1. Write a divide-and-conquer algorithm that finds the minimum of \( n \) integers. Analyze its time complexity assuming the algorithm is run by (1) one processor and (2) by \( n \) processors.

2. Consider the closest-pair algorithm. Suppose we do not sort \( A[i..j] \) by \( y \)-coordinate in \( \text{Closest-Pair}(A[i..j], (p, q), ptr) \), but instead we sort the whole set of \( n \) points (i.e., \( A[1..n] \)) by \( y \)-coordinate into a linked list in the beginning of the algorithm, immediately after sorting them by \( x \). (The procedure \( \text{Closest-Between-Two-Sets} \) remains intact, but the linked list pointed to by \( ptr \) now contains the entire set of \( n \) points.) Does the modified algorithm work correctly? Justify your answer. (If your answer is YES, justify it; if your answer is NO, give a counterexample, which should consist of the entire set of \( n \) points. This question is not to be graded.)

3. Whether the above algorithm is correct or not, what is its time complexity?

4. Let \( A[1..n] \) and \( B[1..n] \) be two arrays of integers, each sorted in nondecreasing order. Write a divide-and-conquer algorithm that finds the \( n \)th smallest of the \( 2n \) combined elements. Your algorithm must run in \( O(\log n) \) time. You may assume that all the \( 2n \) elements are distinct. (Write your algorithm in pseudo-code and explain it in plain English.) **Note:** The input consists of two arrays of the same size. When you divide the problem, make sure that the two (sub)arrays of each subproblem are of equal size.