1. (10 pts) A server has \( n \) customers waiting to be served. The service time required by each customer is known in advance: it is \( t_i \) minutes for customer \( i \). So if, for example, the customers are served in order of increasing \( i \), then the \( i \)th customer has to wait \( \sum_{j=1}^{i} t_j \) minutes.

We wish to minimize the total waiting time

\[
T = \sum_{i=1}^{n} \text{(time spent waiting by customer } i).\]

Describe an efficient algorithm for computing the optimal order in which to process the customers and give a brief justification of its running time and correctness.

2. (10 pts) Consider the task of searching a sorted array \( A[1\ldots n] \) for a given element \( x \): a task we usually perform by binary search in time \( O(\lg n) \). Show that any algorithm that accesses the array only via comparisons (that is, by asking questions of the form “is \( A[i] \leq z \)?”), must take \( \Omega(\lg n) \) steps.

3. A \( k \)-way merge operation. Suppose you have \( k \) sorted arrays, each with \( n \) elements, and you want to combine them into a single sorted array of \( kn \) elements.

   (a) (5 pts) Here’s one strategy: Using the merge procedure, merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of \( k \) and \( n \)?

   (b) (5 pts) Give a more efficient solution to this problem, using divide-and-conquer. What is its time complexity in terms of \( k \) and \( n \)?

4. (a) (10 pts) Show that any array of integers \( x[1\ldots n] \) can be sorted in \( O(n + M) \) time, where \( M = \max_i x_i - \min_i x_i \).

   (b) (Bonus: 5 pts) For small \( M \), this is linear time: why doesn’t the \( \Omega(n \lg n) \) lower bound apply in this case?