University of Waterloo
CS240 - Fall 2018
Assignment 2

Due Date: Wednesday October 3 at 5pm

Please read `http://www.student.cs.uwaterloo.ca/~cs240/f18/guidelines.pdf` for guidelines on submission. Problems 1 – 5(a) are 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. Submit your solution to 5(b) electronically as a file named `report.cpp`.

There are 57 possible marks available. The assignment will be marked out of 55.

Problem 1 [10 marks]

Starting with an empty heap, show the max-heap resulting from insertion of 28, 37, 55, 31, 22, 40, 7. Show the heap, drawn as a binary tree, after each insert operation. Then, perform three `deleteMax` operations, and show the heap, drawn as a binary tree, after each operation.

Problem 2 [5+5+5=15 marks]

Consider a very keen teaching assistant. As students arrive during office hour she assigns a priority number to each student corresponding to their time of arrival and inserts the pair (time, name) into a priority queue, implemented as a min-heap. The next student is determined with a call to `deleteMin`. The student then has five minutes to ask as many questions as possible before being re-inserted in the queue with the current time as priority.

Occasionally students leave before their turn comes up. Describe how to perform a delete key operation in a heap under each of the following assumptions:

a) The input for the delete operation is the student name only, assumed to be unique.

b) The input for the delete operation is the priority value in the heap, again assumed to be unique.

c) The input to the delete operation is the index of the entry in the array where the key resides.

In each case the priority queue is implemented as a heap using an array. After the delete operation the array should still be a heap. For each implementation discuss the time complexity of the operation in the worst case. The running time of all of your methods should be $O(n)$, and at least one of the methods should have running time $o(n)$.

Problem 3 [10 marks]

A sorting algorithm is said to sort *in place* if only a constant number of elements of the input are ever stored outside the array. But suppose are given an array $A[0 \ldots n-1]$ that contains a permutation of the first n non-negative integers. Allowing non comparison based algorithms, give an $O(n)$ in place algorithm to sort A. Analyze the the running time of your method. *Note:* For simplicity we are assuming A is filled with integer keys. Your algorithm must easily extend to work for an array A that is filled with (key,element) pairs, each integer key in the range $0 \ldots n-1$ occurring exactly once.

Problem 4 [4+6=10 marks]

A *deterministic* algorithm is one whose execution depends only on the input. By contrast, the execution of a *randomized* algorithm depends also on some randomly-chosen numbers. A *Las Vegas* randomized algorithm always produces the correct answer, but has a running time which depends on the random numbers chosen (randomized quick-select and quick-sort are of this type). Informally, such algorithms are always correct, and probably fast. A *Monte Carlo* randomized algorithm has running time independent of the random numbers chosen, but may produce an incorrect answer. Informally, such algorithms are always fast, and probably correct.

Given an array A of length n, an element x is said to be *dominant* in A if x occurs at least $\lfloor n/2 \rfloor + 1$ times in A. That is, copies of x occupy more than half of the array.

a) [4 marks] Given an array A that contains a dominant element, describe an in-place Monte Carlo randomized algorithm to find the dominant element. Show that your algorithm has worst-case running time $O(1)$ and returns the correct answer with probability at least 1/2.

b) [6 marks] Given an array A that contains a dominant element, describe an in-place Las Vegas randomized algorithm to find the dominant element. Show that your algorithm always returns the correct answer, and has expected-case running time $O(n)$.

Problem 5 [6+6=12 marks]

You are given an array $A[0 \ldots n-1]$ of integers (not necessarily distinct) that forms a max-heap of size n.

a) Describe an algorithm that takes as input an integer c, not necessarily in the heap, and reports all integers in the heap that are greater than or equal to c. The running time of your algorithm should be $O(k)$, where k is the number of integers reported. Provide a brief explanation for why the running time of your algorithm is $O(k)$.

b) Implement your algorithm from part (a). Your program should read from `cin` the size n, then the n integers in the heap $A[0 \ldots n-1]$, and finally the integer c, and then write to `cout` the integers in the heap that are greater than or equal to c. You may assume that every integer in the input is at least 0 and at most $2^{31} - 1$ (so every integer will fit into a variable of type `int`).

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

```
5
17
15
13
10
3
12
```

then your program should print out the integers `17`, `15`, and `13` in any order (and on separate lines).

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