Fakultät für Informatik
Lehrstuhl 11
Prof. Dr. Kevin Buchin
Dr. Carolin Rehs
Antonia Kalb
Erik Thordsen

Übungen zur Vorlesung

Datenstrukturen, Algorithmen und Programmierung 2

Sommersemester 2024

Übungsblatt 2

Besprechungszeit:
29.4. – 3.5.2024

Aufgabe 2.1 – O-Notation (6 Punkte)

In der Tabelle finden Sie Kombinationen von Landau-Symbolen. Ein ✓ bedeutet, dass die Relation gilt. Ein ✗ bedeutet, dass die Relation nicht gilt. Kreuzen Sie an, ob die angegeben Kombination möglich oder unmöglich ist.

Begründen Sie ihre Antworten. Orientieren Sie sich an dem Beispiel i).

	$f \in o(g)$	$f \in \mathcal{O}(g)$	$f \in \Theta(g)$	$f \in \Omega(g)$	$f \in \omega(g)$	möglich	unmöglich
i)	✓	✗	✗	✗	✓		×
ii)	✗	✓	✓	✓	✗		
iii)	✓	✓	✗	✗	✗		
iv)	✗	✓	✗	✓	✗		
v)	✗	✓	✗	✗	✓		

i) Es gilt $f(n) \in o\left(g(n)\right) \implies f(n) \notin \omega\left(g(n)\right)$, denn es kann nicht gleichzeitig $f\left(n\right) < c \cdot g\left(n\right)$ und $f\left(n\right) > c \cdot g\left(n\right)$ für alle $c \in \mathbb{R}^+$ gelten.

Es gibt nur Punkte, wenn richtig angekreuzt *und* richtig begründet wurde!

Aufgabe 2.2 – O-Notation (15 Punkte)

In der Tabelle finden Sie Paare von Funktionen $f(n)$ und $g(n)$. Bestimmen Sie für jedes Paar und jedes der Landau-Symbole, ob die angegebenen Relationen zutreffen oder nicht, indem Sie *die zutreffenden Relationen mit einem ✓ markieren* und *die nicht-zutreffenden Relationen mit einem ✗ markieren*.

#	$f(n)$	$g(n)$	$f \in o(g)$	$f \in \mathcal{O}(g)$	$f \in \Theta(g)$	$f \in \Omega(g)$	$f \in \omega(g)$
1	$1000n^2$	n^3	✓	✓	✗	✗	✗
2	$(n+1)(n-1)$	n^2					
3	$\sqrt[5]{n^2} + \sqrt[2]{n^5}$	$5n^2 + 2n + 5$					
4	$4n^3 \log_2(n)$	$5n^4 - 3n^2 + 6$					
5	n^n	$4^{n!}$					
6	$2^{n+2} + 2^n$	2^n					

Geben Sie zu jeder der angekreuzten Relationen explizite c und n_0 entsprechend der aus der Vorlesung bekannten Definition an und begründen Sie die Korrektheit.

Es gibt nur Punkte, wenn richtig angekreuzt *und* richtig begründet wurde!

Ein Beispiel für Zeile #1 befindet sich auf der nächsten Seite.

Beispiel für Zeile #1:
Konstanten müssen in der O-Notation nicht betrachtet werden. Das bedeutet, wir vergleichen n^2 mit n^3. Offensichtlich wächst n^3 deutlich schneller als n^2. Wir sagen also $f \in o(g)$.

Um das formal zu begründen, müssen wir nun zeigen, dass es für alle Konstanten c ein $n_0 \in \mathbb{N}^{>0}$ gibt, sodass $f(n) < c \cdot g(n)$ für alle $n > n_0$.

$$\begin{array}{rrcll} & f(n) & < & c \cdot g(n) & \mid \text{Definition einsetzen} \\ \Leftrightarrow & 1000n^2 & < & c \cdot n^3 & \mid n > n_0 > 0 \\ \Leftrightarrow & 1000/c & < & n & \end{array}$$

Wählen wir also $n_0 > 1000/c$, zum Beispiel $n_0 = \lceil \frac{1000}{c} \rceil + 1$, so gilt für alle $n \geq n_0$, dass $f(n) < c \cdot g(n)$, also $f \in o(g)$. Daraus folgt dann $f \in \mathcal{O}(g)\,, f \notin \omega(g)\,, f \notin \Omega(g)$ und $f \notin \Theta(g)$.

Aufgabe 2.3 – Laufzeiten (19 Punkte)

Sei ein aufsteigend vorsortiertes Array $A[1..n]$ von unterschiedlichen ganzen Zahlen gegeben. Ein Element $A[i]$ des Arrays heißt *Hauptelement*, wenn $A[i] = i$ gilt. Wir suchen einen Algorithmus, der ein Hauptelement findet und zurückgibt, wenn ein solches existiert, oder ∞ zurückgibt, wenn es keins gibt.

Hinweis: Wenn es mehrere Hauptelemente gibt, genügt es, eines davon zurückzugeben.

a) Entwerfen Sie einen Bruteforce-Algorithmus mit Laufzeit $\mathcal{O}(n)$ und beschreiben Sie ihn mit eigenen Worten. Geben Sie dann eine Implementiereung Ihres Algorithmus in Pseudocode an. *(4 Punkte)*

b) Beweisen Sie, dass Ihr Algorithmus Laufzeit $\mathcal{O}(n)$ hat. *(2 Punkte)*

c) Entwerfen Sie einen Teile-und-Herrsche Algorithmus mit Laufzeit $\mathcal{O}(\log n)$ und beschreiben Sie ihn mit eigenen Worten. Geben Sie eine Implementierung Ihres Algorithmus in Pseudocode an. *(8 Punkte)*

d) Beweisen Sie, dass Ihr Algorithmus Laufzeit $\mathcal{O}(\log n)$ hat. *(5 Punkte)*