Algorithmen und Datenstrukturen
Sommersemester 2022
Woche 2

Kevin Angele, Tobias Dick, Oskar Neuhuber,
Andrea Portscher, Monika Steidl, Laurin Wischounig

Abgabe bis 22.03.2022 23:59
Besprechung im PS am 24.03.2022

Aufgabe 1 (2 Punkte): Beweis durch Widerspruch

Beweisen Sie durch Widerspruch, dass $2^{2n} \notin \mathcal{O}(2^n)$.

Lösung:
Definition von Big-O: $f(n) \in \mathcal{O}(g(n))$ wenn $c > 0$ und $n_0 \geq 1$ sodass $f(n) \leq c \cdot g(n)$ für $n \geq n_0$.

Nehmen wir an, dass $2^{2n} \in \mathcal{O}(2^n)$. Dann muss es die Konstanten c und n_0 geben, sodass es für alle $n \geq n_0$ wahr ist, dass $2^{2n} \leq c \cdot 2^n$.

Allerdings ist $2^{2n} \leq c \cdot 2^n \Rightarrow 2^n \cdot 2^n \leq c \cdot 2^n \Rightarrow 2^n \leq c$. Es gibt aber keine Konstante c, die größer ist als jedes mögliche 2^n. Daher ist unsere anfängliche Annahme falsch und $2^{2n} \notin \mathcal{O}(2^n)$.

Aufgabe 2 (3 Punkte): Laufzeit-Komplexität von Programmen

Bestimmen Sie die Laufzeit-Komplexität der folgenden Algorithmen in Groß-O-Notation in Abhängigkeit von n.

Algorithmus 1:

```
Input: matrices a,b,c of size n x n
for i ← 0 to n − 1 do
    for j ← 0 to n − 1 do
        for k ← 0 to n − 1 do
            c[i][j] ← c[i][j] + a[i][k] + b[k][j]
        end for
    end for
end for
```

Lösung:
$\mathcal{O}(n^3)$

Algorithmus 2:

```
sum ← 0
for i ← 1 to n do
    for j ← 1 to 10000 do
        sum ← sum + j + i
    end for
end for
```

Lösung:
$\mathcal{O}(n)$

Algorithmus 3:

```
Input: array a of size n
while swapped do
    swapped ← false
    for j ← 1 to n − 1 do
        if a[j − 1] > a[j] then
            swap(a[j − 1], a[j])
            swapped ← true
        end if
    end for
end while
```

Lösung:
$\mathcal{O}(n^2)$, Algorithmus ist Bubblesort. Nach der i-ten Iteration der äußeren Schleife ist das $n-i$-größte Element an der Stelle $n-i$ und wird nicht mehr bewegt. Das heißt, dass nach n Iterationen der äußeren Schleife das Array sortiert ist, *swapped* nicht mehr auf *true* gesetzt wird und die Schleife verlassen wird. Die innere Schleife ist $\mathcal{O}(n)$. Insgesamt ist die Laufzeitkomplexität des Algorithmus $\mathcal{O}(n^2)$.

Aufgabe 3 (2 Punkte): Tilde-Approximation

Bestimmen Sie die Omega-, Theta- und Tilde-Approximation der folgenden Funktionen von n.

- $10n$
- $66n^2 + 17n^3 - 120n$
- $5 + 8\log_2 n$
- $4 \cdot 2^n + 9n^{100}$

Lösung:
$f(n) \sim g(n)$ nur dann, wenn $\lim_{x\to\infty} \frac{f(n)}{g(n)} = 1$

Ω	Θ	$\sim$
n	n	$10n$
n^3	n^3	$17n^3$
$\log n$	$\log n$	$8\log_2 n$
2^n	2^n	2^{n+2}

Aufgabe 4 (3 Punkte): Rekursion

Erstellen Sie einen rekursiven Algorithmus für die folgenden drei Funktionen. Geben Sie (Pseudo)code ab.

1. $pow(a, b)$, gibt a^b zurück. Die Funktion muss nur für natürliche Zahlen definiert sein.
2. $fib(n)$, gibt n-tes Element der Fibonacci-Folge zurück. Probieren sie eine Lösung zu finden, die eine Laufzeitkomplexität von $\mathcal{O}(n)$ hat.
3. $isPalindrome(str)$, Funktion, die bestimmt ob ein String ein Palindrom ist. Ein Palindrom ist ein Wort, das rückwärts gelesen sich selbst ergibt (z.B. "bob", "reittier", "lagerregal", "", "a").

Lösung:
See week02_ex4_solution.py