University of Waterloo
CS240 - Spring 2020
Assignment 2

Due Date: Wednesday June 17, 5pm

The integrity of the grade you receive in this course is very important to you and the University of Waterloo. As part of every assessment in this course you must read and sign an Academic Integrity Declaration before you start working on the assessment and submit it **before the deadline of June 17** along with your answers to the assignment – i.e. **read, sign and submit A02-AcInDe.txt now or as soon as possible**. The agreement will indicate what you must do to ensure the integrity of your grade. If you are having difficulties with the assignment, course staff are there to help (provided it isn't "last minute").

The Academic Integrity Declaration must be signed and submitted on time or the assessment will not be marked.

Please refer to the course webpage for guidelines on submission. Submit your written solutions electronically as a PDF with file name a2Submit.pdf using MarkUs. We will also accept individual question files named a2q1.pdf, a2q2.pdf, . . . if you wish to submit questions as you complete them. There is a programming question in this assignment; don't forget to submit your completed code, named OnlineMedian.cpp.

Problem 1 [4 marks]

You are given a positive integer n of the form $n = 2^h - 1$, for some integer $h \geq 1$. Give an example of an array of length n where the first method of building a max-heap seen in class (using fix-ups) uses $\Omega(n \log(n))$ element comparisons. Give a brief justification.

Problem 2 [4 marks]

Suppose we are working with a max heap, represented as an array A, and we want to remove an element from it that is not necessarily the root. We are given the index i of this element in the array. Describe an algorithm that performs this task, give the pseudo-code, analyse its complexity in terms of the number n of elements in the heap, and briefly justify correctness. Note: if your algorithm runs in time $\Omega(n)$ in the worst case, you will not receive full marks.

Problem 3 [1+3+4+5 marks]

We want to implement a *double-ended priority queue.* This is a collection where we can insert elements, access the minimum or maximum and remove the minimum or maximum; our goal is do to insert, delete-min, delete-max in time $O(\log(n))$ (if there are n elements in the collection) and find-min, find-max in constant time.

To support these requirements, we use two heaps H_1 (a min-heap) and H_2 (a max-heap) stored as arrays. At all times, these heaps should contain the same set of elements, but stored in a different order. You should also use two arrays T_1 and T_2 that specify the correspondence between the elements of H_1 and H_2: $T_1[i]$ gives the index of $H_1[i]$ in H_2, and conversely $T_2[i]$ gives the index of $H_2[i]$ in H_1.

1. Explain how to implement find-min and find-max, and justify the runtime.
2. Give an algorithm (pseudo-code or English) to update the arrays T_1 and T_2 if we swap the elements of indices i and j in H_1. Give (and justify) the cost of this operation, and a brief justification.
3. Give an algorithm (pseudo-code or English) to insert a new key in the data structure. Give (and justify) the cost of this operation, and a brief justification.
4. Give algorithms (pseudo-code or English) to do delete-min and delete-max. Give (and justify) the cost of these operations, and a brief justification.

Problem 4 [1+3+4+4 marks]

Consider n key/value pairs $(k_0, v_0), \ldots, (k_{n-1}, v_{n-1})$. We assume they are stored in an array A of length n, with array indices ranging from 0 to $n-1$, but we do not know in what order: all we know is that for $i = 0, \ldots, n-1$, we have $A[i] = (k_{\sigma(i)}, v_{\sigma(i)})$, for some permutation σ of $\{0, \ldots, n-1\}$. We consider the problem of finding a key k in A, using the following algorithm:

```
find($k$, $A$)
1: $n$ = length($A$)
2: for $i = 0, \ldots, n-1$ do
3:     if $k = A[i]$.key then
4:         return $A[i]$
5:     end if
6: end for
7: return "not found"
```

In the questions below, we are interested in the *average* cost of finding k, where the average is taken over the $n!$ permutations σ of $\{0, \ldots, n-1\}$ (k is fixed when we compute such an average). For the cost analyses, we count only key comparisons, that is, the tests we do at step 3 of the algorithm.

1. Show all possible arrays A for $n = 3$.
2. From now on, n is arbitrary (don't assume $n = 3$ anymore). Suppose that k is not in the array, that is, for all $i \in \{0, \ldots, n-1\}$, $k \neq k_i$. What is the average number of key comparisons we do? Give the exact number (and justify your answer).

3. Suppose now that $k = k_i$ for some indices $i \in \{0, \ldots, n-1\}$. We do not assume that the keys k_i are distinct, so there may be several such i: to be precise, we assume that $k = k_i$ holds for exactly s indices i.

   Take in j in $\{0, \ldots, n-s\}$. Prove that there are exactly
$$\binom{n-s}{j} j! s(n-j-1)!$$
   permutations σ of $\{0, \ldots, n-1\}$ for which in the correspbonding array A,

   - $A[0].\mathrm{key} \neq k$, ..., $A[j-1].\mathrm{key} \neq k$
   - $A[j].\mathrm{key} = k$

   Prove that for $j > n-s$, there are no such permutations.

   How many key comparisons do we do for these permutations?

4. Still under the assumption of the previous question, give the average number of key comparisons we do, in terms of n and s (we want the exact value). We recommend you use the following equality (or equalities)
$$\sum_{j=0}^{n-s}(j+1)\binom{n-j-1}{s-1} = \sum_{j=0}^{n-s}(j+1)\frac{(n-j-1)!}{(n-j-s)!(s-1)!} = \binom{n+1}{s+1},$$
   which you don't have to prove (the first one is trivial, the second one not so much). You can do this questions without completing the previous one.

Problem 5 [3+4+4+20 marks]

The *median* of a sequence $(a_0, \ldots, a_{n-1})$ of integers is defined as follows: assume we sort these integers, and write them as $(a'_0, \ldots, a'_{n-1})$ once sorted. Then their median is $a'_{\lfloor n/2 \rfloor}$. For instance:

- if $n = 1$, the median of (2) is 2
- if $n = 2$, the median of $(5, 1)$ is 5
- if $n = 3$, the median of $(2, 10, 1)$ is 2
- if $n = 4$, the median of $(5, 5, 2, 1)$ is 5, etc

Even though we sort the sequence to *define* the median, it is possible to avoid using any sorting algorithm to *compute* it: for instance, quickselect finds the median of a sequence of length n in average time $O(n)$.

In this problem, we study an *online* algorithm for finding the median of a sequence: we suppose that we receive the entries of the sequence one at a time, and we want to print the medians of all these partial sequences as we go. For instance:

- suppose we first receive **15**. We print **15**, which is the median of the sequence (15)
- next, we receive **10**. We print **15**, which is the median of the sequence $(15, 10)$
- next, we receive **1**. We print **10**, which is the median of the sequence $(15, 10, 1)$
- next, we receive **20**. We print **15**, which is the median of the sequence $(15, 10, 1, 20)$
- next, we receive **30**. We print **15**, which is the median of the sequence $(15, 10, 1, 20, 30)$

Re-computing the median from scratch every time would be too slow. Here is an idea for a better algorithm: use two heaps H_{lo} and H_{hi}, each of which will roughly contain half of the elements seen so far: if we have seen n elements, H_{lo} should contain the $\lfloor n/2 \rfloor$ smallest elements, H_{hi} should contain the $\lceil n/2 \rceil$ largest ones.

1. On the example $(15, 10, \dots)$ above, show us what these heaps would contain at each of the 5 steps (we don't know if these are min-heaps or max-heaps yet, so just tell us what elements they contain).
2. We would like to be able to read off the (current) median using just one access to H_{hi}. What kind of heap should it be, a min-heap or a max-heap? Explain how to find the current median (English of pseudo-code), and give the runtime of this operation, assuming we already have n elements in the collection.
3. Describe how to update the two heaps when inserting the next element (English of pseudo-code). In particular, in which heap do you insert the element, and how do you ensure that H_{lo} and H_{hi} have the required size afterwards? Give the runtime of your update method, with a short justification; it should be $o(n)$. (At this stage, you will have to explain whether H_{lo} should be a min-heap or a max-heap.)
4. Implement heaps and online median; you should of course not use any pre-existing code that would make this problem trivial, such as STL or boost heaps. Using C++ vectors is authorized. The implementation should of course feature the $o(n)$ runtime you obtained in the previous question.

   We give you (on the assignment webpage) a file `OnlineMedian.cpp`. It contains a class `OnlineMedian` that has two public methods (`void insert(int x)`, `int getMedian()`) and a constructor, and a `main`. You can compile it, but notice that the bodies of some functions have been filled with dummy code, just to make compilation possible. You should add to this file all that is needed in order to be able to run the algorithm. In particular, you should create classes for heaps, change `insert`, `getMedian` and the constructor, but do not change `main`. Then, submit your complete `OnlineMedian.cpp` file.

   The `main` method reads lines from `cin` until it sees an EOF (you can assume that each line contains a single integer written in base 10) and displays the current median after each integer is parsed. We also provide a sample input / output (you can pipe `input1.txt` into your program; once everything is implemented, you should obtain what is in `output1.txt`).