COMP 3804: Assignment 1

Due Date: Tuesday, October 1^{st} at 11:59PM

School of Computer Science Carleton University

Your assignment should be submitted online on Brightspace as a single .pdf file. The filename should contain your name and student number. No late assignments will be accepted.You can type your assignment or you can upload a scanned copy of it. Please, use a good image capturing device. Make sure that your upload is clearly readable. If it is difficult to read, it will not be graded.

Question 1 [10 marks]

Find a linear-time algorithm to determine, as efficiently as possible, the smallest and second smallest element in an array of n numbers.

1. After you show that the algorithm runs in linear time, establish eaxactly how many comparison your algorithm executes in worst-case. So, we are interested in the constant looking only at comparisons though.

2. Analogously, the largest and second largest element in that array can be found. What would be your algorithm?

3. Alternately, if you don't want to write a new algorithm for 2., how could you use an implementation of your algorithm from 1. to solve 2 by transforming the input appropriately and in linear time?

4. Now, using the above two algorithms (from 1. and 2.), you should be able to find the median of the elements stored in the array. How? What is the time complexity?

5. Next, using the concept of loop invariant, prove that this algorithm correctly finds the median of a set of n elements.

Question 2 [15 marks]

DT claims that he can perform the operation FindMax in an array using $o(logn)$ comparisons. (FindMax finds the largest element in an array and then deletes it.)

(a) What could you say about the number of comparisons of the most efficient sorting algorithm that is built on precisely this implementation of FindMax?

(b) Using (a) argue that DT must be lying about the time complexity of his implementation of FindMax.

Question 3 [10 marks]

Implement the two algorithms presented in class for computing Finonacci numbers, fib1 and fib2 and the direct way using the Golden Ratio.

Each time you compute a Fibonacci number, measure how many milliseconds it takes, and print this timing information. There is a Java function, System.currentTimeMillis(), which returns a long

result; call it once before you do the computation, and a second time after you do the computation; subtract to get the elapsed time. Be careful to measure only the computation time; any changes to the GUI should be made either before or after you start measuring. Do not do anything else on your computer. From time to time, your computer will do garbage collection, this may explain strange timing results.

What is the maximum value of n for which you can compute Fibonacci for fib1? How long does it take for each n that you can compute? Hand in your code and the timing results.

Question 4 [10 marks]

(a) Write a linear-time, **recursive** algorithm to find the maximum of a set, S, having n numbers.
(b) State and solve the recurrence relation for the time complexity of your algorithm.
(c) Analyze its space complexity.
(e) Prove its correctness.

Question 5 [25 marks]

Consider a really fast computer (outside of our present model of computing) which can do the following operation, called QuickMergeandSort, in O(1) time:

QuickMergeandSort

Input: 2 arrays of n numbers each

Output: a sorted array containing the $2n$ input numbers in sorted order

How fast could this computer sort n^2 numbers?

Question 6 [28 marks]

Part I [20 marks]. Apply the Master's Theorem (general form), where possible, to solve the following recurrence relations and give a Θ-bound for each of them. If the Master's Theorem is not applicable, state why and then solve the recurrence using another method that we learned in class. Assume that T(1) = 1 for all recurrences.

a. $T(n) = T(n-5) + n$

b. $T(n) = T(16n/4) + n^3$.

c. $T(n) = T(9n/3) + n^2$.

d. $T(n) = 5T(n/3) + n^2$.

e. $T(n) = 27T(n/3) + n^3/\log^2 n$

f. $T(n) = 3^n T(n/3) + O(1)$

g. $T(n) = 9T(n/3) + \log n$

h. $T(n) = cT(n/c) + n$ for some positive integer c.

i. $T(n) = 3T(n/9) + n^{0.5}$.

Part II [5 marks]. What can you say about the time complexity of an algorithm whose running time is given by this recurrence $T(n) = 2T(n) + O(\sqrt{n})$?
Part III [5 marks]. Which method might be most appropriate to solve the following recurrence? Use it to give its solution. $T(n) = T(n/4) + T(3n/4) + n$. (You can assume that n is equal to some power of 4.)

End of Assignment 1.