Day 40 / 100 :

Topic - Backtracking, Array

1 Problem statement: Letter Combination of a phone number (Medium)

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order.
A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

Example 2:
Input: digits = ""
Output: []

Example 3:
Input: digits = "2"
Output: ["a","b","c"]

Solutions :
Approach 1 - Backtracking

Intution:
The problem requires finding all possible letter combinations that can be formed by a given sequence of digits. For example, if the input is "23", then the output should contain all possible combinations of the letters 'a', 'b', and 'c' for digit 2, and 'd', 'e', and 'f' for digit 3. This can be solved using a backtracking approach where we start with the first digit and find all the possible letters that can be formed using that digit, and then move on to the next digit and find all the

possible letters that can be formed using that digit, and so on until we have formed a combination of all digits.

Approach:

1. Define a function letterCombinations which takes a string of digits as input and returns a vector of all possible letter combinations.
2. Check if the input string is empty, if it is, return an empty vector as there are no possible letter combinations.
3. Define a mapping vector which contains the letters corresponding to each digit. For example, the mapping vector for the input "23" would be {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"}.
4. Define an empty string combination with the same size as the input digits string.
5. Call the backtrack function with the initial index as 0 and pass the result vector, mapping vector, combination string, and digits string as arguments.
6. In the backtrack function, check if the current index is equal to the size of the digits string. If it is, that means we have formed a complete combination of letters, so we add the current combination string to the result vector and return.
7. Otherwise, get the possible letters corresponding to the current digit using the mapping vector.
8. Iterate through all the possible letters and set the current index of the combination string to that letter.
9. Call the backtrack function recursively with the incremented index.
10. After the recursive call returns, reset the current index of the combination string to its previous value so that we can try the next letter.

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        vector<string> result;
        if (digits.empty()) {
            return result;
        }
        vector<string> mapping = {"", "", "abc", "def", "ghi", "jkl",
"mno", "pqrs", "tuv", "wxyz"};
        string combination(digits.size(), ' ');
        backtrack(result, mapping, combination, digits, 0);
        return result;
    }

private:
    void backtrack(vector<string>& result, const vector<string>& mapping,
string& combination, const string& digits, int index) {
```

```
        if (index == digits.size()) {
            result.push_back(combination);
        } else {
            string letters = mapping[digits[index] - '0'];
            for (char letter : letters) {
                combination[index] = letter;
                backtrack(result, mapping, combination, digits, index + 1);
            }
        }
    }
};
```

2 Problem statement: Combination Sum (Medium)

Given an array of distinct integers candidates and a target integer target, return a list of all unique combinations of candidates where the chosen numbers sum to target. You may return the combinations in any order.
The same number may be chosen from candidates an unlimited number of times. Two combinations are unique if the
frequency
of at least one of the chosen numbers is different.
The test cases are generated such that the number of unique combinations that sum up to target is less than 150 combinations for the given input.

Example 1:
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.

Example 2:
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]

Example 3:
Input: candidates = [2], target = 1
Output: []

Solutions :

Approach 1 - Backtracking

Intuition:

Simply do recursion and generate all the possible combinations to check if that combination result in target if not backtrack.

Approach:

We can use backtracking like the same way we generate all the sub sets of array here the thing is we have to check if the sum of that subset result in target or not . And also we should check repeatedly for the same element also, so not call for i+1 instead i.

Complexity:

Time complexity: $O(n*2n)$

Space complexity: $O(n*2n)$

```
class Solution {
public:
    vector<vector<int>>ans;
    void backtrack(vector<int>&arr,int start,int rem,vector<int>&temp){
        if(rem<0)return;
        if(rem==0)ans.push_back(temp);

        for(int i=start;i<arr.size();i++){
            temp.push_back(arr[i]);
            backtrack(arr,i,rem-arr[i],temp);
            temp.pop_back();
        }
    }
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        vector<int>temp;
        backtrack(candidates,0,target,temp);
        return ans;
    }
```