Assignment 5: Backtracking & Greedy Algorithms

1. **Implement a backtracking algorithm**
   Given a collection of amount values (A) and a target sum (S), find all unique combinations in A where the amount values sum up to S. Return these combinations in the form of a list.
   - Each amount value may be used only the number of times it occurs in list A.
   - The solution set should not contain duplicate combinations.
   - Amounts will be positive numbers.
   - Return an empty list if no possible solution exists.

   Example: **A** = [11,1,3,2,6,1,5]

   Target Sum = 8

   Result = [[3, 5], [2, 6], [1, 2, 5], [1, 1, 6]]

   a. Describe a backtracking algorithm to solve this problem.

   First, to help ensure each value in our collection is not used more than the number of times it occurs, we sort the input collection into an ascending order. Starting at the first value in our list, we will use a helper function to help us recursively try to add each number in our collection to a current combination being built and backtrack if the current combination doesn't lead to a solution or if we've reached the target sum. We will skip over any duplicate combinations in the recursion, and we will keep track of our target with our helper function.

   b. Implement the solution in a function **amount(A, S)**. Name your file **Amount.py**

   c. What is the time complexity of your implementation, you may find time complexity in detailed or state whether it is linear/polynomial/exponential. Etc.?

   $O(2^n)$ exponential

2. **Implement a Greedy algorithm**

   You are a pet store owner and you own a few dogs. Each dog has a specific hunger level given by array hunger_level [1..n] (ith dog has hunger level of hunger_level [i]). You have couple of dog biscuits of size given by biscuit_size [1...m]. Your goal to satisfy maximum number of hungry dogs. You need to find the number of dogs we can satisfy.

   - If a dog has hunger hunger_level[i], it can be satisfied only by taking a biscuit of size biscuit_size [j] >= hunger_level [i] (i.e biscuit size should be greater than or equal to hunger level to satisfy a dog.)
   - If no dog can be satisfied return 0.

   Conditions:

   You cannot give the same biscuit to two dogs.
   Each dog can get only one biscuit.

   Example 1:

   Input: hunger_level[1,2,3], biscuit_size[1,1]
   Output: 1
   Explanation: Only one dog with hunger level of 1 can be satisfied with one cookie of size 1.

   Example 2:

   Input: hunger_level[2, 1], biscuit_size[1,3,2]
   Output: 2
   Explanation: Two dogs can be satisfied. The biscuit sizes are big enough to satisfy the hunger level of both the dogs.

   a. Describe a greedy algorithm to solve this problem

   First, we will sort both the hunger_level and biscuit_size in ascending order to help us efficiently match the smallest biscuit to the dog with the lowest hunger level. We will iterate through both arrays, finding the smallest biscuit that can satisfy the current dog. We will increment a counter if the dog can be satisfied or move on to the next dog if we do not have a biscuit that can satisfy the current dog.

b. Write an algorithm implementing the approach. Your function signature should be **feedDog(hunger_level, biscuit_size)**; hunger_level, biscuit_size both are one dimension arrays . Name your file **FeedDog.py**

c. Analyse the time complexity of the approach.

   O(nlogn)