University of Waterloo
CS240 - Spring 2024
Assignment 2

Due Date: Tuesday June 4, 5pm

Please read the following link for guidelines on submission:

`https://student.cs.uwaterloo.ca/~cs240/s24/assignments.phtml#guidelines`

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.

Question 1 [4 marks]

Starting with an empty heap, show the max-heap resulting from insertion of 20, 4, 200, 100, 0, 50 (show the heap, drawn as a binary tree, after the last `insert` operation). Then, perform three `deleteMax` operations, and show the heap, drawn as a binary tree, after each operation.

Question 2 [4 marks]

A *stable* sorting algorithm is one in which the relative order of all identical entries is the same in the output as it was in the input. Suppose for instance that we sort pairs of keys / values (k, v), using only the key k for comparisons, and that objects $(k, v), (k, v')$ with identical keys are allowed to show up; then, if (k, v) comes before (k, v') in the input, it should come before (k, v') in the output.

Is heapsort, as seen in class, a stable sorting algorithm? If so, prove it; if not, give a counter-example.

Question 3 [2 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.

Question 4 [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 5 [2+4+5 = 11 marks]

Consider n distinct keys $k_0, \ldots, k_{n-1}$ stored in an array A of length n, with array indices ranging from 0 to $n-1$. We consider the problem of finding the first key in A that satisfies a certain condition `test`(k), using the following algorithm:

find(A)

```
1: $n$ = length($A$)
2: for $i = 0, \ldots, n-1$ do
3:     if test($A[i]$) 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 this procedure, where the average is taken over the $n!$ permutations σ of $\{0, \ldots, n-1\}$. For the cost analyses, we count only how many times we call `test` at step 3 of the algorithm.

1. Suppose that no key in the array satisfies `test`. What is the average number of tests we do? Give the exact number (and justify your answer).

2. Suppose now that there are exactly $s \leq n$ keys in A that satisfy `test`. For 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 corresponding array A,

   - `test`$(A[0]), \ldots,$ `test`$(A[j-1])$ are all false
   - `test`$(A[j])$ is true

   How many tests do we do for these permutations?

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

3. 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 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.

Question 6 [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 $A[i] > A[j]$.

1. 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.

2. 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).

3. 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.

4. Suppose a sorting algorithm is only allowed to exchange *adjacent* elements $(i, i + 1)$. Show that its worst-case and average-case complexity is $\Omega(n^2)$; you should use some results of the previous parts in your proof. Can you name such an algorithm?

   Hint: what happens to the number of inversions when you perform such an exchange?