Übungsblatt 2

Abgabe via Moodle.
Deadline Fr. 19ter Mai

Aufgabe 1 (*Rekurrenzen,* $4+4$ *Punkte*)

1. Gegeben sei folgende Rekurrenz:

$$T(n) \leq \begin{cases} 14 & \text{falls } n = 1, \\ 3n + 4 \cdot T(\lceil n/4 \rceil) & \text{falls } n > 1. \end{cases}$$

   Zeigen Sie durch vollständige Induktion, dass $T(n) \leq 20n^2 - 6n$, falls n eine Viererpotenz ist.

2. Gegeben sei folgende Rekurrenz:

$$T(n) = \begin{cases} c_0 n & \text{falls } n \leq n_0,\ n_0 > 20 \text{ geeignet gewählt}, \\ T(\lceil n/2 \rceil) + T(\frac{2}{5}n + 1) + c_1 n & \text{falls } n > n_0. \end{cases}$$

   Finden Sie eine Funktion f, so dass $T(n) = \Theta(f(n))$ gilt und beweisen Sie ihre Behauptung.

Lösung:

1. **Induktionsanfang** $n = 1$**:** $T(1) \leq 14 \quad \leq \quad 20 \cdot 1^2 - 6 \cdot 1 = 14$

   Induktionsvoraussetzung: $T(n) \leq 20n^2 - 6n$ gilt für eine Viererpotenz n

   Induktionsschluss $n \rightsquigarrow 4n$**:**
   $T(4n) \leq 4 \cdot T(n) + 3 \cdot 4n \overset{\text{IV}}{\leq} 4 \cdot (20n^2 - 6n) + 3 \cdot 4n = 80n^2 - 24n + 3 \cdot 4n = 80n^2 - 12n$
   $= \quad 5((4n)^2) - 3(4n)$

   [Für jeden Fehler wird ein Punkt abgezogen.]

2. Wähle z.B. $f(n) := n$. Offensichtlich ist $T(n) \geq \min(c_0, c_1)n$, also $T(n) = \Omega(n)$. Wir beweisen $T(n) = \mathrm{O}(n)$ durch Substitution, c wird am Ende passend gewählt.
   Induktionsanfang $n \leq n_0$**:**
   Dann ist $T(n) = c_0 n \leq cn$,
   Induktionsvoraussetzung:
   $T(k) \leq ck \quad \forall k \leq n - 1$
   Induktionsschluss, $n - 1 \rightsquigarrow n$**,** $n > 20$
   $T(n) = T(\lceil n/2 \rceil) + T(\frac{2}{5}n + 1) + c_1 n$
   $\leq c\lceil \frac{n}{2} \rceil + c(\frac{2}{5}n + 1) + c_1 n$
   $\leq c\frac{n}{2} + c + c\frac{2}{5}n + c + c_1 n$
   $= c\frac{9}{10}n + 2c + c_1 n$

   Damit garantiert ist, dass $c\frac{9}{10}n + 2c + c_1 n \leq cn$ gilt, muss c so gewählt werden können, dass gilt $c_1 n \leq c\underbrace{(\frac{n}{10} - 2)}_{>0 \text{ nötig}}$. Daher musste $n_0 > 20$ gewählt werden.

Aufgabe 2 (*Anwendung Mastertheorem,* $1+1+1+1$ *Punkte*)

Zeigen Sie mit Hilfe des Master-Theorems scharfe asymptotische Schranken für folgende Rekurrenzen:

a) $A(1) := 1$ und für $n = 2^k, k \in \mathbb{N}$: $A(n) = A(n/2) + \tilde{c}n$

b) $B(1) := 1$ und für $n = 3^k, k \in \mathbb{N}$: $B(n) = 9B(n/3) + 4n$

c) $C(1) := 1$ und für $n = 4^k, k \in \mathbb{N}$: $C(n) = C(n/4) + n + 6$

d) $D(1) := 1$ und für $n = 4^k, k \in \mathbb{N}$: $D(n) = 4D(n/4) + C(n)$

Lösung:
Master-Theorem (einfache Form): Für positive Konstanten a, b, c, d, sei $n = b^k$ für ein $k \in \mathbb{N}$.

$$r(n) = \begin{cases} a & \text{falls } n = 1 \text{ Basisfall} \\ cn + dr(n/b) & \text{falls } n > 1 \text{ teile und herrsche.} \end{cases}$$

Es gilt

$$r(n) = \begin{cases} \Theta(n) & \text{falls } d < b \\ \Theta(n \log n) & \text{falls } d = b \\ \Theta\left(n^{\log_b d}\right) & \text{falls } d > b. \end{cases}$$

a) $a = 1$, $b = 2$, $c = \tilde{c}$, $d = 1 \overset{d<b}{\Rightarrow} T(n) = \Theta(n)$

b) $a = 1$, $b = 3$, $c = 4$, $d = 9 \overset{d>b}{\Rightarrow} T(n) = \Theta\left(n^{\log_3 9}\right) = \Theta(n^2)$

c) $a = 1$, $b = 4$, $d = 1$ die Frage ist nun, wie ist c zu wählen. Dazu folgende Überlegungen:
Für $n \geq 1 : n \leq n + 6 \leq 7n$. Definiere

$$C_-(1) := 1,\ C_-(n) := C_-(n/2) + n$$
$$\text{und } C_+(1) := 1,\ C_+(n) := C_+(n/2) + 7n.$$

Dann gilt $C_-(n) \leq C(n) \leq C_+(n)$.
Mit $c = 1 \overset{d<b}{\Rightarrow} C_-(n) = \Theta(n) \Rightarrow C(n) = \Omega(n)$ und
mit $c = 7 \overset{d<b}{\Rightarrow} C_+(n) = \Theta(n) \Rightarrow C(n) = \mathrm{O}(n)$, zusammen $C(n) = \Theta(n)$.

d) $a = 1, b = 4, d = 4$, wie in der Teilaufgabe c ist ist für die Wahl der Konstanten c eine genauere Betrachtung notwendig.
Da $C(n) = \Theta(n)$ existieren c_1, c_2, n_0, so dass $c_1 n \leq C(n) \leq c_2 n \quad \forall n \geq n_0$. Definiere also

$$D_-(1) := 1,\ D_-(n) := 4D_-(n/2) + c_1 n$$
$$\text{und } D_+(1) := 1,\ D_+(n) := 4D_+(n/2) + c_2 n.$$

Dann gilt $D_-(n) \leq D(n) \leq D_+(n) \quad \forall n \geq n_0$.
Mit $c = c_1 \overset{d=b}{\Rightarrow} D_-(n) = \Theta(n \log n)$ und
mit $c = c_2 \overset{d=b}{\Rightarrow} D_+(n) = \Theta(n \log n)$, zusammen $D(n) = \Theta(n \log n)$.

Aufgabe 3 (*Invarianten,* $5+3$ *Punkte*)

1. Das Merging Problem ist folgendermaßen definiert:
   Gegeben: zwei aufsteigend sortierte Arrays $A[1..n_1]$, $B[1..n_2]$ von natürlichen Zahlen
   Gesucht: das aufsteigend sortierte Array $C[1..(n_1 + n_2) =: n]$ von natürlichen Zahlen, dass genau die Zahlen von A und B enthält

Der folgende Algorithmus löst das Problem:

1: **procedure** *merge*(A : Array $[1..n_1]$ of $\mathbb{N}_{\geq 0}$, B : Array $[1..n_2]$ of $\mathbb{N}_{\geq 0}$)
2: **precondition** $A[i] \leq A[j] \quad \forall i \leq j$ mit $i, j \in \{1, ..., n_1\}$
3: **precondition** $B[i] \leq B[j] \quad \forall i \leq j$ mit $i, j \in \{1, ..., n_2\}$
4: $A[n_1 + 1] := \infty$, $B[n_2 + 1] := \infty$
5: $n := n_1 + n_2$
6: $j_A := 1$, $j_B := 1$;
7: **for** $i := 1$ **to** n **do**
8: $C[i] = \min(A[j_A], B[j_B])$
9: **if** $A[j_A] < B[j_B]$ **then**
10: $j_A = j_A + 1$
11: **else**
12: $j_B = j_B + 1$
13: **invariant** $C[1..i]$ enthält genau $A[1..j_A - 1],\ B[1..j_B - 1]$
14: **invariant** $B[k] \leq A[j_A] \quad \forall k \in \{1..j_B - 1\}$, $A[k] \leq B[j_B] \quad \forall k \in \{1..j_A - 1\}$
15: **invariant** $C[1..i]$ ist sortiert
16: **assert** $j_A = n_1 + 1, j_B = n_2 + 1$
17: **postcondition** $C[i] \leq C[j] \quad \forall i \leq j, \quad i, j \in \{1, ..., n\}$
18: **postcondition** $C[1..n]$ enthält genau $A[1..n_1], B[1..n_2]$
19: **return** C

Beweisen sie die Korrektheit des vorgegebenen Algorithmus in dem Sie die vorgegebenen Invarianten und Assertions beweisen. Beweisen Sie außerdem, dass der vorgegebene Algorithmus linearen Zeitverbrauch hat.

2. Gegeben sei $n \in \mathbb{N}$. Geben Sie einen iterativen Algorithmus an, der $n!$ berechnet und beweisen Sie die Korrektheit Ihres Algorithmus über eine Invariante.

Lösung:

1. **Laufzeit:** Jeder Aufruf der Schleife in Zeile 6 benötigt konstant viel Zeit. Da die Schleife von 1 bis n läuft ist daher die Laufzeit offensichtlich $\Theta(n)$.
Korrektheit: Wir zeigen zunächst per Induktion die Invarianten und leiten daraus dann die Korrektheit unseres Algorithmus ab.

   - *Invariante in Zeile 13:*
   Induktionsanfang $i = 1$: Es wird genau eins der Arrays ausgewählt. Danach wird vom ausgewählten Array der Pointer um eins erhöht. D.h. im Fall $A[j_A] < B[j_B]$ wird j_A zu 2 und j_B bleibt 1. Dann gilt die Behauptung. Der andere Fall geht analog.
   Induktionsvoraussetzung: $C[1..i-1]$ enthält genau $A[1..j_A - 1], B[1..j_B - 1]$
   Induktionsschluss $i - 1 \rightsquigarrow i$: Im Fall $A[j_A] < B[j_B]$ wurde $A[j_A]$ hinzugefügt und j_A wurde um eins erhört. Also gilt die Behauptung. Der andere Fall gilt analog.

   - *Invariante in Zeile 14:*
   Induktionsanfang $i = 1$: zu dem Zeitpunkt befindet sich nur ein Element $C[1]$ im Array C. Wegen Zeile 8 ist $C[1] = \min(A[1],\ B[1])$. Im Fall $C[1] = A[1]$ wird j_A um eins erhöht und es gilt offensichtlich $A[1] \leq B[1]$. Die Behauptung gilt also. Der andere Fall geht analog.
   Induktionsvoraussetzung: Die Invariante ist zum Zeitpunkt $i - 1$ erfüllt.
   D.h. $B[k] \leq A[j_A] \quad \forall k \in \{1..j_B - 1\}$, $A[k] \leq B[j_B] \quad \forall k \in \{1..j_A - 1\}$
   Induktionsschluss $i - 1 \rightsquigarrow i$: Wir betrachten den Fall, dass in Zeile 8 $C[i] = A[j_A]$ gewählt wird. In diesem Fall ist $j_{A_{\text{neu}}} = j_{A_{\text{alt}}} + 1$ und $A[j_{A_{\text{alt}}}] \leq B[j_{B_{\text{alt}}}]$. Umstellen liefert $A[j_{A_{\text{neu}}} - 1] \leq B[j_{B_{\text{alt}}}]$. Insbesondere impliziert das zusammen mit der Induktionsvoraussetzung die Behauptung $A[k] \leq B[j_{B_{\text{alt}}}] \quad \forall k \in \{1..j_{A_{\text{neu}}} - 1\}$. Der andere Fall geht analog.

- *Invariante in Zeile 15:*
  Induktionsanfang $i = 1$: zu dem Zeitpunkt befindet sich nur ein Element im Array C. Also ist C sortiert.
  Induktionsvoraussetzung: Die Invariante ist zum Zeitpunkt $i - 1$ erfüllt. D.h. $C[1..i-1]$ ist sortiert.
  Induktionsschluss $i - 1 \rightsquigarrow i$: Da die beiden Arrays A, B aufsteigend sortiert sind, ist $A[k] \leq A[j_A] \quad \forall k \leq j_A$ und $B[k] \leq B[j_B] \quad \forall k \leq j_B$. Mit den vorherigen Invarianten folgt dann, $C[k] \leq A[j_A],\ C[k] \leq B[j_B] \quad \forall k \leq i - 1$. Also folgt insbesondere $C[k] \leq C[i] = \min(A[j_A],\ B[j_B]) \quad \forall k \leq i - 1$. Also ist $C[1..i]$ aufsteigend sortiert.

- *Assertion in Zeile 16:* Nach Ausführung der Schleife wurde j_A n_1 mal erhöht und j_B wurde n_2 mal erhöht. Da beide Werte mit 1 initialisiert wurden gilt die Behauptung.

Insgesamt folgt nach Ausführung der Schleife: $C[1..n]$ enthält genau $A[1..n_1]$ und $B[1..n_2]$ sowie $C[1..n]$ ist sortiert. Unser Algorithmus arbeitet also korrekt.

2. Ein möglicher Algorithmus wäre der folgende:

```
1: procedure fac(n : ℕ)
2: r = 1;
3: for i := 2 to n do
4:     r = r · i
5:     invariant r = i!
6: return r
```

Korrektheit: Wir zeigen zunächst per Induktion die Invariante und leiten daraus dann die Korrektheit unseres Algorithmus ab.

- *Invariante in Zeile 5:*
  Induktionsanfang $i = 1$: Beim ersten Durchlauf der Schleife ist $r = 1$ und $i = 2$. Also hat r nach durchlaufen der Schleife den Wert $r = 2 = 2!$
  Induktionsvoraussetzung: Die Invariant ist zum Zeitpunkt $i - 1$ erfüllt, d.h. $r = (i - 1)!$.
  Induktionsschluss $i - 1 \rightsquigarrow i$: Beim nächsten Durchlauf der Schleife wird $r := r \cdot i = (i - 1)! \cdot i = i \cdot (i - 1)! = i!$

Nach Ausführung der Schleife ist $i = n$ und damit $r = n!$. Also arbeitet unser Algorithmus korrekt.

Aufgabe P2 (*Angry Professor, optional*)

Für die praktischen Übungen verwenden wir die Plattform `www.hackerrank.com`. Hier müssen Sie sich registrieren um an den Übungen teilzunehmen. Unter dem Link

`https://www.hackerrank.com/adsi-2023`

finden die praktischen Übungen in der Form eines Programmierwettbewerbs statt.

Die zweite Challenge heißt "Angry Professor". Hierbei geht es um einen Professor, dessen Studenten sehr undiszipliniert sind und oft zu spät kommen. Er hat entschieden seine Vorlesung nur noch dann zu halten, wenn k Studenten zu Vorlesungsbeginn (Zeitpunkt 0) anwesend sind. Ihre Aufgabe ist es ein Programm zu schreiben, dass entscheidet, ob die Vorlesung stattfindet oder nicht. Hierzu gibt es folgende Eingaben:

- die Anzahl der Studenten n,
- die Grenze k, so viele Studenten müssen zu Vorlesungsbeginn anwesend sein und

- die Ankunftszeiten der Studenten in einem Array a der Länge n.

Die Ausgabe ist entweder "YES", falls die Vorlesung stattfindet, oder "NO", falls nicht.

Eine genauere Beschreibung, sowie ein Beispiel dazu finden Sie auf HackerRank.