University of Waterloo
CS240 Winter 2024
Assignment 2

Due Date: February 13 at 5:00pm

Please read `https://student.cs.uwaterloo.ca/~cs240/w24/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.

Question 1 [2+3+5=10 marks]

a) Let array $A = [6, 2, 10, 9, 4, 11, 27]$. Show the resulting array after applying heapify to A. No explanation is necessary.

b) Let array H of size 2^k store a permutation of integers in $\{1, .., 2^k\}$, where k is integer larger than 1. Array H is also a valid max-heap. What is the largest key that can be stored in a leaf of the heap? Explain.

c) You are given:

- Algorithm $median(A)$ which takes an array A of size n and returns its median element. For example, if $A = [6, 1, 3]$ then $median(A)$ returns 3, and if $A = [6, 2, 4, 5]$ then $median(A)$ returns 4. The running time is $O(n)$.
- Algorithm $partition(A, x)$ which takes an array A of size n and number x and partitions A in such a way that all elements smaller than x are placed at smaller indexes than all elements that are larger than x. For example, $[1, 3, 6]$ is a valid output for $partition([6, 1, 3], 5)$ and $[1, 2, 3, 6]$ is a valid output for $partition([6, 1, 3, 2], 2)$. The running time is $O(n)$.

Design algorithm $CS240Heap$, which given an array A of size n rearranges all the elements in such a way that if we view array A as a min heap, *any* key at level $i - 1$ is smaller than *any* key at level i. For example, if the input array is $A = [5, 3, 8, 1, 4, 2]$, one possible valid output of $CS240Heap$ is $A = [1, 3, 2, 5, 4, 8]$. The running time of $CS240Heap$ must be $O(n)$. You can assume $n = 2^k - 1$ for some integer k, i.e. the last level of the heap is full.

Question 2 [2+3+2+4 = 11 marks]

a) Let H be a heap of height h. How many key comparisons must be performed by $deleteMax()$ in the worst case? Give the exact, not asymptotic, answer (i.e. the answer should be a function of h, for example, $2 + h$, or h^2, as appropriate). Describe the worst case instance where this number of comparisons is achieved.

b) Consider method $deleteMaxFaulty(H)$, implemented as follows. First we remove the item at the root node and store it in a variable to be returned to the user. Then we compare the items at the two children of the root. Whichever item is larger is moved to the root. We repeat this last step at the node of the deleted item until we reach a leaf. The pseudocode for is below.

```
deleteMaxFaulty(H)
H: is an array storing max heap
i ← 0
toReturn ← H[0]
while i is not a leaf do
    if i has right child and i has left child
        H[right child of i].key > H[left child of i].key
            H[i] = H[right child of i].key
            i ← right child of i
        else
            H[i] = H[left child of i].key
            i ← left child of i
    else
        H[i] = H[left child of i].key
        i ← left child of i
H.size ← H.size − 1
return toReturn
```

Explain why $deleteMaxFaulty()$ does not perform deletion of the maximum element correctly.

c) How many key comparisons does $deleteMaxFautly()$ perform in the worst case? Give the exact, not asymptotic, answer.

d) Develop a method $fixAfterDeleteMaxFaulty()$, s.t. if we run it after $deleteMaxFaulty()$, the result is a valid heap. The maximum number of key comparisons allowed for $fixAfterDeleteMaxFaulty()$ is $\log h$. You can modify $deleteMaxFaulty()$ to return more items in the output in addition to the deleted maximum element. You do not have to write pseudo-code, you can describe your algorithm in sufficient detail. Note that we are **not** asking for the runtime of $fixAfterDeleteMaxFaulty()$ to be $O(\log h)$, but putting a restriction on the allowed number of key comparisons.

Question 3 [1+1+5 =7 marks]

Consider the following algorithm.

```
$Mystery(A)$
$A$: an array of size $n \geq 2$ storing distinct numbers
1.  $sum \leftarrow 0$
2.  $sorted \leftarrow true$
3.  for $i = 1$ to $n - 1$
4.      if $A[i-1] > A[i]$
5.          $sorted \leftarrow false$
6.  if $sorted$
7.      for $i = 1$ to $n^4$
8.          $sum \leftarrow sum + i * sum$
9.      return sum
10. if $A[0] > A[1]$ and $A[1] > A[2]$
11.     for $i = 1$ to $n^3$
12.         $sum \leftarrow sum + i * sum$
13.     return sum
14. return sum
```

a) What is the best-case running time? Use Θ notation. Briefly justify.

b) What is the worst-case running time? Use Θ notation. Briefly justify.

c) Derive the average-case running time. Use Θ notation. Your derivation must be based on the definition of the average-case running time (not on an equivalence of running time to some randomized version of $Mystery(A)$).

Question 4 [5 marks]

Consider the algorithm below, where *random(k)* returns an integer from the set of $\{0, 1, 2, \ldots, k-1\}$ uniformly and at random.

```
$ArrayAlg(A)$
$A$: an array storing numbers
1.  if $A.size == 1$
2.      return
3.  $i \leftarrow random(A.size)$
4.  for $j = 0$ to $i^2$ do
5.      ~~print($A[j]$)~~ print($A[i]$)
6.  $ArrayAlg(A[0, ..., A.size - 2])$
```

Let $T^{exp}(n)$ be the expected running time of $ArrayAlg$ of size n. Derive a tight recursive upper bound for $T^{exp}(n)$ and then solve it. Express your final answer using big-O asymptotic notation. Your bound must be asymptotically tight, but you need not prove that it is tight.

Question 5 [5 marks]

Write an in-place partition algorithm called $ModuloPartition(A)$ that takes an array A of n numbers and rearranges A in such a way that all the values that are equivalent to $0 \mod 10$ precede all the values equivalent to $1 \mod 10$, which precede all the values equivalent to 2 mod 10, etc. For example, for an input array $A = [7, 62, 5, 57, 12, 39, 5, 8, 16, 48]$, one possible correct result is $A = [12, 62, 5, 5, 16, 57, 7, 8, 48, 39]$. The runtime of your algorithm must be $\Theta(n)$.

Question 6 [5 marks]

Given an array $A[0 \dots n-1]$ of numbers, show that if $A[i] \geq A[i-j]$ for all $j \geq \log n$ then the array can be sorted in $O(n \log \log n)$ time.

Hint: Partition A into contiguous blocks of size $(\log n)$; i.e. the first $(\log n)$ elements are in the first block, the next $(\log n)$ elements are in the second block, and so on. Then think about how to apply sorting to blocks.

Question 7 [2+2+2+3=9 marks]

a) Show the contents of array $A = [45, 112, 83, 8]$ after one round of LSD sort, where one round means one application of single digit bucket sort. No explanation is required.

b) Perform MSD sort on array $A = [736, 212, 213, 376, 354, 850]$. For each number, underline the digits (if any) which are not examined by MSD sort. You only need to underline the not examined digits, nothing else, and no explanation is required for your answer.

c) What is the running time of bucket-sort when sorting an array of size n^2 containing elements in the range $[0, .., n^{10}]$? Briefly explain.

d) Explain how to sort n integers in the range $[0, n^{100})$ in $O(n)$ time.