Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

6th October 2016

Datenstrukturen & Algorithmen — Exercise Sheet 3 — AS 16

Hand-in: Thursday, 13th October 2016 before the start of the lecture at 10:00 in the entrance area of ML D28. Please staple all sheets together and use this sheet as the cover page. Fill out the first two fields of the form below.

Exercise class (Room & Day):	
Submitted by:	
Corrected by:	
Bonus points:	

Exercise 3.1 *Estimating Asymptotic Running Time I.*

Specify for the following code fragments (as concisely as possible) in Θ notation, how often the function f is called asymptotically depending on $n \in \mathbb{N}$. Justify your answer.

```
for(int i = 1; i <= n/3; i = i+3)
        for(int j = 1; j <= i; j = j+1)
                f();
```

```
for(int i = 1; i <= n; i = i+1) {
        for(int j = 100; j*j >= 1; j = j–1)
                f();
        for(int k = 1; k <= n; k = k*2)
                f();
}
```

```
for(int i = 1; i <= n; i = i+1)
        for(int j = 1; j*j <= n; j = j+1)
                for(int k = n; k >= 2; k = k–1)
                        f();
```

Please turn over.

Exercise 3.2 *Estimating Asymptotic Running Time II.*

Consider the following naively formulated code fragment:

```
Function $f(n)$
If $n = 0$:
    Set $a \leftarrow g(0)$. Set $b \leftarrow g(0)$. Return $a + b$.
If $n = 1$:
    Set $a \leftarrow g(1)$. Return $a$.
If $n \geq 2$:
    Set $a \leftarrow f(n-1)$.
    Set $b \leftarrow f(n-2)$.
    Set $c \leftarrow f(n-2)$.
    Set $z \leftarrow g(a+b+c)$.
    Return $z$.
```

a) Show that the function f with input $n \in \mathbb{N}_0$ calls function g at least 2^{n-1} times in total. Proceed as follows: Function $c(n)$ specifies for input n how often f calls g. Give a recursive definition of $c(n)$ and show using general induction over n that $c(n) \geq 2^{n-1}$.

b) Modify the pseudocode above such that the function computes the same value, but g is called as rarely as possible. How often is g called in your modified code?

Exercise 3.3 *Algorithm Design: Divide-and-Conquer.*

We are given a set of n objects $O_1, \ldots, O_n$. We want to decide if this set has a *majority element.* A set has a majority element if (strictly) more than half of the objects are equal. Design an algorithm for this problem. The only allowed elementary operation is the equality test, i.e. for two objects O_i and O_j you can check in $\mathcal{O}(1)$ time if they are equal. You **cannot** assume that the entries come from an ordered universe (e.g. $\mathbb{N}$), so the operator "$<$" cannot be used.

a) What would be a naive algorithm for the problem? Which running time does it have?

b) Give an algorithm that solves the problem using divide-and-conquer in time $\mathcal{O}(n \log n)$. Prove that the running time of your algorithm is in $\mathcal{O}(n \log n)$.

   Hint: You can assume that $n = 2^k$ for some $k \in \mathbb{N}$. Assume that you split the set of objects into two subsets of the same size. What can you deduce from the solutions for these subsets for the solution of your problem?

c) **Bonus question:** Is the solution from exercise b) best possible or can you give a better solution?

Please describe your algorithm clear and/or in pseudocode **with comments**, so that your idea is comprehensible. Please refrain from describing your solution in Java code.