University of Waterloo
CS240 - Spring 2023
Assignment 2

Due Date: Wednesday June 7, 5pm

You should have submitted AID01 before the due date of this assignment. 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 read `https://student.cs.uwaterloo.ca/~cs240/s23/assignments.phtml#guidelines` for guidelines on submission. **Each question must be submitted individually to MarkUs as a PDF** with the corresponding file names: a2q1.pdf, a2q2.pdf, ... It is a good idea to submit questions as you go so you aren't trying to create several PDF files at the last minute.

Late Policy: Assignments are due at 5:00pm, with the grace period until 11:59pm. Assignments submitted after 11:59pm on the due date will not be accepted but may be reviewed (by request) for feedback purposes only.

Question 1 [10 marks]

Starting with an empty heap, show the max-heap resulting from insertion of 20, 4, 24, 100, 0, 50. 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.

Question 2 [5 marks]

Suppose we are working with a max heap, represented as an array, 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.

Question 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.

(a) Explain how to implement find-min and find-max, and justify the runtime.

(b) Give an algorithm to update the arrays T_1 and T_2 if we swap the elements of indices i and j in H_1 (we do not worry whether swapping these two elements breaks the heap condition in H_1). Give (and justify) the cost of this operation, and a brief justification of its correctness.

(c) Give an algorithm to insert a new key in the data structure. Give (and justify) the cost of this operation, and a brief justification of correctness.

(d) Give algorithms to do delete-min and delete-max. Give (and justify) the cost of these operations, and a brief justification of correctness (you may want to use your algorithm of Question 2 here).

Question 4 [1+4+5 marks]

We want to design a variant of MergeSort that divides the array A into k subarrays rather than two, with the division taking place at indices n/k, $2n/k$, …, $(k-1)n/k$. Here, k is a fixed constant, which we will take of the form $k = 2^s$ (s integer), to simplify some calculations.

In this problem, *we only count comparisons between elements in our arrays.* We will deviate slightly from the asymptotic point of view that hides constants: we want to understand the impact of the choice of k, so we will be a bit more explicit in the estimates.

(a) With $k = 2$, show that you can merge two sorted arrays A_1 and A_2 into a single sorted array A using at most n element comparisons, if we assume that A_1 and A_2 both have length $n/2$ (you can assume n even).

(b) More generally, for k of the form $k = 2^s$, show that you can merge k sorted arrays $A_1, \dots, A_k$ into a single sorted array A using at most $ns = n\log_2(k)$ element comparisons, if we assume that $A_1, \dots, A_k$ all have length n/k (you can assume n is a multiple of k).

(c) Give the variant of MergeSort that subdivides the input array into k subarrays, and give an exact (but simple) upper bound on the number of element comparisons it does (for $k = 2$, you should obtain $n \log_2(n)$). You can assume that n is a power of k.