Subset Sum II

BRUTE FORCE :

Generate all subsets and store them in set then copy the set into vector<vector<int>> to return the ans.

```
#include <bits/stdc++.h>
void subsetSum(int ind,int n,vector<int>& arr,vector<int> temp,set<vector<int>> &s)
{
    if(ind==n)
    {
        sort(temp.begin(),temp.end());
        s.insert(temp);
        return;
    }
    temp.push_back(arr[ind]);
    subsetSum(ind+1,n,arr,temp,s);
    temp.pop_back();
    subsetSum(ind+1,n,arr,temp,s);

}
vector<vector<int>> uniqueSubsets(int n, vector<int> &arr)
{
    // Write your code here.
    set<vector<int>> s;
    vector<int> temp;
    subsetSum(0,n, arr, temp, s);
    vector<vector<int>> ans(s.begin(),s.end());
    return ans;
}
```

- Time Complexity : O(2^n * (k*log(x))*2^n)
- Space Complexity :O(2^n * k)

Optimal Approach :

Initially sort the array so the duplicates occur side by side.

Run a loop inside recursive function to generate vector of length 0,1,2,… and so on and keep comparing whether the current element is equal to the previous element , if it is we will not consider it and continue the loop. There is no need for base condition as the loop ends the recursive function calls stops.

```
#include <bits/stdc++.h>
void helper(int ind,int n,vector<int>& arr,vector<int> temp,vector<vector<int>>& ans)
{
    ans.push_back(temp);
    for(int i=ind;i<n;i++)
    {
        if(i!=ind && arr[i]==arr[i-1]) continue;
        temp.push_back(arr[i]);
        helper(i+1,n,arr,temp,ans);
        temp.pop_back();
    }
}
vector<vector<int>> uniqueSubsets(int n, vector<int> &arr)
{
    // Write your code here.
    vector<vector<int>> ans;
    vector<int> temp;
    sort(arr.begin(),arr.end());
    helper(0,n,arr,temp,ans);
    return ans;
}
```

- Time Complexity : O(k*2^N), **O(k)** to insert every subset in another data structure if the average length of every subset is **k.**
- Space Complexity : O(2^N *k)