Karlsruher Institut für Technologie

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

Zusatzaufgaben 01

Algorithmen I – Sommersemester 2023

Gesamtpunkte: 26

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

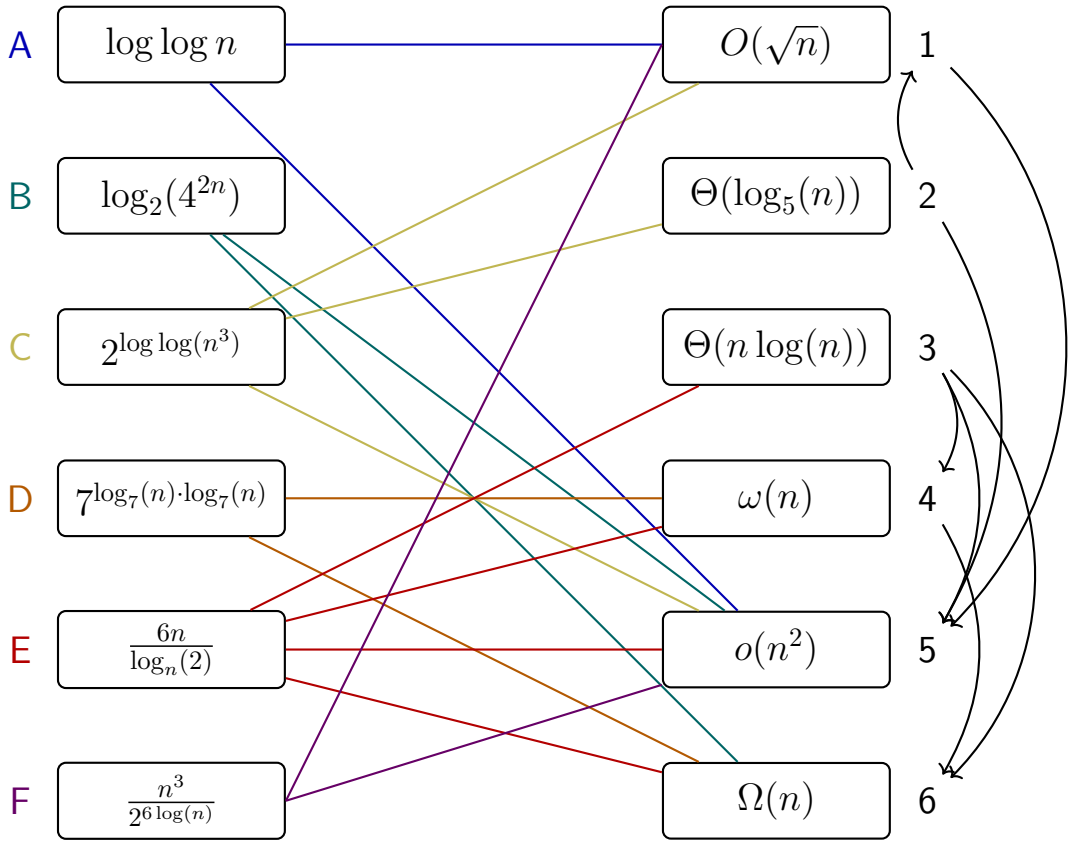
Aufgabe 1 - Verbindungen knüpfen (5 Punkte)

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 1



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 - Dr. Meta räumt auf (5 Punkte)

Da Dr. Meta ein sehr paranoider Superschurke ist, hält er jeden Biberfeind und jeden Biberfreund sehr genau im Auge. Um dies zu bewerkstelligen hat er eigens eine Datenstruktur D angelegt, in der er alle ihm jemals bekannten Biber hält. Die Ordnung in dieser Datenstruktur ist dadurch bestimmt, wie wohlgesonnen ein Biber seinem großen Meisterplan ist. Dem wohlgesonnensten Biber ist hierbei der Wert high zugewiesen und dem rebellischsten Biber der Wert low.
Zusätzlich zu dieser Ordnung werden auch die bekannten Freunde eines jeden Bibers in D gespeichert. Hierfür bietet sein Überwachungsapparat die Funktion SPIONIERE an, welche eine noch nicht bekannte Freundschaft zwischen zwei Bibern in $\Theta(1)$ in D einträgt.
Da nicht jeder Biber dem Plan zuträglich ist kommt es manchmal vor, das Bibern gekündigt wird. Die entsprechende Funktion EXPIRE beendet das Arbeitsverhältnis eines Bibers und entfernt ihn aus D in $\Theta(1)$. Zusätzlich wird, wenn Biber b entlassen wird, jeder mit b befreundete Biber als verdächtig markiert und die Freundschaftsbeziehung aus D entfernt. Dies benötigt Zeit

in $\Theta(\texttt{friends}(b))$, wobei $\texttt{friends}(b)$ die Anzahl der Freunde von b bezeichnet.
Das Einfügen eines Bibers in D lässt sich durch INSERTMIDDLE in $\mathcal{O}(1)$ bewerkstelligen.

1. Zeige mittels einer Methode deiner Wahl, dass EXPIRE eine amortisiert konstante Laufzeit hat. Du darfst dabei annehmen, dass D zu Beginn leer ist. (3 Punkte)

Zusätzlich zum Ausspionieren kann Dr. Metas Überwachungsapparat auch Biber mittels QUESTION befragen. Wird ein Biber befragt, so wird er in D in $\Theta(1)$ als verdächtig markiert. Wird aber ein schon als verdächtig eingetragener Biber befragt, so wird diesem mit EXPIRE gekündigt.

* Zeige, dass QUESTION eine amortisierte Laufzeit in $\Theta(1)$ hat. (2 Punkte)

Lösung 3

1. Wir zeigen zwei mögliche Herangehensweisen:
   - **Kontomethode**:
     Die Kosten von INSERTMIDDLE sowie SPIONIERE liegen bei einem Token, die von EXPIRE bei $\texttt{friends}(b)+1$ Token. SPIONIERE zahlt zusätzlich 1 Token ein.
     INSERTMIDDLE amortisiert sich von selber. Nach k Aufrufen von SPIONIERE liegen $2k - k = k$ Token auf dem Konto. Die $k+1$-te Operation ist EXPIRE. Zu diesem Zeitpunkt wurden k Freundschaften in D eingetragen. Es gilt also $\texttt{friends}(b) \leq k$. Daraus folgt für das Konto: $k + 1 - (\texttt{friends}(b) + 1) = k - \texttt{friends}(b) \geq 0$. Das Konto ist somit nie negativ und EXPIRE liegt amortisiert in $\mathcal{O}(1)$
   - **Potentialmethode:**
     Es sei p die Anzahl an eingetragenen Freundschaften. Als Potentialfunktion wählen wir $\phi(D) = p$.
     SPIONIERE: Da sich die Anzahl an eingetragenen Freundschaften um 1 erhöht hat, ist $1 + \phi(D_i) - \phi(D_{i-1}) = 1 + p + 1 - p = 2$ und somit amortisiert konstant.
     INSERTMIDDLE: Die Anzahl an eingetragenen Freundschaften hat

sich nicht verändert. $1 + \phi(D_i) - \phi(D_{i-1}) = 1 + p - p = 1$ und auch amortisiert konstant.
EXPIRE: Die Anzahl an eingetragenen Freundschaften hat sich um $l \leq p$ reduziert. $l + \phi(D_i) - \phi(D_{i-1}) = l + (p - l) - p = 0$. EXPIRE hat also auch amortisiert konstante Laufzeit.

* Wir verwenden die Kontomethode:
  Die Kosten von INSERTMIDDLE, SPIONIERE und EXPIRE bleiben gleich.
  Für die Kosten von QUESTION müssen wir eine Fallunterscheidung durchführen:
  - Fall 1: Der befragte Biber ist noch nicht verdächtig. QUESTION ändert also nur einen Eintrag mit Kosten von einem Token. Also verändert sich in diesem Fall der Kontostand nicht.
  - Fall2: Der befragte Biber wird schon verdächtigt. QUESTION ruft EXPIRE auf. Wie bei EXPIRE sind vor dem Aufruf von QUESTION k Tokens auf dem Konto. Für den Kontostand nach dem Aufruf gilt $k + 1 - (\texttt{friends}(b) + 1) = k - \texttt{friends}(b) \geq 0$.

  Da in beiden Fällen der Kontostand nie negativ wird und jede Operation nur konstant viele Tokens einzahlt, hat auch QUESTION amortisiert konstante Laufzeit.

Aufgabe 4 - Das Genie beherrscht das Chaos (7 Punkte)

Dr. Meta ist ein sehr kreativer Kopf. Ständig fallen ihm neue Möglichkeiten ein, die Weltherrschaft an sich zu reißen (oder Biber zu koordinieren - ein nie endendes Unterfangen). Um seine ganzen Ideen zu sammeln, hat er sich eine nagelneue Datenstruktur angelegt; doch nicht nur irgendeine Datenstruktur! Auf seine neuesten Einfälle möchte Dr. Meta besonders schnell zugreifen können, denn er hält eine Idee, wenn sie ihm neu einfällt, immer für besonders gut. Als großes Genie hat er natürlich keine schlechten Ideen, doch er weiß auch, dass er schnell den Überblick verlieren kann. Schweren Herzens beschließt er sich deswegen, stets höchstens wurzelig viele Ideen schnell zugreifbar zu machen. Wir bezeichnen die Anzahl der neuesten Ideen mit $n_{\mathsf{new}} \in \mathbb{N}$ und die Anzahl der übrigen Ideen mit $n \in \mathbb{N}$, wobei $n_{\mathsf{new}} \leq \sqrt{n}$ gilt.
Insgesamt kann Dr. Meta folgende Operationen durchführen:

- INSERT(`idea`)
  Fügt die neue Idee `idea` ein. Gilt $n_{\text{new}} < \sqrt{n}$ Einträge, so wird `idea` in die Menge der neuesten Ideen eingefügt, was $\Theta(1)$ Zeit benötigt. Ansonsten muss Dr. Meta aufräumen. Dazu fügt er die neuesten Ideen in die Menge aller anderen Ideen ein, um Platz zu schaffen. Außerdem löscht er alle Ideen, die veraltet sind. Das Aufräumen benötigt Zeit in $\Theta(n + n_{\text{new}})$.
  Beachte: Nach dem Aufräumen gilt $n_{\text{new}} = 0$.

- HAS(`idea`)
  Sucht nach einer Idee. Dazu werden zunächst in $\Theta(n_{\text{new}})$ alle neusten Ideen überprüft. Wurde die Idee dort nicht gefunden, wird wird die Menge der übrigen Ideen in $\Theta(\log n)$ Zeit nach ihr durchsucht.

- DELETE(`idea`)
  Löscht die Idee `idea`. Nunja... fast. Dr. Meta ist faul und markiert erst einmal Ideen nur als veraltet. Wir bezeichnen mit n_{old} die Anzahl der als veraltet markierten Ideen. Ist $n_{\text{old}} < \sqrt{n}$, so wird `idea` wie bei HAS gesucht. Ist `idea` eine der neuesten Ideen, wird sie sofort gelöscht, ansonsten als veraltet markiert; beides benötigt Zeit in $\Theta(1)$.
  Ist $n_{\text{old}} \geq \sqrt{n}$, so ist es wieder dringend Zeit zum Aufräumen. Das passiert wie bei INSERT.
  Beachte: Nach dem Umsortieren gilt $n_{\text{old}} = 0$.

1. Wir betrachten nun den Zustand der Datenstruktur direkt nach einer Aufräumaktion, d.h. es gilt $n_{\text{new}} = 0$. Wie oft hintereinander kann Dr. Meta nun neue Ideen einfügen, bis er wieder aufräumen muss? Wie oft kann er andererseits Ideen löschen, bis er wieder aufräumen muss? (2 Punkte)

2. Dr. Meta fragt sich, wie unaufgeräumt seine Datenstruktur eigentlich werden kann. Beschreibe den Zustand der Datenstruktur, von dem aus möglichst viele Ideen gelöscht werden können, ehe aufgeräumt werden muss. Wie viele Ideen können in Abhängigkeit von n, n_{new} gelöscht werden? (1 Punkt)

3. Zeige mit einer Methode deiner Wahl, dass alle Operationen eine amortisierte Laufzeit in $O(\sqrt{n})$ haben. (4 Punkte)

Lösung 4

1. Da $n_{\text{new}} = 0$ gilt, können $\sqrt{n}$ neue Ideen eingefügt werden, bevor aufgeräumt werden muss. Außerdem können bis zu $\sqrt{n}$ „alte“ Ideen gelöscht werden, bevor $n_{\text{old}} = \sqrt{n}$ und aufgeräumt werden muss.

2. Gilt $n_{\text{new}} = \sqrt{n}$ und $n_{\text{old}} = 0$, so können alle neuen und $\sqrt{n}$ alte Ideen gelöscht werden, bevor aufgeräumt werden muss, also insgesamt $n_{\text{new}} + \sqrt{n} = 2 \cdot \sqrt{n}$.

3. Wir betrachten die Operationen getrennt voneinander. Das ist hier möglich, denn:

   - Die Laufzeit von HAS ist stets in $\Theta(\sqrt{n} + \log(n)) = \Theta(\sqrt{n})$, d.h. insbesondere in $O(\sqrt{n})$. Damit ist für HAS nichts zu zeigen.
   - Ob eine teure DELETE-Operation auftritt, hängt nur von n_{old} ab. Da der Wert von n_{old} nur durch DELETE-Operationen verändert wird, müssen wir in der Analyse von DELETE andere Operationen nicht betrachten.
   - Ob eine teure INSERT-Operation auftritt, hängt nur von n_{new} ab. Da der Wert von n_{new} nur durch INSERT-Operationen erhöht wird, können durch eventuelle DELETE-Opertionen nur mehr günstige INSERT-Operationen auftreten, nicht mehr teure. Wir sehen gleich, warum das in der Analyse unproblematisch ist.

   Auch hier zeigen wir zwei mögliche Lösungswege:

   - **Kontomethode**:
     Günstige INSERT-Operationen heben keine Tokens ab, zahlen aber $\sqrt{n}+1$ Tokens auf das Konto ein. Teure INSERT-Operationen heben $n + \sqrt{n}$ Tokens vom Konto ab. Wie wir in Teilaufgabe 1 festgestellt haben, müssen vor jeder teuren INSERT-Operation mindestens $\sqrt{n}$ günstige INSERT-Operationen erfolgt sein (mehr sind möglich, wenn auch neue Ideen gelöscht wurden). Damit liegen dann mindestens $\sqrt{n} \cdot (\sqrt{n} + 1) = n + \sqrt{n}$ Tokens auf dem Konto und die Kosten für den teuren INSERT-Aufruf können gedeckt werden. Außerdem bleibt der Kontostand stets nichtnegativ.
     Die Analyse von DELETE erfolgt analog zu der von INSERT: günstige

Operationen heben nicht vom Konto ab und zahlen $\sqrt{n}+1$ Tokens ein, teure Operationen heben $n+\sqrt{n}$ Tokens ab. Da vor jeder teuren Operation mindestens $\sqrt{n}$ günstige Operationen erfolgt sein müssen, sind die Kosten für teure Operationen stets gedeckt und der Kontostand wird nie negativ.

- **Potentialmethode**:
  Wir nennen unsere Datenstruktur im Folgenden D. Für die amortisierte Analyse von INSERT wählen wir die Potentialfunktion

  $$\phi_1(D) = n + \sqrt{n} \cdot n_{\text{new}}$$

  Da n und n_{new} stets nichtnegativ sind, gilt das auch für den Wert von ϕ_1. Damit ist ϕ_1 eine gültige Potentialfunktion, mit der sich die folgenden amortisierten Kosten ergeben:

  - INSERT im günstigen Fall:

    $$\begin{aligned} & 1 + (n + \sqrt{n} \cdot (n_{\text{new}} + 1)) - (n + \sqrt{n} \cdot n_{\text{new}}) \\ = \; & 1 + \sqrt{n} \\ \in \; & O(\sqrt{n}) \end{aligned}$$

  - INSERT im teuren Fall:
    Zunächst stellen wir fest: Da markierte Ideen gelöscht werden und neue Ideen eingepflegt werden, gilt

    $$n_{\text{nach}} = n_{\text{vor}} + \sqrt{n_{\text{vor}}} - n_{\text{old}}$$

    Damit bekommen wir:

    $$\begin{aligned} & n + \sqrt{n} + \phi(D_{\text{nach}}) - \phi(D_{\text{vor}}) \\ = \; & n + \sqrt{n} + (n + \sqrt{n} - n_{\text{old}}) - (n + \sqrt{n} \cdot \sqrt{n}) \\ = \; & 2n + 2\sqrt{n} - n_{\text{old}} - 2n \\ = \; & 2\sqrt{n} - n_{\text{old}} \\ \in \; & O(\sqrt{n}) \end{aligned}$$

    Beachte: Es gilt $0 \leq n_{\text{old}} \leq \sqrt{n}$.

  Für die amortisierte Analyse von DELETE wählen wir die Potential-

funktion

$$\phi_2(D) = n + \sqrt{n} \cdot n_{\text{old}}$$

Diese ist nach dem gleichen Argument wie für ϕ_1 eine gültige Potentialfunktion. Mit ihr erhalten wir die folgenden amortisierten Kosten:

- DELETE im günstigen Fall:

$$\begin{aligned} & \sqrt{n} + (n + \sqrt{n} \cdot (n_{\text{old}} + 1)) - (n + \sqrt{n} \cdot n_{\text{old}}) \\ = \; & \sqrt{n} + \sqrt{n} \\ \in \; & O(\sqrt{n}) \end{aligned}$$

- DELETE im teuren Fall:
  Analog zu der Betrachtung von INSERT:

$$\begin{aligned} & n + \sqrt{n} + \phi(D_{\text{nach}}) - \phi(D_{\text{vor}}) \\ = \; & n + \sqrt{n} + (n + n_{\text{new}} - \sqrt{n}) - (n + \sqrt{n} \cdot \sqrt{n}) \\ = \; & 2n + n_{\text{new}} - 2n \\ = \; & n_{\text{new}} \\ \in \; & O(\sqrt{n}) \end{aligned}$$

  Beachte: Es gilt $0 \leq n_{\text{new}} \leq \sqrt{n}$.

Aufgabe 5 - Wieder und wieder und wieder (6 Punkte)

Bestimme die Laufzeit der folgenden rekurrsiven Algorithmen.

Dazu kannst du die vorgestellte Technik verwenden, in dem du die folgenden Fragen für den Rekursionsbaum beantwortest:

- Wie viele Lagen gibt es?
- 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.

$$T_1(n) = \begin{cases} 1 & \mid n = 1 \\ 243 \cdot T_1(\frac{n}{9}) + n^2\sqrt{n} & \mid \text{sonst} \end{cases}$$

2.

$$T_2(n) = \begin{cases} 1 & \mid n = 1 \\ 2 \cdot T_2(\frac{n}{3}) + n\log(n) & \mid \text{sonst} \end{cases}$$

3.

```
1:  ZERKLEINERN(L : List⟨ℕ⟩): ℕ
2:      n : ℕ = L.LENGTH()
3:      if n == 1 then
4:          return 1
5:      end
6:      L_1 : LIST⟨ℕ⟩
7:      L_2 : LIST⟨ℕ⟩
8:      L_3 : LIST⟨ℕ⟩
9:      L_4 : LIST⟨ℕ⟩
10:     i = 0
11:     for x ∈ L do
12:         if 0 ≤ i ≤ ⌊n/2⌋ then
13:             L_1.PUSHBACK(x)
14:         end
15:         if ⌊n/2⌋ ≤ i ≤ n then
16:             L_2.PUSHBACK(x)
17:         end
18:         if ⌊n/4⌋ ≤ i ≤ ⌊3n/4⌋ then
19:             L_3.PUSHBACK(x)
20:         end
21:         if (0 ≤ i ≤ ⌊n/4⌋) ∨ (⌊3n/4⌋ ≤ i ≤ n) then
22:             L_4.PUSHBACK(x)
23:         end
24:         i++
25:     end
26:     a =ZERKLEINERN(L_1)
27:     b =ZERKLEINERN(L_2)
28:     c =ZERKLEINERN(L_3)
29:     d =ZERKLEINERN(L_4)
30:     return a + b + c + d
```

Lösung 5

1. Wir beantworten zuerst die Fragen über den Rekursionsbaum, um damit die Gesamtkosten herzuleiten.

- Insgesamt gibt es $\log_9(n)$ Lagen
- Auf Lage i sind 243^i Knoten
- Auf Lage i sind die Knoten $\frac{n}{9^i}$ groß
- Auf Lage i setzen wir in die Kostenfunktion $f(n) = n^2\sqrt{n}$ die Knotengröße $\frac{n}{9^i}$ ein und erhalten dann Kosten von $(\frac{n}{9^i})^2\sqrt{\frac{n}{9^i}}$

Damit erhalten wir Gesamtkosten von

$$\begin{aligned}\sum_{i=0}^{\log_9(n)} 243^i \cdot (\frac{n}{9^i})^2 \cdot \sqrt{\frac{n}{9^i}} &= \sum_{i=0}^{\log_9(n)} 243^i \cdot \frac{n^2}{9^{2i}} \cdot \frac{\sqrt{n}}{\sqrt{9^i}} \\ &= n^2\sqrt{n} \cdot \sum_{i=0}^{\log_9(n)} \frac{243^i}{81^i \cdot 3^i} \\ &= n^2\sqrt{n} \cdot \sum_{i=0}^{\log_9(n)} 1 \\ &= n^2\sqrt{n}\log_9(n) \in \Theta(n^2\sqrt{n}\log(n))\end{aligned}$$

2. Wir beantworten zuerst die Fragen über den Rekursionsbaum, um damit die Gesamtkosten herzuleiten.
   - Insgesamt gibt es $\log_3(n)$ Lagen
   - Auf Lage i sind 2^i Knoten
   - Auf Lage i sind die Knoten $\frac{n}{3^i}$ groß
   - Auf Lage i setzen wir in die Kostenfunktion $f(n) = n\log(n)$ die Knotengröße $\frac{n}{2^i}$ ein und erhalten dann Kosten von $\frac{n}{3^i}\log(\frac{n}{3^i})$

Damit erhalten wir Gesamtkosten von

$$\begin{aligned}\sum_{i=1}^{\log_3(n)} 2^i \cdot \frac{n}{3^i}\log(\frac{n}{3^i}) &= \sum_{i=1}^{\log_3(n)} n \cdot \frac{2^i}{3^i}(\log(n) - i\log(3)) \\ &= n \cdot \sum_{i=1}^{\log_3(n)} (\frac{2}{3})^i(\log(n) - i\log(3)) \\ &= (n\log(n) \cdot \sum_{i=1}^{\log_3(n)} (\frac{2}{3})^i) - (n\log(3) \cdot \sum_{i=0}^{\log_2(n)} (\frac{2}{3})^i i)\end{aligned}$$

Die linke Summe wird durch den ersten Term dominiert und ist daher in $\Theta(1)$. Die rechte Summe ist durch den Hinweis am Anfang des Blattes ebenfalls in $\Theta(1)$. Daher haben wir $T(n) \in \Theta(n\log(n) - n\log(3)) = \Theta(n\log(n))$.

3. Zuerst leiten wir die Parameter für die Rekursionsgleichung her um damit die Fragen über den Rekursionsbaum zu beantworten.
   - In jedem rekurrsiven Aufruf haben unsere Listen ungefähr die Hälfte der Größe der vorherigen Liste, das heißt unsere Knotengröße halbiert sich.
   - Pro Aufruf von ZERKLEINERN rufen wir die Funktion vier mal rekurrsiv auf
   - Pro Aufruf gehen wir die gegebene Liste einmal durch und haben daher zusätzliche Kosten von $\Theta(n)$.

   Insgesamt erhalten wir dadurch folgende Rekursionsgleichung:

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

   Nun können wir die Fragen über den Rekursionsbaum beantworten und dann die Laufzeit herleiten:
   - Insgesamt gibt es $\log_2(n)$ Lagen
   - Auf Lage i sind 4^i Knoten

- Auf Lage i sind die Knoten $\frac{n}{2^i}$ groß
- Auf Lage i setzen wir in die Kostenfunktion $f(n) = n$ die Knotengröße $\frac{n}{2^i}$ ein und erhalten dann Kosten von $\frac{n}{2^i}$

Damit erhalten wir Gesamtkosten von

$$\begin{aligned}\sum_{i=1}^{\log_2(n)} 4^i \cdot \frac{n}{2^i} &= \sum_{i=1}^{\log_2(n)} (\frac{4}{2})^i n \\ &= n \sum_{i=1}^{\log_2(n)} 2^i \in \Theta(n \cdot 2^{\log_2(n)}) = \Theta(n^2)\end{aligned}$$