

Data Structures and Algorithms

Exercise 2

Markus Innerebner, Romans Kasperovics
dsa(a)inf.unibz.it

Free University of Bozen - Bolzano, Italy

Thursday, March 5rd, 2009

Assignment 02

Task 1 (CLRS1-1)

For each function $f(n)$ and time t in the following table, determine the largest size n of a problem that can be solved in time t , assuming that the algorithm to solve the problem takes $f(n)$ milliseconds.

	1 sec	1 min	1 hour	1 day	1 month	1 year
$\lg n$						
\sqrt{n}						
n						
$n \ln n$						
n^2						
n^3						
2^n						
$n!$						

Exercises

Language / Time

- Exercises are taught every Thursday (except holidays)
 - 08:30-10:30 – in Italian by Markus Innerebner (room **E531**)
 - 08:30-10:30 – in English by Romans Kasperovics (room **E431**)
 - 14:00-16:00 – in German by Markus Innerebner (room **E531**)
- There are also office hours for each instructor when you can come with additional questions
 - R. Kasperovics's office hours are on Mondays, 14:00-15:00
 - M. Innerebner's office hours are on Mondays, 14:00-15:00
- **Please write an e-mail before you come**

Assignment 02

Task 2

Order the following functions according to their asymptotic complexity:

- $30 \cdot \log_2 n$
- $5 \cdot n + n^2 + 1$
- $\log_{10}^2 n$
- $2^n + 5$
- $4\sqrt{n}$
- $5^3 \cdot n$
- $3 \cdot \log_{10} n$
- $(n + 1)!$
- $4^{\log_2 n}$
- \sqrt{n}

Assignment 02

Task 3

Let A_1 be a sorted array of length n_1 . Let A_2 be an unsorted array of length n_2 . A_1 and A_2 store integers and do not contain duplicates.

- 3a) Write an algorithm that prints the integers that are present in both arrays. The asymptotic complexity of your algorithm must be less than $O(n_1 \cdot n_2)$. The arrays may not be changed and your solution may not use more than a constant amount of additional memory.
- 3b) Determine the exact running time of your algorithm.
- 3c) Determine a tight asymptotic bound for the runtime of your algorithm.
- 3d) Give at least two examples of special cases of the input data specifying the type of the special case

	Special case	A_1	A_2
1			
2			

Hints

Task 2: Ordering functions

- $f(n)$ goes before $g(n)$, if $f(n) = O(g(n))$
- if in doubts, use the definition of Big-O:
 $f(n) = O(g(n))$ iff exist constants $c > 0$ and $n_0 > 0$, such that $f(n) \leq cg(n)$ for all $n > n_0$
- if still in doubts, calculate both $f(n)$ and $g(n)$ for some very big n

Hints

Task 1: Comparison of running times

- 1 second = 1000 of milliseconds, 1 minute = 60 000 of milliseconds, ...
- if $\lg n = 1000$, $n = 2^{1000}$ (here, \lg is \log_2)
- if $\sqrt{n} = 1000$, $n = 1000000$
- ...

Hints

Task 3: Searching for duplicates

- What are the functions that are asymptotically less than $n_1 \cdot n_2$?
- How would you solve this problem in $n_1 \cdot n_2$ time? How can you optimize it knowing that one array is sorted?

```
#include <stdio.h>
#define n1 8
#define n2 8
int A1[n1] = {1,3,4,9,12,25,27,39};
int A2[n2] = {7,1,15,48,22,27,5,70};
int main() {
    // searching for duplicates
    // and printing them
    return 0;
}
```