University of Waterloo
CS240, Spring 2015
Assignment 2

Due Date: Monday, June 1, 5:00pm

Please read `http://www.student.cs.uwaterloo.ca/~cs240/s15/guidelines.pdf` for guidelines on submission. For the written problems, submit your solutions electronically as a PDF with file name a02wp.pdf using MarkUs. We will also accept individual question files named a02q1w.pdf, a02q2w.pdf, ... , a02q5w.pdf if you wish to submit questions as you complete them. Problem 5(d) is a programming problem; submit your solution electronically as a file named `report.cpp`.

There are 56 marks available. The assignment will be marked out of 50.

Problem 1 [4 marks]

Assume the order of bubble-down operation is changed in the heapify algorithm covered in class to get the following alternative heapify solution:

```
alter-heapify(A)
A: an array
1.    n ← size(A) − 1
2.    for i ← 0 to ⌊n/2⌋ do
3.        bubble-down(A, i)
```

Is the above solution correctly heapify a given array? If yes, briefly justify your answer. If no, provide a small example (of size 7) that shows the above procedure is not correct.

Problem 2 [1+6 = 7 marks]

In the minimum spanning tree problem, the input is a set of n points, with arbitrary coordinates, in the plane. The output is a set of segments that connect these points to form a connected tree, called spanning tree. The goal is to form a spanning tree in which the total length of segments in the tree is minimum. For example, consider the set of points $\{A = (0,0), B = (0,2), C = (2,0), D = (1,2), E = (-3,0), F = (-2,-2)\}$; the minimum spanning tree is formed by segments $\{(A,B),(A,C),(A,E),(B,D),(E,F)\}$ (see Figure 1).

a) What is the length of minimum spanning tree in Figure 1? **Note the change in the tree (the previously posted one was not a minimum spanning tree). We will accept correct solutions for any of the posted trees.**

b) Assume an algorithm A solves the minimum spanning tree problem. Prove that A has a time complexity of $\Omega(n \log n)$. Note that we do not make any assumption (e.g., integer coordinates) for the points that define the minimum spanning tree problem. [Hint: Consider a problem that is known to have $\Omega(n \log n)$ complexity and show that the minimum spanning tree problem cannot have a solution with better complexity.]

In other words, you can assume, in the contrary, that there is a minimum spanning tree algorithm that runs in $o(n \log n)$. You use that algorithm as a black box to solve a problem P that is known to have $\Omega(n \log n)$ complexity in $o(n \log n)$ (hence, a contradiction). The black-box returns minimum spanning tree in the form of a rooted tree; the root is the first point (at index 0) in the array of input-points. In this example, assuming point B is the first point in the array, the output will be $(B \to A), (B \to D), (A \to C), (A \to F), (F \to E)$.

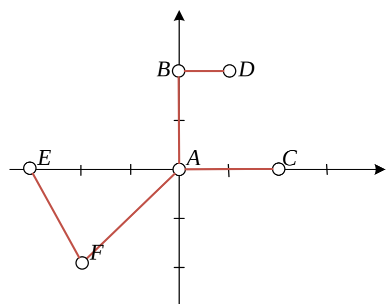

Figure 1: An example of minimum spanning tree of $n = 6$ points.

Problem 3 [4+2+2+4 = 12 marks]

Let A be an array of n distinct integers. An *inversion* is a pair of indices (i, j) such that $i < j$ and $A[i] > A[j]$.

a) Determine the maximum number (i_{max}) and minimum number (i_{min}) of inversions in an array of n distinct integers. Characterize what the arrays that attain these maxima and minima look like.

b) Given a pair of distinct indices (i, j), show that the probability that (i, j) is an inversion is $1/2$ (the average is computed over all $n!$ permutations of the n integers in A).

c) Determine the average number (i_{avg}) of inversions in an array of n distinct integers. The average is computed over all $n!$ permutations of the n integers in A (you might use the result in part (b) in your proof).

d) Suppose a sorting algorithm is only allowed to exchange *adjacent* elements. Show that its worst-case and average-case complexity is $\Omega(n^2)$ (you might use the result in the previous parts in your proof).

Problem 4 [3+3+5 = 11 marks]

Consider the selection problem for an array of n distinct integers, i.e., given an integer $k \leq n$ of numbers, we would like to report the value of the k'th smallest number. The following randomized algorithm selects a random index and checks whether its entry is the desired value. If it is, it returns the index; otherwise, it recursively calls itself.

Recall that $random(n)$ returns an integer from the set of $\{0, 1, 2, \ldots, n-1\}$ uniformly and at random.

random-select(A, n, k)

$i \leftarrow random(n)$
if $A[i]$ is the k'th smallest item **then**
    return $A[\ i\ ]$
else
    return *random-select*$(A, n,\ k)$.
end if

In your answers below, be as precise as possible. You may use order notation when appropriate. Briefly justify your answers.

a) What is the **best-case** running time of random-select?

b) What is the **worst-case** running time of random-select?

c) Let $T(n)$ be the expected running time of random-select. Write down a recurrence for $T(n)$ and then solve it.

Problem 5 [3+4+3+12 = 22]

A clever student (let's call her Sara) thinks she can avoid the worst-case behaviour of Quicksort by employing the following pivot-selection procedure. First, compute the mean $\bar{n}$ of the elements in the array. Then choose as the pivot the element x of the array, such that $|x - \bar{n}|$ is minimized, i.e., pick the element closest to the average value in the array. Everything else is the same as Quicksort. She calls her modified Quicksort algorithm SaraSort.

a) Write down the recurrence for running time $T(n)$ of SaraSort. In doing so, assume x is placed at index i of the partitioned array.

b) Assume that the elements of the array form an arithmetic sequence (i.e., have the form a, $a+k$, $a+2k$, $a+3k$, $\ldots$, $a+(n-1)k$), scrambled in some order. Show that, under this distribution of array elements, SaraSort always runs in $\Theta(n \log n)$ time.

c) Unfortunately, Sara's scheme is not as clever as it looks. Give an example of an array for which SaraSort runs in $\Theta(n^2)$ time, and explain why the worst-case running time is achieved.

d) Implement SaraSort for sorting an array of numbers in increasing order. Your program should read from `cin` the size n, then the n values which form the input array, and then write to `cout` the sorted array. You may assume that every value will fit into a variable of type `double`).

Every value in the input and output should be on a separate line. So for instance if the input consists of the following lines:

```
6
12.8232
15.1312
13.1532
10.2121
3.143
12.2143
```

then your program should print out:

```
3.143
10.2121
12.2143
12.8232
13.1532
15.1312
```

Submit the code for your `main` function, along with any helper functions, in a file called `report.cpp`.