Combination of Number with Size K LeetCode

This program is designed to generate all possible combinations of a specific number of elements from a given set of numbers. The combinations are generated using a recursive backtracking approach.

Approach 1: Function to find all combinations of 'combination' elements from 'num' numbers using Backtracking

1. The program starts by defining a recursive function **findCombinations** that aims to generate combinations of 'combination' elements from 'num' numbers.
2. The base case of the recursion is when the output vector has reached the desired size of 'combination'. At this point, the current combination is added to the answer.
3. The **findCombinations** function iterates through the numbers from the given 'index' up to 'num'. For each number, it adds the number to the output vector, recursively calls itself to find the remaining combinations, and then backtracks by removing the last added number to explore other possibilities.
4. The **combinations** function initializes the answer vector and checks if the required 'combination' size is larger than the available 'num' numbers. If so, it returns an empty result, as generating combinations with more elements than available numbers is not possible.
5. In the **main** function, the user is prompted to enter the number of elements and the desired combination size. The **combinations** function is then called to generate all possible combinations.
6. The generated combinations are printed using the **printArray** function, which formats and prints each combination.

Time Complexity: The time complexity of this program is determined by the number of possible combinations that need to be generated. In the worst case, when 'num' is large and 'combination' is nearly equal to 'num', the program could potentially generate a significant number of combinations. The number of possible combinations is given by the binomial coefficient "num choose combination," which is O(num! / (combination! * (num - combination)!)). **Therefore, the time complexity is O(num! / (combination! * (num - combination)!)).**

Space Complexity: The space complexity involves the space required for the answer vector to store all the generated combinations and the space used by the recursive call stack during the backtracking process. The depth of the recursive call stack depends on the combination size, and the answer vector can store multiple combinations. **Therefore, the space complexity is O(combination + num! / (combination! * (num - combination)!)).**