Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement of Computer Science
Markus Püschel, David Steurer
François Hublet, Goran Zuzic, Tommaso d'Orsi, Jingqiu Ding

17 October 2022

Algorithms & Data Structures Exercise sheet 4 HS 22

The solutions for this sheet are submitted at the beginning of the exercise class on 24 October 2022.

Exercises that are marked by * are "challenge exercises". They do not count towards bonus points.

You can use results from previous parts without solving those parts.

Master Theorem. The following theorem is very useful for running-time analysis of divide-and-conquer algorithms.

Theorem 1 (Master theorem). *Let $a, C > 0$ and $b \geq 0$ be constants and $T : \mathbb{N} \to \mathbb{R}^+$ a function such that for all even $n \in \mathbb{N}$,*

$$T(n) \leq aT(n/2) + Cn^b. \tag{1}$$

Then for all $n = 2^k$, $k \in \mathbb{N}$,

- *If $b > \log_2 a$, $T(n) \leq O(n^b)$.*
- *If $b = \log_2 a$, $T(n) \leq O(n^{\log_2 a} \cdot \log n)$.*
- *If $b < \log_2 a$, $T(n) \leq O(n^{\log_2 a})$.*

If the function T is increasing, then the condition $n = 2^k$ can be dropped. If (1) holds with "$=$", then we may replace O with Θ in the conclusion.

This generalizes some results that you have already seen in this course. For example, the (worst-case) running time of Karatsuba algorithm satisfies $T(n) \leq 3T(n/2) + 100n$, so $a = 3$ and $b = 1 < \log_2 3$, hence $T(n) \leq O(n^{\log_2 3})$. Another example is binary search: its running time satisfies $T(n) \leq T(n/2) + 100$, so $a = 1$ and $b = 0 = \log_2 1$, hence $T(n) \leq O(\log n)$.

Exercise 4.1 *Applying Master theorem.*

For this exercise, assume that n is a power of two (that is, $n = 2^k$, where $k \in \{0, 1, 2, 3, 4, \ldots\}$).

a) Let $T(1) = 1$, $T(n) = 4T(n/2) + 100n$ for $n > 1$. Using Master theorem, show that $T(n) \leq O(n^2)$.

b) Let $T(1) = 5$, $T(n) = T(n/2) + \frac{3}{2}n$ for $n > 1$. Using Master theorem, show that $T(n) \leq O(n)$.

c) Let $T(1) = 4$, $T(n) = 4T(n/2) + \frac{7}{2}n^2$ for $n > 1$. Using Master theorem, show that $T(n) \leq O(n^2 \log n)$.

The following definitions are closely related to O-Notation and are also useful in running time analysis of algorithms.

Definition 1 (Ω-Notation). Let $n_0 \in \mathbb{N}$, $N := \{n_0, n_0 + 1, \ldots\}$ and let $f : N \to \mathbb{R}^+$. $\Omega(f)$ is the set of all functions $g : N \to \mathbb{R}^+$ such that $f \in O(g)$. One often writes $g \geq \Omega(f)$ instead of $g \in \Omega(f)$.

Definition 2 (Θ-Notation)**.** Let $n_0 \in \mathbb{N}$, $N := \{n_0, n_0+1, \ldots\}$ and let $f : N \to \mathbb{R}^+$. $\Theta(f)$ is the set of all functions $g : N \to \mathbb{R}^+$ such that $f \in O(g)$ and $g \in O(f)$. One often writes $g = \Theta(f)$ instead of $g \in \Theta(f)$.

Exercise 4.2 *Asymptotic notations.*

a) Give the (worst-case) running time of the following algorithms in Θ-Notation.

1) Karatsuba algorithm.
2) Binary Search.
3) Bubble Sort.

b) **(This subtask is from January 2019 exam).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	**true**	**false**
$\frac{n}{\log n} \leq O(\sqrt{n})$	☐	☐
$\log(n!) \geq \Omega(n^2)$	☐	☐
$n^k \geq \Omega(k^n)$, if $1 < k \leq O(1)$	☐	☐
$\log_3 n^4 = \Theta(\log_7 n^8)$	☐	☐

c) **(This subtask is from August 2019 exam).** For each of the following claims, state whether it is true or false. You don't need to justify your answers.

claim	**true**	**false**
$\frac{n}{\log n} \geq \Omega(n^{1/2})$	☐	☐
$\log_7(n^8) = \Theta(\log_3(n^{\sqrt{n}}))$	☐	☐
$3n^4 + n^2 + n \geq \Omega(n^2)$	☐	☐
$(*) \quad n! \leq O(n^{n/2})$	☐	☐

Note that the last claim is challenge. It was one of the hardest tasks of the exam. If you want a 6 grade, you should be able to solve such exercises.

Sorting and Searching.

Exercise 4.3 *One-Looped Sort* **(1 point)**.

Consider the following pseudocode whose goal is to sort an array A containing n integers.

Algorithm 1 Input: array $A[0 \dots n-1]$.

```
$i \leftarrow 0$
**while** $i < n$ **do**
    **if** $i = 0$ or $A[i] \geq A[i-1]$ **then**:
        $i \leftarrow i + 1$
    **else**
        swap $A[i]$ and $A[i-1]$
        $i \leftarrow i - 1$
```

(a) Show the steps of the algorithm on the input $A = [10, 20, 30, 40, 50, 25]$ until termination. Specifically, give the contents of the array A and the value of i after each iteration of the while loop.

(b) Explain why the algorithm correctly sorts any input array. Formulate a reasonable loop invariant, prove it (e.g., using induction), and then conclude using invariant that the algorithm correctly sorts the array.

Hint: *Use the invariant "at the moment when the variable i gets incremented to a new value $i = k$ for the first time, the first k elements of the array are sorted in increasing order".*

(c) Give a reasonable running-time upper bound, expressed in O-notation.

Exercise 4.4 *Searching for the summit* **(1 point)**.

Suppose we are given an array $A[1 \dots n]$ with n **unique** integers that satisfies the following property. There exists an integer $k \in [1, n]$, called the *summit index*, such that $A[1 \dots k]$ is a strictly increasing array and $A[k \dots n]$ is a strictly decreasing array. We say an array is **valid** is if satisfies the above properties.

(a) Provide an algorithm that find this k with worst-case running time $O(\log n)$. Give the pseudocode and give an argument why its worst-case running time is $O(\log n)$.

Note: Be careful about edge-cases! It could happen that $k = 1$ or $k = n$, and you don't want to peek outside of array bounds without taking due care.

(b) Given an integer x, provide an algorithm with running time $O(\log n)$ that checks if x appears in the array of not. Describe the algorithm either in words or pseudocode and argue about its worst-case running time.

Exercise 4.5 *Counting function calls in loops (cont'd)* **(1 point)**.

For each of the following code snippets, compute the number of calls to f as a function of n. Provide **both** the exact number of calls and a maximally simplified, tight asymptotic bound in big-O notation.

Algorithm 2

(a)

```
$i \leftarrow 0$
**while** $2^i < n$ **do**
    $j \leftarrow i$
    **while** $j < n$ **do**
        $f()$
        $j \leftarrow j + 1$
    $i \leftarrow i + 1$
```

Algorithm 3

(b)

```
$i \leftarrow n$
**while** $i > 0$ **do**
    $j \leftarrow 0$
    $f()$
    **while** $j < n$ **do**
        $f()$
        $k \leftarrow j$
        **while** $k < n$ **do**
            $f()$
            $k \leftarrow k + 1$
        $j \leftarrow j + 1$
    $i \leftarrow \lfloor \frac{i}{2} \rfloor$
```