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
Johannes Lengler, Gleb Novikov, Chris Wendler

14. October 2019

Algorithms & Data Structures — Exercise sheet 4 — HS 19

Exercise Class (Room & TA): _____

Submitted by: _____

Peer Feedback by: _____

Points: ____

The solutions for this sheet are submitted at the beginning of the exercise class on October 21st.

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

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

Theorem 1 (Master theorem). *Let $T : \mathbb{N} \to \mathbb{R}^+$ be a non-decreasing function such that for all $k \in \mathbb{N}$ and $n = 2^k$,*

$$T(n) \leq aT(n/2) + \mathcal{O}(n^b)$$

for some constants $a > 0$ and $b \geq 0$. Then

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

This theorem generalizes some results that you have seen in this course. For example, the running time of Karatsuba algorithm satisfies $T(n) \leq 3T(n/2) + 100n$, so $a = 3$ and $b = 1 < \log_2 3$, hence $T(n) \in \mathcal{O}(n^{\log_2 3})$. The other 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 $T(n) \in \mathcal{O}(\log n)$.

In parts a), b) and c) of the first exercise you will see some examples of recurrences that can be analyzed in $\mathcal{O}$-Notation using Master theorem. These three examples show that the bounds in Master theorem are tight.

Exercise 4.1 *Solving Recurrences* **(2 points)**.

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

a) Let $f : \mathbb{N} \to \mathbb{R}^+$ be defined as $f(1) = c$, and $f(n) = 2f\left(\frac{n}{2}\right) + d \cdot n^2$ for $n \geq 2$. Using mathematical induction show that $f(n) = 2d \cdot n^2 + (c - 2d) \cdot n$.

b) Let $f : \mathbb{N} \to \mathbb{R}^+$ be defined as $f(1) = c$, and $f(n) = 2f(\frac{n}{2}) + d \cdot n$, for $n \geq 2$. Using mathematical induction show that $f(n) = c \cdot n + d \cdot n \cdot \log_2 n$.

c) Let $f : \mathbb{N} \to \mathbb{R}^+$ be defined as $f(1) = c$, and $f(n) = 8f(\frac{n}{2}) + d \cdot n^2$, for $n \geq 2$. Using mathematical induction show that $f(n) = (c + d) \cdot n^3 - d \cdot n^2$.

d) Consider the following recursive function:

Algorithm 1 $g(m)$

```
if m > 1 then
    g(m − 1)
    g(m − 1)
    g(m − 1)
else
    f()
```

Determine the number of calls of the function f in $\mathcal{O}$-notation. Prove your result.

e) Consider the following recursive function (recall that in this exercise $n = 2^k$, where $k \in \mathbb{N}_0$):

Algorithm 2 $h(n)$

```
if n > 1 then
    f()
    h(n/2)
    f()
    h(n/2)
else
    f()
```

Using Master theorem, determine the number of calls of the function f in $\mathcal{O}$-notation.

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

Definition 1 (Ω-Notation)**.** Let $f : N \to \mathbb{R}^+$. $\Omega(f)$ is a set of all functions $g : N \to \mathbb{R}^+$ such that there exists $C > 0$ (that may depend on g) such that for all $n \in N$, $g(n) \geq Cf(n)$.

Definition 2 (Θ-Notation)**.** Let $f : N \to \mathbb{R}^+$. $\Theta(f) := \mathcal{O}(f) \cap \Omega(f)$.

As for $\mathcal{O}$-Notation, typically $N = \mathbb{N}$, but sometimes we will consider other sets, e.g. $N = \{2, 3, 4, \ldots\}$.

We will usually write $g \leq \mathcal{O}(f)$ instead of $g \in \mathcal{O}(f)$, $g \geq \Omega(f)$ instead of $g \in \Omega(f)$, and $g = \Theta(f)$ instead of $g \in \Theta(f)$.

Exercise 4.2 *Asymptotic notations.*

a) Show that $g \geq \Omega(f)$ if and only if $f \leq \mathcal{O}(g)$.

b) Describe the (worst-case) running time of the following algorithms in Θ-Notation.

1) Karatsuba algorithm.

2) Binary Search.

3) Bubble Sort.

c) **(This subtask is from January 2019 exam).** For each of the following claims, state whether it is true or false.

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

d) **(This subtask is from August 2019 exam).** For each of the following claims, state whether it is true or false.

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.

Exercise 4.3 *Search with an oracle* **(1 point)**.

Let $f : \mathbb{N} \to \mathbb{R}^+$ be a strictly increasing function such that $f(1) = 1$ and $\lim_{x\to\infty} f(x) = \infty$. You don't know anything else about f, but you can call f at any input you want and get its value. You are given some number $n \in \mathbb{N}$, your goal is to find $m \in \mathbb{N}$ such that $f(m) \leq n$, and $f(m+1) > n$ (note that for any n such m always exists). Describe a procedure of finding such m.

A trivial solution to this problem is to call f at $i = 2, 3, 4, \ldots$ until you find an i with $f(i) > n$. Then you output $m := i - 1$. This algorithm needs $\Theta(m)$ calls, where m is the *output* of the algorithm.

Find a procedure that is more efficient, i.e., that needs asymptotically strictly less than $\Theta(m)$ calls. How many calls does your solution need asymptotically?

Hint: *Start with finding some number M such that $f(M) > n$. The trivial algorithm needs $\Theta(m)$ steps to achieve this. Find a more efficient strategy.*