1. There are \( n \) items in an array. It is easy to see that their minimum can be found using \( n - 1 \) comparisons and that \( n - 1 \) are actually required. It is also easy to see that finding the max can similarly be done using \( n - 1 \) comparisons with \( n - 1 \) required.

Design an algorithm that finds both the minimum and the maximum using at most \( \frac{3}{2}n + c \) comparisons where \( c > 0 \) can be any constant you want.

**Note:** Although it is harder to prove, \( \frac{3}{2}n + c \) comparisons is actually a lower bound.

2. Prove that insertion in a binary search tree requires at least \( O(\log n) \) comparisons (in the worst case) per insertion, where \( n \) is the number of items in the search tree.

*Hint: What lower bounds have we learned in class? Suppose you built the search tree using insertions. What can you do with it?*

3. Build a Binary Search Tree for the items
   8, 4, 6, 13, 3, 9, 11, 2, 1, 12, 10, 5, 7
   and draw the final tree.
   Now, delete 3, 9, 4 in order and draw the resulting trees.

4. The maximum item in a set of \( n \) real-valued keys is well defined. The maximum item in a set of \( n \) 2-dimensional real-valued points is not.
   One definition that is used in database theory is that of **skyline vectors**. These are also known as **maximal points** or **maximal vectors**.
   Let \( S = \{p_1, p_2, \ldots, p_n\} \) be a set of 2-d points where \( p_i = (x_i, y_i) \). A point \( p \in S \) is a **skyline vector** if no other point is bigger than it in both \( x \) and \( y \) dimensions.
   Formally, \( p_j \) **dominates** \( p_i \) if
   \[
   x_i < x_j \quad \text{and} \quad y_i < y_j.
   \]
   \( p = (x, y) \) is a **skyline vector** in \( S \) if no \( p_i \) in \( S \) dominates \( p \).
   In the example below, the 3 filled points are the skyline ones.

   ![Skyline Vectors Example](image)

   (a) Give an algorithm that finds the skyline vectors in a set \( S \) of \( n \) points in \( O(n \log n) \) time.
(b) Suppose that the points all have integer coordinates in the range \([1, \ldots, n^2]\). Give an \(O(n)\) algorithm for solving the same problem.