CS 344 - Spring 2021
Homework 3.
100 points total + 10 points EC

Note on 344 library For the rest of this class, you may use heaps and dictionaries on any HW or exam problem, unless I explicitly say that you can't.

1 Problem 1 (25 points)

Say there are n *sorted* arrays $A_1, A_2, ..., A_n$, each with n numbers (so n^2 numbers between all of them.) Write pseudocode that uses the heap data structure to find the n smallest numbers between all the arrays in time $O(n \log(n))$. You must make it clear why the run-time is $O(n \log(n))$ by analyzing how often you call each heap operation.

REMINDER: Heaps are now in our virtual library. For this problem, you can use a min-heap or a max-heap without explaining how the heap works.

HINT: for this problem, you will want to use that, just like in dictionaries, when you add an element to a heap you can add a (key, value) pair, where the key is the number that actually goes into the heap, while the value stores extra info about the number. So in the pseudocode you could write something like heap.add(key,value). And then heap.find-min().value would tell you the value corresponding to the minimum key in the heap.

2 Problem 2 (20 points)

Given an array A, we define a *subsequence* of A to be any sequence $A[i_1], A[i_2], ..., A[i_k]$ where $i_1 < i_2 < ... < i_k$. Note that this is different from the term subarray in HW 2, because in a subarray $A[a...b]$ all the elements must be right next to each in the other array.

Now, we say that a subsequence $A[i_1], A[i_2], ..., A[i_k]$ is *serial* if $A[i_1] = A[i_2] - 1 = A[i_3] - 2 = A[i_4] - 3 = ... = A[i_k] - k + 1$. Note that any one element on its own is a serial subsequence of length 1, so every array A always contains a serial subsequence of length at least 1. Now consider the following problem

- INPUT: unsorted array A of length n. You can assume all numbers in A are distinct.
- OUTPUT: the length of the longest serial subsequence.

For example, if $A = 1, 13, 7, 3, 8, 2, 20, 9, 4$ then the longest serial subsequence is $7, 8, 9$, so the output of the algorithm would be 3. Note that $1, 2, 3, 4$ is NOT a serial subsequence of A because 2 comes after 3 in A.

The Problem Write pseudocode for an algorithm that solves the longest serial subsequence problem in time $O(n)$.

NOTE: you only have to return of the longest serial subsequence, not the sequence itself.

3 Problem 3 (25 points)

Consider the following problem, where we are given an array A with some duplicate elements, and want to find the number of distinct elements in each interval of size k.

- Input: a positive integer k, and an unsorted A with n numbers, some of them repeating.
- Output: an array B of length $n - k + 1$, where $B[i]$ should contain the number of *distinct* elements in the interval $A[i], A[i+1], ..., A[i+k-1]$.

EXAMPLE: if A = 3,2,7,3,5,3,5,7,2 and k = 4 then we are looking at intervals of size k = 4, so B[1] = 3 because the subarray [3,2,7,3] has 3 distinct elements; B[2] = 4 because the subarray [2,7,3,5] has four distinct elements. More generally, the output array B = 3,4,3,2,3,4

The question: Write pseudocode for an algorithm for this problem with expected running time $O(n)$. Note that your expected running time should be $O(n)$ even if k is large. A run-time of $O(nk)$ is too slow, and will receive very little credit.

HINT: similar to the algorithm for sorting a k-sorted array, you will want to take advantage of the fact that two consecutive intervals $A[i]...A[i+k]$ and $A[i+1]...A[i+k+1]$ only differ by a small number of elements.

4 Problem 4 (30 points)

Let A be some array of n integers (possibly negative), with no duplicated elements, and recall that Rank(x) = k if x is the kth *smallest* element of A. Now define InverseRank(x) = n + 1 - Rank(x). It is easy to see that if InverseRank(x) = k, then x is the kth *largest* element of A.

Now, let us define a number x in A to be *special* if x = InverseRank(x). For example, if $A = -9, 8, 1, -1, 2$, then 2 is special because 2 is the 2nd largest number in the array, so InverseRank(2) = 2.

Consider the following problem:

- Input: unsorted array A of length n
- Output: return a special number x in A, or return "no solution" if none exists.

Questions:

- **Part 1 (5 points):** Give pseudocode for a $O(n \log(n))$ algorithm for the above problem.
- **Part 2 (25 points):** Give pseudocode for a $O(n)$ algorithm for the above problem. Give a brief justification for why the algorithm is correct. Make sure to analyze the running time of the algorithm.

  HINT 1: the algorithm I have in mind is recursive, and the recurrence formula is one we have seen before. Recall that if we've seen a recurrence formula before then you don't have to solve it again, you can just say what $T(n)$ ends up being.

  HINT 2: use the high-level approach of Median Recursion described in class.

  HINT 3: write pseudocode for an algorithm FindSpecial(A, offset), which looks for a number x such that x = InverseRank(x) + offset. In the initial call you just have offset = 0, but as you recurse, you will want to change the offset value. This is similar to how in Selec(A,k), when we recurse we sometimes need to change the rank k that we are searching for.

5 Problem 5 – Extra Credit 10 points

Consider the following problem:

- INPUT: An array A of length n.
- OUTPUT: A pair of indices (i, j), with $j \geq i$ such that $\sum_{k=i}^{j} = A[i] + A[i+1] + ... + A[j] = 100$, or "no solution" if no such pair of indices exists. (If there exist many such pair of indices i, j, you only have to return one of them.)

Give an $O(n)$ time algorithm for this problem.