You can use set to remove duplicate answer

classmate Date Page

(VI) Combination Sum II: Duplicate are present

→ element of array is used/picked once in the combination
→ find all unique combination with sum == target.
→ Note: the soln set must not contain duplicate combinations

I/P → [10,1,2,7,6,1,5] t = 8
O/P → [[1,1,6], [1,2,5], [1,7], [2,6]]

Without complexity is $O(2^n \times k \times \log m)$
Without $O(2^n \times k)$

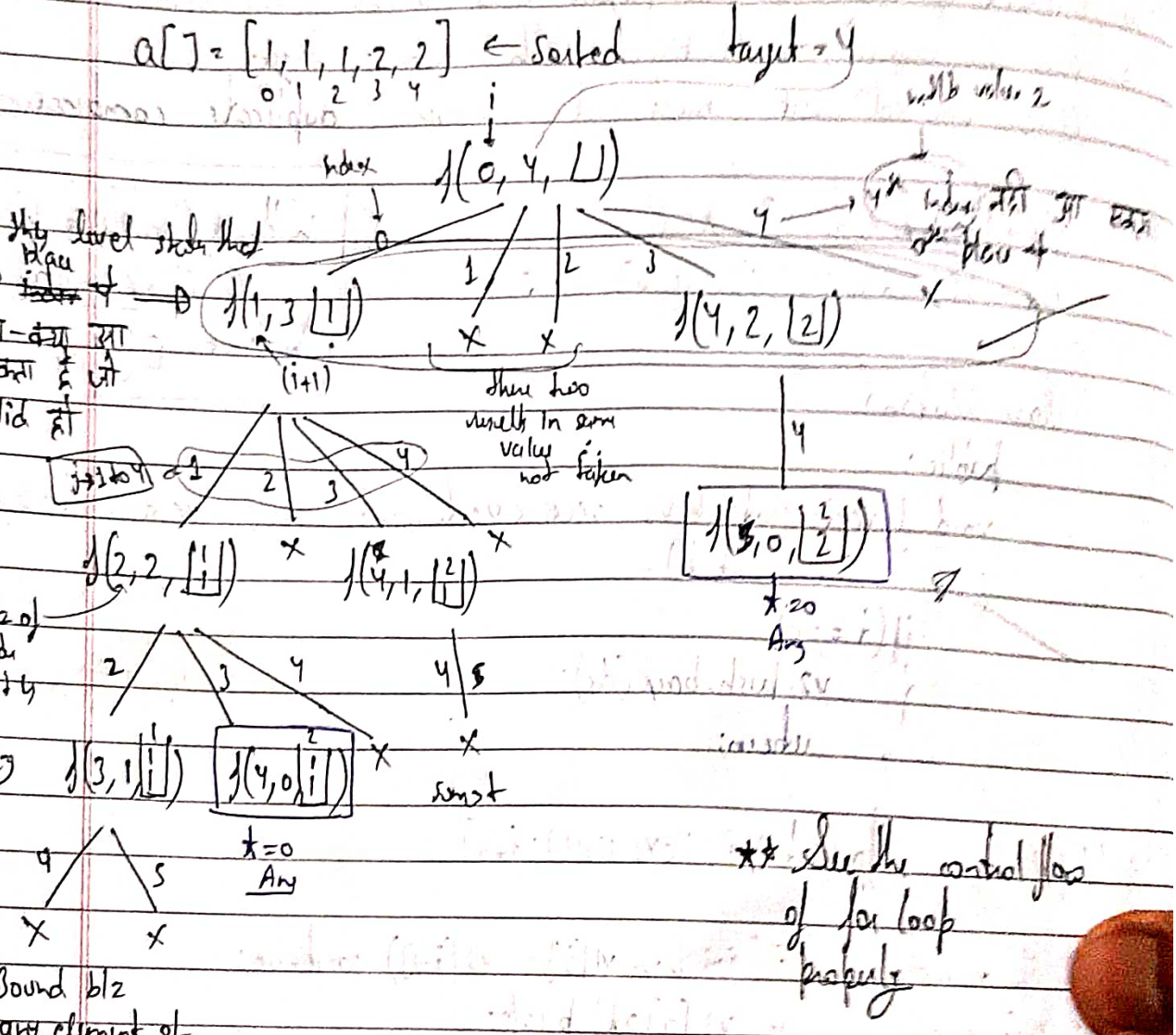
```
class Solution{
    public:
    void f(vector<int> &v1, vector<vector<int>> &v2, vector<int> &ds,
           int i, int t)
    {
        if(t==0)
        {
            v2.push_back(ds);
            return;
        }
        for(int j=i; j<v1.size(); j++)
        {
            if(j>i && v1[j]==v1[j-1]) continue;
            if (v1[j]>t) break;
            ds.push_back(v1[j]);
            f(v1, v2, ds, j+1, t-v1[j]);
            ds.pop_back(); *
        }
    }
    public:
        vector<vector<int>> CombinationSum(vector<int> &v1, int t){
        vector<vector<int>> v2;
        vector<int> ds;
        sort(v1.begin(), v1.end());
        f(v1, v2, ds, 0, t);
        return v2;
    }
};
```

for loop approach

[1,1,1,2,2]
0 1 2 3 4

(j>i) b/z no matter v1[4]==v1[3] we will always pick 1 for 1st recursion call

→ Sorting it b/z we want the combination in Leexgraphic manner ie [1,1,2] not [2,1,1]



** See the control flow of for loop properly

Remember two thing

① When t==0 We get ans that is the base case

② When a[j] > t, we don't go further checking simply break, since the array is sorted.

Imagine properly the Control Flow

∴ Using Break

→ With for loop we go to next level (one level down)

this ∮ happened b/z of for loop from j=0 to 4 i.e
→ the first for loop.

Time Complexity

$(2^n \times k)$ → for pushing in vector v2
↳ avg length of combination

Space Complexity

$k \times x$
↳ total combination

→ (j>i) or (j!=i) for pushing/picking up the first call means next level के call को रखा जाे और पहला है उसे

→ You can't use set method b/z TLE occurs

Int to String
to_string(n);

(VII) Subset Sum I:

print Sum of all subsets in increasing Order

Eg: I/P → a[2,3]

O/P → 0 2 3 5

```
class Solution
{
    public:
    void f(vector<int> &v1, vector<int> &v2, int i, int n, int sum)
    {
        if(i==n)
        {
            v2.push_back(sum);
            return;
        }
        f(v1, v2, i+1, n, sum+v1[i]);
        f(v1, v2, i+1, n, sum);
    }
    public:
    vector<int> SubsetSum(vector<int> &v1, int n)
    {
        vector<int> v2;
        f(v1, v2, 0, n, 0);
        sort(v2.begin(), v2.end());
        return v2;
    }
};
```

```
sort(v1.begin(), v1.end());
f(                        );
return v2;
```

Time Complexity: $2^n + 2^n \log(2^n)$ → $2^n + n\log n$

(2^n: ½ of sort; can be reduced to if we pass the sorted v2 v1)

Space Complexity: 2^n

Better TC than Power Set Approach ($2^n \times n$)

accumulate (~~first~~ a.begin(), a.end(), 0);
To Sum Vector — initial sum value

classmate Date Page

(VIII) Subset Sum II :

- Array contain [duplicate item], return all possible subset
- The solution set must not contain duplicate ~~items~~ subsets.
- return soln in any order.

Logic is: at every function call first we will generate/print the subset. (At every step we generate subset)

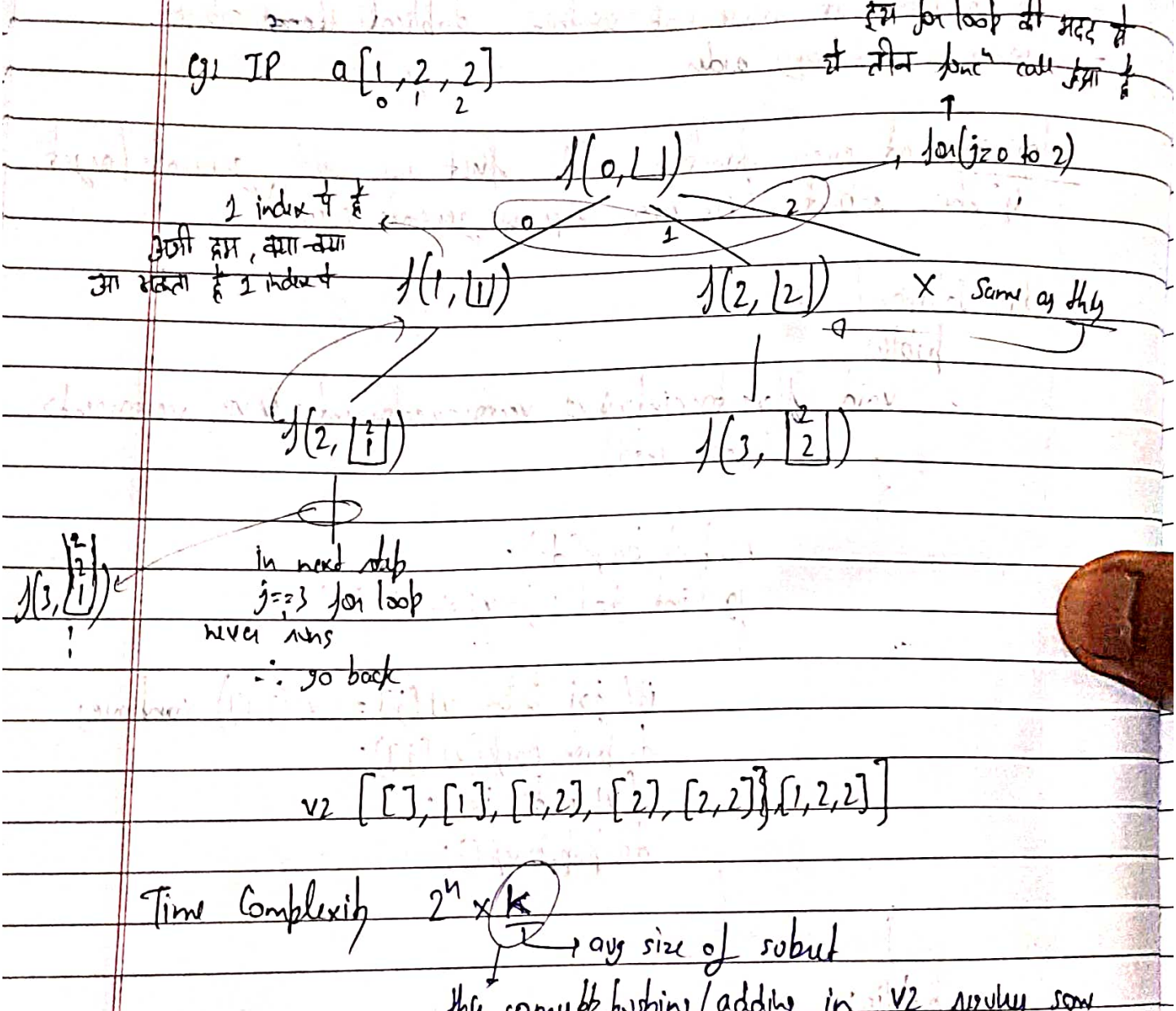
```
class Solution{
    public:
        void f(vector<int>&v1, vector<vector<int>>&v2, vector<int>
            &ds, int i)
        {
            v2.push_back(ds);
            for(int j=i ; j<v1.size();j++)
            {
                if( j>i && v1[j] == v1[j-1]) continue;
                ds.push_back(v1[j]);
                f(v1,v2,ds, j+1);
                ds.pop_back();
            }
        }
    public:
        vector<vector<int>> SubsetwithDup(vector<int> &v1){

            vector<vector<int>> v2;
            vector<int> ds;
            sort(v1.begin(), v1.end());
            f(v1, v2, ds, 0);
            return v2;
        }
};
```

** To apply the recursive logic for duplicate element array, we have to sort to make the element side-by-side for comparison & continue it

→ Other Approach for Subset Sum II is use set for pushing Unique Subset;

Eg: IP a[1, 2, 2]
0 1 2



v2 [[], [1], [1,2], [2], [2,2]] [1,2,2]

Time Complexity $2^n \times k$ → avg size of subset
this comes for pushing/adding in v2, recursion some time

Space Complexity $(2^n) \times k$

Auxillary Space: O(n)
↓
Space due to the recursion depth/level

(IX) Print all Permutations of a String / Array (I^st Approach)

→ Given distinct integer vector
→ return all permutation

n! → no. of permutations for n size vector

Eg: I/P a[1,2,3]
O/P [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

```
void f(vector<int> &v1, vector<vector<int>> &v2, int n, int i)
{
    if(i==n)
    {
        v2.push_back(v1);
        return;
    }
    for(int j=i; j<n; j++)
    {
        swap(v1[j], v1[i])
        f(v1, v2, j+
```

vector of ↑

Approach: that we maintain a map datastructure such a way that whenever we pick that element we mark the index in map as 0.

Initialize map [1,1,1,-]

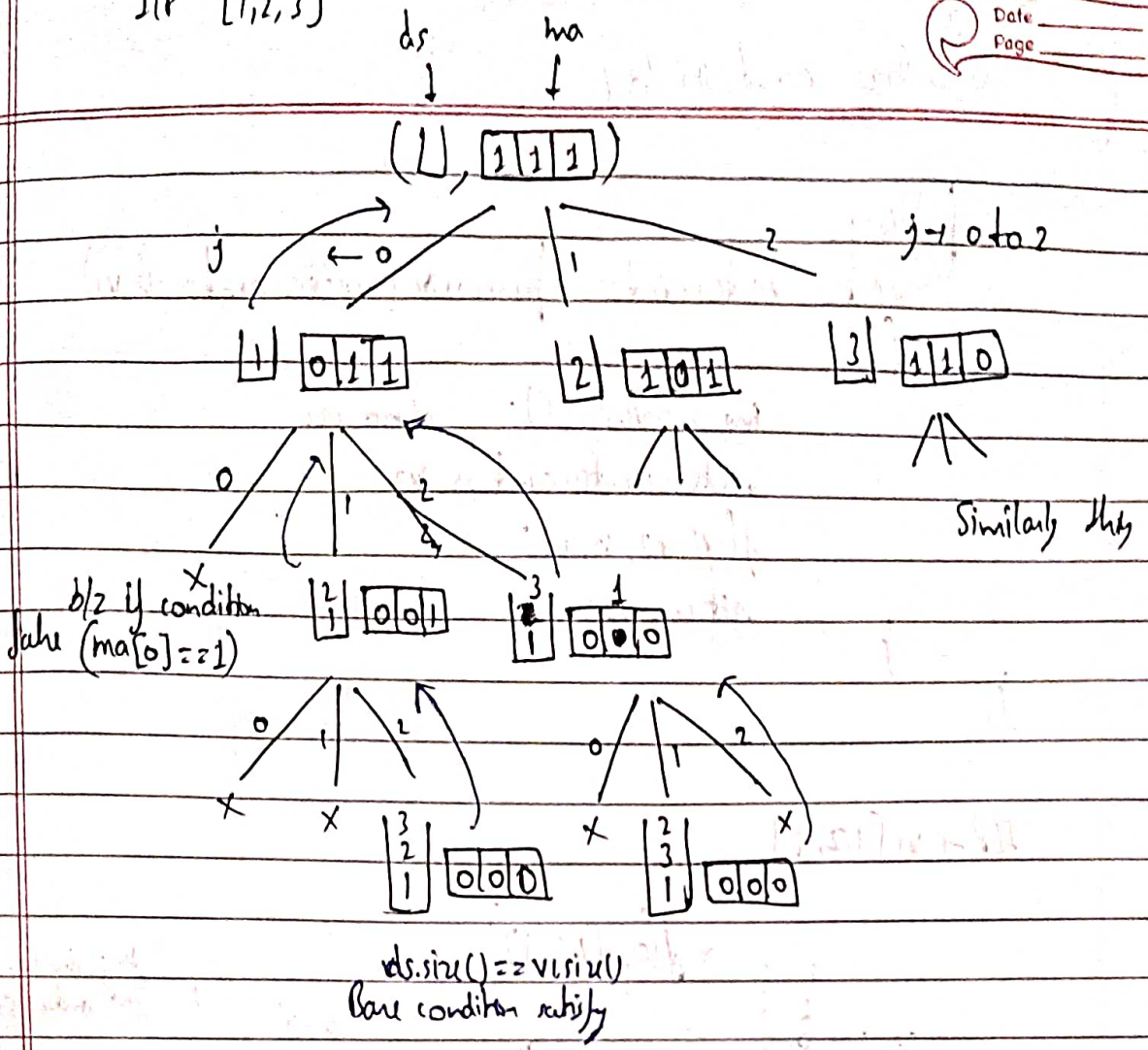
Time Complexity n! × (n → everytime loop from 0 to n)

SC O(n) + O(n) + O(n!)
ds ma ans

Auxiliary space O(n)

```
class Solution{
public:
    void f(vector<int> &v1, vector<vector<int>> &v2, vector<int>
        &ds, int i, vector<int> ma)
    {
        if( ds.size() == v1.size())
        {
            v2.push_back(ds);
            return;
        }
        for( j=0; j< v1.size(); j++)
        {
            if(ma[j] == 1)
            {
                ds.push_back(v1[j]);
                ma[j]=0;
                f(v1, v2, ds, ma);
                ds.pop_back();
                ma[j] = 1;
            }
        }
    }
public:
    vector<vector<int>> permute(vector<int> &v1){
        vector<int> ds, ma(v1.size(), 1);
        vector<vector<int>> ans; v2;
        f(v1, v2, ds, ma);
        return v2;
    }
};
```

I/P [1,2,3]



(V) Print all Permutation II^nd Approach : (Swap method)

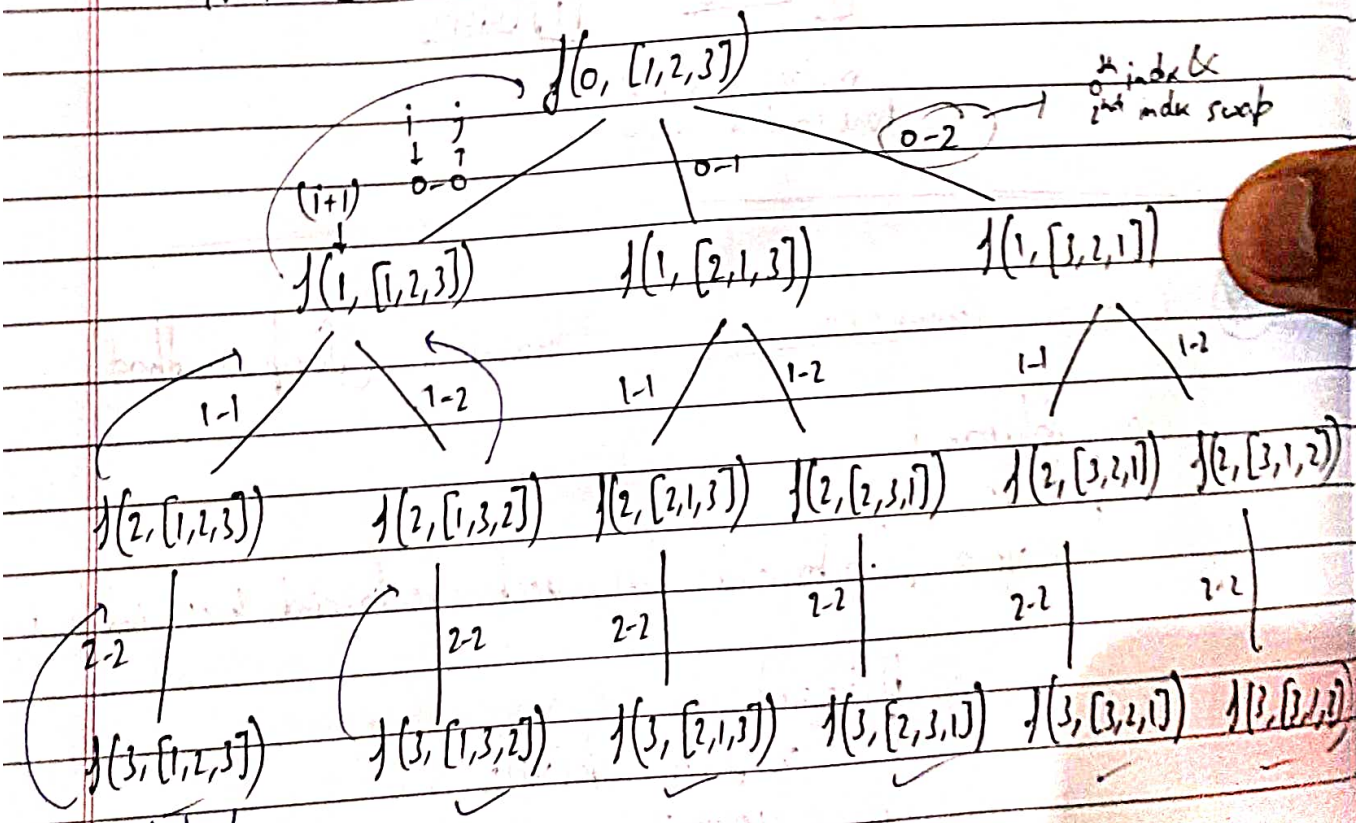
```
class Solution{
    public:
        void f(vector<int> &v1, vector<vector<int>> &v2, int n, int i)
        {
            if(i==n)
            {
                v2.push_back(v1);
                return;
            }
            for(int j=i; j< v1.size(); j++)
            {
                swap(v1[j], v1[i]);
                f(v1, v2, n, i+1)
                swap(v1[j], v1[i]);
            }
        }
```

`const int m=1e9+7;` 10^9+2

To declare const in C++

```
public:
    vector<vector<int>> permute(vector<int>& v1)
    {
        int n=v1.size();  //no use
        vector<vector<int>> v2;
        f(v1,v2,n,0);
        return v2;
    }
};
```

I/P → v1 [1,2,3]



swap back to get original vector v1

Set Method

Implementation

(XI) Permutations II — Duplicate present

Given vector, return all possible unique permutations in any order

```
class Solution{
    public:
    void f(vector<int> &v1, set<vector<int>> &v2, int i)
    {
        if(i==v1.size())
        {
            v2.insert(v1);
            return;
        }
        for(int j=i ; j<v1.size(); j++)
        {
            swap(v1[j], v1[i]);
            f(v1,v2, i+1);
            swap(v1[j], v1[i]);
        }
    }
    public :
        vector<vector<int>> permuteUnique(vector<int> &v1){
            set<vector<int>> v2;
            sort(v1.begin(), v1.end());
            f(v1, v2, 0);
            vector<vector<int>> ans;
            for(auto it: v2)
                ans.push_back(it);
            return ans;
        }
};
```

Point To Remember

** Set method can be used in Subset Sum II problem also, it can be used when we have duplicacy in array.

** Problem with set is that time Complexity increases.