CS 344 - Fall 2022
Exam 1
100 points total + 10 points EC

Instructions

Academic Integrity Policy You must complete this exam entirely alone. You may not discuss the exam with any students in the class, or with anyone outside the class.

The exam is open notes, so you can look at any files/videos posted on canvas and your own notes. You may also use either of the two books recommended for this course. You are not allowed to use any other sources, *and in particular you are not allowed to use any online source.*

Academic Integrity Statement You should type the following text on the first page of your midterm.

I have not discussed this exam with anyone except Professor Aaron Bernstein. I have not used any online resources during the exam except for those accessible from the Canvass website.

What to Write

- On the first page write your full name, netID, and the academic integrity statement above. Do not write anything else.
- Start each problem on a new page.
- Typed assignments receive 2 extra points.
- If your assignment is hand-written please write very legibly.
- Assignment should be submitted as a single PDF on Canvas in the same way as a HW problem. You get infinite re-submissions on Canvas, same as with HW.

Skipping Problems Remember that if you skip a problem you get 25% credit. If you give an answer that is on the right track you will get more than 25% in partial credit, so don't be afraid to attempt problems if you think you know how to approach them. But keep in mind that if you simply don't know the answer, writing a very wrong answer may get 0%.

The only exception is the extra credit problem: you do not get any credit for leaving that blank and it's harder to get partial credit on that problem.

Suggestion: Leave the extra credit problem for last, as it is harder than the rest and worth fewer points.

The 344 Library: Throughout the test, any algorithm you design may use any of the functions or data structures we covered in class without explanation. The most important ones are as follows; details can be found in the lecture notes and class recordings

- Sort(A) returns a array with the elements of A sorted from smallest to largest element. Runtime is $O(n \log(n))$.
- Select(A,k) returns element of rank k in A. The runtime in $O(n)$. Note that Select(A,$\lceil n/2 \rceil$) returns the median of A in $O(n)$ time.
- Partition(A,x) returns arrays A_1, A_2, where A_1 contains all elements in A that are $< x$ and A_2 returns all elements in A_2 that are $> x$. Note that the number x itself does not appear in A_1 or A_2. The runtime is $O(n)$ time.
- You can use dictionaries on the exam, same as for HW 3. See lecture notes for a list of dictionary operations.

1 Problem 1 (20 points total)

For both problems below, you can assume a base case of $T(1) = T(2) = T(3) = 1$.

- Part 1 (10 points) Say you had an algorithm with recurrence formula $T(n) \leq 27T(n/3) + n^2$. Use a recursion tree (or recursion table) to solve for $T(n)$. *No need for induction on this one.*
- (10 points) Say you had an algorithm with recurrence formula $T(n) \leq 4T(n/3) + 5T(n/4) + n^4$. Use *induction* to show that $T(n) = O(n^4)$. *No need for a recursion tree on this one – only induction.*

2 Problem 2 (25 points)

Recall the Select(A,k) from class where we broke the array into chunks of size 5, found the median of each chunk, recursively computed the median of the medians and so on.

Let us say that we use a different algorithm where we break the array into chunks of size 13 rather than size 5. Other than that the algorithm is the same. So the algorithm is:

1. Break A into chunks of size 13
2. Compute the median of each chunk
3. Recursively compute the median of the chunk-medians (call it m)
4. Partition A around m
5. Recurse to one side

What is the recurrence formula for this algorithm? That is, with chunks of size 5 we showed in class that the recurrence formula is $T(n) \leq T(n/5) + T(7n/10) + O(n)$. So what is the recurrence formula if we use chunks of size 13 instead? *Make sure to explain why your recurrence formula is the correct one.* As in class, you can assume that all elements of the array A are distinct (no duplicates).

NOTE: you do NOT need to prove that the recurrence solves to $O(n)$. You do NOT need to write pseudocode for the algorithm. All you need to do is

state the recurrence formula and explain why that is the recurrence formula. The explanation should be at the level of what we did in class to show that for chunks of size 5 the reccurence formula is $T(n) \leq T(n/5) + T(7n/10) + O(n)$.

3 Problem 3 (25 points)

Consider the following problem. We are given an array $A[0...n-1]$ of positive numbers with $len(A) = n$ (duplicates allowed). We say that a subarray $A[i]...A[j]$ is *special* if *both* the following two properties hold:

- $j - i + 1 = n/5$ (i.e. the length of the subarray is $n/5$) AND
- there is some number x such that x appears *at least* 100 times in the subarray $A[i]...A[j]$.

Now consider the following problem:

FindSpecialSubarray(A)

- INPUT: an array $A[0...n-1]$ of positive numbers (duplicates allowed).
- OUTPUT: indices i, j such that subarray $A[i...j]$ is special. Or "no solution" if no special subarray exists. If there exists multiple special subarrays $A[i...j]$ you only have to return one of them.

The Problem: Write pseudocode for an algorithm that solves FindSpecial(A) in $O(n)$ time.

HINT: use a dictionary.

4 Problem 4 – 30 points

Consider the following problem

FindCutoff(A)

- INPUT: an array $A[0...n-1]$ of length n. You can assume that all numbers in A are positive and unique (i.e. no duplicates). You can NOT assume that A is sorted.
- OUTPUT: find the *largest* number x such that A contains at least x elements that are $\geq 10x$.

Example : Consider the array $A = 30, 2, 61, 45, 9, 38, 82, 40, 13, 26, 51, 77, 5$. Note that $x = 4$ works because there are in fact at least 4 numbers that are ≥ 40 (namely $61, 45, 82, 40, 51, 77$). But $x = 5$ does NOT work because there aren't 5 numbers ≥ 50 (in fact there are only 4 of them; $61, 82, 51, 77$). Thus, the correct answer is FindCutoff(A) = 4

4.1 Part 1 (5 points)

Write pseudocode for an algorithm that solves FindCutoff(A) in $O(n \log(n))$ time.

4.2 Part 2 (25 points)

Write pseudocode for an algorithm that solves FindCutoff(A) in $O(n)$ time.

HINT: For Part 2, you will want to use median recursion. Also, as with similar problems we've seen, the easiest way to solve part 2 is to use an offset variable.

In particular, the algorithm I have in mind actually solves the following more general problem. Your pseudocode should solve this more general problem FindCutoff(A,offset), which has the following input and output:

- INPUT: same array A as above and also a non-negative integer offset
- OUTPUT: largest number x such that A contains at least x numbers that are ≥ 10(x + offset).

Once you solve this more general problem with an offset variable, you can then call FindCutoff(A,0) to solve the original problem.

Suggestion: Of the non-extra-credit problems, I think this one (part 2 in particular) will be the most challenging for the majority of students. So I recommend solving all of the other problems first, and only starting problem 4 part 2 one once you are happy with your answer to all the others.

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.)

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