L12 Subsets I or Subsets Sum I

Friday, December 23, 2022 5:31 PM

Given an integer array nums of **unique** elements, return *all possible subsets (the power set)*.

→ popular algorithm for finding the All unique subset powerset Algorithm that we learn in Bit Manipulation

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

Example 1:
Input: nums = [1,2,3]
Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]

↳ this problem on leetcode Name of subset basically our task is finding all possible subset including Null set also

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

Example 1:
Input:
N = 2
arr[] = {2, 3}

Output: 0 2 3 5

Explanation:
When no elements is taken then Sum = 0.
When only 2 is taken then Sum = 2.
When only 3 is taken then Sum = 3.
When element 2 and 3 are taken then
Sum = 2+3 = 5.

↳ this problem can also by the Name of Subsets Sum

↳ the only one difference of leetcode and this problem is on the leetcode written all the possible subset

→ and on GFG Return Sum of all the possible subsets

Thought Process

→ Again we follow the Concept of pick and Not pick that we learn previously Indepth

arr = [3, 1, 2]
n = 3

formula = 2^n → unique possible Subsets

[3, 1, 2]	Subsets
x x x	[ ]
✓ x x	[1]
x ✓ ✓	[1,2]
x x ✓	[3]
✓ ✓ x	[1,3]
✓ x ✓	[2,3]
✓ ✓ ✓	[1,2,3]

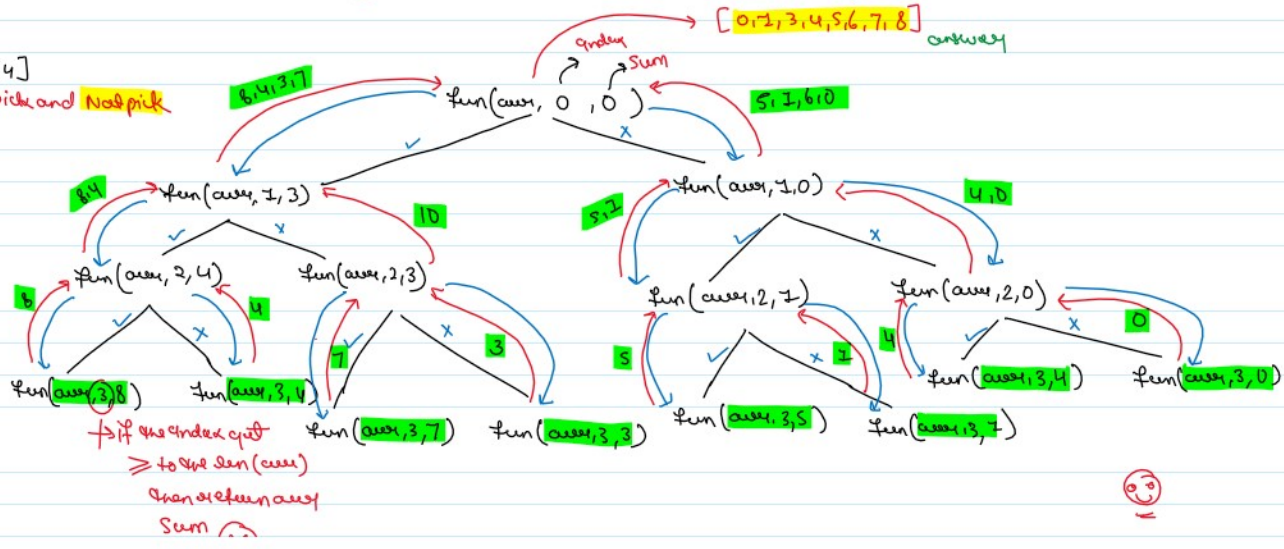
8 possible Combination

→ If this problem Comes in the Interview then first start with brute force Approach Powerset Algorithm used for Generate All the possible Subset

Time Complexity = $2^n \times N$
using the bit Manipulation

→ Now the Interviewer Not want to the Solve (N) time then jump on the Recursion Solution / Optimized Approach

arr = [3, 1, 4]
→ Follow the pick and Not pick Approach



>= to the len(arr) then return any sum

myfunction(index, arr, sum, ans) {

if index >= len(arr) {
ans.append(sum)
return
}

Base case

Time Complexity = $2^n + 2^n \log(2^n)$

→ here we have every position 2 option

→ sorting the array

because the size of the array depends on the combination and every combination is 2^n at max that's why

pick case → myans.append(arr[index])
myfunction(index+1, arr, sum, ans)

Not pick case → myans.remove(arr[index])
myfunction(index+1, arr, sum, ans)

Space Complexity = $2^n + O(N)$

→ for the all combination that we stored into the array ds

→ Recursive stack space

```python
class Solution:

    def leetCodeSolution(self,index,arr,myans,ans):

        if index >= len(arr):
            ans.append(myans[:])
            return

        myans.append(arr[index])
        self.leetCodeSolution(index+1,arr,myans,ans)
        myans.remove(arr[index])
        self.leetCodeSolution(index+1,arr,myans,ans)

    def GFGSolution(self, index, arr, Sum, ans):

        if index >= len(arr):
            ans.append(Sum)
            return

        self.GFGSolution(index + 1, arr, Sum+arr[index], ans)

        self.GFGSolution(index + 1, arr, Sum, ans)

    def subsets(self, arr):

        # leetcode
        ans = []
        self.leetCodeSolution(0,arr,[],ans)
        ans.sort()

        # gfg
        gfgans = []
        self.GFGSolution(0,arr,0,gfgans)
        gfgans.sort()

        return ans
```

Recursive Array of Array

Recursive Sum of Array

```java
import java.util.*;
public class L12_Subset_Sums_I {
    public static void main(String[] args) {

        System.out.println("L12_Subset_Sums_I");

    }
}


no usages
class Solution4{
    no usages
    public ArrayList<Integer> subsetSums(ArrayList<Integer> arr, int N){


        ArrayList<ArrayList<Integer>> ans = new ArrayList<>();
        List<Integer> ds = new ArrayList<>();
        leetcodeSolution(arr, index: 0,ans,ds);


        ArrayList<Integer> sumans = new ArrayList<>();
        GFGSolution( index: 0,arr, sum: 0,sumans);
        return sumans;
    }

    2 usages
    private void GFGSolution(int index, ArrayList<Integer> arr, int sum, ArrayList<Integer> ans) {

        if(index >= arr.size()){
            ans.add(sum);
            return;
        }

        GFGSolution( index: index+1,arr, sum: sum-arr.get(index),ans);
        GFGSolution( index: index+1,arr,sum,ans);
    }


    3 usages
    public void leetcodeSolution(ArrayList<Integer> arr, int index, List<List<Integer>> ans, List<Integer> ds) {

        if(index >= arr.size()){
            ans.add(new ArrayList<>(ds));
            return;
        }

        ds.add(arr.get(index));
        leetcodeSolution(arr, index: index+1,ans,ds);
        ds.remove( index: ds.size() - 1);
        leetcodeSolution(arr, index: index+1,ans,ds);

    }
}
```

GFG Solution

LC Solution