University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithms and Datastructures
Winter Term 2022
Sample Solution Exercise Sheet 2

Due: Wednesday, November 2nd, 2pm

Exercise 1: $\mathcal{O}$-notation *(9 Points)*

Prove or disprove the following statements. Use the *set definition* of the $\mathcal{O}$-notation (lecture slides week 2, slides 11 and 12).

(a) $2n^3 + 4n^2 + 7\sqrt{n} \in \mathcal{O}(n^3)$ *(1 Point)*

(b) $n \cdot \log_3(n) \in \omega(n \cdot \log_5(n))$ *(2 Points)*

(c) $2^n \in o(n!)$ *(2 Points)*

(d) $2\log_2\left(n^2\right) \in \Omega((\log_2 n)^2)$ *(2 Points)*

(e) $\max\{f(n), g(n)\} \in \Theta(f(n) + g(n))$ for non-negative functions f and g. *(2 Points)*

Sample Solution

(a) True. Choose $n_0 = 1$ and $c = 13$. For all $n \geq n_0$ we have $n^3 \geq n^2 \geq \sqrt{n}$ and hence $2n^3 + 4n^2 + 7\sqrt{n} \leq 13n^3 = cn^3$.

(b) False. Consider some $c > \frac{1}{\log_5(3)}$. Then for all n we have $n \cdot \log_3(n) = n \cdot \frac{\log_5(n)}{\log_5(3)} < c \cdot n \cdot \log_5(n)$.

(c) True. For $n \geq 2$ we have

$$(n-1)! = (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \geq 2^{n-2}$$

and hence

$$4(n-1)! \geq 2^n$$

Let $c > 0$. Choose $n_0 = \max\left\{2, \left\lceil \frac{4}{c} \right\rceil\right\}$. Then for all $n \geq n_0$ we have

$$2^n \leq 4(n-1)! \leq c \cdot n \cdot (n-1)! = c \cdot n!$$

(d) False. Let $c > 0$. We have

$$\begin{array}{cccc} & 2\log(n^2) & \geq & c(\log n)^2 \\ \Leftrightarrow & 4\log(n) & \geq & c(\log n)^2 \\ \Leftrightarrow & 4 & \geq & c\log n \\ \Leftrightarrow & \frac{4}{c} & \geq & \log n \\ \Leftrightarrow & 16^{\frac{1}{c}} & \geq & n \end{array}$$

So for a given $n_0 \geq 1$ choose $n = \max\{n_0, 16^{\frac{1}{c}}\} + 1$. For this n we have $n > n_0$ but $2\log(n^2) < c(\log n)^2$.

(e) True. Choose $n_0 = 1$, $c_1 = \frac{1}{2}$ and $c_2 = 1$. For $n \geq n_0$ we have

$$c_1 \cdot (f(n) + g(n)) \leq \max\{f(n), g(n)\} \overset{f,g \geq 0}{\leq} c_2(f(n) + g(n))$$

Exercise 2: Sorting by asymptotic growth *(6 Points)*

Sort the following functions by their asymptotic growth. Write $g <_{\mathcal{O}} f$ if $g \in \mathcal{O}(f)$ *and* $f \notin \mathcal{O}(g)$. Write $g =_{\mathcal{O}} f$ if $f \in \mathcal{O}(g)$ *and* $g \in \mathcal{O}(f)$ (no proof needed).

$\sqrt{n}$	2^n	$n!$	$\log(n^3)$
3^n	n^{100}	$\log(\sqrt{n})$	$(\log n)^2$
$\log n$	$10^{100}n$	$(n+1)!$	$n \log n$
$2^{(n^2)}$	n^n	$\sqrt{\log n}$	$(2^n)^2$

Sample Solution

$$\sqrt{\log n} <_{\mathcal{O}} \log(\sqrt{n}) =_{\mathcal{O}} \log n =_{\mathcal{O}} \log(n^3) <_{\mathcal{O}} (\log n)^2 <_{\mathcal{O}} \sqrt{n} <_{\mathcal{O}} 10^{100}n <_{\mathcal{O}} n \log n$$
$$<_{\mathcal{O}} n^{100} <_{\mathcal{O}} 2^n <_{\mathcal{O}} 3^n <_{\mathcal{O}} (2^n)^2 <_{\mathcal{O}} n! <_{\mathcal{O}} (n+1)! <_{\mathcal{O}} n^n <_{\mathcal{O}} 2^{(n^2)}$$

Exercise 3: Stable Sorting *(5 Points)*

A sorting algorithm is called stable if elements with the same key remain in the same order. E.g., assume you want to sort the following tuples with respect to their integer key:

$$[(3, \text{"}blue\text{"}), (1, \text{"}green\text{"}), (1, \text{"}red\text{"}), (7, \text{"}gray\text{"}), (4, \text{"}yellow\text{"}), (3, \text{"}orange\text{"}), (4, \text{"}white\text{"}), (3, \text{"}black\text{"})]$$

A *stable* sorting algorithm must generate the following output:

$$[(1, \text{"}green\text{"}), (1, \text{"}red\text{"}), (3, \text{"}blue\text{"}), (3, \text{"}orange\text{"}), (3, \text{"}black\text{"}), (4, \text{"}yellow\text{"}), (4, \text{"}white\text{"}), (7, \text{"}gray\text{"})]$$

A sorting algorithm is not stable (with respect to the sorting key) if it outputs, e.g., the following:

$$[(1, \text{"}red\text{"}), (1, \text{"}green\text{"}), (3, \text{"}black\text{"}), (3, \text{"}blue\text{"}), (3, \text{"}orange\text{"}), (4, \text{"}yellow\text{"}), (4, \text{"}white\text{"}), (7, \text{"}gray\text{"})]$$

(a) Give an example that shows that `QuickSort` is not stable. *(1 Point)*

(b) Describe a method to make any *comparison-based* sorting algorithm stable, without changing the *asymptotic* runtime. Explain. *(4 Points)*

Sample Solution

(a) Consider as input the array $[x, y, z, w]$ with $x.key = 1$, $y.key = z.key = 2$ and $w.key = 0$ and assume x is taken as pivot. In the first divide step, y and w are swapped (i.e., we first get $[x, w, z, y]$ and then the pivot will be swapped s.t. the array looks like $[w, x, z, y]$) and the array is divided into $[x, w]$ and $[z, y]$. Recursive sorting yields $[x, w]$ and $[z, y]$ and thus $[w, x, z, y]$ will be returned. So y and z have been swapped.

(b) Add the number i to the key of the i-th element in the array (i.e., set $A[i].key = (A[i].key, i)$). Now run the given (non-stable) sorting algorithm according to the lexicographic ordering[1] on this new set of keys. That is, we sort according to the original keys and use the index in A as tie breaker.

Changing the keys takes time $O(n)$. Additionally, each comparison between two elements is prolonged by an additional $O(1)$ steps. As any sorting algorithm takes $\Omega(n)$, the asymptotic runtime does not change.

[1] Let $(A, <_A)$ and $(B, <_B)$ be ordered sets. The lexicographic ordering $<_{lex}$ on $A \times B$ is defined by $(a, b) <_{lex} (a', b') :\Leftrightarrow a <_A a' \vee (a = a' \wedge b <_B b')$