12) COMBINATIONAL SUM:

METHOD:

LINK OF EXPLANATION: L8. Combination Sum | Recursion | Leetcode | C++ | Java

1) This sum is the same as that of the unbounded knapsack problem.
2) In this sum, for every particular element we have three choices:
   The first choice is to select the element and don't move,
   The second choice is to select the element and move,
   The third choice is we do not select the element and move ahead.
3) So we get to understand that we have to pass three recursion functions.
4) Also this depends on the sum, i.e. if taking the element exceeds the sum than the sum which is required we don't take an element .
5) But this can be reduced to the dead end if we keep the array sorted.
   This is because at any position we encounter that taking an element exceeds the required sum, then the elements ahead of it will also exceed. So we return from there.

The below code showed me an error of TLE at test case 105/215 as we don't include the dead end condition.

```
class Solution {
  public:
    //Function to return a list of indexes denoting the required
    //combinations whose sum is equal to the given number.
    map<vector<int>,int> mp;

    void fun(vector<int> &a,int index,int currentsum,int b,vector<int>
&c,vector<vector<int>> & answers){
        if(currentsum==b){
            if(mp[c]==1){
                return;
            }
            answers.push_back(c);
            mp[c]=1;
            return;
        }
        else if(index==a.size()){
            return;
        }
        else{
            if(currentsum+a[index]<=b){
                c.push_back(a[index]);
                fun(a,index,currentsum+a[index],b,c,answers);
                c.pop_back();
                c.push_back(a[index]);
                fun(a,index+1,currentsum+a[index],b,c,answers);
                c.pop_back();
                fun(a,index+1,currentsum,b,c,answers);
            }
            else{
                fun(a,index+1,currentsum,b,c,answers);
            }
```

```
            }
        }

        vector<vector<int> > combinationSum(vector<int> &a, int b) {
            mp.clear();
            sort(a.begin(),a.end());
            vector<vector<int>> answer;
            vector<int> c;
            fun(a,0,0,b,c,answer);
            return answer;
        }
    };
```

Even after adding the dead end condition the code showed TLE at 109/215 test case.

- The main reason behind the TLE was that, we only needed to call 2 recursive functions.
- The 1 extra included recursion was calling the function for the one which have to be selected more than once.
- The intuition was that if we select a particular element then we don't have to move because calling the same function recursively will call the select and non select option for the same element again.
- Also we do not require map because the problem of getting the same combination again and again is due to the duplicate elements in the original array.So instead of using the maps we remove the duplicates before only.

THE CORRECT CODE IS:

```
class Solution {
  public:
    //Function to return a list of indexes denoting the required
    //combinations whose sum is equal to given number.

    void fun(vector<int> &a,int index,int currentsum,int b,vector<int> &c,vector<vector<int>> &
answers){
        if(currentsum==b){
            answers.push_back(c);
            return;
        }
        else if(index==a.size()){
            return;
        }
        else{
            if(currentsum+a[index]<=b){
                c.push_back(a[index]);
                fun(a,index,currentsum+a[index],b,c,answers);
                c.pop_back();
            }
            fun(a,index+1,currentsum,b,c,answers);
            return;
```

```
        }
    }

    vector<vector<int> > combinationSum(vector<int> &a, int b) {
        sort(a.begin(),a.end());
        vector<int> d;
        d.push_back(a[0]);
        for(int i=1;i<a.size();i++){
            if(a[i]!=a[i-1]){
                d.push_back(a[i]);
            }
        }
        vector<vector<int>> answer;
        vector<int> c;
        fun(d,0,0,b,c,answer);
        return answer;
    }
};
```