1. **Decision Trees and Lower Bounds.** In class, you saw how the running time of a comparison-based sorting algorithm can be bounded from below (i.e., “lower-bounded”) by analyzing the height of the decision tree associated with the sorting algorithm. Let us apply the same kind of analysis to **FindMax**, the problem of finding the largest number among \( n \) numbers. Consider a comparison-based algorithm that solves **FindMax**.

   a. Describe the decision tree \( T \) associated with such an algorithm. In particular, what does its internal nodes represent? What does its external nodes represent?
   
   b. Suppose the input is \( x_1, x_2, x_3, x_4 \). Show the decision tree associated with the tournament-style algorithm for finding the maximum of \( x_1, x_2, x_3, x_4 \).
   
   c. Based on your description in part (a), what is a lower bound on the height of \( T \). Hence, provide a function \( f(n) \) so that any algorithm that solves **FindMax** should run in \( \Omega(f(n)) \) time.

2. Given a set of \( n \) numbers, suppose we want to find the \( i \) smallest numbers in sorted order. We have several options for doing this:

   OPTION 1: Sort the \( n \) numbers and list the \( i \) smallest numbers.
   
   OPTION 2: Build a heap with the \( n \) numbers and apply removeMin \( i \) times.
   
   OPTION 3: Use QuickSelect to find the \( i \)th smallest number \( s \), partition the \( n \) numbers using \( s \), and sort the \( i \) smallest numbers.

   For each of these options, find the algorithm that implements it in the best worst-case running time, and analyze the running time in terms of \( n \) and \( i \).


4. C-4.15.