CS 344 - FA 2022
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 (30 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 the indices do not have to be right next to each other; we could have something like $i_1 = 3, i_2 = 7$.

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 $A = 13, 7, 10, 8, 3, 1, 20, 4, 9, 2$ 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 e.g. 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 the length of the longest serial subsequence, not the sequence itself.

2 Problem 2 (30 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[0] = 3 because the subarray [3,2,7,3] has 3 distinct elements; B[1] = 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: 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.

3 Problem 3 (40 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, let us define a number x in A to be *special* if Rank(x) = -x. (Note the minus sign on the right hand sign.) For example, if $A = 1, -2, 4, 7, -5$, then -2 is special because it is the second smallest number, so Rank$(-2) = 2$, so we have Rank$(-2) = -(-2)$. Note that a non-negative number can never be special.

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 (10 points):** Give pseudocode for a $O(n \log(n))$ algorithm for the above problem.

  HINT: your solution to this one should be short. 8 lines max (you can get away with less.)

- **Part 2 (30 points):** Give pseudocode for a recurrence $O(n)$ algorithm for the above problem. Give a brief justification for why the algorithm is correct (see Hint 2 below). Make sure to also state the recurrence formula of the algorithm.

  HINT 1: the recurrence formula is one we have seen before, and we have shown in class that it solves to $T(n) = O(n)$.

  HINT 2: use the high-level approach of Median Recursion described in class. You will want to show that by looking at the median, you can figure out one side to recurse because the other cannot have special numbers *The only justification of correctness you need to include for this problem is justification of why you know for certain that there can't be special numbers on the side you don't recurse to. You don't need any other justification. A couple sentences is enough.*

  HINT 3: you will want to add an extra input parameter for this problem. In particular, you should write pseudocode for an algorithm FindSpecial(A, offset), which finds a number x such that -x = rank(x) + offset. In the initial call you have offset = 0, but as you recurse, you will sometimes 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.

4 Problem 4: finding the two lightest toys. Extra Credit – 10 points

Say that you have 1024 toys $t_1, ..., t_{1024}$. You an assume that all toys have different weights. The only measurement tool you have available is a scale: you put two toys on the scale and it shows you which is lighter, which is heavier. We call this a comparison. Your goal for this problem is to find simultaneously both the lightest toy AND the second-lightest toy.

The Problem: Show that using at most 1050 comparisons you can simultaneously find both the lightest and second-lightest toy.

NOTE: this is actually pretty surprising! One can prove that you need at least 1023 comparisons to find the lightest toy. So what this algorithm is telling is that finding the *two* lightest toys is barely harder than finding the one lightest toy.

NOTE: you can actually do a bit better than 1050.

HINT: you will want to use a recursive approach for this problem.