Striver’s SDE Sheet

Day 9: Recursion

Subset Sums

Given a list arr of N integers, print sums of all subsets in it.

Ans:

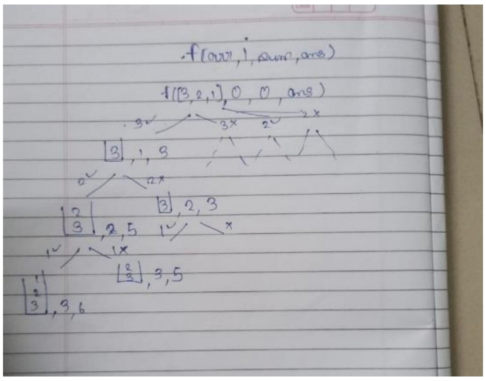
This question can be solved by two approach one is power set and second one using recursion

Power set:

```
vector<int> res;
        for(int i=0; i<(1<<N); i++) {
            int sum=0;
            for(int j=0; j<N; j++) {
                if(i&(1<<j)) sum+=arr[j];
            }
            res.push_back(sum);
        }
        return res;
```

and second approach will be recursion one:

recursion tree :pick non pick approach



Code:

```
class Solution
{
public:
   void f(int i, vector<int>arr,int sum, vector<int>& ans){
      if(i >= (int)arr.size()){
           ans.push_back(sum);
           sum = 0;
           return;
      }
      sum += arr[i];
      f(i+1, arr, sum, ans);
      sum -= arr[i];
      f(i+1, arr, sum, ans);
}

    vector<int> subsetSums(vector<int> v, int N)
    {

    vector<int> ans;
      f(0, v, 0, ans);
      sort(ans.begin(), ans.end());

      return ans;
    }
};
```

Variation:

In subset some time give some conditions

Subsets II

Given an integer array nums that may contain duplicates, return *all possible*

subsets

(the power set).

The solution set must not contain duplicate subsets. Return the solution in any order.

Ans:

There are two approach from which we can solve this problem one si using extra space like covert list into set

Or just little bit change in our recursion to get answer

Code for second approach :

```
 class Solution {
public:

   void
subset(vector<vector<int>>&ans,vector<int>&a,vector<int>&cont,int
idx){
        ans.push_back(cont);
    for(int i=idx;i<a.size();i++){
        if(i!=idx&&a[i]==a[i-1])continue;
        cont.push_back(a[i]);
        subset(ans,a,cont,i+1);
        cont.pop_back();
    }
}
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        vector<vector<int>>v;
        vector<int>ds;
        set<vector<int>>st;
        sort(nums.begin(),nums.end());
        subset(v,nums,ds,0);
        return v; } };
```

recursion tree :

just the change is if(i!=idx&&a[i]==a[i-1])continue;

variation:

variation of previous question and in second approach it’s a variation of pick non pick approach

Combination Sum

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

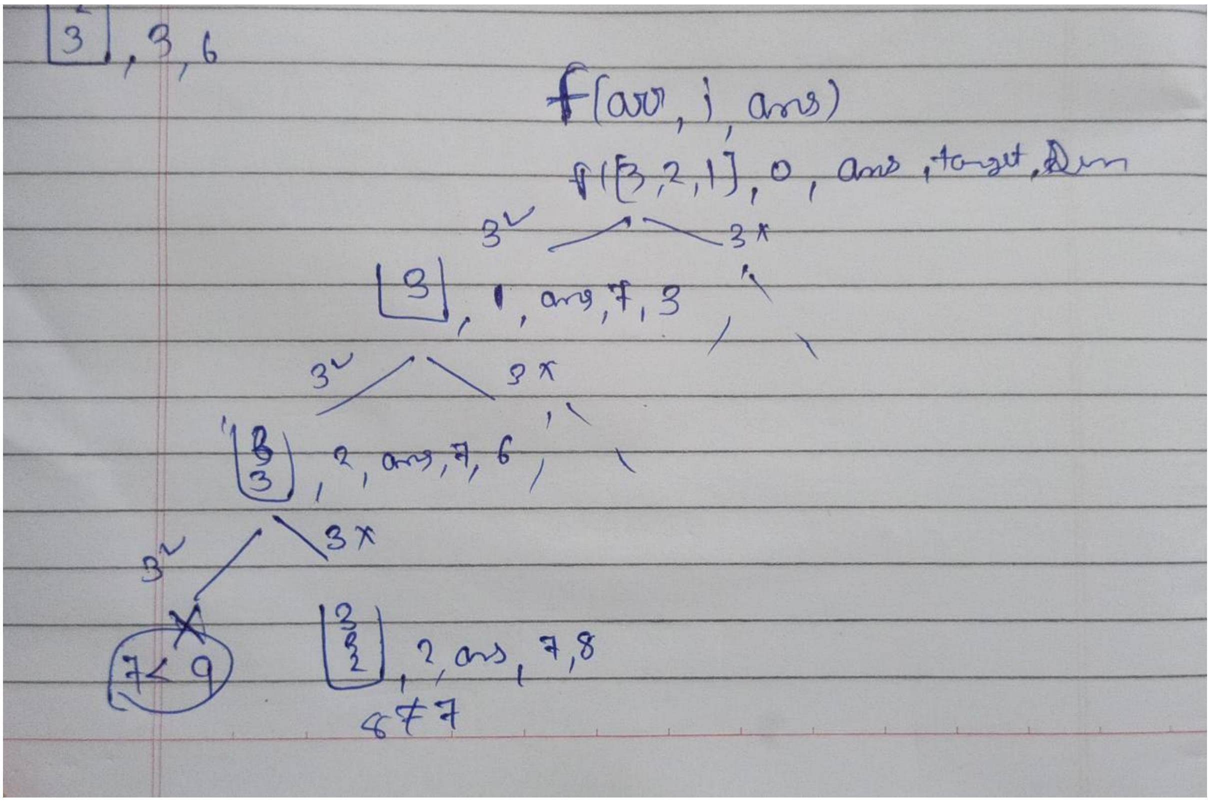
of at least one of the chosen numbers is different.

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.

Ans:

It's a straight forward question of trying out all possibility

Recursion tree: pick non pick approach



Code:

```
class Solution {
public:
    // for the sake of convience, not to include again and again in my function call, I declare target here
    int target;

    vector<vector<int>> ans; // 2-D vector to store our answer

    void solve(vector<int>& arr, int i, int sum, vector<int> op)
    {
        // if i crosses the array size, we will return saying that no more possibilty is left to choose

        if(i >= arr.size())
        {
            return;
        }

        // if value at ith index + sum becomes equal to target, then we will store it in our answer array, saying that yes it is a possible combination
        if(arr[i] + sum == target)
        {
            op.push_back(arr[i]);
            ans.push_back(op);
            return;
        }

        // if value at ith index + sum is less than target, then we have two choices i.e whether to include this value in our possible combiation array or not include that,
        if(arr[i] + sum < target)
        {
            // we make two output vector, one for calling function at
same index and anthor for calling function frm next index. Because for
every element we have unlimited choices, that it will contribute in
making our sum any number of times.
            vector<int> op1 = op;
            vector<int> op2 = op;

            op2.push_back(arr[i]);
            solve(arr, i, sum + arr[i], op2);
            solve(arr, i + 1, sum, op1);
        }
        else
        {
            solve(arr, i + 1, sum, op); // call for the next index
        }
    }

    vector<vector<int>> combinationSum(vector<int>& arr, int
required_target) {
        ans.clear(); //clear global array, make to sure that no
garbage value is present in it

        target = required_target; // give target what he wants
        vector<int> op; // op array to try all possible combination
        sort(arr.begin(),arr.end()); // sort the array in ascending
order
        solve(arr, 0, 0, op); // call function

        return ans; // return the final answer array
    }
};
```

Variation:

Variation is only in second recursion solution

Combination Sum II

Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target.

Each number in candidates may only be used once in the combination.

Note: The solution set must not contain duplicate combinations.

Ans:

Just the trick is maintain set and easier then previous question:

Same recursion tree as previous to previous question

Code:

```
class Solution {

public:

     void findCombinations(int idx, vector<int> &arr, int target,
vector<int> &ds, vector<vector<int>> &ans){

        if(target == 0){

            ans.push_back(ds);

            return;

        }

        for(int i=idx; i<arr.size(); i++){

            if(i>idx && arr[i] == arr[i-1]) continue;

            if(arr[i]>target) break;

            ds.push_back(arr[i]);

            findCombinations(i+1, arr, target-arr[i], ds, ans);

            ds.pop_back();

        }

    }


    vector<vector<int>> combinationSum2(vector<int>& candidates, int
target) {

        vector<vector<int>> ans;

        vector<int> ds;

        sort(candidates.begin(), candidates.end());
        findCombinations(0, candidates, target, ds, ans);
        return ans;
    }
};
```

Palindrome Partitioning

Given a string s, partition s such that every Substring of the partition is a palindrome. Return *all possible palindrome partitioning of* s.

Ans:

Same as above question but the change here is to check if palindrome then push to ans

Code:

```
class Solution {

public:

     vector<vector<string>> partition(string s) {

        vector<vector<string>> res;

        vector<string> path;

        helper(0, s, path, res);

        return res;

    }

    void helper(int index, string s, vector<string> &path,
vector<vector<string>> &res){

        if(index == s.size()){

            res.push_back(path);

            return;

        }

        for(int i = index; i < s.size(); i++){

            if(isPalindrome(s, index, i)){

                path.push_back(s.substr(index, i - index + 1));

                helper(i + 1, s, path, res);

                path.pop_back();

            }

        }

    }

    bool isPalindrome(string s, int start, int end){

        while(start <= end){

            if(s[start++] != s[end--]) return false;
        }
        return true;
    }
};
```

Variation from above question is just to check if palindrome or not

Permutation Sequence

The set [1, 2, 3, ..., n] contains a total of n! unique permutations.

By listing and labeling all of the permutations in order, we get the following sequence for n = 3:

1. "123"
2. "132"
3. "213"
4. "231"
5. "312"
6. "321"

Given n and k, return the kth permutation sequence.

Ans:

Pick no pick with some variation :

Code:

```
class Solution {

public:

    void recur(string s, string temp, vector<bool> &vis, vector<string> &ans,
int &cnt, int k){

      if(temp.size() == s.size()){

        ans.emplace_back(temp);

        cnt++;

        if(cnt == k) return;

      }

      for(int i = 0; i < s.size(); i++){

        if(!vis[i]){

          vis[i] = true;

          temp += s[i];

          recur(s, temp, vis, ans, cnt, k);

          temp.pop_back();

          vis[i] = false;

        }

        if(cnt == k) return;

      }

    }

    string getPermutation(int n, int k) {
      vector<string> ans;
      string s = "", temp = "";
      for(int i = 1; i < n + 1; i++)
        s += to_string(i);
      vector<bool> vis(n, false);
      int cnt = 0;
      recur(s, temp, vis, ans, cnt, k);
      return ans.back();
    }
};
```