Diskrete und Geometrische Algorithmen

1. Übung am 12.10.2020

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

Aufgabe 1. Überlegen Sie sich einen Pseudocode für die folgenden Algorithmen und bestimmen Sie die Anzahl der notwendigen Schritte, die (in Ihrem Pseudocode) nötig sind, um eine n-elementige Menge zu sortieren. Wenden Sie die Algorithmen auf den Datensatz $6, 77, 45, 103, 4, 17$ an.

(a) Selection Sort: der Algorithmus sucht zunächst das kleinste Element und bringt es an die erste Position. Anschließend sucht er das zweitkleinste Element und bringtes an die zweite Position, usw.

(b) Bubble-Sort: Der Algorithmus vergleicht der Reihe nach je zwei benachbarte Zahlen und vertauscht diese, falls sie nicht in der richtigen Reihenfolge angeordnet sind. Dieses Verfahren wird so lange wiederholt, bis alle Zahlen der Eingabe sortiert sind.

Lösung.

(a)

```
1 : Prozedur SELECTION-SORT(A)
2 :   n := A.Länge
3 :   Für i := 1, ..., n − 1
4 :     i_min := i
5 :     Für j := i + 1, ..., n
6 :       Wenn A[j] < A[i_min]
7 :         i_min := j
8 :       Ende Wenn
9 :     Ende Für
10 :    A[i_min], A[i] := A[i], A[i_min]
11 :  Ende Für
12 : Ende Prozedur
```

$Z := \{2, \dots, 7, 10\}$ ist die Menge der Zeilen, die beim Algorithmus wesentliche Operationen ausführen. Wir können also voraussetzen dass für $z \in Z$ eine Konstante c_z existiert, welche die Zeit angibt, die zur Ausführung von der z-ten Zeile benötigt wird. Sei A das Eingabedatenfeld und n die Länge von A. Dann belaufen sich die Kosten der Anwendung von Selection-Sort auf das Datenfeld A auf

$$T(A) = c_2 + \sum_{i=1}^{n-1} \left(c_3 + c_4 + \sum_{j=i+1}^{n} (c_5 + c_6 + c_7 \cdot t_{ij}) + c_{10} \right),$$

wobei t_{ij} mit 1 oder 0 angibt, ob die 7-te Zeile ausgeführt wird oder nicht. Die t_{ij} hängen offensichtlich von A ab. Um die Anzahl der Schritte zu bekommen, setzen wir für alle $z \in Z$ einfach $c_z := 1$.

$$S(A) = 1 + \sum_{i=1}^{n-1} \left(3 + \sum_{j=i+1}^{n} (2 + t_{ij}) \right)$$

Im worst-case gilt die Bedingung in der 6-ten Zeile immer und

$$\forall\, i = 1, \ldots, n-1 : \forall\, j = i+1, \ldots, n : t_{ij} = 1.$$

Wir setzen dies in $S(A)$ ein und erhalten

$$\begin{aligned} S_{\mathrm{s}}(A) = 1 + \sum_{i=1}^{n-1} \left(3 + \sum_{j=i+1}^{n} 3 \right) &= 1 + \sum_{i=1}^{n-1} (3 + 3(n-i)) = 1 + \sum_{i=1}^{n-1} 3 + \sum_{i=1}^{n-1} 3n - \sum_{i=1}^{n-1} 3i \\ &= 1 + 3(n-1) + 3n(n-1) - 3\frac{(n-1)n}{2} = \frac{3n^2}{2} + \frac{3n}{2} - 2 = \mathcal{O}\big(n^2\big). \end{aligned}$$

Im best-case ist die Liste aufsteigend sortiert. Die Bedingung in der 6-ten Zeile hält also nie und

$$\forall\, i = 1, \ldots, n-1 : \forall\, j = i+1, \ldots, n : t_{ij} = 0.$$

Wir setzen dies in $S(A)$ ein und erhalten

$$\begin{aligned} S_{\mathrm{b}}(A) = 1 + \sum_{i=1}^{n-1} \left(3 + \sum_{j=i+1}^{n} 2 \right) &= 1 + \sum_{i=1}^{n-1} (3 + 2(n-i)) = 1 + \sum_{i=1}^{n-1} 3 + \sum_{i=1}^{n-1} 2n - \sum_{i=1}^{n-1} 2i \\ &= 1 + 3(n-1) + 2n(n-1) - 2\frac{(n-1)n}{2} = n^2 + 2n - 2 = \mathcal{O}\big(n^2\big). \end{aligned}$$

(b)

```
1 : Prozedur BUBBLE-SORT(A)
2 :   n := A.Länge
3 :   Für i := 1, ..., n
4 :     Für j := 1, ..., n − i
5 :       Wenn A[j + 1] < A[j]
6 :         A[j], A[j + 1] := A[j + 1], A[j]
7 :       Ende Wenn
8 :     Ende Für
9 :   Ende Für
10 : Ende Prozedur
```

$$\implies T(A) = c_2 + c_3 + \sum_{i=1}^{n} \left(c_3 + c_4 + \sum_{j=1}^{n-i} (c_4 + c_5 + c_6 \cdot t_{ij}) \right)$$

$$\implies S(A) = 2 + \sum_{i=1}^{n} \left(2 + \sum_{j=1}^{n-i} (2 + t_{ij}) \right)$$

$$\implies S_{\mathrm{s}}(A) = 2 + \sum_{i=1}^{n} \left(2 + \sum_{j=1}^{n-i} (2+1) \right) = 2 + \sum_{i=1}^{n} (2 + 3(n-i)) = 2 + \sum_{i=1}^{n} 2 + \sum_{i=1}^{n} 3n - \sum_{i=1}^{n} 3i$$
$$= 2 + 2n + 3n^2 - \frac{3n(n+1)}{2} = \frac{3n^2}{2} + \frac{n}{2} + 2 = \mathcal{O}\big(n^2\big)$$

$$\implies S_{\mathrm{b}}(A) = 2 + \sum_{i=1}^{n} \left(2 + \sum_{j=1}^{n-i} (2+0) \right) = 2 + \sum_{i=1}^{n} (2 + 2(n-i)) = 2 + \sum_{i=1}^{n} 2 + \sum_{i=1}^{n} 2n - \sum_{i=1}^{n} 2i$$
$$= 2 + 2n + 2n^2 - n(n+1) = n^2 + n + 2 = \mathcal{O}\big(n^2\big)$$

Aufgabe 2. Sequentielle Suche: Gegeben sei ein n-elementiger Datenfeld $A[1, \ldots, n]$ und ein Wert x. Überlegen Sie sich einen Pseudocode, der x durch sukzessive Vergleiche mit den Elementen $A[1], A[2], \ldots$ sucht und einen Wert $j \in \{1, \ldots n\}$ ausgibt, falls $x = A[j]$, oder NIL ausgibt, falls x nicht in der Liste A enthalten ist.

(a) Beweisen Sie, dass Ihr Algorithmus ein korrektes Ergebnis liefert (etwa mit Zuhilfenahme einer Schleifeninvariante).

(b) Machen Sie für diesen Algorithmus eine best-case-Analyse, eine worst-case-Analyse und eine average-case-Analyse (für die average-case-Analyse soll das Modell der Zufallspermutationen verwendet werden, d.h. alle Permutationen von $\{1, \ldots, n\}$ sind gleich wahrscheinlich. Weiters soll angenommen werden, dass x im Datensatz enthalten ist).

Lösung.

```
1 : Prozedur SEQUENTIELLE SUCHE(A, x)
2 :   n := A.Länge
3 :   i := NIL
4 :   Für j := 1, ..., n
5 :     Wenn x = A[j]
6 :       i := j
7 :       Abbruch
8 :     Ende Wenn
9 :   Ende Für
10 : Ende Prozedur
```

(a) Bezeichne $\mathtt{n}$ die Länge eines Datenfelds $\mathtt{A}$, $\mathtt{i}$ den (potentiellen) kleinsten Index von einer Variable $\mathtt{x}$ (vielleicht) aus $\mathtt{A}$, und $\mathtt{j}$ die laufende Variable der Schleife der 4-ten Zeile. Unsere Schleifeninvariante lautet wie folgt.

$$I(\mathtt{A}, \mathtt{x}, \mathtt{i}, \mathtt{j}) : \mathtt{j} \leq \mathtt{n} \wedge ((\mathtt{i} = \text{NIL} \wedge \forall\, \mathtt{k} < \mathtt{j} : \mathtt{x} \neq \mathtt{A}[\mathtt{k}]) \vee \mathtt{x} = \mathtt{A}[\mathtt{i}])$$

1. Zu Beginn des ersten Durchlaufs der Schleife ist $\mathtt{i} = \text{NIL}$ und $\mathtt{j} = 1$. $I(\mathtt{A}, \mathtt{x}, \mathtt{i}, \mathtt{j})$ ist hier also wahr.
2. Es gelte $I(\mathtt{A}, \mathtt{x}, \mathtt{i}, \mathtt{j})$. Falls $\mathtt{x} = \mathtt{A}[\mathtt{i}]$, dann hat der Algorithmus bereits terminiert. Weil die Schleife in der 4-ten Zeile nach oben zählt, ist $\mathtt{j}' = \mathtt{j} + 1$. Falls $\mathtt{j} = \mathtt{n}$, dann terminiert der Algorithmus, weil dann $\mathtt{j}' > \mathtt{n}$. Ansonsten gelten $\mathtt{j}' \leq \mathtt{n}$, $\mathtt{i} = \text{NIL}$, und $\forall\, \mathtt{k} < \mathtt{j} : \mathtt{x} \neq \mathtt{A}[\mathtt{k}]$, und die Schleife in der 4-ten Zeile wird ein weiteres mal instanziert. Offensichtlich gilt $\mathtt{A}' = \mathtt{A}$ sowie $\mathtt{x}' = \mathtt{x}$ weil sich $\mathtt{A}$ und $\mathtt{x}$ nie ändern. Für den Wert von $\mathtt{i}'$ machen wir eine Fallunterscheidungs.
   I. Fall ($\mathtt{x}' = \mathtt{A}'[\mathtt{j}']$): Dann wird die 6-te Zeile ausgeführt und $\mathtt{i}' = \mathtt{j}'$. Anschließend terminiert der Algorithmus.
   II. Fall ($\mathtt{x}' \neq \mathtt{A}'[\mathtt{j}']$): Dann wird die 6-te Zeile nicht ausgeführt und es bleibt $\mathtt{i}' = \mathtt{i}$. Wegen $I(\mathtt{A}, \mathtt{x}, \mathtt{i}, \mathtt{j})$ gilt $\forall\, \mathtt{k} < \mathtt{j} : \mathtt{x} \neq \mathtt{A}[\mathtt{k}]$ und wegen der Fallunterscheidungsbedingung gilt $\mathtt{x}' \neq \mathtt{A}'[\mathtt{j}']$. Insgesamt folgt daher $\forall\, \mathtt{k} < \mathtt{j}' : \mathtt{x} \neq \mathtt{A}[\mathtt{k}]$.

   Also gilt auch $I(\mathtt{A}', \mathtt{x}', \mathtt{i}', \mathtt{j}')$.
3. Offensichtlich ist $\mathtt{i}$ für $\mathtt{j} = \mathtt{n}$ das gewünschte Ergebnis.

(b) Sei A ein Datenfeld der Länge n und $x \in A$.

1. Worst-Case:
   x taucht erst im letzten Eintrag von A auf, d.h. $A[n] = x$.

$$\implies T_{\mathrm{s}}(A, x) = c_2 + c_3 + (c_4 + c_5)n + c_6 = \mathcal{O}(n)$$

2. Average-Case:
   Nachdem im Skriptum vorausgesetzt wird (ohne darauf hinzuweisen), dass die Einträge von A verschieden sind und damit o.B.d.A $\{1, \ldots, n\}$, werden wir das hier auch tun. Ansonsten müsste man die Wahrscheinlichkeit berechnen, dass $i = 1, \ldots, n$ der kleinste Index ist, sodass $A[i] = x$.

$$W(A[i] = x \wedge \forall\, j = 1, \ldots, i-1 : A[j] \neq x \mid \exists\, j = 1, \ldots, n : A[j] = x)$$

   Für $\pi \in S_n$ und $\pi(i) = x$ mit $i = 1, \ldots, n$, muss die Wenn-Bedingung in der 5-ten Zeile genau i-Mal überprüft werden. Weil alle Permutationen aus S_n gleich wahrscheinlich sind, erhalten wir für die durchschnittliche Anzahl der Wenn-Bedingungs-Überprüfungen

$$\overline{t} = \sum_{i=1}^{n} \frac{i}{n} = \frac{1}{n} \frac{n(n+1)}{2} = \frac{n+1}{2}.$$

$$\implies T_{\mathrm{d}}(A, x) = c_2 + c_3 + (c_4 + c_5)\frac{n+1}{2} + c_6 = \mathcal{O}(n)$$

3. Best-Case:
   x taucht erst im ersten Eintrag von A auf, d.h. $A[1] = x$.

$$T_{\mathrm{b}}(A, x) = c_2 + c_3 + c_4 + c_5 + c_6 = \mathcal{O}(1)$$

Lösung.

1 : **Prozedur** SEQUENTIELLE SUCHE(A, x)
2 : $j := 1$
3 : **Solange** $j \leq A.Länge$ *und* $A[j] \neq x$
4 : $j := j + 1$
5 : **Ende Solange**
6 : **Wenn** $j = A.Länge + 1$
7 : $j := NIL$
8 : **Ende Wenn**
9 : **Ende Prozedur**

Wir wollen nun einen Korrektheitsbeweis führen. Dafür bezeichnen wir mit k den Schleifendurchlauf und definieren die Schleifeninvariante $I(j) : \forall k < j : A[k] \neq x$. Nun gilt

(1) Zu Beginn der Schleife ist $j = 1$ also gibt es gar kein $k \in \mathbb{N}$ das $k < j$ erfüllt, damit ist $I(j)$ erfüllt.

(2) Sei nun $I(j)$ erfüllt. Da die Schleife durchlaufen wird gilt $A[j] \neq x$. Insgesamt zu Beginn des $n+1$-ten Schleifendurchlaufs also aus der Voraussetzung und dem Argument eben $\forall k < j + 1 : A[k] \neq x$.

(3) Bei Beendigung der Schleife ist $j = A.Länge + 1$ oder $A[j] = x$. Im ersten Fall gilt $\forall k < A.Länge + 1 : A[k] \neq x$ also ist x nicht in A enthalten, der Algorithmus endet mit NIL. Sonst gibt der Algorithmus klarerweise ein j aus, für das $A[j] = x$ gilt.

Nun wollen wir noch den Aufwand des Algorithmus analysieren. Sei dazu t die Anzahl der Ausführungen des Schleifenkopfes und s die Anzahl der Ausführungen von Zeile 7. Allgemein gilt nun

$$T(A, x) = (c_3 + c_5)t + c_4(t - 1) + c_7 s + c_2 + c_6 + c_8$$

für den Aufwand.

(1) Im besten Fall ist $A[1] = x$, folglich $s = 0$ und $t = 1$. Damit ergibt sich $T(A, x) = c_2 + c_3 + c_5 + c_6 + c_8 = \mathcal{O}(1)$.

(2) Im schlechtesten Fall ist x nicht in der Liste A enthalten und es ist $s = 1$ und $t = n + 1$. Es ergibt sich ein Aufwand von $T(A, x) = (c_3 + c_5)(n + 1) + nc_4 + c_2 + c_6 + c_7 + c_8 = \mathcal{O}(n)$.

(3) Zur Analyse des durchschnittlichen Falls nehmen wir an, dass x in der Liste A enthalten ist und an jeder Stelle mit der gleichen Wahrscheinlichkeit, nämlich $\frac{1}{n}$ vorzufinden ist. Mit Sicherheit ist also $s = 0$ und wir berechnen

$$\bar{t} = \sum_{i=1}^{n} \frac{i}{n} = \frac{n+1}{2}.$$

und erhalten mit $t = \bar{t}$ den Aufwand $T(A, x) = (c_3 + c_5)\frac{n+1}{2} + c_4\frac{n-1}{2} + c_2 + c_6 + c_8 = \mathcal{O}(n)$.

Aufgabe 3.

(a) Binäre Suche: Gegeben sei ein (aufsteigend) sortiertes Datenfeld $A[1, \ldots, n]$ und ein Wert x. Das sogenannte Suchproblem, also einen Index j mit $x = A[j]$ auszugeben, falls x in A enthalten ist, und einen speziellen Wert NIL auszugeben, falls x nicht in A vorkommt, kann hier mittels Divide-and-Conquer gelöst werden. Man vergleicht x mit dem mittleren Element des Datenfelds und ist nach diesem Vergleichen entweder fündig geworden oder braucht nur noch das halbe Datenfeld mit der gleichen Prozedur zu durchsuchen. Schreiben Sie ein Programm in Pseudocode für die binäre Suche. Begründen Sie, warum die Laufzeit der binären Suche im schlechtesten Fall $\mathcal{O}(\log n)$ ist.

(b) Beim Algorithmus Einfügesortieren wird die sequentielle Suche verwendet, um das bereits sortierte Teilfeld $A[1, \ldots, n]$ (rückwärts) zu durchsuchen. Kann stattdessen die binäre Suche verwendet werden, um die worst-case-Laufzeit von Insertion Sort auf $\mathcal{O}(\log n)$ zu verbessern?

Lösung.

(a)

```
1 : Prozedur DIVIDE-AND-CONQUER-SUCHE(A, x)
2 :   a := 1
3 :   b := A.Länge
4 :   j := ⌊(b − a)/2⌋
5 :   Solange b − a > 0 und A[a + j] ≠ x
6 :     Wenn A[a + j] < x
7 :       a := a + j + 1
8 :     Sonst
9 :       b := a + j − 1
10 :    Ende Wenn
11 :    j := ⌊(b − a)/2⌋
12 :  Ende Solange
13 :  Wenn j = 0
14 :    j := NIL
15 :  Sonst
16 :    j := j + a
17 :  Ende Wenn
18 : Ende Prozedur
```

In jedem Schritt des Solange-Block wird $(b - a)$ mindestens halbiert (oder die Schleife bricht ab):
Fall 1: $A[a + j] < x$

$$b' - a' = b - a - j - 1 = b - a - \lfloor (b - a)/2 \rfloor - 1 \leq \frac{b - a}{2}$$

Fall 2: $A[a + j] > x$

$$b' - a' = a + j - 1 - a = j - 1 \leq \frac{b - a}{2}$$

Fall 3: $A[a + j] = x$
Die Solange-Bedingung ist nicht mehr erfüllt und wir haben den letzten Schleifendurchlauf erreicht.
Insgesamt ist nach $\lceil \log_2(n) \rceil$ in jedem Fall die Solange-Bedingung verletzt und somit wird der Solange-Block maximal $\lceil \log_2(n) \rceil$ oft ausgeführt, was uns insgesamt auf einen Aufwand von $\mathcal{O}(n)$ führt.

```
1 : Prozedur DIVIDE-AND-CONQUER-SUCHE(A, x)
2 :   a := 0
3 :   b := A.Länge + 1
4 :   j := ⌊(b − a)/2⌋
5 :   Solange b − a > 0 und A[a + j] ≠ x
6 :     Wenn A[a + j] < x
7 :       a := a + j + 1
8 :     Sonst
9 :       b := a + j
10 :    Ende Wenn
11 :    j := ⌊(b − a)/2⌋
12 :  Ende Solange
13 :  Wenn j = 0
14 :    j := NIL
15 :  Ende Wenn
16 : Ende Prozedur
```

Es ist nicht ganz klar, was in der Angabe mit begründen gemeint ist. Ein sauberer Beweis oder die Bemerkung, dass die Länge des Datenfeldes sich stets halbiert, die Schleife also sicher nicht öfter als $\lceil \log_2(n) \rceil$ Mal ausgeführt wird?

(b)

Algorithmus 2 Einfügesortieren

```
1:  Prozedur EINFÜGESORTIEREN(A)
2:    Für j := 2, ..., A.Länge
3:      x := A[j]
4:      i := j − 1
5:      Solange i ≥ 1 und A[i] > x
6:        A[i + 1] := A[i]
7:        i := i − 1
8:      Ende Solange
9:      A[i + 1] := x
10:   Ende Für
11: Ende Prozedur
```

Beim Algorithmus Einfügesortieren wird die äußere Schleife stets $(n-1)$-Mal durchlaufen. Ersetzt man die innere Schleife durch ein ähnliches Verfahren wie die Prozedur Divide-and-Conquer-Suche, so wird im schlechtesten Fall die innere Schleife im j-ten Durchlauf der äußeren Schleife $\lceil \log_2 j \rceil$-Mal durchlaufen. Es ergibt sich so folgender Aufwand.

$$\sum_{j=2}^{n} \lceil \log_2 j \rceil \geq \sum_{j=2}^{n} \log_2 j \geq \int_1^n \log_2 x \, \mathrm{d}x = \frac{1}{\log 2} \int_1^n \log x \, \mathrm{d}x$$
$$= \frac{1}{\log 2}((n \log n - n) - (1 \log 1 - 1)) = \frac{1}{\log 2}(n(\log n - 1) + 1) = \mathcal{O}(n \log n) > \mathcal{O}(\log n)$$

Die worst-case-Laufzeit von Insertion Sort lässt sich daher nicht einmal durch anwenden der binären Suche auf $\mathcal{O}(\log n)$ verbessern.

Aufgabe 4. Zeigen Sie, dass die harmonischen Zahlen $H_n = \sum_{k=1}^{n} \frac{1}{k}$ die Abschätzung $H_n = \mathcal{O}(\log n)$ gilt, indem Sie

1. die Summe durch $N = \lfloor \log_2 n \rfloor$ Blöcke der Gestalt $\sum_{j=0}^{2^i-1} \frac{1}{2^i+j}$ (wobei $i = 1, \ldots, N$) abschätzen und anhand dieser Aufteilung $\sum_{k=1}^{n} \frac{1}{k} \leq 1 + \log_2 n$ verifizieren.
2. das Cauchy'sche Integralkriterium verwenden.

Lösung.

1.

$$\{1, \ldots, n\} = \bigcup_{i=0}^{\lfloor \log_2 n \rfloor - 1} \left\{2^i, \ldots, 2^{i+1} - 1\right\} + \left\{2^{\lfloor \log_2 n \rfloor}, \ldots, 2^{\log_2 n}\right\}$$
$$= \bigcup_{i=0}^{\lfloor \log_2 n \rfloor - 1} \left\{2^i + 0, \ldots, 2^i + (2^i - 1)\right\} + \left\{2^{\lfloor \log_2 n \rfloor}, \ldots, n\right\} \subseteq \bigcup_{i=0}^{\lfloor \log_2 n \rfloor} \left\{2^i, \ldots, 2^i + (2^i - 1)\right\}$$

$$\implies H_n = \sum_{k=1}^{n} \frac{1}{k} = \sum_{i=0}^{\lfloor \log_2 n \rfloor - 1} \left(\sum_{j=0}^{2^i-1} \frac{1}{2^i + j} \right) + \sum_{k=2^{\lfloor \log_2 n \rfloor}}^{n} \frac{1}{k} \leq \sum_{i=0}^{\lfloor \log_2 n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i + j} \leq \sum_{i=0}^{\lfloor \log_2 n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i}$$
$$= \sum_{i=0}^{\lfloor \log_2 n \rfloor} \frac{2^i}{2^i} = 1 + \lfloor \log_2 n \rfloor = 1 + \left\lfloor \frac{\log n}{\log 2} \right\rfloor \leq \mathcal{O}(\log n)$$

2. Definiere $f(x) = \frac{1}{x}$. Betrachte die Zerlegung $\mathcal{Z}_n = \{1, 2, 3, \ldots, n\}$ des Intervalls $[1, n]$. Da f monoton fallend ist, ist $\sum_{k=2}^{n} \frac{1}{k}$ die zugehörige Untersumme von f zu $\mathcal{Z}_n$ und es gilt

$$H_n = 1 + \sum_{k=2}^{n} \frac{1}{k} \leq 1 + \int_1^n \frac{1}{x} \, \mathrm{d}x = 1 + \log n - \log 1 = 1 + \log(n) = \mathcal{O}(\log n).$$

Aufgabe 5. Das Horner-Schema dient zur Auswertung von Polynomen. Die Grundidee dahinter ist die Umformung $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + x a_n) \ldots))$.

(i) Wiederholen Sie die Funktionsweise des Horner-Schemas und schreiben sie einen Pseudocode für die Auswertung von Polynomen mittels Horner-Schema.

(ii) Schreiben Sie einen Pseudocode für die direkte Auswerung von Polynomen (Einsetzen).

(iii) Vergleichen Sie die Schrittzahlen beider Codes.

Lösung.

(i) Wir nehmen an, dass Polynome als Arrays übergeben werden mit $p[0] = a_0, \ldots, p[n] = a_n$.

```
1 : Prozedur HORNER-SCHEMA(p, x)
2 :     n := grad p
3 :     y := p[n] · x
4 :     Für i = n − 1, . . . , 1
5 :         y := y + p[i]
6 :         y := x · y
7 :     Ende Für
8 :     y := y + p[0]
9 : Ende Prozedur
```

```
1 : Prozedur HORNER-SCHEMA(p, x)
2 :     n := grad p
3 :     y := p[n]
4 :     Für i = n − 1, . . . , 1
5 :         y := p[i] + x · y
6 :     Ende Für
7 : Ende Prozedur
```

(ii)

```
1 : Prozedur POLYNOM AUSWERTEN(p, x)
2 :     n := grad p
3 :     y := 0
4 :     z := 1
5 :     Für i = 0, . . . , n
6 :         y := y + p[i] · z
7 :         z := z · x
8 :     Ende Für
9 : Ende Prozedur
```

(iii) Wir zählen als Schritte nur reine Rechenschritte und nicht jede ausgeführte Zeile Code: Additionen / Multiplikationen Horner-Schema (in Abhängigkeit vom Grad des Polynoms):

$$A_{\text{Horner-Schema}}(n) = n, \quad M_{\text{Horner-Schema}}(n) = n$$

Additionen / Multiplikationen Polynom-Auswerten (in Abhängigkeit vom Grad des Polynoms):

$$A_{\text{gewöhnlich}}(n) = n + 1, \quad M_{\text{gewöhnlich}}(n) = 2(n + 1).$$

Da Additionen wesentlich billiger sind als Multiplikationen fallen diese nicht so ins Gewicht. Das Horner-Schema spart also fast die Hälfte der Zeit im Vergleich zur gewöhnlichen Auswertung des Polynoms.

Aufgabe 6.

(a) Zeigen Sie mittels vollständiger Induktion, dass ein Algorithmus, dessen Laufzeit $T(n)$ (wobei $n = 2^k$, $k \in \mathbb{Z}^+$) der Rekursion

$$T(n) = \begin{cases} 1 & \text{für } n = 2 \\ 2T(\frac{n}{2}) + 1 & \text{für } n = 2^k, k > 1 \end{cases}$$

genügt, $T(n) = n - 1$ erfüllt.

(b) Zeigen Sie mittels vollständiger Induktion, dass ein Algorithmus, dessen Laufzeit $T(n)$ (wobei $n = 2^k$, $k \in \mathbb{Z}^+$) der Rekursion

$$T(n) = \begin{cases} 2 & \text{für } n = 2 \\ 2T(\frac{n}{2}) + n & \text{für } n = 2^k, k > 1 \end{cases}$$

genügt, $T(n) = n \log_2(n)$ erfüllt.

Lösung.

(a) Wir führen Induktion nach k (und schreiben immer 2^k statt n).
IA($k = 1$):

$$\implies T(2^1) = T(2) = 1 = 2 - 1 = 2^1 - 1$$

IV: $T(2^{k-1}) = 2^{k-1} - 1$
IS($k - 1 \mapsto k$):

$$\implies T(2^k) = 2T(2^{k-1}) + 1 \overset{\text{IV}}{=} 2(2^{k-1} - 1) + 1 = 2^k - 2 + 1 = 2^k - 1$$

(b) Wir führen Induktion nach k (und schreiben immer 2^k statt n).
IA($k = 1$):

$$\implies T(2^1) = T(2) = 2 = 2^1 \log_2 2^1$$

IV: $T(2^{k-1}) = 2^{k-1} \log_2 2^{(k-1)}$
IS($k - 1 \mapsto k$):

$$\implies T(2^k) = 2T(2^{k-1}) + 2^k \overset{\text{IV}}{=} 2(2^{k-1} \log_2 2^{k-1}) + 2^k = 2^k(k-1) + 2^k = 2^k k - 2^k + 2^k = 2^k \log_2(2^k)$$

Diskrete und Geometrische Algorithmen

2. Übung am 19.10.2020

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

Aufgabe 7. Zum asymptotischen Vergleich von Folgen:

(a) Vergleichen Sie das asymptotische Verhalten von $f(n) = n!$ und $g(n) = (n+2)!$, also überlegen Sie sich ob eine (welche) der Funktionen ein $o, \mathcal{O}, \omega, \Omega, \Theta$ der anderen Funktion ist.

(b) Vergleichen Sie das asymptotische Verhalten von $f(n) = n^{\log_2(4)}$ und $g(n) = 3^{\log_2(n)}$, also überlegen Sie sich ob eine (welche) der Funktionen ein $o, \mathcal{O}, \omega, \Omega, \Theta$ der anderen Funktion ist.

(c) Zeigen Sie an Hand der Defintion, dass für positive Funktionen f und g die Beziehung

$$\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$$

gilt.

(d) Gilt selbige Beziehung ebenfalls für $\min\{f(n), g(n)\}$?

(e) Folgt aus $f(n) = \mathcal{O}(g(n))$, dass $2^{f(n)} = \mathcal{O}(2^{g(n)})$?

(f) Gilt für alle positiven Funktionen f die Beziehung $f(n) = \mathcal{O}(f(n)^2)$?

(g) Finden Sie eine Funktion f, sodass weder $f(n) = \mathcal{O}(n)$ noch $f(n) = \Omega(n)$ gilt.

Lösung.

(a)

$$\frac{f(n)}{g(n)} = \frac{n!}{(n+1)!} = \frac{1}{(n+2)(n+1)} \xrightarrow{n\to\infty} 0$$

$$\overset{1.3.7}{\iff} f \in o(g) \overset{1.3.1}{\subseteq} \mathcal{O}(g)$$
$$\overset{1.3.5}{\iff} g \in \omega(f) \overset{1.3.1}{\subseteq} \Omega(f)$$

$$\implies \frac{g(n)}{f(n)} \xrightarrow{n\to\infty} \infty$$

$$\overset{1.2.3}{\iff} g \notin \mathcal{O}(f) \overset{1.3.2}{\supseteq} o(f)$$
$$\overset{1.2.2}{\iff} f \notin \Omega(g) \overset{1.3.2}{\supseteq} \omega(g)$$

$$\implies g \notin \mathcal{O}(f) \cap \Omega(f) \overset{2.1.1}{=} \Theta(f)$$
$$\implies f \notin \mathcal{O}(g) \cap \Omega(g) \overset{2.1.1}{=} \Theta(g)$$

(b)

$$\log_2 4 = \log_2 2^2 = 2$$
$$\log_2 n = \frac{\log_3 n}{\log_3 2}$$
$$2 - \frac{1}{\log_3 2} > 0 \iff 2 > \frac{1}{\log_3 2} \iff \log_3 4 = \log_3 2^2 = 2\log_3 2 > 1 = \log_3 3^1$$

$$\implies \frac{f(n)}{g(n)} = \frac{n^{\log_2 4}}{3^{\log_2 n}} = n^2 / \left(3^{\log_3 n}\right)^{1/\log_3 2} = n^{2-\frac{1}{\log_3 2}} \xrightarrow{n\to\infty} \infty \implies \frac{g(n)}{f(n)} \xrightarrow{n\to\infty} 0$$

(b) ist also (a), verkehrt rum.

(c) Seien $c := 1/2$ und $d := 1$, dann gilt $\forall^\infty n \in \mathbb{N}$:

$$\begin{aligned} c(f(n)+g(n)) \leq \max\{f(n),g(n)\} &= \frac{f(n)+g(n)+|f(n)-g(n)|}{2} \\ &\leq \frac{f(n)+g(n)}{2} + \frac{f(n)+g(n)}{2} = d(f(n)+g(n)). \end{aligned}$$

(d) Gegenbeispiel: Seien $f \equiv 0$ und $g \equiv 1$, dann gilt $\forall\, c, d > 0$:

$$c\underbrace{(f(n)+g(n))}_{1} \not\leq \underbrace{\min\{f(n),g(n)\}}_{0} < d\underbrace{(f(n)+b(n))}_{1}.$$

$$\implies \min\{f,g\} \notin \Theta(f+g)$$

(e) Gegenbeispiel: Seien $f(n) = n$ und $g(n) = -n$, für $n \in \mathbb{N}$, dann gilt wegen $|f| = |g|$ zwar $f \in \mathcal{O}(g)$, aber

$$\limsup_{n\to\infty} \left|\frac{2^{f(n)}}{2^{g(n)}}\right| = \limsup_{n\to\infty} \left|\frac{2^n}{2^{-n}}\right| = \limsup_{n\to\infty} 2^{2n} = \infty \iff f \notin \mathcal{O}(g).$$

Gegenbeispiel: Seien $f = \log_2$ und $g = \log_4$, für $n \in \mathbb{N}$, dann gilt wegen $|f| = \log_2 4|g|$ zwar $f \in \mathcal{O}(g)$, aber

$$\limsup_{n\to\infty}\left|\frac{2^{f(n)}}{2^{g(n)}}\right| = \limsup_{n\to\infty}\left|\frac{2^{\log_2(n)}}{2^{\log_4(n)}}\right| = \limsup_{n\to\infty}\left|\frac{n}{2^{\frac{\log_2(n)}{\log_2(4)}}}\right| = \limsup_{n\to\infty}\left|\frac{n}{(2^{\log_2(n)})^{1/2}}\right|$$
$$= \limsup_{n\to\infty}\left|\frac{n}{\sqrt{n}}\right| = \limsup_{n\to\infty}\sqrt{n} = \infty$$

(f) Gegenbeispiel: Sei $f(n) = 1/n$.

$$\implies \limsup_{n\to\infty}\left|\frac{f(n)}{f(n)^2}\right| = \limsup_{n\to\infty}\left|\frac{1/n}{1/n^2}\right| = \limsup_{n\to\infty}|n| = \infty \iff f \notin \mathcal{O}(f^2)$$

(g) Beispiel:

$$f(n) := \begin{cases} 0, & n \in 2\mathbb{N}, \\ n^2, & n \in 2\mathbb{N}-1 \end{cases}$$

$$\implies \liminf_{n\to\infty}\left|\frac{f(n)}{n}\right| = \sup_{n\in\mathbb{N}}\inf_{k\geq n}\left|\frac{f(k)}{k}\right| = 0 \iff f(n) \neq \Omega(n),$$
$$\limsup_{n\to\infty}\left|\frac{f(n)}{n}\right| = \inf_{n\in\mathbb{N}}\sup_{k\geq n}\left|\frac{f(k)}{k}\right| = \infty \iff f(n) \neq \mathcal{O}(n)$$

Aufgabe 8. Zeigen Sie:

$$f(n) := 2^{2^{\lfloor \log_2(\log_2 n)\rfloor}} = \mathcal{O}(n)$$

und bestimmen Sie die größte Zahl $c > 0$, sodass

$$2^{2^{\lfloor \log_2(\log_2 n)\rfloor}} = \Omega(n^c)$$

Lösung.

1.
$$\implies \forall n \in \mathbb{N} : 2^{2^{\lfloor \log_2(\log_2 n)\rfloor}} \leq 2^{2^{\log_2(\log_2 n)}} = 2^{(\log_2 n)} = n = \mathcal{O}(n)$$

2. Wir behaupten, dass
$$1/2 = c := \sup C, \quad C := \left\{d > 0 : 2^{2^{\lfloor \log_2(\log_2 n)\rfloor}} = \Omega(n^d)\right\}.$$

„$\leq$“:
$$\implies \forall n \in \mathbb{N} : 2^{2^{\lfloor \log_2(\log_2 n)\rfloor}} \geq 2^{2^{\log_2(\log_2 n)-1}} = 2^{2^{\log_2(\log_2(n))}/2} = (2^{\log_2 n})^{1/2} = \sqrt{n} = \Omega(n^{1/2})$$
$$\implies 1/2 \in C$$

„$\geq$“: Sei $d > 1/2$, dann ist $1/2 - d < 0$. Betrachte die Folge $a_n := 2^{2^n} - 1$.

$$\implies f(a_n) = 2^{2^{\lfloor \log_2 (\log_2 (a_n)) \rfloor}} = \underbrace{2^{2^{\lfloor \log_2 \left(\log_2 \left(2^{2^n}-1\right)\right) \rfloor}}}_{\overset{\text{scharf}}{<} 2^{2^{\lfloor \log_2 \log_2 2^{2^n} \rfloor} = 2^{2^n}}} = 2^{2^{n-1}}$$

$$\implies \frac{f(a_n)}{a_n^d} = \frac{2^{2^{n-1}}}{(2^{2^n}-1)^d} = \frac{(2^{2^n})^{1/2}}{(2^{2^n}-1)^d} \leq \frac{(2^{2^n})^{1/2}}{(2^{2^n}/2)^d} \leq 2^d \frac{(2^{2^n})^{1/2}}{(2^{2^n})^d} = 2^d (2^{2^n})^{1/2-d} \xrightarrow{n\to\infty} 0$$

$$\implies \liminf_{n\to\infty} \frac{f(n)}{n^d} = 0 \iff f(n) \neq \Omega(n^d)$$

$$\implies d \notin C$$

Aufgabe 9. In einer Menge von n Personen können 10 Personen Deutsch, 9 Englisch, 9 Russisch, 5 Deutsch und Englisch, 7 Deutsch und Russisch, 4 Englisch und Russisch, 3 alle drei Sprachen. Wie groß ist n? (Hinweis: Prinzip von Inklusion und Exklusion.)

Lösung.

Satz 2.1 (Prinzip von Inklusion und Exklusion)**.** *Seien $A_1, \ldots, A_n$ endliche Mengen, dann ist*

$$|\bigcup_{i=1}^{n} A_i| = \sum_{\emptyset \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} |\bigcap_{i\in I} A_i|$$

Satz 2.17: Additionstheorem

$A_1, \ldots, A_n$ seien Mengen aus einem Ring mit einem Inhalt μ, und $\mu(A_i)$ sei endlich für alle $i = 1, \ldots, n$. Dann gilt:

$$\mu(\bigcup_{i=1}^{n} A_n) = \sum_{I\subseteq\{1,\ldots,n\}, I\neq\emptyset} (-1)^{|I|-1} \mu(\bigcap_{i\in I} A_i) =$$

$$\mu(A_1) + \ldots + \mu(A_n) - \mu(A_1 \cap A_2) - \ldots \mu(A_{n-1} \cap A_n) + \mu(A_1 \cap A_2 \cap A_3) + + - - \ldots +$$
$$(-1)^{n-1} \mu(A_1 \cap \ldots \cap A_n).$$

Abbildung 1: Grill - Maß- und Wahrscheinlichkeitstheorie

Wir definieren die Mengen D, E, R jeweils als die Menge aller Deutsch-, Englisch-, Russisch-Sprachler. Sei Ω die Menge aller Personen.

$$\implies |\Omega| = \underbrace{|(D \cup E \cup R)^C|}_{0} + |D \cup E \cup R|$$
$$= |D| + |E| + |R| - |D \cap E| - |E \cap R| - |R \cap D| + |D \cap E \cap R| = 10 + 9 + 9 - 5 - 7 - 4 + 3 = 15$$

Aufgabe 10. Gegeben sei die Adjazenzmatrix eines gerichteten Graphens $G = (V, E)$, welcher keine Schlingen (also Kanten $(v, v), v \in V$) und keine Mehrfachkanten enthält. Eine universelle Senke in solch einem gerichteten Graphen G ist ein Knoten s mit Hingrad $d^-(s) = |V| - 1$ und Weggrad $d^+(s) = 0$. Man zeige, dass es möglich ist, durch Untersuchen der Adjazenzmatrix A in Laufzeit $O(|V|)$ festzustellen, ob G solch eine universelle Senke enthält oder nicht.

Definition 2.1. Ein *gerichteter Graph* ist ein Paar $G = (V, E)$ wobei V eine beliebige Menge ist und $E \subseteq V \times V$.

Definition 2.3. Die *Adjazenzmatrix* eines gerichteten Graphen $G = (\{1, \dots, n\}, E)$ ist die Matrix $M = (m_{i,j})_{1 \leq i,j \leq n}$ wobei

$$m_{i,j} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{falls } (i,j) \notin E \end{cases}$$

Lösung.
Offensichtlich sind universelle Senken (u.S.) eindeutig. Wir führen weiters folgende Übersetzungen durch:

- keine Schlingen:
  d.h. alle Diagonaleinträge sin 0
  d.h. $\forall\, i = 1, \dots, |V| : A_{ii} = 0$
- keine Mehrfachkanten:
  d.h. gespiegelte 1-Einträge sind 0-Einträge
  d.h. $\forall\, i, j = 1, \dots, |V| : A_{ij} = 1 \implies A_{j,i} = 0$
- s u.S.
  d.h. s-te Zeile hat nur 0er & s-te Spalte hat genau einen 0er
  d.h. $A_s^T = 0 \wedge A_s = (\underbrace{1, \dots, 1}_{(s-1)\text{-mal}}, 0, \underbrace{1, \dots, 1}_{|V|\text{-mal}})^T$

	1	2	3	$\cdots$	$s-1$	s	$s+1$	$\cdots$	$\lvert V\rvert-2$	$\lvert V\rvert-1$	$\lvert V\rvert$
1	0					1					
2		0				1					
3			0			1					
$\vdots$				$\ddots$		$\vdots$					
$s-1$					0	1					
s	0	0	0	$\cdots$	0	0	0	$\cdots$	0	0	0
$s+1$						1	0				
$\vdots$						$\vdots$		$\ddots$			
$\lvert V\rvert-2$						1			0		
$\lvert V\rvert-1$						1				0	
$\lvert V\rvert$						1					0

```
1 : Prozedur UNIVERSELLE SENKE(G)
2 :   (V, E) := G
3 :   A := ADJAZENZMATRIX(G)
4 :   i := 1
5 :   j := 2
6 :   Solange max{i, j} ≤ |V|
7 :     Wenn A_ij = 0
  :     ⟹ j-te Spalte hat mehr als (nicht genau) einen 0er!
  :     ⟹ j keine u.S.
8 :       s := i
9 :       j := max{i, j} + 1
10 :    Sonst
  :     ⟹ A_ij = 1
  :     ⟹ i-te Zeile hat nicht nur 0er!
  :     ⟹ i keine u.S.
11 :      s := j
12 :      i := max{i, j} + 1
13 :    Ende Wenn
14 :  Ende Solange
15 :  Für r := 1, ..., |V|
16 :    Wenn r ≠ s
  :     ⟹ (r, s) kein Diagonaleintrag
17 :      Wenn A_rs = 0
  :       ⟹ s-te Spalte hat mehr als (nicht genau) einen 0er!
  :       ⟹ s keine u.S.
18 :        s := NIL
19 :        Ende Prozedur
  :       Sonst
  :       ⟹ A_rs = 1
  :       ⟹ A_sr = 0
20 :      Ende Wenn
  :     Sonst
  :     ⟹ r = s =: t
  :     ⟹ (t, t) Diagonaleintrag
  :     ⟹ A_tt = 0
21 :    Ende Wenn
22 :  Ende Für
  :   ⟹ ∀ r = 1, ..., |V| : (A_sr = 0) ∧ (s ≠ r ⟹ A_rs = 1)
  :   ⟹ s-te Zeile hat nur 0er & s-te Spalte hat genau einen 0er
  :   ⟹ s u.S.
23 : Ende Prozedur
```

Lösung. Wir bemerken, dass wir das Problem eine universelle Senke zu finden, reformulieren können zu: Finde Index i, sodass die i-te Zeilensumme von A gleich 0 und die i-te Spaltensumme gleich $|V|-1$ ist.
Anschaulich gesprochen, wird bei jedem Matrixeintrag außerhalb der Diagonale den wir überprüfen ein Senkenkandidat ausgeschlossen. Das liegt daran, dass für $A[i,j]=1$ für i die Bedingung, dass die Zeilensumme 0 sein muss verletzt wird, und für $A[i,j]=0$ für j die Bedingung, dass die Spaltensumme gleich $|V|-1$ sein muss verletzt wird. Damit bleibt uns nach $n-1$ Überprüfungen nur noch 1 möglicher Senkenkandidat übrig. Von diesem müssen wir noch überprüfen, ob er tatsächlich die Senkenbedingung erfüllt, was in maximal $2n$ Überprüfungen gelingt.
Weiter unten ist ein Algorithmus ausgeführt, der diese eben beschriebenen Schritte präzisiert.

```
1 : Prozedur FINDE UNIVERSELLE SENKE(A)
2 :   n := A.Zeilenanzahl
3 :   Senke := NIL
4 :   i := 1
5 :   j := 2
6 :   s := 1
7 :   Solange max{i,j} ≤ n :
8 :     Wenn A[i,j] = 0 :
9 :       j := max{i,j} + 1
10 :      s := i
11 :    Ende Wenn
12 :    Wenn A[i,j] = 1 :
13 :      i := max{i,j} + 1
14 :      s := j
15 :    Ende Wenn
16 :  Ende Solange
17 :  Für k = 1,...,n :
18 :    Wenn k ≠ s :
19 :      Wenn A[k,j] = 0 :
20 :        s := NIL
21 :      Ende Wenn
22 :    Ende Wenn
23 :  Ende Für
24 : Ende Prozedur
```

Der Solange-Block wird maximal $|V|$-Mal ausgeführt, ebenso die Für-Schleife weiter unten, insgesamt entscheidet der Algorithmus also in $\mathcal{O}|V|$ Schritten, ob es eine universelle Senke gibt.

Lösung. Alternativer Code:

```
1 : Prozedur UNIVERSELLE SENKE(G)
2 :   (V, E) := G
3 :   A := ADJAZENZMATRIX(G)
4 :   i := 1
5 :   j := 2
6 :   s := 1
7 :   Solange j ≤ |V|
8 :     Wenn A_ij = 0
9 :       j := j + 1
10 :    Sonst
11 :      s := j
12 :      i, j := j, j + 1
13 :    Ende Wenn
14 :  Ende Solange
15 :  Für r := 1, ..., |V| und r ≠ s
16 :      Wenn A_rs = 0
17 :        s := NIL
18 :        Ende Prozedur
19 :      Ende Wenn
20 :  Ende Für
21 : Ende Prozedur
```

Aufgabe 11. Sei G der vollständige Graph auf 6 Knoten (also eine Kante zwischen je 2 von ihnen). Jede Kante ist rot oder blau gefärbt. Zeigen Sie: Es existiert in dem Graphen ein Dreieck (induzierter Teilgraph mit 3 Knoten) mit nur roten Kanten oder ein Dreieck mit nur blauen Kanten.
(Hinweis: Schubfachprinzip)

> Das *Schubfachprinzip* (engl. *pigeonhole principle*) besagt Folgendes: Seien A und B endliche Mengen mit $|A| > |B|$, dann existiert keine injektive Funktion $f : A \to B$. Es kann mit einem einfachen Induktionsargument bewiesen werden. Ein Beispiel für seine Anwendung liefert der folgende Satz.

Abbildung 2: Schubfachprinzip

Lösung. Bezeichne die Knoten von G mit A, B, C, D, E, F.
Vom Knoten A gehen insgesamt 5 Kanten aus, also gibt es mindestens 3 Kanten, die die gleiche Farbe haben.
O.b.d.A. seien die Kanten $\overline{AB}, \overline{AC}, \overline{AD}$ allesamt rot.
Nun betrachte die Kanten $\overline{BC}, \overline{CD}, \overline{BD}$.
Fall 1: Alle diese Kanten sind blau. Dann besteht das Dreieck $\overline{BCD}$ nur aus blauen Kanten.
Fall 2: Es existiert eine rote Kante unter den dreien: Dann besteht eines der Dreiecke $\overline{ABC}, \overline{ABD}, \overline{ACD}$ nur aus roten Kanten.

Aufgabe 12. Beweisen Sie, dass es unter je neun Punkten in einem Würfel der Kantenlänge 2 stets zwei Punkte gibt, deren Abstand höchstens $\sqrt{3}$ ist.
(Hinweis: Schubfachprinzip)

Lösung. Man partitioniere den Würfel W in 8 Teilwürfel aus $\mathcal{W} := \{W_{abc} : a, b, c = 1, 2\}$ der Kantenlänge 1.

$$\implies W = \sum_{a,b,c=1}^{2} W_{abc}$$

Sei $P := \{p_1, \ldots, p_9\} \subseteq W$ die Menge der 9 Punkte. Nun ist aber $|P| = 9 > 8 = |\mathcal{W}|$. Laut dem Schubfachprinzip existiert daher keine injektive Funktion $P \to \mathcal{W}$. Das bedeutet, es gibt einen Teilwürfel $\widetilde{W} \in \mathcal{W}$, in welchem mindestens zwei verschiedene Punkte $\widetilde{p_1}, \widetilde{p_2} \in P \cap \widetilde{W}$ zu finden sind. Der maximale Abstand zweier Punkte im Einheitswürfel (also auch in $\widetilde{W} \subseteq \mathbb{R}^3$) ist $\operatorname{diam}_2 \widetilde{W} = \sqrt{3}$. Insbesondere, ist daher auch $d_2(\widetilde{p_1}, \widetilde{p_2}) \leq \sqrt{3}$.

Diskrete und Geometrische Algorithmen

3. Übung am 9.11.2020

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

Aufgabe 13. Gegeben sei ein zusammenhängender ungerichteter Graph $G = (V, E)$ mit einer geraden Anzahl an Knoten. Zeigen Sie, dass es einen (nicht notwendigerweise zusammenhängenden) Untergraph mit Knotenmenge V gibt (also einen Graph $G' = (V, E')$ mit $E' \subseteq E$), in dem alle Knotengrade ungerade sind. (Hinweis: beweisen Sie die Behauptung für Bäume und begründen Sie, warum diese Annahme reicht.)

Lösung. Der Grad eines Knoten $v \in V$ ist definert als

$$\text{grad}(x) = |\{\{x, y\} : \{x, y\} \in E\}|.$$

Wir beweisen die Aussage mit Induktion nach $n := |E| = |V - 1|$.

IA($n = 1$):
Es gibt genau eine Kante, welche die beiden Knoten verbindet. Also gilt die Aussage bereits für $E' = E$.

IS($n \mapsto n + 2$):
Betrachte einen beliebigen Baum $G = (V, E)$ mit einer geraden Anzahl an Knoten. O.B.d.A. $\exists\, v \in V$: $\text{grad}(v) = 2k$, $k \in \mathbb{N}$, sonst wäre die Aussage schon erfüllt. Wir betrachten Menge aller Kanten aus E, die an v angrenzen.

$$\{x_1, \ldots, x_{2k}\} = \{x \in E : \{x, v\} \in V\}$$

Löscht man diese Kanten und den Knoten v aus G, so zerfällt der neue Graph in Zusammenhangskomponenten $G'_1, \ldots, G'_{2k}$. Für $i = 1, \ldots, 2k$, ergänzen wir G'_i um den Knoten v und Kante $\{x_i, v\}$ zum „Ast“ G_i.

G besteht aus einer geraden Anzahl von Knoten und somit aus einer ungeraden Anzahl an Kanten. Also gibt es also mindestens einen (von v ausgehenden) Ast G_i, der eine gerade Kanten-Zahl hat. (Wenn alle Äste (gerade viele) ungerade Kanten-Zahl hätten, wäre die Gesamt-Kanten-Zahl gerade mal ungerade, also gerade!) „$G \setminus G_i$“ hat dann eine ungerade Kanten-Zahl, weil „$G = G_i + G \setminus G_i$“ ja eine ungerade Kanten-Zahl hat.

Wir löschen von G nun die Kante $\{x_i, v\}$ (des Asts G_i (mit gerader Kanten-Zahl)). Dann erhalten wir zwei Zusammenhangskomponenten mit ungerader Kantenanzahl $\leq n$. Die sind wieder Bäume. Somit können wir die Induktionsvoraussetzung auf Beide anwenden. Wir vereinigen die resultierenden Graphen und erhalten $G' = (V, E')$.

Sei $G = (V, E)$ ein beliebiger zusammenhängender Graph mit gerader Knotenanzahl. Ein Baum ist genau ein minimal zusammenhängender Graph. Wir finden einen solchen Teilgraphen („Spannbaum“) $G_0 = (V, E_0)$ mit $E_0 \subseteq E$. Auf den wenden wir das oben gezeigte an.

□

Aufgabe 14. Sei $A[1,\ldots,n]$ ein Feld mit n verschiedenen Zahlen. Das Paar (i,j) wird Inversion genannt, wenn $i<j$ und $A[i]>A[j]$ gilt.

(a) Welches Feld mit Elementen der Menge $\{1,\ldots,n\}$ besitzt die meisten Inversionen und wie viele Inversionen sind in diesem Feld enthalten?

(b) Welche Beziehung gibt es zwischen der Anzahl von Inversionen im Eingabefeld und der Laufzeit von Insertion-Sort (Einfügesortieren)?

(c) Geben Sie einen Algorithmus an, der die Anzahl von Inversionen in einer Permutation von n Elementen bestimmt und dessen Laufzeit im schlechtesten Fall $\Theta(n\log n)$ ist. (Hinweis: Modifizieren Sie Merge-Sort (Sortieren durch Verschmelzen) in passender Weise)

Lösung. Wenn wir die Werte des Datenbereichs auf $\{1,\ldots,n\}$ einschränken, bezeichnet die Anzahl der Inversionen genau die Anzahl der Fehlstände, der durch $A[1,\ldots,n]$ induzierten Permutation.

(a) Betrachte das Feld $A[1,\ldots,n]=[n,\ldots,1]$. Klarerweise gilt $\forall\, i,j=1,\ldots,n : i<j : A[i]>A[j]$.

$$\begin{aligned}\implies |\{(i,j):1\le i<j\le n, A[i]>A[j]\}| &= |\{(i,j):1\le i<j\le n\}|\\ &= \left|\sum_{j=1}^{n}\{(i,j):1\le i<j\}\right| = \sum_{j=2}^{n}(j-1) = \sum_{j=1}^{n-1} j = \frac{(n-1)n}{2} = \binom{n}{2}\end{aligned}$$

(b) Für jedes $j=2,\ldots,n$ entspricht die Anzahl der inneren Schleifendurchläufe genau der Anzahl aller $i<j$ mit $A[i]>A[j]$. Insgesamt ist die Anzahl der inneren Schleifendurchläufe also genau die Anzahl der Inversionen des Datenfelds.

(c) Wir Modifizieren den Algorithmus „Merge-Sort“ in passender Weise. Man erhält, zusätzlich zum sortierten Datenfeld, die Anzahl der Fehlstände, mit Laufzeit $\Theta(n\log n)$ im Worst-Case.

```
1 : Prozedur Inversions-Verschmelzen(A, B)
2 :   Sei C ein neues Datenfeld der Länge A.Länge + B.Länge
3 :   i := 1
4 :   j := 1
5 :   n := 0
5 :   Für k := 1, ..., A.Länge + B.Länge
6 :     Falls j > B.Länge oder (i ≤ A.Länge und A[i] ≤ B[j]) dann
7 :       C[k] := A[i]
8 :         i := i + 1
9 :     Sonst
10 :      C[k] := B[j]
11 :      j := j + 1
12 :      Falls i ≤ A.Länge
  :      ⟹ B[j] < A[i] ≤ A[i + 1] ≤ ··· ≤ A[A.Länge]
13 :        n := n + (A.Länge − i + 1)
14 :      Ende Falls
15 :    Ende Falls
16 :  Ende Für
17 :  Antworte C, n
18 : Ende Prozedur
```

Streng genommen, ist die Zeile 12 redundant. Die Prozedur wird dadurch nur besser lesbar.

```
1 : Prozedur VInversionszähler(A)
2 :   Falls A.Länge = 1
3 :     Antworte A, 0
4 :   Sonst
5 :     m := ⌈A.Länge / 2⌉
6 :     L := A[1, ..., m]
7 :     R := A[m + 1, ..., A.Länge]
8 :     L', n_L := VInversionszähler(L)
9 :     R', n_R := VInversionszähler(R)
10 :    A', n_A := Inversions-Verschmelzen(L', R')
11 :    n := n_L + n_R + n_A
12 :    Antworte A', n
13 :  Ende Falls
14 : Ende Prozedur
```

□

Aufgabe 15. Natural Merge-Sort ist eine Variante von Merge-Sort, die bereits vorsortierte Teilfolgen (sogenannte runs) ausnutzt. Ein run ist eine Teilfolge aufeinanderfolgender Glieder $x_i, x_{i+1}, \ldots, x_{i+k}$ mit $x_i \leq x_{i+1} \leq \cdots \leq x_{i+k}$. Die Basis für den Verschmelzen-Vorgang bilden hier nicht die rekursiv oder iterativ gewonnenen Zweiergruppen, sondern die runs. Im ersten Durchlauf des Algorithmus bestimmt man die runs, anschließend fügt man die runs mittels VERSCHMELZEN zusammen.

(a) Sortieren Sie folgende Liste mittels Natural Merge-Sort:

$$2, 4, 3, 1, 7, 6, 8, 9, 0, 5 \tag{1}$$

(b) Schreiben Sie einen Pseudocode für Natural Merge-Sort.

(c) Machen Sie eine Best- sowie eine Worst-Case-Analyse für das Laufzeitverhalten von Natural Merge-Sort bei einem Eingabefeld der Größe n.

Lösung.

(a) Wir haben dafür ein Python-Programm geschrieben. Der Output lautet wie folgt.

```
# ------------------------------------------------------------ #

[[2, 4], [3], [1, 7], [6, 8, 9], [0, 5]]

[2, 4]
[2, 3, 4]
[1, 2, 3, 4, 7]
[1, 2, 3, 4, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# ------------------------------------------------------------ #
```

Jetzt machen wir's manuell. Zuerst bestimmen wir die Runs.

$$[2, 4], [3], [1, 7], [6, 8, 9], [0, 5]$$

Nun verschmelzen wir.

$$\begin{aligned}[2, 4], [3], [1, 7], [6, 8, 9], [0, 5] &\rightsquigarrow [2, 3, 4], [1, 7], [6, 8, 9], [0, 5] \\ &\rightsquigarrow [1, 2, 3, 4, 7], [6, 8, 9], [0, 5] \\ &\rightsquigarrow [1, 2, 3, 4, 6, 7, 8, 9], [0, 5] \\ &\rightsquigarrow [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]\end{aligned}$$

(b)

```
1 : Prozedur Runs Verschmelzen(R)
2 :   m := R.Länge
3 :   C := R[1]
4 :   Für i = 2, ..., m
5 :     C := Verschmelzen(C, R[i])
6 :   Ende Für
7 :   Antworte C
8 : Ende Prozedur
```

```
1 : Prozedur Natural Merge Sort(A)
2 :   Sei R ein neuer Datenfeld der Länge 0.
3 :   n := A.Länge
4 :   i := 1
5 :   Solange i ≤ n
6 :     j := 0
7 :     r := [A[i]]
8 :     Solange i + j + 1 ≤ n und A[i + j] ≤ A[i + j + 1]
9 :       r.append(A[i + j + 1])
10 :      j := j + 1
11 :    Ende Solange
12 :    R.append(r)
13 :    i := i + j + 1
14 :  Ende Solange
15 :  Antworte Runs Verschmelzen(R)
16 : Ende Prozedur
```

Man könnte den Algorithmus evtl. noch optimieren, indem man die Runs aufsteigend nach ihrer Länge sortiert. Damit spart man sich beim Verschmelzen etwas Aufwand.

(c) Sei A ein Datenfeld der Länge n. Das Erstellen der Runs gelingt immer in linearem Aufwand.

- Best-Case:
  Sei das Datenfeld bereits aufsteigend sortiert. Der Aufwand liegt also nur in der Erstellung der Runs. Somit erhalten wir linearen Aufwand.
- Worst-Case:
  Sei das Datenfeld absteigend sortiert. Wir erhalten also n Runs der Länge 1. Abgesehen vom Erstellen der Runs, müssen wir die Prozedur Verschmelzen für $i = 2, \ldots, n$, also $(n-1)$-mal, auf die Datenfelder C und $R[i]$ der Länge $i-1$ bzw. 1 anwenden. Verschmelzen hat linearen Aufwand. Wir kommen also insgesamt auf quadratischen Aufwand.

□

Aufgabe 16. Die Fibonacci-Zahlen seien durch die Rekursion $F_n = F_{n-1} + F_{n-2}$ für $n \geq 2$ mit Anfangswerten $F_0 = 0$ und $F_1 = 1$ definiert. Die Fibonacci-Zahlen können effizient mittels folgender auf Matrizenmultiplikation beruhender Formel berechnet werden:

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n \quad \text{für } n \geq 1$$

a. Beweisen Sie diese Formel durch vollständige Induktion.

b. Überlegen Sie sich einen Algorithmus, der $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ in nur logarithmisch vielen Schritten berechnet.

Lösung. Wir nennen die linke Matrix L_n und die rechte R_n.

a. IA($n = 1$):

$$\implies F_2 = F_1 + F_0 = 1 + 0 \implies L_1 = \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^1 = R_1$$

IS($n \mapsto n + 1$):

$$\begin{aligned} \implies R_{n+1} = R_n \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = L_n \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} &= \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \\ &= \begin{pmatrix} F_{n+1} + F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{pmatrix} = \begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix} = L_{n+1} \end{aligned}$$

b. Sei $k \in \mathbb{N}$.

$$\implies P_k := \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{2^k} = \underbrace{\left(\left(\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \cdots \right) \right)^2}_{k\text{-mal}} = P_{k-1}^2 \implies P_0 = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Stelle $n \in \mathbb{N}_0$ (eindeutig) binär dar, d.h.

$$n = \sum_{k=0}^{m} a_k 2^k, \quad m = \lfloor \log(n) \rfloor, \quad a_0, \ldots, a_m \in \{0, 1\}.$$

$$\implies F_n := \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \prod_{k=0}^{m} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{a_k 2^k} = \prod_{\substack{k=0 \\ a_k = 1}}^{m} P_k$$

```
1 : Prozedur Fibonacci Matrix(n)
2 :   a := Binärkoeffizienten(n)
3 :   m := a.Länge
4 :   F := ( 1  0 )
          ( 0  1 )
5 :   P := ( 1  1 )
          ( 1  0 )
6 :   Für k = 0, ..., m
7 :     Falls a_k = 1
8 :       F := F · P
9 :     Ende Falls
11 :    Falls k < m
12 :      P := P^2
13 :    Ende Falls
14 :  Ende Für
15 :  Antworte F
16 : Ende Prozedur
```

Um die Binätdarstellung zu umgehen, wird man vielleicht dazu verleitet, folgende Prozedur zu verwenden.

```
1 : Prozedur Matrix Potenzieren(M, n)
2 :   Falls n = 1
3 :     Antworte M
4 :   Sonst
5 :     Antworte Matrix Potenzieren(M, ⌊n/2⌋) · Matrix Potenzieren(M, ⌈n/2⌉)
6 :   Ende Falls
7 : Ende Prozedur
```

Wenn man aber diese Prozedur mit M^{2^k} testet, merkt man, dass der Aufwand dennoch linear ist. Man gewinnt durch das „teilen und herrschen“ hier leider Nichts. (Dazu kommt, dass rekursive Funktionen den stack schnell sehr zumüllen.)

□

Aufgabe 17.

(a) Wie schnell könnte man eine $(kn \times n)$-Matrix A mit einer $(n \times kn)$-Matrix B multiplizieren, d.h. $C = A \cdot B$ berechenen, wenn man Strassens Algorithmus als Unterprogramm verwendet?

(b) Benantworten Sie die gleiche Frage, wenn die Reihenfolge der Eingabematrizen vertauscht ist, man also $\tilde{C} = B \cdot A$ bestimmen möchte.

Lösung.

(a) Man kann die Matrizen als Blockmatrizen auffassen.

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_k \end{pmatrix}, \quad B = (B_1 \cdots B_k), \quad A_1, \ldots, A_k, B_1, \ldots, B_k \ (n \times n)\text{-Matrizen}$$

Man muss somit, zur Berechnung der $(kn \times kn)$-Matrix C, für k^2 Blöcke der Größe $n \times n$ durch je eine $(n \times n)$-Matrix-Multiplikation durchführen. Dies kann jeweils mit Strassens Algorithmus gemacht werden.

$$C = AB = \begin{pmatrix} A_1 \\ \vdots \\ A_k \end{pmatrix} (B_1 \cdots B_k) = \begin{pmatrix} A_1B_1 & \cdots & A_1B_k \\ \vdots & \ddots & \vdots \\ A_kB_1 & \cdots & A_kB_k \end{pmatrix}$$

Sei n eine Zweierpotenz. Strassens Algorithmus hat den Aufwand $S(n) = \Theta(n^{\log 7})$. Es ergibt sich also eine Laufzeit von $T(n, k) = k^2 S(n) = \Theta(k^2 n^{\log 7})$.

(b) Analog zu oben betrachten wir wieder die $(n \times n)$-Matrix-Multiplikation von $(n \times n)$-Blockmatrizen. Hier benötigen wir k viele $(n \times n)$-Matrix-Multiplikationen und $k - 1$ viele $(n \times n)$-Matrix-Additionen.

$$\tilde{C} = BA = (B_1 \cdots B_k) \begin{pmatrix} A_1 \\ \vdots \\ A_k \end{pmatrix} = \sum_{i=1}^{k} B_i A_i$$

Es ergibt sich also eine Laufzeit von $T(n, k) = kS(n) = \Theta(kn^{\log 7})$.

□

Aufgabe 18. Zeigen Sie, wie man komplexe Zahlen $a + bi$ und $c + di$ mit nur drei Multiplikationen reeller Zahlen multiplizieren kann. Der Algorithmus sollte a, b, c und d als Eingabe bekommen und den Realteil $ac - bd$ sowie den Imaginärteil $ad + bc$ getrennt ausgeben.

Lösung.

1 : **Prozedur** KOMPLEXE MULTIPLIKATION(a, b, c, d)
2 : $P_1 := (a - b)d$
3 : $P_2 := (d - c)a$
4 : $P_3 := (c + d)b$
5 : **Antworte** $P_1 - P_2, P_1 + P_3$
6 : **Ende Prozedur**

Der Algorithmus leistet das Gewünschte, da

$$P_1 - P_2 = (ad - bd) - (ad - ac) = ac - bd, \quad P_1 + P_3 = (ad - bd) + (bc + bd) = ad + bc.$$

□

Diskrete und Geometrische Algorithmen

4. Übung am 16.11.2020

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

Aufgabe 19. Lösen Sie die Rekursion $na_n = (n+1)a_{n-1} + 2n$ mit $a_0 = 0$.

Lösung. Wir dividieren auf beiden Seiten durch n und erhalten die Rekursionsgleichung

$$a_n = \frac{n+1}{n} a_{n-1} + 2$$

Dies ist eine aus der Vorlesung, siehe dazu Satz 4.1 im Skriptum, bekannte Form mit $b_0 = 0, b_n = 2, n \geq 1, c_n = \frac{n+1}{n}$. Die Lösung ist auch aus der Vorlesung bekannt und hat die Form

$$a_n = \sum_{i=0}^{n} b_i \prod_{j=i+1}^{n} c_j = 2\sum_{i=1}^{n} \prod_{j=i+1}^{n} \frac{j+1}{j} = 2\sum_{i=1}^{n} \frac{n+1}{i+1} = 2(n+1)\sum_{i=2}^{n+1} \frac{1}{i}$$

Eine Lösung der Rekursionsgleichung ist. Mit Induktion nach n erhalten wir für festes i

$$\prod_{j=i+1}^{i} = 1 = \frac{i+1}{i+1}, \quad \prod_{j=i+1}^{n} \frac{j+1}{j} = \frac{n+1}{n}\frac{n}{i+1} = \frac{n+1}{i+1}$$

Machen wir noch die Probe.

$$\begin{aligned} a_0 &= 0, \\ 2n(n+1)\sum_{i=2}^{n+1} \frac{1}{i} = na_n &\overset{!}{=} (n+1)a_{n-1} + 2n = (n+1)2n\sum_{i=2}^{n} \frac{1}{i} + 2n \\ &= 2n(n+1)\sum_{i=2}^{n} \frac{1}{i} + 2n(n+1)\frac{1}{n+1} = \quad 2n(n+1)\sum_{i=2}^{n+1} \frac{1}{i} \end{aligned}$$

□

Aufgabe 20. Lösen Sie folgendes System von Rekursionen

$$a_{n+1} = 2a_n + b_n, \quad b_{n+1} = 4a_n - b_n, \quad \text{für } n \geq 1$$

mit den Anfangsbedingungen $a_0 = 1, b_0 = 0$ indem Sie das System in eine äquivalente Rekursion 2. Ordnung umschreiben.

Lösung. Wir formen zuerst die erste Rekursion um und setzen in die zweite ein

$$\begin{aligned} & a_n = 2a_{n-1} + b_{n-1} \Leftrightarrow b_{n-1} = a_n - 2a_{n-1} \\ \Rightarrow\ & b_n = 4a_{n-1} - b_{n-1} = 4a_{n-1} - (a_n - 2a_{n-1}) = 6a_{n-1} - a_n \\ \Rightarrow\ & b_{n-1} = 6a_{n-2} - a_{n-1} \end{aligned}$$

Setzen wir nun b_{n-1} in die Gleichung aus der ersten Zeile ein erhalten wir also

$$a_n = 2a_{n-1} + b_{n-1} = 2a_{n-1} + 6a_{n-2} - a_{n-1} = a_{n-1} + 6a_{n-2}, \quad \text{für } n \geq 2$$

Mit den Anfangsbedingungen $a_0 = 1, a_1 = 2a_0 + b_0 = 2$. Diese Rekursion 2. Ordnung lösen wir nun mithilfe von Satz 4.2, das charakteristische Polynom unserer Rekursionsgleichung lautet:

$$\chi(z) = z^2 - z - 6 = (z-3)(z+2)$$

Allgemeine Lösungen unserer Rekursion haben also die Form:

$$a_n = c_0(-2)^n + c_1 3^n$$

Jetzt lösen wir noch für die Anfangsbedingungen:

$$\begin{aligned} 1 = a_0 &\overset{!}{=} c_0 + c_1 \\ 2 = a_1 &\overset{!}{=} -2c_0 + 3c_1 = 2(c_1 - 1) + 3c_1 = 5c_1 - 2 \iff c_1 = \frac{4}{5} \iff c_0 = \frac{1}{5}. \end{aligned}$$

Die konkrete Lösung lautet also für $n \geq 1$

$$\begin{aligned} a_n &= \frac{1}{5}(-2)^n + \frac{4}{5}3^n \\ b_n &= 6a_{n-1} - a_n = \frac{6}{5}(-2)^{n-1} + \frac{24}{5}3^{n-1} - \frac{1}{5}(-2)^n - \frac{4}{5}3^n \\ &= \frac{8}{5}(-2)^{n-1} + \frac{12}{5}3^{n-1} = \frac{4}{5}3^n - \frac{4}{5}(-2)^n \end{aligned}$$

□

Aufgabe 21. Gegeben sei die durch folgenden Algorithmus definierte Funktion

$F(n, a, b, c)$:
if $n = 0$ **then**
  return 1
end if
if $n = 1$ **then**
  return 2
end if
$A = F(n-2, b, c, a)$
$B = F(n-2, c, a, b)$
$C = F(n-2, b, a, c)$
if $F(n-1, A, B, C) \equiv 0 \pmod 3$ **then**
  return $F(n-1, A-1, B, C+1) + F(0, A, B, C)$
else
  return $F(n-1, B-1, A+B+C, A \cdot C) + F(0, A, B, C)$
end if

Bezeichne $a(n)$ die Anzahl der Aufrufe von $F(0, x, y, z)$ mit irgendwelchen Parametern x, y und z bei der Berechnung von $F(n, a, b, c)$. Bestimmen Sie eine Rekursionsgleichung (inklusive Anfangsbedingungen) für $(a(n))_{n\in\mathbb{N}}$ und lösen Sie diese.

Lösung. Klarerweise gilt $a(0) = 1, a(1) = 0$. Wir erhalten außerdem die Rekursion

$$a(n) = 2a(n-1) + 3a(n-2) + 1.$$

Diese inhomogene, lineare Rekursionsgleichung 2-ter Ordnung mit konstanten Koeffizienten lässt sich durch $a(n) = y(n) - \frac{d}{C-1} = y(n) - \frac{1}{4}$ lösen, wobei $y(n)$ Lösung von

$$\begin{aligned} y(0) &= a(0) + \frac{1}{4} = \frac{5}{4}, \quad y(1) = a(1) + \frac{1}{4} = \frac{1}{4}, \\ y(n) &= 2y(n-1) + 3y(n-2), \quad n \geq 2. \end{aligned}$$

Das charakteristische Polynom davon lautet

$$\chi(\lambda) = \lambda^2 - 2\lambda - 3 \implies \lambda_{1,2} = 1 \pm 2.$$

Also erhalten wir die Lösung

$$y(n) = c_0(-1)^n + c_1(3)^n.$$

Lösen wir die Konstanten nach den Anfangsbedingungen auf:

$$\begin{aligned} \frac{5}{4} = y(0) &\stackrel{!}{=} c_0 + c_1 \\ \frac{1}{4} = y(1) &\stackrel{!}{=} -c_0 + 3c_1 = 3c_1 + c_1 - \frac{5}{4} \iff c_1 = \frac{3}{8} \iff c_0 = \frac{7}{8} \end{aligned}$$

Schließlich erhalten wir für $a(n)$:

$$a(n) = y(n) - \frac{1}{4} = \frac{7}{8}(-1)^n + \frac{3}{8}3^n - \frac{1}{4}$$

□

Aufgabe 22. Die Türme von Hanoi: Gegeben seien drei Stäbe A, B, C und n verschieden große Scheiben. Anfangs seien alle Scheiben der Größe nach auf Stab A aufgereiht, die größte ganz unten. Dieser Turm von Scheiben soll nun unter folgenden Regeln von A nach B transferiert werden:

- In jedem Zug darf nur eine Scheibe bewegt werden.
- Eine größere Scheibe darf nie über einer kleineren platziert werden.

Sei a_n die minimale Anzahl der benötigten Züge. Ermitteln Sie a_n durch Aufstellen und Lösen einer Rekursion.

Lösung. In den Regeln in der Angabe ist nicht explizit verlangt, dass immer nur die oberste Scheibe bewegt werden kann. Also bewegen wir einfach die unterste zuerst und erhalten einfach $a_n = n$. :)
Ernsthafte Lösung:
Wir haben in EProg einen rekurisven Algorithmus dafür programmiert, der folgende Rekursion verwendet:

1. Verschiebe die obersten $n-1$ Scheiben von Pfosten A auf Pfosten B.
2. Verschiebe die größte Scheibe von Pfosten A auf Pfosten C.
3. Verschiebe die $n-1$ Scheiben von Pfosten B auf Pfosten C.

Also erhalten wir für a_n folgende Rekursionsgleichung:

$$a_n = 2a_{n-1} + 1, \quad n \geq 2.$$

mit der expliziten Lösung

$$a_n = \sum_{i=1}^{n} \prod_{j=i+1}^{n} 2 = 2^n - 1.$$

Warum ist die Anzahl der Züge minimal? Für $n = 0$ ergibt klarerweise unsere Formeln mit 0 Zügen sicher die minimale Anzahl an Zügen zurück. Das Weitere folgt induktiv. Beginnend mit $n+1$ Scheiben auf A müssen wir, um die größte Scheibe auf C zu bringen, zuerst alle n anderen Scheiben auf B legen. Dafür brachen wir mindestens a_n Schritte. Danach mindestens nocheinmal a_n um diese n von B auf C zu bringen. □

Aufgabe 23. In Übungsblatt 3 haben wir gesehen, wie sich die Fibonacci-Zahlen mithilfe von Potenzen der Matrix $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ effizient berechnen lassen. Man kann diese Methode adaptieren, um Zahlenfolgen, die andere Rekursionsgleichungen erfüllen, zu berechnen. Wir betrachten im Folgenden die Potenzen der Matrix

$$M = \begin{pmatrix} 3 & 2 \\ 2 & 0 \end{pmatrix}.$$

Definiere

$$M_n := M^n = \begin{pmatrix} 3 & 2 \\ 2 & 0 \end{pmatrix}^n = \begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix}$$

und betrachte die Koeffizienten dieser Matrix.

a) Geben Sie eine Rekursion für die Zahlenfolge $(a_n)_{n\geq 0}$ an.

b) Lösen Sie diese Rekursion, um eine explizite Formel für a_n zu erhalten.

c) Geben Sie einen Algorithmus zur effizienten Berechnung von M^n an und analysieren Sie dessen Laufzeit in Abhängigkeit von n ($\mathcal{O}$-Notation genügt).

Lösung.

a)

$$\begin{pmatrix} a_n & b_n \\ c_n & d_n \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 2 & 0 \end{pmatrix} \cdot \begin{pmatrix} a_{n-1} & b_{n-1} \\ c_{n-1} & d_{n-1} \end{pmatrix} = \begin{pmatrix} 3a_{n-1} + 2c_{n-1} & 3b_{n-1} + 2d_{n-1} \\ 2a_{n-1} & 2b_{n-1} \end{pmatrix}$$

Wir erhalten die Rekursion

$$a_n = 3a_{n-1} + 2c_{n-1} = 3a_{n-1} + 4a_{n-2}.$$

b) Nullstellen des charakteristischen Polynoms:

$$\chi(\lambda) = \lambda^2 - 3\lambda - 4, \quad \lambda_{1,2} = \frac{3}{2} \pm \sqrt{\frac{9}{4} + 4} = \frac{3}{2} \pm \frac{5}{2}$$

allgemeine Lösung:

$$a_n = \alpha_0(-1)^n + \alpha_1 4^n$$

Anfangsbedingungen:

$$a_0 = 1 \stackrel{!}{=} \alpha_0 + \alpha_1$$
$$a_1 = 3 \stackrel{!}{=} -\alpha_0 + 4\alpha_1 = 4\alpha_1 + \alpha_1 - 1 \iff \alpha_1 = \frac{4}{5} \iff \alpha_0 = \frac{1}{5}.$$

explizite Lösung:

$$a_n = \frac{1}{5}(-1)^n + \frac{4}{5}4^n$$
$$c_n = 2a_{n-1} = \frac{2}{5}(-1)^{n-1} + \frac{8}{5}4^{n-1} = -\frac{2}{5}(-1)^n + \frac{2}{5}4^n$$

c) Wir erkennen aus der Darstellung in a), dass wir b_n, d_n durch die selbe Rekursiongleichung berechnen können mit Anfangswerten $b_0 = 0, b_1 = 2$.

$$b_0 = 0 \stackrel{!}{=} \beta_0 + \beta_1$$
$$b_1 = 2 \stackrel{!}{=} -\beta_0 + 4\beta_1 = 4\beta_1 + \beta_1 \iff \beta_1 = \frac{2}{5} \iff \beta_0 = -\frac{2}{5}.$$

explizite Lösung von b_n, d_n:

$$b_n = -\frac{2}{5}(-1)^n + \frac{2}{5}4^n$$
$$d_n = 2b_{n-1} = -\frac{4}{5}(-1)^{n-1} + \frac{4}{5}4^{n-1} = \frac{4}{5}(-1)^n + \frac{1}{5}4^n$$

Also besteht der Aufwand des Algorithmus lediglich darin, die 4-er Potenzen zu berechnen, sowie eine konstante Anzahl von Additionen und Multiplikationen. Wie beim Algorithmus zur expliziten Berechnung der Fibonacci-Zahlen können wir $n = \sum_{i=0}^{k} a_i 2^i$ in Binärform darstellen und $4^n = \prod_{i=0}^{k} 4^{a_i 2^i}$ in logarithmischer Zeit berechnen, insgesamt erhalten wir also $\mathcal{O}(\log(n))$ für den Aufwand. Im folgenden Algorithmus ist $m \leq \lfloor \log(n) \rfloor$.

```
1 : Prozedur ZahlPotenzieren(x, n)
2 :     u := Binärkoeffizienten(n)
3 :     m := u.Länge
4 :     y := 1
6 :     Für k = 0, ..., m
7 :         Falls a_k = 1
8 :             y := y · x
9 :         Ende Falls
11 :        Falls k < m
12 :            x := x^2
13 :        Ende Falls
14 :    Ende Für
15 :    Antworte y
16 : Ende Prozedur
```

□

Aufgabe 24.

a) Lösen Sie die Rekursion $T(1) = 2$ und $T(n) = T(n-1) + 2$ für $n \geq 2$, indem Sie widerholt in die Rekursion einsetzen, bis Sie $T(n)$ erkennen. Verifizieren Sie ihr Ergebnis anschließend mit der Substitutionsmethode.

b) Lösen Sie die Rekursion $T(1) = 2$ und $T(n) = 3T(n-1) + 2$ für $n \geq 2$, indem Sie widerholt in die Rekursion einsetzen, bis Sie $T(n)$ erkennen. Verifizieren Sie ihr Ergebnis anschließend mit der Substitutionsmethode.

Lösung.

a) Setzen wir also zuerst ein paar mal ein:

$$T(1) = 2$$
$$T(2) = 4$$
$$T(3) = 6$$

Die Vermutung ist, dass $T(n) = 2n$. Mit Induktion lässt sich dies leicht bestätigen:
IA($n = 2$):

$$T(2) = 4 = 2 \cdot 2$$

IV: $T(n) = 2n$

IS:$n \mapsto n + 1$

$$T(n+1) = T(n) + 2 \overset{IV}{=} 2n + 2 = 2(n+1)$$

b) Einsetzen in die Rekursion liefert

$$T(1) = 2$$
$$T(2) = 8$$
$$T(3) = 26$$
$$T(4) = 80$$

Die Vermutung ist, dass $T(n) = 3^n - 1$. Wiederum zeigen wir das mit Induktion:
IA($n = 2$):

$$T(2) = 8 = 3^2 - 1$$

IV: $T(n) = 3^n - 1$

IS:$n \mapsto n + 1$

$$T(n+1) = 3T(n) + 2 \overset{IV}{=} 3(3^n - 1) + 2 = 3^{n+1} - 1$$

□

Diskrete und Geometrische Algorithmen

5. Übung am 23.11.2020

Richard Weiss Florian Schager Christian Sallinger Fabian Zehetgruber
Paul Winkler Christian Göth

Aufgabe 25. Lösen Sie die Rekursion $T(1) = 1$ und $T(n) = 3T(\frac{n}{2}) + n^2 + n$ für $n = 2^k \geq 2$, indem Sie wiederholt in die Rekursion einsetzen, bis Sie $T(n)$ erkennen. Verifizieren Sie ihr Ergebnis anschließend mit der Substitutionsmethode.

Lösung. Setzen wir also zuerst ein paar mal ein (und addieren dabei nicht direkt damit man das Schema erkennen kann):

$$
\begin{aligned}
&T(2^0) = 1\\
&T(2^1) = 3 + 4 + 2 = 2 + 3 + 2^2\\
&T(2^2) = 3 \cdot 2 + 3^2 + 3 \cdot 2^2 + 2^4 + 2^2 = 2^2 + 2 \cdot 3 + 3^2 + 2^4 + 3 \cdot 2^2\\
&T(2^3) = 3^2 \cdot 2 + 3^3 + 2^2 \cdot 3^2 + 3 \cdot 2^4 + 3 \cdot 2^2 + 2^6 + 2^3 = 2^3 + 2^2 \cdot 3 + 2 \cdot 3^2 + 3^3 + 2^4 + 2^6 + 3 \cdot 2^4 + 3^2 \cdot 2^2
\end{aligned}
$$

Wir stellen die Vermutung

$$T(2^k) = \sum_{i=0}^{k} 2^{k-i}3^i + \sum_{i=0}^{k-1} 2^{2(k-i)}3^i$$

auf. Um diese zu Verifizieren verwenden wir Induktion (über k).
IA($k = 0$):

$$T(1) = 1 = 2^0 \cdot 3^0$$

IV: $T(2^k) = \sum_{i=0}^{k} 2^{k-i}3^i + \sum_{i=0}^{k-1} 2^{2(k-i)}3^i$

IS: $k \mapsto k + 1$

$$
\begin{aligned}
T(2^{k+1}) &= 3T(2^k) + 2^{2(k+1)} + 2^{k+1} \overset{IV}{=} 3(\sum_{i=0}^{k} 2^{k-i}3^i + \sum_{i=0}^{k-1} 2^{2(k-i)}3^i) + 2^{2(k+1)} + 2^{k+1}\\
&= \sum_{i=0}^{k} 2^{k-i}3^{i+1} + \sum_{i=0}^{k-1} 2^{2(k-i)}3^{i+1} + 2^{2(k+1)} + 2^{k+1} = \sum_{i=1}^{k+1} 2^{k-(i-1)}3^i + \sum_{i=1}^{k} 2^{2[k-(i-1)]}3^i + 2^{2(k+1)} + 2^{k+1}\\
&= \sum_{i=0}^{k+1} 2^{k+1-i}3^i + \sum_{i=0}^{k} 2^{2[(k+1)-i]}3^i
\end{aligned}
$$

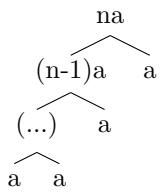
□

Aufgabe 26. Verwenden Sie einen Rekursionsbaum, um eine asymptotische obere Schranke für die Rekursion

$$T(n) = T(n-a) + T(a) + n \quad \text{für } n > a, \quad T(n) = 0 \quad \text{für } n < a,$$

zu bestimmen ($a \geq 1$). Überprüfen Sie Ihre Schranke mittels Substitutionsmethode.

Lösung.

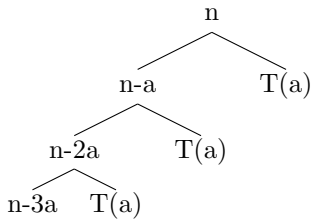


Aus ebenjenem Baum liest man leicht unsere Vermutung für die asymptotische obere Schranke ab.
Wir zeigen mittels vollständiger Induktion: $T(na) \leq n^2 + nT(a)$.

$$\begin{aligned} n = 1: &\quad T(a) \leq 1^2 + 1T(a) \\ n \rightsquigarrow n+1: &\quad T((n+1)a) = T(na) + T(a) + n \leq n^2 + nT(a) + T(a) + n \leq (n+1)^2 + (n+1)T(a). \end{aligned}$$

Daher erhalten wir $T(na) = \mathcal{O}(n^2)$ □

Lösung.



Der Baum hat eine Tiefe von $\frac{n}{a}$ und auf jeder Höhe hat er zwei Blätter also insgesamt $2\frac{n}{a}$ Blätter. Außerdem ist der Aufwand in Tiefe k gegeben durch $n - ka + T(a)$. Wir berechenen

$$\sum_{k=0}^{\frac{n}{a}-1} (n - ka + T(a)) + 2\frac{n}{a} = a \sum_{k=1}^{\frac{n}{a}-1} k + \frac{n}{a}(T(a) + 2)$$

Also ist die Vermutung $T(n) = \mathcal{O}(n^2)$. Wir wollen also zeigen

$$\exists c > 0 \; \exists n_0 \in \mathbb{N} \; \forall n \geq n_0 \; T(n) \leq cn^2$$

Für alle $n < a$ gilt dies können wir ein beliebiges $c > 0$ aussuchen, da $T(n) = 0$. Außerdem wollen wir c hinreichend groß, dass $T(a) \leq ca^2$ gilt. Unter der Annahme, dass für $n > a$ alle $k < n$ gilt, dass $T(k) \leq ck^2$ wünschen wir uns

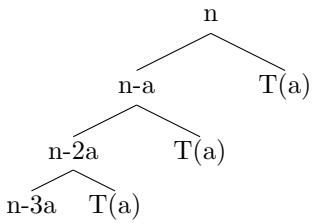
$$\begin{aligned} cn^2 \overset{!}{\geq} T(n) = T(n-a) + T(a) + n \leq c(n-a)^2 + ca^2 + n = cn^2 - 2cna + 2ca^2 + n \\ \Leftarrow 2ca(n-a) \geq n \Leftarrow c \geq \frac{n}{2a(n-a)} \end{aligned}$$

Zusammengefasst wollen wir

1. $n_0 = 0$
2. $c \geq \frac{T(a)}{a^2}$
3. $\forall n > a \ (c \geq \frac{n}{2a(n-a)})$

□

Lösung.



Der Baum hat eine Tiefe von $\frac{n}{a}$ und auf jeder Höhe hat er zwei Blätter also insgesamt $2\frac{n}{a}$ Blätter. Außerdem ist der Aufwand in Tiefe k gegeben durch $n - ka + T(a)$. Wir berechenen

$$\sum_{k=0}^{\frac{n}{a}-1}(n - ka + T(a)) + 2\frac{n}{a} = \frac{n^2}{a} - a\sum_{k=0}^{\frac{n}{a}-1} k + \frac{n}{a}(T(a) + 2) \leq \frac{n^2}{a} + \frac{n}{a}(T(a) + 2)$$

Also ist die Vermutung $T(n) = \mathcal{O}(n^2)$. Wir wollen also zeigen

$$\exists c > 0 \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 : \ T(n) \leq cn^2 + ca^2 + T(a)$$

Für unseren Induktionsanfang sei $n_0 = 0 \implies T(n_0) = 0 \leq ca^2 + T(a)$. Nehmen wir nun also an, dass die Ungleichung für alle $k < n$ gilt. Dann gilt auch

$$T(n) = T(n-a) + T(a) + n \leq c(n-a)^2 + ca^2 + T(a) + n = cn^2 - 2cna + T(a) + n \leq cn^2 + ca^2 + T(a)$$

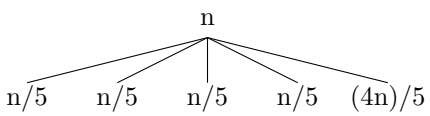
Wobei wir hier $\forall n \geq n_0 : \ 2cna \geq n$ als Bedingung an unser c stellen.
Damit haben wir also gezeigt $T(n) = \mathcal{O}(cn^2 + ca^2 + T(a)) = \mathcal{O}(n^2)$. □

Aufgabe 27. Verwenden Sie einen Rekursionsbaum, um eine asymptotische untere sowie eine asymptotisch obere Schranke (brauchen nicht dieselbe Größenordnung zu haben) für die Rekursion

$$T(n) = 4T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + n$$

zu finden. Verifizieren Sie Ihre Schranke mittels Substitutionsmethode.

Lösung.



$$T(n) = n + \frac{8n}{5} + \frac{64n}{25} + \cdots$$

obere Schranke: $T(n) \leq n^2$

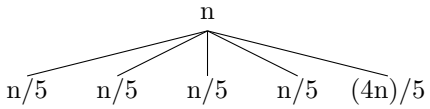
$$T(5n) = 4T(n) + T(4n) + n \leq 4n^2 + 16n^2 + n \leq (5n)^2.$$

untere Schranke: $T(n) \geq n\log(n)$ für $n > 2$:

$$\begin{aligned} T(5n) &= 4T(n) + T(4n) + n \geq 4n\log(n) + 4n\log(4n) + n = 4n\log(4n^2) + n = 8n\log(2n) + n \\ &= 5\log((2n)^{8/5}) + n \geq 5\log(5n) \end{aligned}$$

□

Lösung.



Der Baum hat Arität 5 und eine Tiefe von $\log_{\frac{5}{4}}(n)$. Die Anzahl der Blätter können wir abschätzen durch $5^{\log_{\frac{5}{4}}(n)} = n^{\log_{\frac{5}{4}}(5)}$. Durch das Masterheorem kann man sich

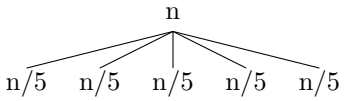
$$T(n) = 4T\Big(\frac{n}{5}\Big) + T\Big(\frac{4n}{5}\Big) + n \leq 5T\Big(\frac{4n}{5}\Big) + n = \mathcal{O}\Big(n^{\log_{\frac{5}{4}}(5)}\Big)$$

$$T(n) = 4T\Big(\frac{n}{5}\Big) + T\Big(\frac{4n}{5}\Big) + n \geq 5T\Big(\frac{n}{5}\Big) + n = \Omega(n\log(n))$$

klar machen. Wir wollen uns das ganze auch noch mit dem Rekursionsbaum ansehen, aus diesem erhält man (mit entsprechenden Vereinfachungen) für die obere Schranke:

$$T(n) = \sum_{i=0}^{\log_{\frac{5}{4}}(n)-1} \Big(\frac{8}{5}\Big)^i n + n^{\log_{\frac{5}{4}}(5)} = \mathcal{O}\Big(n^{\log_{\frac{5}{4}}(5)}\Big)$$

Für die untere Schranke betrachten wir



Hier haben alle Ebenen den Aufwand n, die Tiefe ist $\log_5(n)$ und die Anzahl der Blätter $5^{\log_5(n)} = n$. Somit

$$T(n) = \sum_{i=0}^{\log_5(n)-1} n + n = \Omega(n\log n)$$

Hier jedenfalls der Beweis. Für den Induktionsanfang wollen wir $\forall m \in \{n_0, \ldots, 5n_0 - 1\}$ $\left(T(m) \overset{!}{\leq} c_0 m^{\log_{\frac{5}{4}}(5)}\right)$ beziehungsweise $T(m) \geq c_1 m\log(m)$. Für alle $n \geq 5n_0$ wünschen wir uns

$$\begin{aligned} c_0 n^{\log_{\frac{5}{4}}(5)} \overset{!}{\geq} T(n) = 4T\Big(\frac{n}{5}\Big) + T\Big(\frac{4n}{5}\Big) + n &\leq 4c_0\Big(\frac{n}{5}\Big)^{\log_{\frac{5}{4}}(5)} + c_0\Big(\frac{4n}{5}\Big)^{\log_{\frac{5}{4}}(5)} + n \\ &\Leftarrow c_0 n^{\log_{\frac{5}{4}}(5)}\left(1 - 4\Big(\frac{1}{5}\Big)^{\log_{\frac{5}{4}}(5)} - \Big(\frac{4}{5}\Big)^{\log_{\frac{5}{4}}(5)}\right) \geq n \end{aligned}$$

wobei $\left(\frac{4}{5}\right)^{\log_{\frac{5}{4}}(5)} = \left(\frac{5}{4}\right)^{-\log_{\frac{5}{4}}(5)} = \frac{1}{5}$. Es sieht so aus als müssten wir keine Bedingungen an n_0 stellen, für c_0 wollen wir

1. $\forall m \in \{n_0, \ldots, 5n_0 - 1\} : \left(T(m) \leq c_0 m^{\log_{\frac{5}{4}}(5)}\right)$
2. $\forall n \geq 5n_0 : \left(c_0 n^{\log_{\frac{5}{4}}(5)}\left(1 - 4\left(\frac{1}{5}\right)^{\log_{\frac{5}{4}}(5)} - \left(\frac{4}{5}\right)^{\log_{\frac{5}{4}}(5)}\right) \geq n\right)$

Nun das gleiche noch für die untere Schranke.

$$\begin{aligned} c_1 n \log(n) \overset{!}{\leq} T(n) = 4T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + n &\geq c_1 \frac{4n}{5} \log\left(\frac{n}{5}\right) + c_1 \frac{4n}{5} \log\left(\frac{4n}{5}\right) + n \\ &\Leftarrow c_1 4n(\log(n) - \log(5)) + c_1 4n(\log(4) + \log(n) - \log(5)) + 5n - c_1 5n \log(n) \geq 0 \\ &\Leftarrow c_1 n(3\log(n) - 8\log(5) + 4\log(4)) + 5n \geq 0 \end{aligned}$$

Wir wünschen uns also n_0 und c_1 mit

1. $\forall n \geq n_0 : \ (3\log(n) - 8\log(5) + 4\log(4) \geq 0)$
2. $\forall m \in \{n_0, \ldots, 5n_0 - 1\} : \ (T(m) \geq c_1 m \log(m))$
3. $\forall n \geq 5n_0 : \ (c_1 n(3\log(n) - 8\log(5) + 4\log(4)) + 5n \geq 0)$

□

Aufgabe 28. Finden Sie für folgende Rekursionen $T(n)$ asymptotische untere und obere Schranken mittels Master-Theorem.

a) $T(n) = 2T(\frac{n}{3}) + n \log n$

b) $T(n) = T(\frac{8n}{9}) + n$

c) $T(n) = 11T(\frac{n}{3}) + n^{1.5}$

d) $T(n) = 4T(\frac{n}{2}) + n$

e) $T(n) = 4T(\frac{n}{2}) + n^2$

f) $T(n) = 4T(\frac{n}{2}) + n^3$

Lösung.

> **Satz 4.4** (Master-Theorem für Teile-und-herrsche-Rekursionsgleichungen)**.** *Seien $a_0, a_1 \in \mathbb{N}$ mit $a = a_0 + a_1 \geq 1$, sei $b > 1$ und sei $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Dann hat die Rekursionsgleichung*
>
> $$T(n) = a_0 T(\lfloor \tfrac{n}{b} \rfloor) + a_1 T(\lceil \tfrac{n}{b} \rceil) + f(n)$$
>
> *die folgende asymptotische Lösung.*
>
> 1. *Falls $f(n) = O(n^{\log_b a - \varepsilon})$ für ein $\varepsilon > 0$, dann $T(n) = \Theta(n^{\log_b a})$.*
> 2. *Falls $f(n) = \Theta(n^{\log_b a})$, dann $T(n) = \Theta(n^{\log_b a} \log n)$.*
> 3. *Falls es ein $c < 1$ gibt, so dass für alle bis auf endlich viele $n \in \mathbb{N}$ die Ungleichung $a_0 f(\lfloor \frac{n}{b} \rfloor) + a_1 f(\lceil \frac{n}{b} \rceil) \leq c f(n)$ gilt, dann ist $f(n) = \Omega(n^{\log_b a + \varepsilon})$ für ein $\varepsilon > 0$ und $T(n) = \Theta(f(n))$.*

a) Es ist $a = 2, b = 3$ und wir sind im Fall 3. (mit $c := \frac{2}{3}$), denn

$$\forall n \in \mathbb{N} : 2f\left(\frac{n}{3}\right) = \frac{2}{3} n \log\left(\frac{n}{3}\right) = \frac{2}{3} n (\log n - \log 3) \leq \frac{2}{3} n \log n$$

Also gilt $T(n) = \Theta(n \log n)$.

b) Es ist $a = 1, b = \frac{9}{8}$ und wir sind im Fall 3. (mit $c := \frac{8}{9}$), denn

$$f\left(\frac{8n}{9}\right) = \frac{8}{9}n$$

Also gilt $T(n) = \Theta(n)$.

c) Es ist $a = 11, b = 3$ und wir sind im Fall 1. (mit $\varepsilon := \log_3(11) - 1.5$), denn

$$\log_3(11) \approx 2.182 > 1.5 \implies \varepsilon > 0$$
$$f(n) = n^{3/2} = \mathcal{O}(n^{\log_3(11)-\varepsilon})$$

Also gilt $T(n) = \Theta(n^{\log_3(11)})$.

d) Es ist $a = 4, b = 2$ und wir sind im Fall 1. (mit $\varepsilon := 1$), denn

$$f(n) = n = \mathcal{O}(n^{\log_2(4)-1})$$

Also gilt $T(n) = \Theta(n^2)$.

e) Es ist $a = 4, b = 2$ und wir sind im Fall 2., denn

$$f(n) = n^2 = \Theta(n^2)$$

Also gilt $T(n) = \Theta(n^2 \log n)$.

f) Es ist $a = 4, b = 2$ und wir sind im Fall 3. (mit $c := \frac{1}{2}$), denn

$$4f\left(\frac{n}{2}\right) = 4\left(\frac{n}{2}\right)^3 = \frac{1}{2}n^3$$

Also gilt $T(n) = \Theta(n^3)$

□

Aufgabe 29. Gegeben ist die Divide-and-Conquer-Rekursion

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

wobei $f(n)$ eine nichtnegative Funktion mit asymptotischen Wachstum $\Theta(n^{\log_b(a)} \log(n))$ ist. Zeigen Sie für den Fall $n = b^k, k \in \mathbb{N}$, dass $T(n) = \Theta(n^{\log_b(a)} \log^2(n))$

Lösung. Wir gehen wie im Beweis des Master-Theorems vor:

$$
\begin{aligned}
T(n) &= \Theta\left(n^{\log_b(a)}\right) + \underbrace{\sum_{w \in X_I(n)} f\left(\frac{n}{b^{|w|}}\right)}_{=:g(n)} \\
g(n) &= \Theta\left(\sum_{w \in X_I(n)} \left(\frac{n}{b^{|w|}}\right)^{\log_b(a)} \log\left(\frac{n}{b^{|w|}}\right)\right) \\
&= \Theta\left(\sum_{j=0}^{\lfloor \log_b(n) \rfloor - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b(a)} \log\left(\frac{n}{b^j}\right)\right) \\
&= \Theta\left(n^{\log_b(a)} \sum_{j=0}^{\lfloor \log_b(n) \rfloor - 1} \left(\frac{a}{b^{\log_b(a)}}\right)^j \log\left(\frac{n}{b^j}\right)\right) \\
&= \Theta\left(n^{\log_b(a)} \sum_{j=0}^{\lfloor \log_b(n) \rfloor - 1} \log(n) - j\log(b)\right) \\
&= \Theta\left(n^{\log_b(a)} \lfloor \log_b(n) \rfloor \left(\log(n) - \log(b)\frac{\lfloor \log_b(n) \rfloor - 1}{2}\right)\right) \\
&= \Theta\left(n^{\log_b(a)} \lfloor \log_b(n) \rfloor^2\right) = \Theta\left(n^{\log_b(a)} \log^2(n)\right)
\end{aligned}
$$

□

Lösung. Wir schauen uns noch einen alternativen Beweis an. Dafür definieren wir für alle $k \in \mathbb{N}$ die Funktion $S(k) := T(b^k)$ und erhalten für $k \in \mathbb{N}^+$

$$S(k) = T(b^k) = aT(b^{k-1}) + f(b^k) = aS(k-1) + f(b^k)$$

Nun wollen wir $S(k) = \Theta(a^k k^2 (\log(b))^2)$ beweisen, wobei $f(b^k) = \Theta(a^k k \log(b))$. Wir wählen zuerst ein $c_0 > 0$ und ein $k_0 \in \mathbb{N}$, sodass für alle $k \geq k_0$ gilt, dass $f(b^k) \leq c_0 a^k k \log(b)$. Weiters wünschen wir uns für alle $k \geq k_0$

$$
\begin{aligned}
c_1 a^k k^2 (\log(b))^2 \overset{!}{\geq} S(k) = aS(k-1) + f(b^k) &\leq c_1 a^k (k-1)^2 (\log(b))^2 + c_0 a^k k \log(b) \\
&\Leftarrow c_1 a^k (\log(b))^2 (k^2 - (k-1)^2) \geq c_0 a^k k \log(b) \\
&\Leftarrow c_1 \geq \frac{c_0 k}{(2k-1)\log(b)}
\end{aligned}
$$

und zuletzt wollen wir noch, um beim Induktionsanfang keine Probleme zu bekommen, $S(k_0) \geq c_1 a^{k_0} k_0^2 (\log(b))^2$. Für die andere Abschätzung wollen wir $c_2 > 0$ und $k_1 \in \mathbb{N}$, sodass für alle $k \geq k_0$ die Ungleichung $f(b^k) \geq c_2 a^k k \log(b)$ gilt. Diesmal handeln wir den Induktionsanfang gleich ab und wünschen uns $c_3 > 0$ mit $S(k_0) \geq c_3 a^{k_0} k_0^2 (\log(b))^2$. Darüber hinaus wollen wir noch für alle $k > k_0$

$$
\begin{aligned}
c_3 a^k k^2 (\log(b))^2 \overset{!}{\leq} S(k) = aS(k-1) + f(b^k) &\geq c_3 a^k (k-1)^2 (\log(b))^2 + c_2 a^k k \log(b) \\
&\Leftarrow c_3 \leq \frac{c_2 k}{(2k-1)\log(b)}
\end{aligned}
$$

□

Aufgabe 30. Gegeben ist die Rekursion

$$a_n = a_{f(n)} + a_{g(n)} + a_{h(n)} + 1$$

für $n > t$ und $a_n = 1$ für $n \leq t$ mit $f, g, h : \mathbb{N} \to \mathbb{N}^+$ und $f(n) + g(n) + h(n) = n$. Beweisen Sie, dass $a_n = \Theta(n)$ gilt.

Lösung. Wir bemerken zu Beginn, dass $t \geq 2$, sonst haben wir undefinierte Werte. Als Induktionsanfang erhalten wir

$$\forall n \in \{1, \ldots, \lfloor t \rfloor\} \; (a_n = 1 \leq 2n - 1)$$

Nun zeigen wir noch den Induktionsschritt um schließlich $a_n \leq 2n - 1$ zu erhalten:

$$a_n = a_{f(n)} + a_{g(n)} + a_{h(n)} + 1 \leq 2(f(n) + g(n) + h(n)) - 3 + 1 \leq 2n - 1.$$

Untere Schranke: $a_n \geq \frac{n}{t}$:

$$a_n = a_{f(n)} + a_{g(n)} + a_{h(n)} + 1 \geq \frac{f(n) + g(n) + h(n)}{t} + 1 = \frac{n}{t} + 1 \geq \frac{n}{t}.$$

Bei der Abschätzung nach unten wollen wir für den Induktionsanfang noch

$$\forall m \in \{1, \ldots, \lfloor t \rfloor\} \left(a_m = 1 \geq \frac{m}{t}\right)$$

was für $m \leq t$ natürlich erreicht ist. Damit haben wir insgesamt $a_n = \Theta(n)$ gezeigt. □