1. Consider the following numerical questions game. In this game, player 1 thinks of a number in the range 1 to \( n \). Player 2 has to figure out this number by asking the fewest number of true/false questions. For example, a question may be "Is your number larger than \( x \)?" Assume that nobody cheats.

(a) What is an optimal strategy if \( n \) is known?

(b) What is a good strategy if \( n \) is not known?

2. Suppose that we are given a sequence of \( n \) values \( x_1, x_2, \ldots, x_n \) in an arbitrary order and seek to quickly answer repeated queries of the form: given an arbitrary pair \( i \) and \( j \) with \( 1 \leq i < j \leq n \), find the smallest value in \( x_1, \ldots, x_j \). Design a data structure and an algorithm that answer each query in \( O(\log n) \) time.

3. The input is a sequence of real numbers \( x_1, x_2, \ldots, x_n \) in an arbitrary order where \( n \) is even. Design an \( O(n \log n) \) time algorithm to partition the input into \( n/2 \) pairs in the following way. For each pair, we compute the sum of its numbers. Denote these \( n/2 \) sums by \( s_1, s_2, \ldots, s_{n/2} \). The objective is to find a partition that minimizes the maximum sum value in \( \{s_1, s_2, \ldots, s_{n/2}\} \). Describe an \( O(n \log n) \) time algorithm.

4. The input is two strings of characters \( A = a_1 a_2 \cdots a_n \) and \( B = b_1 b_2 \cdots b_n \). Design an \( O(n) \) time algorithm to determine whether \( B \) is a cyclic shift of \( A \). In other words, the algorithm should determine whether there exists an index \( k, 1 \leq k \leq n \) such that \( a_i = b_{(k+i) \mod n} \), for all \( i, 1 \leq i \leq n \).

5. We consider disjoint sets and wish to perform two operations on these sets.

   (1) **Union:** If \( S_i \) and \( S_j \) are two disjoint sets, then their union is \( S_i \cup S_j = \{x \mid x \} \) is either in \( S_i \) or \( S_j \), and the sets \( S_i \) and \( S_j \) do not exist independently.

   (2) **Find(\( i \)):** Given an element \( i \), find the set containing \( i \).

We assume that each set is represented using a directed tree such that nodes are linked from children to parents. Three algorithms to perform the union operation \( UNION(S_i, S_j) \) were discussed in class. Now, consider the following algorithm.

**Algorithm 4:** Make the root of the tree with lower depth be a son of the root of the tree with higher depth. (If two trees have the same depth, choose arbitrarily.) Show that \( UNION \) can be done \( O(1) \) time and \( FIND \) can be done in \( O(\log n) \) time.
(Note that Algorithm 3 discussed in class considers the size of each tree while Algorithm 4 considers the depth of each tree.)