Monsoon 2019
Tuesday, 27 August

Graduate Algorithms: Test-I

Max: 50 poin
Duration: 80 min

Name: **Roll No.:**

no explanations necessary

Q1 [$\frac{1}{2} \times 5 + 3 \times 2 = 11$ points] Let A and B be two sorted arrays of $2n+1$ elements each. Let m_a and m_b be the media elements of A and B respectively. Consider the case when $m_a < m_b$. Since $C = A \cup B$ has even number of elements let m denote the smaller median of C (assume that A and B do not contain any common element). Give tight bound (upper or lower, as suitable) on the following; use appropriate signs from "$=$", "$\leq$", "$<$", "$\geq$", "$>$" (no need to include trivial bounds like "> 0", "$\leq 4n+2$").

(a) The rank of m in C: 2n+1

(b) For any element x in $A[1\dots n]$, the rank of x in C: rank(x) <= (4n+2)-2(n+1) = 2n

(c) For any element x in $A[n+2\dots 2n+1]$, the rank of x in C: rank(x) > n+1

(d) For any element x in $B[1\dots n]$, the rank of x in C: rank(x) <= 4n+2 - (n+1) = 3n+1

(e) For any element x in $B[n+2\dots 2n+1]$, the rank of x in C: rank(x) > 2n+2

no need to give bounds not given -- those are trivia deduct -0.5 for gi an incorrect boun

(f) For each of the subarrays, specify if m can/cannot/must belong to that subarray.

cannot $A[1\dots n]$, can $A[n+1]$, can $A[n+2\dots 2n+1]$

(g) For each of the subarrays, specify if m can/cannot/must belong to that subarray.

can $B[1\dots n]$, cannot $B[n+1]$, cannot $B[n+2\dots 2n+1]$

Q2 [3 points] Given an array $A[1\dots n]$ of distinct positive integers and an integer $i \in \{1,\dots,n\}$, give a recursive expression to compute $Solve(i)$ which is defined as the length of the longest increasing subsequence of $A[1\dots i]$ that <u>ends</u> with $A[i]$.

$Solve(i) =$ 1 + max { Solve(j) : j < i and A[j] < A[i] }

Q3 [3 points] Denote the $n \times n$ Vandermonde matrix by V_n. Write the sum of all the elements of the k-th row of $(V_n)^{-1}$.

Ans: 0

Q4 [3 points] Let $DFT(A, k, n)$ denote the recursive algorithm we studied in class for evaluation a polynomial of degree $n-1$ at all the k k-th roots of unity. Does the *same* (unmodified) algorithm work for $k = 7$ and $n = 8$? If "yes", write all calls to $DFT(poly(x), k = ?, n = ?)$ that are made (I am only interested in the values of k and n in the recursive calls). If "no", briefly explain why in the space provided.

No. Since DFT(A,7,n) is evaluation of a polynomial at 7 7-th roots of unity and it calls Odd(w^2i) for i=0 to 6 which are actually evaluations of the Odd polynomial at 7 different values. So we are not recursively solving a smaller problem.

- no marks for only yes/no answers
- ~~1 for saying no but not giving any useful answer (e.g., stating whatever is said in the question is not~~ adding any useful information)

ROUGH

- if reason is roughly "book says so"
- 2 if any reasonable explanation is given
- 3/3 if any reasonable explanation is given
- padding requires modification of the algorithm and is incorrect: 0
- correctly saying "no" but wrong reason: 0
- if answer is almost correct but has glaring mistake: 1.5-2

Q5 [5 points] Let $H(n)$ denote the number of min-heaps that can be created using the elements $\{1, 2, \ldots, 2^n - 1\}$. Note that $H(1) = 1$ since there is only min-heap possible using $\{1\}$. Give a recursive expression to compute $H(n)$.

$H(n) =$ {(2^{n})-2 choose (2^{n-1})-1} x H(n-1) x H(n-1)

If H(2^{n-1}-1) is used instead of H(n-1), give full marks (even though it is incorrect)

Given an array A of positive distinct intgers, a subsequence $\langle a_{i_1} a_{i_2} \ldots \rangle$ (where, $i_1 < i_2 < i_3 \ldots$) of A is said to be oscillating if the first pair is increasing $a_{i_1} < a_{i_2}$, the second pair is decreasing $a_{i_2} > a_{i_3}$, the third pair is increasing, the fourth pair is decreasing, and so on. For example, given a sequence 9, 16, 5, 12, 1, 7, 3, 6, 8, 4, one of the oscillating subsequences is 9, 12, 1, 7, 3, 8, 4 — observe that the first, third, etc. pairs $(9, 16), (5, 12)$ are increasing, and the second, fourth, etc. pairs $(16, 5), (12, 1)$ are decreasing. Subsequences with only one element are considered (trivially) oscillating.

For $i < j$, let $inclosat(i, j)$ denote the length of the longest osc. subseq. present in the subarray $A[i \ldots j]$ such that the first pair of the subsequence is an increasing pair ("increasing pair" is explained above) and $A[i]$ is the first element of the subsequence. $declosat(i, j)$ is defined similarly except that the first pair is decreasing.

Q6 [5+5=10 points] Give recursive expressions to compute *inclosat* and *declosat*. Do not write pseudocode/algorithm to compute those values. Cross-check with the help of an example to avoid errors. You are allowed to compute *inclosat* recursively using *inclosat* and *declosat* values of smaller subarrays (and similarly for *declosat*).

Write **"I don't know"** if you cannot answer this question and you will get 2/10.

$inclosat(i, j) =$ 1 + max { declosat(A[k .. j]) : i < k <= j, A[i] < A[k] }

$declosat(i, j) =$ 1 + max { inclosat(A[k ... j]) : i < k <= j, A[k] < A[i] }

2+3+5=10

Q7 [2+5+3=10 points] Let $T(i, j)$ denote the time-complexity to compute both $inclosat(i, j)$ and $declosat(i, j)$. Give a recursive expression for computing $T(i, j)$.

Write **"I don't know"** if you cannot answer this question and you will get 2/10.

Base case(s) (could be multiple): T(i,i+1) = 1 for i=1...n-1

Recursive expression: $T(i, j) =$ T(i+1,j) + T(i+2,j) + ... + T(j-1,j)

Solution of recurrence $T(1, n)$ =(show derivation) T(i,j) - T(i+1,j) = T(i+1,j) + c

T(i,j) = 2T(i+1,j) + c = ... (some steps) = 2^{j-i}*c

If there is no derivation for the solution, give 2-3 partial marks.

efficient

Q8 [5 points] Explain an algorithm to compute the longest oscillating subsequence of A using all the $inclosat(i, j)$ and $declosat(i, j)$ values, for all $i < j$. You are allowed to use sentinels if necessary. Explain the time complexity assuming inc dec values are already computed.

Approach A: Max { inclosat(i,n), declosat(i,n) for all i} : O(n)

Approach S: Set A[0] = 0, return inclosat(0,n) : O(1)

ROUGH

- If sentinels are used, it must be clearly mentioned that A[0]=...
- Using flag/global variables : 0
- Not explaining time complexity - no penalty (only for this time)
- Using only inclosat/declosat and getting O(n) solution as above : 3.5
- Using sentinel and giving O(1) solution as above : 5
- Algorithm recursively computes LOS (and also uses inclosat/declosat) : 3
- Algorithms that are computing LOS by looking at adjacent pairs are usually incorrect : 0
- Half-true/incorrect approach - 1.5 to 2