Program Design and Data Structures, 20.0 c

Course code: 1DL201, Report code: DL201, 67%, DAG, NML,
week: 44 - 02 Semester: Autumn 2017
week: 03 - 11 Semester: Spring 2018

LAB ASSIGNMENT 6

This information is not available in English. Now showing the Swedish version.

AFTER THIS LAB YOU SHOULD BE ABLE TO...

- Have a sense of the steepness of various common complexity functions,
- understand the consequences for algorithm design, and
- prove the big-Theta value for various functions.

BEFORE THE LAB

- We will use terminology introduced in lecture 11.
- Brushing up on high school math (logarithms in particular) might be useful.

INSTRUCTIONS

- Read *the whole* assignment first.
- You should ***not*** write any Haskell code in this lab.
- Exhibit your reasoning!
- You do not need to use the computer at all; pen, paper and a calculator should suffice.
- Complete the feedback form at the end of the lab.

THE ASSIGNMENT

Problem 1: Time Impact of Larger Size

Complete Table 1 below, which shows the CPU times on three problem instances of different sizes using algorithms of the given time complexity functions (note that those functions do not have any constant factors or lower-order terms). Assume that the linear-time algorithm can solve the problem instance of size $n = 1$ in one microsecond (1 μs). Give the times in the largest unit among nanoseconds, microseconds, milliseconds, seconds, minutes, days, years, and centuries, so that the amount is at least 1. Two digits of decimal precision suffice. Only consider years of 365 days. Exhibit your reasoning!

Table 1: CPU times as the problem instance size increases

time complexity function	n = 10	n = 30	n = 60
1			
$\log_2 n$			
n			
$n * \log_2 n$			
n^2			
n^3			
2^n			
n!			

Problem 2: Size Impact of Longer Time

Complete Table 2 below, which shows the largest sizes of the inputs of problem instances that can be solved, on a given computer, in various amounts of CPU time, by using algorithms of the given time complexity functions (note that those functions do not have any constant factors or lower-order terms). So, for example, if an algorithm has linear time complexity and on can solve a problem instance of size *D* in one CPU hour, what is the size, expressed in terms of the symbolic constant *D*, of the largest problem instance that one can solve on that computer in 10 or 100 CPU hours? (A *CPU hour* is one hour of time consumed by one particular program running on a CPU; this may correspond to more than one hour of wall clock time, whose amount may vary from run to run because the CPU may be dividing its time among several programs.) Exhibit your reasoning!

Hint: the unit of measure of the complexity functions is not given, and should not be needed. In particular, you cannot assume that the unit of measure is CPU hours.

Table 2: Size of the largest problem instance solvable in a given amount of CPU time

time complexity function	largest size in 1 CPU hour	largest size in 10 CPU hours	largest size in 100 CPU hours
1	A		
$\log_2 n$	B		
$(\log_2 n)^2$	C		
n	D		
n^2	E		
n^3	F		

2^n	G
3^n	H

You can use Wolfram Alpha to help with calculations. For example, open Wolfram Alpha and enter into the box, `solve n * log_2 n = 100`, to work out the value of `n` that makes `n * log_2 n` equal 100.

Problem 3

Reflect on the consequences of your solutions to Problem 1 and 2. What do the contents of the filled-out tables teach us about algorithm design? Furthermore, do you know any algorithms that work in any of the time complexities in Problems 1 and 2? If so, are these algorithms good or bad from a time complexity point of view?

Problem 4

Determine the big-Θ value $g(n)$ for the following functions and prove $f(n) = \Theta(g(n))$. This means that you need to find positive numbers n_0, a c_1 and a c_2 such that for all $n > n_0$ it is the case that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$.

a) $f(n) = 4n$.

b) $f(n) = 5n^2 + n + 10$.

c) **(Challenge)** $f(n) = 3n^2 + n \sin n + 1000$.

WHEN YOU ARE DONE

When you are done with all problems (the explorations below are optional), raise your hand or approach an assistant to have your solution graded.

If you pass this lab at least 30 minutes early and other groups are still working on it, we ask you to **help one other group**. Do not simply share your solution with them, but try to understand the (partial) solutions that they have developed so far (which may be different from yours) and the difficulties that they have. Assist them in coming up with their own solutions. Once the group that you are helping completes the lab assignment, you may leave. (If it is still early, the group that received your help should then stay to help another group.)

EXPLORATIONS

If you have finished the lab exercises and are waiting to be graded, or if you wish to explore programming in Haskell further at any time, or want to get some extra practice, have a look at the Explorations page.

Exploration problems are optional and should only be attempted after other problems are completed.
You don't have to show these answers to the lab assistants for grading.