ALP3 Übung 3

Tutor: Fabian Halama

Leon Preusker & Jannik Hein & Sebastian Bünger

8. November 2018

1. (a) Sortierung:
   $\log_2 n < \sqrt{n} < n < n(\log_2 n)^2 < n^2 < n^3 < 1,8^n < 3^n$
   Begründungen:
   • $\log_2 n < \sqrt{n}$
   $\log_2 n \leq \sqrt{n} \Leftrightarrow \log_2 n \leq n^{\frac{1}{2}}$
   Nach Regel $(\log n)^\alpha \in O(n^\beta)$, setzte $\alpha = 1, \beta = \frac{1}{2}$
   Damit ist $\log_2 n \in O(\sqrt{n})$

   •$\sqrt{n} < n$ Sei $c = 1, n_0 = 1$
   $\forall n \geq n_0 : 0 \leq \sqrt{n} = n^{\frac{1}{2}} \leq n = n \cdot c$
   Nach Def von O ist $\sqrt{n} \in O(n)$

   •$n < n(\log_2 n)^2$
   Sei $c = 1, n_0 = 2$
   $\forall n \geq n_0 : 1 \leq \log_2 n$
   $0 \leq n = n \cdot 1 \leq n \cdot (\log_2 n)^2 = n(\log_2 n)^2 \cdot c$
   Nach Def von O ist $n \in O(n(\log_2 n)^2)$

   •$n(\log_2 n)^2 < n^2$
   $n \cdot (\log_2 n)^2 \leq n \cdot n$
   $\Leftrightarrow (\log_2 n)^2 \leq n$
   Es gilt $(\log n)^\alpha \in O(n^\beta)$,
   also existiert c, n_0 so dass $\forall n \geq n_0 : 0 \leq n \cdot (\log_2 n)^2 \leq n \cdot n \cdot c$ gilt.
   Damit $n(\log_2 n)^2 \in O(n^2)$

   •$n^2 < n^3$
   Sei $n_0 = 2,\ c = 1$
   $\forall n \geq n_0 : 0 \leq n^2 = n^2 \cdot 1 \leq n^2 \cdot n = n^3 = n^3 \cdot c$
   Nach Def. von O ist $n^2 \in O(n^3)$

   •$n^3 < 1,8^n$
   $n^3 \leq 1,8^n$
   $\Leftrightarrow (\log_{1,8} 1,8^n)^3 \leq 1,8^n$
   Es gilt $(\log n)^\alpha \in O(n^\beta)$, setzte $n = 1,8^n$, $\alpha = 3$, $\beta = 1$.
   Damit ist $(\log_{1,8} 1,8^n)^3 = n^3 \in O(1,8^n)$

$\bullet 1,8^n < 3^n$ Sei $c = 1$, $n_0 = 1$
$\forall n \geq n_0 : 0 \leq 1,8^n \leq 3^n \leq 3^n \cdot c$
Nach Def von O ist $1,8^n \in O(3^n)$

(b) Wenn Computer 1000 mal schneller sind, kann man mit den Zeitkosten einer Operation heute 1000 Operationen in der Zukunft durchführen. $f(n) = n$
$f(20) \cdot 1000 = 20 \cdot 1000 = 20000 = f(20000)$
In 10 Jahren kann man Probleme von Größe $n = 20000$ lösen mit der selben Geschwindigkeit wie $n = 20$ heute.
$f(n) = n^3$
$f(20) \cdot 1000 = 20^3 \cdot 1000 = 8000000$
$\sqrt[3]{8000000} = 200$
$8000000 = 200^3 = f(200)$
In 10 Jahren kann man Probleme von Größe $n = 200$ lösen mit der selben Geschwindigkeit wie $n = 20$ heute.
$f(n) = 3^n$
$f(20) \cdot 1000 = 3^{20} \cdot 1000 = 3486784401000$
$\log_3 3486784401000 \approx 26.2877 \approx 26$
$3486784401000 \approx 3^{26} = f(26)$
In 10 Jahren kann man Probleme von Größe $n = 26$ lösen mit der selben Geschwindigkeit wie $n = 20$ heute.

2. (a) Naiver Linearer Ansatz:

```
getMinMaxLinear(arr) {
    max := arr[0]
    min := arr[0]

    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i];
        }
        if (arr[i] < min) {
            min = arr[i]
        }
    }
    return (min, max)
}
```

Komplexität: $n - 1$ Schleifendurchläufe in jedem Schleifendurchlauf 2 Vergleiche d.h. $T(n) = 2n - 2$ Vergleiche. $O(n)$

(b) Divide-and-Conquer:

```
getMinMaxDC(arr, low, high) {
    if (high - low <= 1) {
```

```
        return (Math.min(arr[low], arr[high]),
        Math.max(arr[low], arr[high]));
    }

    /* If there are more than 2 elements */
    mid = (low + high)/2;
    mml = getMinMaxDC(arr, low, mid);
    mmr = getMinMaxDC(arr, mid+1, high);

    return (Math.min(mml.min, mmr.min), Math.max(mml.max, mmr.max));
}
```

$$T(n) = \begin{cases} T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 2 & \text{für } n > 2 \\ 1 & \text{für } n = 2 \\ 0 & \text{für } n = 1 \end{cases}$$

3. Siehe java-code