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, Ulysse Schaller

12. October 2020

Algorithms & Data Structures — Exercise sheet 4 — HS 20

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 19th.

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 $a, C > 0$ 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 $b > \log_2 a$, $T(n) \leq \mathcal{O}(n^b)$.*
- *If $b = \log_2 a$, $T(n) \leq \mathcal{O}(n^{\log_2 a} \cdot \log n)$.*
- *If $b < \log_2 a$, $T(n) \leq \mathcal{O}(n^{\log_2 a})$.*

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

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 $T(n) \in \mathcal{O}(n^{\log_2 3})$. 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 $T(n) \in \mathcal{O}(\log n)$.

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* **(1 point)**.

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) Consider the following algorithm:

Algorithm 1 $g(n)$

```
if n > 1 then
    for i = 1, ..., 5 do
        for j = 1, ..., n do
            for k = 1, ..., n do
                f()
    g(n/2)
    g(n/2)
else
    f()
```

The number of calls of f is given by the recurrence relation $T(1) = 1$ and $T(n) = 2T(\frac{n}{2}) + 5n^2$ for $n \geq 2$. Using mathematical induction show that $T(n) = 10 \cdot n^2 - 9n$.

b) Consider the following algorithm:

Algorithm 2 $g(n)$

```
if n > 1 then
    for i = 1, ..., 3n do
        f()
    g(n/2)
    g(n/2)
else
    f()
    f()
```

The number of calls of f is given by the recurrence relation $T(1) = 2$ and $T(n) = 2T(\frac{n}{2}) + 3n$, for $n \geq 2$. Using mathematical induction show that $T(n) = 2n + 3n \log_2 n$.

c) Consider the following algorithm:

Algorithm 3 $g(n)$

```
if n > 1 then
    for i = 1, ..., 8 do
        g(n/2)
    for i = 1, ..., 4 do
        for j = 1, ..., n do
            for k = 1, ..., n do
                f()
else
    f()
    f()
    f()
```

The number of calls of f is given by the recurrence relation $T(1) = 3$ and $T(n) = 8T(\frac{n}{2}) + 4n^2$, for $n \geq 2$. Using mathematical induction show that $T(n) = 7n^3 - 4n^2$.

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

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

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

Exercise 4.2 *Asymptotic notations.*

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

b) As a reminder, we write $g = \Theta(f)$ if $g \leq \mathcal{O}(f)$ and $g \geq \Omega(f)$ at the same time. 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. You don't need to justify your answers.

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. 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.

Exercise 4.3 *Proving an invariant* **(1 point)**.

Let $n \in \mathbb{N}$ be an odd integer. Consider the following algorithm that starts with the list of all integers from 1 to $2n$ and returns a single integer:

Algorithm 4 $A(n)$

$L \leftarrow [1, 2, \ldots, 2n]$
while $\text{length}(L) > 1$ **do**
  Choose *any* two different elements a and b in L.
  Remove a and b from L, and add $|a - b|$ to L.
return $L[1]$

Here $\text{length}(L)$ denotes the number of elements contained in the list L, and $L[1]$ denotes its first element.

The goal of this exercise is to prove that, no matter how the two elements a and b are chosen, the algorithm will never return a zero.

a) Explain briefly why $A(n)$ always terminates. How many times does it enter the **while** loop ?

b) Let $S(L) := \sum_{k \in L} k$ be the sum of all elements of L. Prove that the parity of $S(L)$ is an invariant of the algorithm, i.e. that after each iteration of the **while** loop, the value of S mod 2 is the same.

c) Deduce that $A(n)$ never returns the number 0.

Exercise 4.4 *Finding fake coins with a balance scale* **(1 point)**.

Figure 1: Balance scale.

Imagine that you are given n '1 franc' coins of which k coins are fake. The fake coins are slightly heavier than the real ones, but all fake coins have the same weight. In order to determine which coins are fake, you are allowed to use a balance scale (see. Figure 1). Using the balance scale you can determine whether the coins you put onto the left side are heavier, lighter, or the same weight as the ones you put on the right side.

a) Consider the problem with $n = 9$ and $k = 1$. Draw a decision tree (called 'Entscheidungsbaum' in the lecture) for a strategy of your choice.

b) Prove that for $k = 1$ even the best possible algorithm requires at least $\log_3(n) - 1$ comparisons to find the fake coin in the worst case.

c) For $k = 1$, provide an algorithm that finds the fake coin and requires exactly $\lceil \log_3(n) \rceil$ comparisons in the worst case.

d*) Prove that for $k \geq 1$ even the best algorithm requires at least $k \log_3(\frac{n}{k}) + (n - k) \log_3(\frac{n}{n-k}) - \mathcal{O}(\log(n))$ comparisons in the worst case.

For this exercise, you may use the so-called *Stirling approximation*

$$\ln(n!) = n \ln n - n + \Theta(\ln n)$$

without further justification.

Hint: Show using an decision tree that the number w of maximally required comparisons satisfies the inequality $3^{w+1} > \binom{n}{k}$.