University of Waterloo
CS240 - Spring 2025
Assignment 2

Due Date: Tuesday June 3, 5pm

Please read the following link for guidelines on submission:

`https://student.cs.uwaterloo.ca/~cs240/assignments.phtml#guidelines`

Late Policy: Assignments are due at 5:00pm, with a grace period until 8:00pm.

Question 1 [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 worst-case complexity in terms of the number n of elements in the heap (give a big-O), and briefly justify correctness (you can take for granted that the algorithms for insert and delete in a heap are correct without reproving them, of course). Note: if your algorithm runs in time $\Omega(n)$ in the worst case, you will not receive full marks.

Question 2 [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. Big-Os are sufficient.

(d) Give algorithms to do delete-min and delete-max. Give (and justify) the cost of these operations, and a brief justification of correctness. Big-Os are sufficient.

Question 3 [4+2 = 6 marks, plus 6 bonus marks]

We are going to analyse a recursive algorithm for selection (Call it R-QUICK-SELECT). On input a size n array $A[0..n-1]$ of distinct integers entries and k in $\{0,\ldots,n-1\}$, we want to return what would be the entry $A[k]$ if A was sorted in increasing order.

The algorithm proceeds as follows: if $n = 1$, we are done. Else, we call CHOOSE_PIVOT(A) = R-QUICK-SELECT($A[0..2M], M$). This means that we do a recursive call with inputs $A[0..2M]$ and M, for $M = M(n)$ a given function of n that takes values in $\{0,\ldots,\lfloor(n-2)/2\rfloor\}$ (note: with $0 \le M \le \lfloor(n-2)/2\rfloor$, we always have $0 \le 2M \le n-2$). The result of this recursive call is a value v (CHOOSE_PIVOT(A)), and we use v as pivot-value to partition A. Then, if needed, do a second recursive call as in the algorithm on the left/right part of A. For function $M(n)$, you can assume the time is constant.

(a) Assume that the function $M(n)$ has been chosen, and let $T(n)$ be the worst-case runtime of this algorithm, for all possible inputs A of size n (with distinct entries) and all possible values of k. Prove that $T(n) \le T(2M(n)+1) + T(n-M(n)-1) + cn$, for some constant c. You can assume that $T(n)$ is non-decreasing.

(b) For this question and the next one, use sloppy recurrences. Find the value of $M(n)$ (as a function of n) that balances the two terms in the right-hand side of the recurrence, and write the (sloppy) recurrence for this choice of $M(n)$.

(c) (bonus) Solve the recurrence you got to find a big-O estimate for $T(n)$. Being sloppy is OK, but you can't use results on the analysis of algorithms that have not been proved in class.

Question 4 [3+2+2+4 = 11 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 and minimum number of inversions in an array of n distinct integers. Explain what the arrays that attain these maxima and minima look like.

(b) Given a pair of distinct indices (i,j), determine the number of permutations for which (i,j) is an inversion (do not just give the number; we want a justification).

(c) Determine the average number of inversions in an array of n distinct integers. The average is computed over all $n!$ permutations of the n integers in A. Hint: indicator variables indexed by i,j.

(d) Give the average runtime of insertion sort (as a Theta bound), assuming that in size n you run it over possible permutations of $1,\ldots,n$.

Question 5 [6 marks]

Consider the following algorithm, where random(2) returns either 0 or 1, both with probability 1/2. What is the (worst case) expected number of lines that this algorithm prints in total? (assume each print statement at step 5 uses exactly one line) Give a $\Theta(\ )$ expression (if you end up with a recurrence relation, you can use the sloppy version).

```
algo(A)
A[1..n]: array of size n
1.  if n = 0 then
2.      return
3.  end if
4.  for i from 1 to n do
5.      print A[i]
6.  end for
7.  r = random(2)
8.  if r = 0 then
9.      algo(A[1..⌊n/2⌋])
10.     algo(A[1..⌊n/2⌋])
11. else
12.     algo(A)
13. end if
```