Algorithmen und Datenstrukturen, Übung 3

Marouane Soussi, Lars Happel, Mustafa Miresh

April 2022

Aufgabe 1

siehe .java Datei

Aufgabe 2

a)

1.

Der Bruch ist immer positiv, so hebt sich den absoluten Wert auf.

$\lim_{n\to\infty} \frac{\pi 3^{n/2}}{2^n} = \lim_{n\to\infty} \pi(\frac{3^{1/2}}{2})^n$.
Da $0 < \frac{3^{1/2}}{2} < 1 \implies \lim_{n\to\infty}(\frac{3^{1/2}}{2})^n = 0$.
Somit auch

$$\lim_{n\to\infty} \frac{\pi 3^{n/2}}{2^n} = 0$$

.
Es folgt dann laut Definition, dass $f \in O(g)$ und $f \notin \Omega(g)$, weil $lim > 0$ ist

2.

$$\lim_{x\to\infty} |\frac{n * log(n)}{n * log(n^{\frac{1}{3}})}| = \lim_{n\to\infty} |\frac{log(n)}{log(n^{\frac{1}{3}})}| = \lim_{n\to\infty} |\frac{log(n)}{\frac{1}{3}log(n)}| = 3$$

$0 < 3 < \infty$ also: $f(n) \in \Theta g(n)$

3.

$$\lim_{n\to\infty} |\frac{\frac{log(n)}{n}}{\frac{\sqrt{n}}{n}}| = \lim_{n\to\infty} |\frac{log(n)}{\sqrt{n}}|$$

Setzen wir für $\sqrt{n} = t \implies n = t^2$. Außerdem wenn n gegen ∞ geht, dann geht auch t gegen ∞. Wenn wir das umschreiben, erhalten wir :

$$\lim_{t\to\infty} |\frac{log(t^2)}{t}| = \lim_{t\to\infty} 2|\frac{log(t)}{t}|$$

Wir erhalten also einen Standard Grenzwert von log die gegen 0 konvergiert. Somit auch das Ganze konvergiert gegen 0.

$$\lim_{t\to\infty} \frac{log(t)}{t} = 0$$

Das heißt, dass auch

$$\lim_{n\to\infty} |\frac{\frac{log(n)}{n}}{\frac{\sqrt{n}}{n}}| = 0$$

Also daraus folgt dass

$$f \in O(g)$$

b)

Sei $g(n) = \frac{1}{n}$. Behauptung: Es existiert ein $f(n) \in o(g)$.
Beweis: Es gilt $lim_{n\to\infty}|\frac{f(n)}{g(n)}| = 0 \Leftrightarrow f \in o(g)$. Es wird also ein f gesucht, so dass $lim_{n\to\infty}|\frac{f(n)}{g(n)}| = 0$.
Sei $f(n) = \frac{1}{n^2}$, dann gilt $\frac{f(n)}{g(n)} = \frac{\frac{1}{n^2}}{\frac{1}{n}} = \frac{n}{n^2}$.
Da $lim_{n\to\infty}\frac{n}{n^2} = 0$ ist also $f \in o(g)$

Aufgabe 3 - Aktienkurs

a)

Eine Möglichkeit wäre, zwei Zeiger durch das Array laufen zu lassen und für jede mögliche Kombination von 2 Tagen den Gewinn zu berechnen. Am Ende gibt man den höchsten gefundenen Wert zurück.

```
max := 0 // Bisheriges Maximum merken
for i = 1 to n-1:
    for j = i+1 to n: // 2 Zeiger um jede Kombi abzudecken
        diff = A[j] - A[i] // Möglichen Gewinn ausrechnen
        if diff > max: // Ersetzen falls besser
            max = diff
```

Die Laufzeit wäre hierbei $O(n^2)$ da für jede Position des vorderen Zeigers (oben i), der hintere Zeiger (oben j) noch einmal das Array von i bis n durchlaufen muss.

b)

Input: Ein Array A[1 .. n] mit positiven Zahlen die Aktienkurse repräsentieren sollen. Output: Der größtmögliche Gewinn, also gegeben zwei Indexe a und b, mit $a < b$, die Werte für a und b, so dass $A[b] - A[a]$ maximal ist.
Pseudocode für MaxWin:

```
MaxWin(A, p, r, t1, t2)
    q := Math.floor((p+r)/2) // Mittleren Index finden
    n1 := q-p+1
    n2 := r-q
    for i:= 1 to n1:
        A1[i] := A[p+i]
    for j:= 1 to n2:
        A2[j] := A[q+j+1]
    A1[n1+1]=Integer.MAX_VALUE A2[n2+1]=Integer.MIN_VALUE
    win = Max(A2) - Min(A1) // Möglichen Gewinn aus Kauf in 1. Hälfte
                            // und Verkauf in 2. Hälfte finden
    return Max(t1, t2, win)
```

Die Laufzeit ist $O(n*log(n))$ da Aufgrund von Divide Conquer das Array in jedem Rekursionsschritt halbiert wird. Jedoch muss für die Ermittlung des Gewinns zwischem 1. und 2. Hälfte das Minimum bzw. das Maximum jeder Arrayhälfte gesucht werden, was für jedes Subarray jeweils in $O(n)$ geschieht.