Please answer the following 4 questions, some of which have multiple parts.

1. (a) Suppose we are given two sorted arrays \( A[1..n] \) and \( B[1..n] \). Describe an algorithm to find the median element (the element of rank \( n \)) in the union of \( A \) and \( B \) in \( O(\log n) \) time. You may assume that the arrays contain no duplicate elements. [Hint: Compare \( A[\lfloor n/2 \rfloor] \) and \( B[\lfloor n/2 \rfloor] \). How can you reduce your search space to two sorted arrays of size \( \lceil n/2 \rceil \)?]

(b) Now suppose we are given two sorted arrays \( A[1..m] \) and \( B[1..n] \) with no duplicate elements and an integer \( k \) where \( 1 \leq k \leq m+n \). Describe an algorithm to find the \( k \)th smallest element in \( A \cup B \) in \( O(\log(m+n)) \) time. [Hint: Now compare \( A[\lfloor m/2 \rfloor] \) and \( B[\lfloor n/2 \rfloor] \) but only reduce the search space in one of the two arrays.]

2. Suppose you are given a set \( P = \{p_1, \ldots, p_n\} \) of \( n \) points in the plane, represented by two arrays \( X[1..n] \) and \( Y[1..n] \). Specifically, point \( p_i \) has coordinates \( (X[i], Y[i]) \). A point \( p_i \in P \) is called **Pareto optimal** if there exists no point \( p_j \in P \) with \( i \neq j \) such that both \( X[j] \geq X[i] \) and \( Y[j] \geq Y[i] \). In other words, each Pareto optimal point of \( P \) has no other point of \( P \) both above and to its right.

Describe and analyze an \( O(n \log n) \) time algorithm to output the set of Pareto optimal members of \( P \). (Any reasonable output describing these points is fine; for example, you could output an array \( Z[1..h] \) where each element of \( Z \) is the index \( i \) of a Pareto optimal point \( p_i \).) [Hint: Use divide-and-conquer.]

3. Suppose we are given an array \( A[1..n] \) of numbers, which may be positive, negative, or zero, and which are not necessarily integers. We are going to design a dynamic programming algorithm that finds the largest sum of elements in a contiguous subarray \( A[i..j] \).

For example, if we are given the array \([-6, 12, -7, 0, 14, -7, 5]\), our algorithm should return 19 for the contiguous subarray \( A[2..5] \). Given the one-element array \([-374]\) as input, our algorithm should return 0 (the empty interval is still an interval!) For the sake of analysis, we’ll assume that comparing, adding, or multiplying any pair of numbers takes \( O(1) \) time.

(a) Unless it is empty, the maximum contiguous subarray must consist of some last element \( A[j] \) along with 0 or more elements preceding \( A[j] \). Accordingly, let \( \text{maxSum}(j) \) equal the largest sum of elements in a contiguous subarray of \( A[1..j] \) whose last member is \( A[j] \).

Give a recursive definition for \( \text{maxSum}(j) \). Don’t forget the base cases!
(b) What would be the running time of a dynamic programming algorithm that computes $\text{maxSum}(j)$ for all $j$ from 1 to $n$ using your recursive definition? [Hint: You should be able to answer this question without having to describe an iterative algorithm.]

(c) Describe and analyze an efficient algorithm that finds the largest sum of elements in a contiguous subarray of $A[1..n]$. [Hint: Use parts (a) and (b).]

(d) Now suppose in addition to $A[1..n]$, you are given an additional integer $X \geq 0$. Describe and analyze an algorithm that finds the largest sum of elements in a contiguous subarray of $A$ whose length is at most $X$. [Hint: You’ll want to start over by slightly modifying the function from part (a).]

4. For each of the following problems, the input consists of two arrays $X[1..k]$ and $Y[1..n]$ where $k \leq n$.

(a) Describe and analyze an algorithm to decide whether $X$ is a subsequence of $Y$. For example, the string \texttt{PPAP} is a subsequence of the string \texttt{PENPINEAPPLEAPPLEPEN}.

(b) Suppose the input also includes a third array $C[1..n]$ of numbers, which may be positive, negative, or zero, where $C[i]$ is the cost of $Y[i]$. Describe and analyze an algorithm to compute the minimum cost of any occurrence of $X$ as a subsequence of $Y$. That is, we want to find the minimum total cost $\sum_{j=1}^{k} C[I[j]]$ among all arrays $I[1..k]$ such that $I[j] < I[j + 1]$ and $X[j] = Y[I[j]]$ for every index $j$.

(c) Describe and analyze an algorithm to compute the total number of (possibly overlapping) occurrences of $X$ as a subsequence of $Y$. For purposes of analysis, assume we can add two arbitrary integers in $O(1)$ time. For example, the string \texttt{PPAP} appears exactly 23 times as a subsequence of the string \texttt{PENPINEAPPLEAPPLEPEN}. If all characters in $X$ and $Y$ are equal, your algorithm should return $\binom{n}{k}$.