CS601 Algorithms and Complexity **2021 Jul-Nov**

Assignment 1

Total Marks: 30 *Deadline: August 22*

- Assignment needs to be done individually. Submit your solution on Moodle. If the solutions are hand-written, please make sure your writing is clear, and it is also clearly readable from the scan/image.
- Keep your answers succinct and to the point. Give clear description of your algorithms using English text and pseudocode, if needed. Any pseudocode needs to come with sufficient explanation, for example, what a variable supposed to store after each iteration, how the update statement is ensuring that, what a function supposed to compute and so on.
- You need to give arguments for correctness of your strategy/algorithm/observations/claims, whenever it is not obvious (to any fellow student).
- You are supposed to solve the assignment completely on your own, without any discussions with anyone. Any immoral acts will attract severe consequences.
- You don't need to necessarily follow the given hints.

Que 1 [9+1 marks]. (a) Suppose there is a trader for a particular commodity, whose license allows them to buy it only once and sell it only once during a season. Naturally, the commodity can be sold only after it is bought. The price of the commodity fluctuates every day. Once the trader buys the commodity, they have to also pay a rent on per day basis till they sell it. The trader wants to compute the maximum profit they could have made in the last season.

Design an $O(n)$-time algorithm, where the input is the list of prices $\{p_1, p_2, \ldots, p_n\}$ for the n days in the last season and a rent r (per day rate), and the output is the maximum possible profit. Assume addition, subtraction, comparison etc are unit cost. Only six marks if the running time is $O(n \log n)$. No marks for $O(n^2)$ time.

Sample input: Prices: 70, 100, 140, 40, 60, 90, 120, 30, 60. Rent per day: 15

Output: Max profit: 40

Buying at price 70, selling at 140, paying rent for the two days gives net profit of 40. Another option, for example, was buying at 40 and selling at 120, but then one has to pay the rent for three days, which means a lower profit of 35).

(b) If instead, the rent was charged as a fixed percentage of the selling price, can you still design an $O(n)$-time algorithm? Yes or No? Give one or two sentences justifying your answer. No need to give the actual algorithm.

Que 2 [9+1 marks]. (a) Suppose there is a rectangular field with multiple cameras installed at various points in the field. Let the four corners of the field be $(0,0), (N,0), (0,M), (N,M)$ for some $N, M \geq 0$. Each camera has a 90 degrees *field of view* and is oriented such that a camera at point (α, β) can cover all points in the rectangle described by $0 \leq x \leq \alpha$ and $0 \leq y \leq \beta$ (see figure 1). A point is said to be covered, if it is covered by at least one camera.

Design an algorithm that takes the list of 2-D coordinates for the n cameras as input and outputs the total area of the covered points. The running time should be $O(n \log n)$, where comparison, addition, multiplication etc are assumed to be unit cost. Only three marks if the running time is $O(n^2)$, no marks for a higher running time.

Sample input: (4,5), (3,9), (7,4), (5,7) . Output: 49 (see Figure 1)

Hint: When the target running time is $O(n \log n)$, your two friends are sorting and divide and conquer. You can take help from one of them or both.

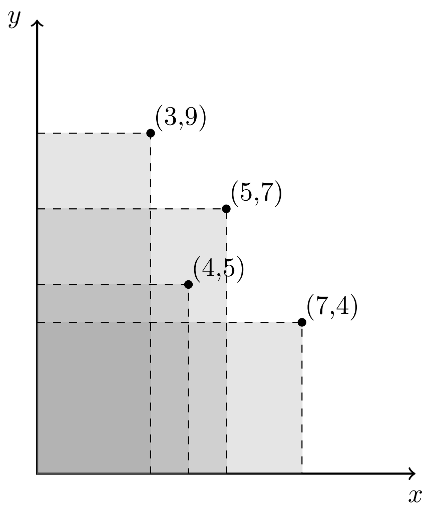

Figure 1: Shaded area is covered by the cameras.

(b) Note that if we have a camera at $(5, 7)$ then it is useless to have another camera at $(4, 5)$, because the field of view of the second camera is completely covered by the first camera. If we assume that there are no cameras at such 'useless' locations, does the problem become easier? Yes or No? Give one or two sentences justifying your answer. No need to give the algorithm.

Que 3 [9+1 marks]. (a) Design a randomized algorithm for finding the third largest element in a given array. The expected number of comparisons should be $n + O(\log n)$ for array size n and you also need to prove that.

Hint: Here is one way to do it. First randomly permute the array. Then go over the array iteratively while maintaining the top 3 elements seen so far. Design the algorithm so that in most iterations, it is more likely that only one comparison happens. To find the expected number of total comparisons, we can first find the expected number of comparisons in each iteration and then add them all up. To analyze a particular iteration, you will need to find out the probabilities of having one comparison, two comparisons, three comparisons and so on. Then expectation for a particular iteration can be computed by simply applying the expectation formula.

(b) If you want to design such an algorithm to find the k-th largest element, what can be the best possible expected number of comparisons? $n+O(k \log n)$, $n+O(k^2 \log n)$, $n+O(k \log k \log n)$, or some other function of n, k ? Don't need to give a detailed analysis, just write one or two sentences to justify your answer.