Gewichte}}{=} \sum_{s_i < s_g} \frac{1}{n} \overset{\text{Def.: } n_k}{=} \frac{n_k}{n} \overset{\text{Def.}}{\leq} \frac{1}{2}$$

$$\Leftrightarrow n_k <= \frac{n}{2}$$

Kleinere Elemente, sei n_g die Anzahl der Elemente, die größer sind als s_g sind:

$$\sum_{s_i > s_g} w_i \overset{\text{Gewichte}}{=} \sum_{s_i > s_g} \frac{1}{n} \overset{\text{Def.: } n_g}{=} \frac{n_k}{n} \overset{\text{Def.}}{<} \frac{1}{2}$$

$$\Leftrightarrow n_g < \frac{n}{2}$$

Da gelten muss, das $n_g + n_k + 1 = n$ erfüllen unsere beiden Ergebnisse die Bedingungen für den normalen Median.

(b) Zeigen Sie, wie man den gewichteten Median in $O(n \cdot \log n)$ Zeit berechnen kann.

Beweis: Wir wählne uns einen geeigneten Sortieralgorithmus. Hier bietet sich Mergesort an. Aus dem letzten Übungszettel wissen wir, dass $T_{\text{Mergesort}}(n) = n \cdot \log n - (n-1)$ ist.
Nachdem wir unseren Liste sortiert haben tun wir folgendes:

Schritt 1: Erzeuge $sum = 0$ Laufsummenvariable mit 0 und indexvariable $i = 0$

Schritt 2: Rechne $sum \leftarrow sum + w_i$.

Schritt 3: Ist $sum <= \frac{1}{2}$ inkrementiere i um eins und mache mit Schritt 2 weiter.

Schritt 4: (Ist dies nicht der Fall) Gebe s_i zurück.

Was wir als erstes sehen, ist dass die Summer der Gewichte, die kleiner als der *gewichtete Median* sind, $\leq \frac{1}{2}$ da wir beim ersten mal, wenn es größer ist aus der Schleife gehen und dort nur den Media drauf gerechnet haben. Da wir beim finden über $\frac{1}{2}$ gekommen sind, wissen wir, dass aus der Bedingung, dass die Gesammtgewichte $= 1$ sind, dass die Gewichte der Elemente, die größer sind $\geq \frac{1}{2}$ sein müssen.

Laufzeit: Die erste Überlegung ist, ob die Schleife abbricht.
Eine einfache Antwort wäre, dass wir alle Gewichte aufsummieren, d.h. wir wissen im n-ten Schritt, dass $sum = 1$ ist. Die Abbruchbedingung wird also spätestens nach n Schritten erfüllt. Da wir aber äußerste Beweisfetischisten sind, weden wir für die Terminierung noch einen formellen Beweis am Ende des Übungsblattes geben.
Für die Laufzeit ergibt sich nun : $T(n) = T_{\text{Mergesort}} + n \cdot (c) + d$, wobei c die konstanten kosten sind, für das aufsummieren der Gewichte und inkrementieren des indexes sind und d die Fixkosten für das anlegen der beiden Variablen sind.
$\Rightarrow T(n) = n \cdot \log n - (n-1) + n \cdot c + d \in O(n \log n)$.

(c) Zeigen Sie, wie man den gewichteten Median in $O(n)$ Zeit finden kann, wenn ein Linearzeitalgorithmus zum Finden des normalen Medians zur Verfügung steht.

Wir geben zunächst eine veränderte Variante des K-SELECT Algorithmuses an. Die Eingaben dafür sind eine Menge S von Tupeln $\{(s_i, w_i)\}$ und ein Wert G, wobei $0 \leq G \leq 1$ gilt. Der Aufruf für den *gewichteten Median* ist dann:
GSELECT(S , 1/2).

```
GSELECT (S , K)
  IF |S| < 100 THEN
    RETURN BRUTEFORCE(S , K)
  (q, w_q) ← MEDIAN (S) // Auf ersten Teil der Elemente
     angewandt.
  S_l ← {s ∈ S|s < q}
  S_g ← {s ∈ S|s > q}
  w_l = Σ_{(s,w)∈S_l} w
  w_g = Σ_{(s,w)∈S_g} w
  IF w_l > K THEN
    RETURN GESLECT (S_l , K)
  ELSE IF w_l = K THEN
    RETURN q
  ELSE
    RETURN GESELCT (S_g, K −w_l − w_q)
```

Bei diesem Algorithmus teilen wir die gegebene Menge in 2 Teile (die aufgrund der Medianeigenschaft gleich groß sind). Danach iterieren wir über die Listen um die

Gesamtgewichte zu berechnen. Können wir feststellen, in welcher Liste der gewichtete Median liegen muss. Ist das Gewicht der kleineren Liste größer als das gesuchte, müssen wir darin weiter suchen. Sollte es schon $\frac{1}{2}$ sein, ist der Median unser gewichtete Median. Sollte es kleiner sein, ist der gewichtete Median im größeren Teil. Wenn wir darin weiter suchen, müssen wir aber das schon berechnete Gewicht des kleineren Teils vom gesuchten Gewicht abziehen.

Laufzeit: Diese Überlegungen führen zu folgender Laufzeit:
Wir nehmen zur Analyse wieder eine Liste mit einer Zweierpotenz als Länge an. Wobei $c, d > 0$ Konstanten sind.

$$T(n) \leq \begin{cases} d & , n < 100 \\ c \cdot n + T(\frac{n}{2}) & , sonst \end{cases}$$

Lösen wir das ganze auf.

$$\begin{aligned} T(n) &\leq T(\tfrac{n}{2}) + cn \\ &\leq T(\tfrac{n}{2^2}) + \tfrac{c}{2}n + \tfrac{c}{1}n \\ &\quad \text{Nach k Schritten} \\ &\leq T(\tfrac{n}{2^k}) + c \cdot n \cdot \sum_{i=0}^{k-1} \left(\tfrac{1}{2}\right)^i \end{aligned}$$

Der erste Term wird nach spätestens nach $\log n$ Schritt den Anker erreichen und damit konstant. Die Reihe konvergiert, da $\sum_{i=0}^{\infty} q^i$ für $0 \leq q < 1$ konvergiert. Im speziellen konvergiert diese Reihe gegen 2.
Es gilt also:

$$T(n) \leq d + 2cn \in O(n)$$

Unser gefundener Algorithmus für den *gewichteten Median* hat also eine lineare Laufzeit.

Aufgabe 2

In der Vorlesung wurden zur Berechnung des BFPRT-Algorithmus die Menge in 5er Gruppen unterteilt.

(a) Was passiert bei 7er Blöcken.

Splitter: Als erstes prüfen wir, ob die Splittereigenschaft noch erfüllt ist (analog zur Untersuchung aus der VL).
Elemente, die kleiner sind:
Wir haben $n/7$ Gruppen aufgerunded. Davon wissen wir wieder, dass die Hälft Dieser Gruppen einen kleineren Median hat als unser gefundener Splitter. In jeder dieser Gruppen sind 4 Elemente kleiner als der jeweilige Median (und durch Transitivität kleiner als der Splitter). Dies gilt für alle Gruppen, bis auf die nicht voll besetzte (da können 3 Fehlen) und die Gruppe des Splitters (in der der Splitter fehlt)
$\Rightarrow n_k = 4 \cdot \left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{7} \right\rceil \right\rceil - 4$ Elemente, die größer sind:
Wir haben hier exakt die selbe Abschätzung, wie oben. $n/7$ Gruppen, von denen die Hälte größer ist, diese besitzen jeweils 4 Elemente, von denen wir die Größe relativ zum Splitter kennen. Davon gehen wieder 4 Elemente ab. Aus der letzten Gruppe und der Splitter selber.
$\Rightarrow n_g = 4 \cdot \left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{7} \right\rceil \right\rceil - 4$

Nun können wir untersuchen, wie viele Elemente wir wegschmeißen können:
Wir schätzen das ganze erstmal nach unten ab:

$$\begin{array}{rrcl} & 4 \cdot \left\lceil \frac{1}{2} \cdot \left\lceil \frac{n}{7} \right\rceil \right\rceil - 4 \geq 4 \cdot \frac{1}{2} \cdot \frac{n}{7} - 4 & \geq & \frac{1}{4}n \\ \Leftrightarrow & \frac{2}{7}n - \frac{1}{4}n & \geq & 4 \\ \Leftrightarrow & \frac{8-7}{28}n & \geq & 4 \\ \Leftrightarrow & n & \geq & 112 \end{array}$$

Beweis: Nun bestimmen wir noch die Laufzeit. Als erstes gilt $\Omega(n)$ als Komplexität des Problems (nicht des Algorithmus). Dies gilt, das wir jedes Element mindestens einmal anfassen müssen, um es zu prüfen. Sollten wir eine echt sublineare Laufzeit erreichen, würden wir einige Elemente nicht betrachten. Wir könnten also einen Fall konstruieren, in dem wir den Median nicht betrachten. Das Ergebnis dieses Algorithmus würde also nicht korrekt sein. $\Rightarrow \Omega(n)$ ist untere Schranke für die Komplexität des Problems.
Für die Beschrenkung nach oben, zeigen wir, dass immer noch gilt $T(n) \in O(n)$
Als Laufzeit können wir ersteinmal, da der Splitter immer noch seine Eigenschaften erfüllt, die Formel aus der VL nehmen:

$$T(n) \leq \begin{cases} O(1) & , n < 120 \\ O(n) + T\left(\left\lceil \frac{n}{7} \right\rceil\right) + T\left(\frac{3}{4}n\right) & , sonst \end{cases}$$

Dies Beweisen wir mit einer Induktion über n.

Behauptung: $\exists \alpha > 0 \ : \ T(n) \leq \alpha \cdot n$

I.A. Für $n < 120$ gilt es, da die Laufzeit konstant ist.

I.S.

$$\begin{array}{rcl} T(n) & \underset{I.V.}{\leq} & O(n) + T\left(\left\lceil \frac{n}{7} \right\rceil\right) + T\left(\frac{3}{4}n\right) \\ & \leq & O(n) + \alpha \left\lceil \frac{n}{7} \right\rceil + \alpha \cdot \frac{3}{4}n \\ & \leq & c \cdot n + \alpha \frac{n}{7} + \alpha + \alpha \frac{3}{4}n \leq \alpha n \end{array}$$

Zeigen wir nun den letzten Schritt:

$$\begin{array}{rrcl} & cn + \alpha \cdot \left(\frac{n}{7} + \frac{3}{4}n + 1\right) & \leq & \alpha n \\ \Leftrightarrow & cn & \leq & \alpha \cdot \left(n - \frac{n}{7} - \frac{3}{4}n - 1\right) \\ \Leftrightarrow & cn & \leq & \alpha \left(\frac{3}{28}n - 1\right) \\ \Leftrightarrow & c \cdot \frac{n}{\frac{3}{28}n - 1} & \leq & \alpha \end{array}$$

Damit muss α größer sein, als eine Konstante mal einen Wert, der einen Grenzwert kleiner unendlich besitzt. Von konvergenten Folgen wissen wir, dass diese ein Maximum besitzen. Wählen wir dieses, können wir unser α einfach größer als (c·max) wählen und damit kann α konstant sein.

(b) Was passiert bei 3er Blöcken.

Splitter: Wir wählen wieder den selben Ansatz zur Analyse des Splitters. Wir haben $\lceil \frac{n}{3} \rceil$ Blöcke und davon sind 0.5 größer (oder kleiner). Dadrin sind jeweils 2 Elemente, von denen wir wissen, dass sie kleiner (größer) als der Splitter sind. Dies gilt wieder für die letzte Gruppe und die Gruppe des Splitters nicht:

$$\begin{aligned} 2 \cdot \lceil \tfrac{n}{2} \lceil \tfrac{n}{3} \rceil \rceil &\geq \tfrac{1}{4}n \\ \tfrac{n}{3} - 2 &\geq \tfrac{1}{4}n \\ \tfrac{1}{12}n &\geq 2 \\ n &\geq 24 \end{aligned}$$

Also haben wir für Listen, die länger als 24 sind, die Splittereigenschaft erfüllt.
Die Laufzeit ist also:

$$T(n) = \begin{cases} d & , n < 30 \\ cn + T\left(\lceil \tfrac{n}{3} \rceil\right) + T\left(\tfrac{3}{4}n\right) & , sonst \end{cases}$$

Annahme: $T(n) \in O(n \cdot \log n)$

Induktion: $\exists \alpha > 0 : T(n) \leq \alpha \cdot n \log n$ (Für das c wählen wir die selbe Konstante α, da wir das Maximum der beiden Nehmen können, falls es existiert)

I.A. : Gilt, da für $n < 24$ konstant.

I.S. :

$$\begin{aligned} T(n) &\underset{I.V.}{=} O(n) + T\left(\lceil \tfrac{n}{3} \rceil\right) + T\left(\tfrac{3}{4}n\right) \\ &\leq \alpha n + \alpha \cdot \lceil \tfrac{n}{3} \rceil \cdot \log \lceil \tfrac{n}{3} \rceil + \alpha \tfrac{3}{4} n \cdot \log(\tfrac{3}{4}n) \\ &= \alpha \cdot \left(n + \lceil \tfrac{n}{3} \rceil \cdot \log \lceil \tfrac{n}{3} \rceil + \tfrac{3}{4} n \cdot \log(\tfrac{3}{4}n)\right) \\ &\leq \alpha \cdot n \log n \end{aligned}$$

Wenn gilt:

$$\left(n + \lceil \tfrac{n}{3} \rceil \cdot \log \lceil \tfrac{n}{3} \rceil + \tfrac{3}{4} n \cdot \log(\tfrac{3}{4}n)\right) \leq n \log n$$

Da wir an dieser Stelle zu lange bräuchten um die Umformung konkret aufzuschreiben, verweisen wir hier auf ein positives Ergebnis von Wolframalpha.
Wir können den Term auch noch abschätzen nach:
$n + 2 \cdot \frac{3}{4} n \cdot \log(\frac{3}{4}n) = \frac{3}{2} n \log \frac{3}{4} n + n = \frac{3}{2} n \log n + n \cdot (\frac{3}{2} \log \frac{3}{4} + 1)$ Für große n kann man sehen, dass es kleiner sein muss $n \log n$.

Annahme: $T(n) \in \Omega(n \log n)$

Induktion: $\exists \alpha > 0 \ : \ T(n) \geq \alpha \cdot n \log n$ (Selber trick mit c wie oben.)

I.A. Für kleine n ist $n \log n$ konstant.

I.S.

$$\begin{aligned} T(n) &\underset{I.V.}{=} O(n) + T\left(\lceil \tfrac{n}{3} \rceil\right) + T\left(\tfrac{3}{4}n\right) \\ &\geq \alpha n + \alpha \cdot \lceil \tfrac{n}{3} \rceil \cdot \log \lceil \tfrac{n}{3} \rceil + \alpha \tfrac{3}{4} n \cdot \log(\tfrac{3}{4}n) \\ &\geq \alpha n + \alpha \cdot \tfrac{n}{3} \cdot \log \tfrac{n}{3} + \alpha \tfrac{3}{4} n \cdot \log(\tfrac{3}{4}n) \\ &= \alpha n \cdot \left(1 + \tfrac{1}{3} \log \tfrac{1}{3} + \tfrac{1}{3} \log n + \tfrac{3}{4} \log \tfrac{3}{4} + \tfrac{3}{4} \log n\right) \\ &\geq \alpha n \cdot \left(\tfrac{4+9}{12} \log n\right) \\ &\geq \alpha n \cdot \log n \end{aligned}$$

Damit haben wir gezeigt, dass $T(n) \in \Omega(n \log n) \ \wedge \ T(n) \in O(n \log n)$ gilt.
$\Rightarrow T(n) \in \Theta(n \log n)$

Aufgabe 3

(a) Für zwei ganzzahlige Vektoren $x = (x_1; x_2; ...; x_n)$ und $y = (y_1; y_2; ...; y_n))$ mit $0 \leq x_i, y_i \leq M$ und einen Wert $u > M$ betrachten wir die Zahlen

$$a = x_1 u^n + x_2 u^{n-1} + ... + x_n u^1$$

und

$$b = y_1 u^n + y_2 u^{n-1} + ... + y_n u^1.$$

Zeigen Sie: (*) $a = b \Leftrightarrow x = y$, (**) $a < b \Leftrightarrow x < y$ (lexikographisch).

Beweis *:

Wir fassen die Vektoren $x, y \in \mathbb{Z}[u]$, also als Vektoren des Polynomrings auf. Damit folgt die Aussage direkt (wie in Mafi 2 gezeigt).

Beweis **:

Wir fassen die Zahlen a, b als u-adische Zahlen auf. Dies können wir machen, da jede "Ziffer" kleiner ist als die Basis.
Bilden wir nun die Differenz :

$$x - y = (x_1 - y_1, x_2 - y_2, x_3 - y_3, x_4 - y_4, ..., x_n - y_n)$$

$$= (x_1 - y_1) \cdot u^n + (x_1 - y_1) \cdot u^{n-1} + ... + (x_n - y_n) = a - b < 0$$

Da wir annehmen können, dass x lexikographisch kleiner ist als y, existiert ein j als kleinster Index, an dem sich beide Zahlen das erste mal unterscheiden. Damit gilt für die Zahl, dass es auch bei $(x_j - y_j) \cdot u^{n-j}$. Aus der Zahldarstellung wissen wir, dass wenn die Zahl, an dieser Stelle positiv ist, wird jede Subtraktion von Stellen darunter die Zahl nicht mehr positiv machen können. Dies gilt auch im Umkehrschluss. Wenn die Differenz kleiner 0 ist, gibt es eine Stelle t an der sich beide Zahlen unterscheiden. dort ist $x_t < y_t$. Da dies wiederum der kleinste Index (größte Stelle nach Zahlendarstellung) ist, ist damit x lexikographisch kleiner als y.

(b) Entwerfen Sie einen Algorithmus, der für zwei gegebene Folgen nichtnegativer Zahlen $x = (x_1; ...; x_m)$ und $y = (y_1; ...; y_n)$ in "linearer" Zeit entscheidet, ob x als Teilfolge $(y_{i+1}, ..., y_{i+m})$ in y vorkommt $(0 \leq i \leq n - m)$. Berechnen Sie die Kosten des Algorithmus im EKM und im LKM.

Algorithmus:

Schritt 1: Falls $n < m$ gib FALSE zurück
Schritt 2: Wähle $u = \max\{x_1, ..., x_m, y_1, ..., y_n\} + 1$
Schritt 3: Berechne $a = x_1 \cdot u^m + x_2 \cdot u^{m-1} + ... + x_m$
Schritt 4: Berechne $b = y_1 \cdot u^m + y_2 \cdot u^{m-1} + ... + y_m$
Schritt 5: Setzte Indexvariable $i = m$
Schritt 6: Falls $a = b$ gib TRUE zurück
Schritt 7: Rechne $b \leftarrow (b \mod u^m) \cdot u + y_{i+1}$
Schritt 8: Setze $i \leftarrow i + 1$
Schritt 9: Ist $i > n - m$ Gib FALSE zurück
Schritt 10: Springe zu Schritt 6

Laufzeit (EKM):

Schritt 1: Ist ein konstanter Vergleich $O(1)$

Schritt 2: Wir iterieren einmal über die Liste $\Rightarrow O(n+m)$

Schritt 3/4: Wir rechnen pro Stelle einmal Plus und 2 mal Mal $\Rightarrow O(m)$

Schritt 5: Konstant $O(1)$

Schleife: Die Schleife wird im Worstcase $n-m$ mal ausgeführt:

Schritt 6: Dauert Konstant viel Zeit $\Rightarrow O(1)$

Schritt 7: Einmal modulo (u^n kann von 3./4. genommen werden), einmal Multiplizieren und eine Addition $\Rightarrow O(1)$

Schritt 8,9,10: Konstant $O(1)$

Damit ist die gesamte Laufzeit
$T(n) = O(1) + O(n+m) + 2 \cdot O(m) + (m-n) \cdot O(1) = O(m+n)$
Nehmen wir an, dass $n >> m$ gilt:
$T(n) = O(n)$

Laufzeit (LKM): Wir schätzen die Komponeten von x und y durch u ab.

Schritt 1: Ist ein Vergleich $O(\log n)$

Schritt 2: Wir iterieren einmal über die Liste $\Rightarrow O((n+m)\log u)$

Schritt 3/4: Wir rechnen pro Stelle einmal Plus und 2 mal Mal $\Rightarrow O(m \log u)$

Schritt 5: $O(\log m)$

Schleife: Die Schleife wird im Worstcase $n-m$ mal ausgeführt:

Schritt 6: $O(m \cdot \log u)$

Schritt 7: Einmal modulo, einmal Multiplizieren und eine Addition $\Rightarrow O(m \log u)$

Schritt 8: Kosten $O(\log n)$ (betrachtet über die ganze Schleife hinweg allerdings $O(1)$)

Schritt 9 $O(\log n)$

Schritt 10: Konstant $O(1)$ (oder auch $\log 6$:D)

Damit ergibt sich eine Gesamtlaufzeit von:
$T(n) = O((n+m)\log u + (n-m)m \log u + \log n)$
Nehmen wir einmal an, dass $n >> m$ gilt:
$T(n) = O(n \log u + \log n)$