Winter 2024
Feb 26, 2024

Algorithm Design and Analysis: Midsem (Sec-A)

Max: 70 Marks
120 minutes

Roll No: _____ **Name:** _____

Please write solutions independent of each other. This is a closed book test. You can not use books or lecture notes. Please note that your solution must fit in the space provided. Extra sheet will be provided only *for roughwork . So try to be precise and brief. Meaningless blabber fetches negative credit.*

Part A

Question	1	2	3	Total
Marks				

Part B

Question	1	2	3	Total
Marks				

Total Marks:

Part A

1. **(5 Marks)** Suppose that your algorithm divides an input instance of size n into 2 different subproblems each of size $\lfloor n/2 \rfloor$ and 2 additional subproblems each of size $\lceil n/2 \rceil$. Then, recursively solves each of these subproblems and combines those solutions in $12n^2 + 16n$-time. If your input has size at most 4, then it solves your problem correctly using some brute-force approach with at most 20 steps. Then, what is the running time of your algorithm? Give a short explanation of your claimed running time.

2. **(10 Marks)** Recall that an algorithm to count the number of inversions in an array A of n distinct numbers that was discussed in the class. We used a subroutine **Count-Split-Inversion**(X, Y, n_x, n_y) that takes two sorted arrays X of n_x numbers and Y of n_y numbers as input and returns the number of split-inversions, i.e. $|\{(a, b) \mid a \in X, b \in Y\}|$ between X and Y in $O(n_x + n_y)$-time and outputs a sorted array Z of size $n_x + n_y$ that contains all the numbers of X and Y. We modify that algorithm and propose a different divide and conquer algorithm.

Count-Inversion(A, n)

(i) If$(n = 1)$ then (fill up here) _____;

(ii) If$(n = 2)$ then (fill up here)

(iii) If$(n \geq 3)$ then

- $t_1 \leftarrow \lfloor n/3 \rfloor$ and $t_2 \leftarrow \lceil n/3 \rceil$;
- $A_L \leftarrow A[1, \ldots, t_1]$; $A_M \leftarrow A[t_1 + 1, \ldots, t_1 + t_2]$; $A_R \leftarrow A[t_1 + t_2 + 1, \ldots, n]$;
- $(d_L, A_L) \leftarrow$ Count-Inversion(A_L, t_1);
- $(d_M, A_M) \leftarrow$ Count-Inversion(A_M, t_2);
- $(d_R, A_R) \leftarrow$ Count-Inversion$(A_R, n - t_1 - t_2)$;
- (fill up here, you can directly invoke the subroutine **Count-Split-Inversion** in an instruction. You do not have to describe it)

(iv) Since the algorithm above is recursive, what is the the running time of the above algorithm? Explain with recurrence relation. Just write down the recurrence relation and the running time.

3. **(10 Marks)** Recall the algorithm to identify the k-th smallest element from an array A of n numbers that was done in the class. Suppose that the algorithm is modified and designed using the following approach.

   Function $\mathsf{Select}(A, n, k)$ where $n = |A|$ and $k \leq n$.

   (i) If $n < 10$, then use brute-force approach and find the k-th smallest element.

   (ii) Divide A into $\lfloor n/7 \rfloor$ groups of 7 elements each. Additional elements are placed in their own group of size r such that r is the remainder after you divide n by 7.

   (iii) Find median of each of the groups (sort the elements of each group and pick the median). If there are r elements in a group, them median is the $\lfloor r/2 \rfloor$-th smallest element in that group.

   (iv) $B \leftarrow$ the array containing the medians found in Step (ii).

   (v) Recursively compute $pivot \leftarrow \mathsf{Select}(B, |B|, \lfloor |B|/2 \rfloor)$.

   (vi) Compute the array A_L that contains all elements that are smaller than *pivot*.

   (vii) Compute the array A_R that contains all elements that are larger than *pivot*.

   (viii) If $|A_L| = k - 1$, then (fill up here) _____

   (ix) If $|A_L| < k$, then (fill up here)

   (x) If $|A_L| > k$, then (fill up here)

   (a) What is the recurrence relation of the recursive algorithm described above?

(b) Explain the running time of your algorithm.

Part B

1. (10 **Marks**) You are given two sorted arrays $A[]$ and $B[]$ of positive integers. The sizes of the arrays are not given. Accessing any index beyond the last element of the arrays returns -1. The elements in each arrays are distinct but the two arrays may have common elements. An intersection point between two arrays is an element that is common to both, i.e. p be an intersection point if there is i and j such that $A[i] = B[j] = p$. Given an integer x, design an algorithm (in pseudocode) to check if x is an intersection point of A and B. You will be awarded zero marks if your algorithm

   - runs in time linear in $\max\{|A|, |B|\}$.
   - if you use any C++/Java/Python library function that returns the size of an array in a single instruction.

   (a) Pseudocode of your algorithm. The instructions of your pseudocode can be in plain text if required.

(b) Give an explanation of the running time of your algorithm.

2. **(15 Marks)** Consider an ordered sequence of balls $b_1, b_2, \ldots, b_n$ and each ball is assigned a positive weight $\mathsf{wt}(b_i)$. A subset of balls S is said to be a *competent* set if for every $i \in \{2, \ldots, n-1\}$ either $b_i \in S$ or $b_{i-1} \in S$ or $b_{i+1} \in S$. Similarly, for the ball b_1 either $b_1 \in S$ or $b_2 \in S$. Moreover, for the ball b_n either $b_n \in S$ or $b_{n-1} \in S$. The *weight* of a set S of balls is defined as $\sum_{b_i \in S} \mathsf{wt}(b_i)$. A competent set S is *optimal* if the weight of S is the smallest among all possible competent set of balls. Design an algorithm that runs in time polynomial in n and computes the weight of a competent set of balls that is optimal.

   (a) A brief description of your algorithm (if your algorithm uses dynamic programming then write down steps by steps, i.e. subproblem definition, recurrence of subproblem, the specific subproblem solving the actual problem, algorithm description, and running time explanation)

(b) Explanation of the running time of your algorithm:

3. **(20 Marks)** Suppose that you are working on a consulting business, just you, two associates and a rented equipment. Your clients are distributed in two parts of the country, north part and south part. In each month, you either run your business from an office in Delhi (DEL) or from an office in Bangalore (BLR). In the i-th month, if you run your business from DEL, then you will incur an operating cost of d_i; if you run your business from BLR, then you will incur an operating cost of r_i. However, if you run the business in one city in the i-th month and move it to another city in the $(i+1)$-th month, then you incur a fixed moving cost M to switch the base offices. Given a sequence of n months, a *plan* is a sequence of n locations - each one either to DEL or BLR - such that the i-th location indicates a city in which you will be based in the i-th month. The cost of the plan is the sum of the operating cost in each month, plus a moving cost each time you switch the city. The plan can begin in any of the two cities. A plan is *optimal* if it has minimum cost among all possible plans. Design a dynamic programming based algorithm that runs in time polynomial in n and outputs the cost of an optimal plan.

   Example: Suppose that $n = 4, M = 10$ and the operating costs are given by the following example. The values are $d_1 = 1, d_2 = 3, d_3 = 20, d_4 = 30, r_1 = 50, r_2 = 20, r_3 = 2$ and $r_4 = 2$. The plan of a minimum cost would be the sequence of locations (DEL, DEL, BLR, BLR) with total cost $d_1 + d_2 + M + r_3 + r_4 = 18$. A different plan (DEL, BLR, BLR, DEL) has cost $d_1 + M + r_2 + r_3 + M + d_4 = 73$ since this plan requires moving between city two times.

   (i) Definition of your subproblem:

   (ii) Recurrence of the subproblem:

   (iii) The subproblem that solves the final problem:

(iv) Algorithm Description:

(v) Explanation of the running time: