Subset Sum : Sum of all Subsets

Problem Statement: Given an array print all the sum of the subset generated from it, in the increasing order.
Examples:

Example 1:

Input: N = 3, arr[] = {5,2,1}

Output: 0,1,2,3,5,6,7,8

Explanation: We have to find all the subset's sum and print them.in this case the generated subsets are [[], [1], [2], [2,1], [5], [5,1], [5,2]. [5,2,1],so the sums we get will be 0,1,2,3,5,6,7,8

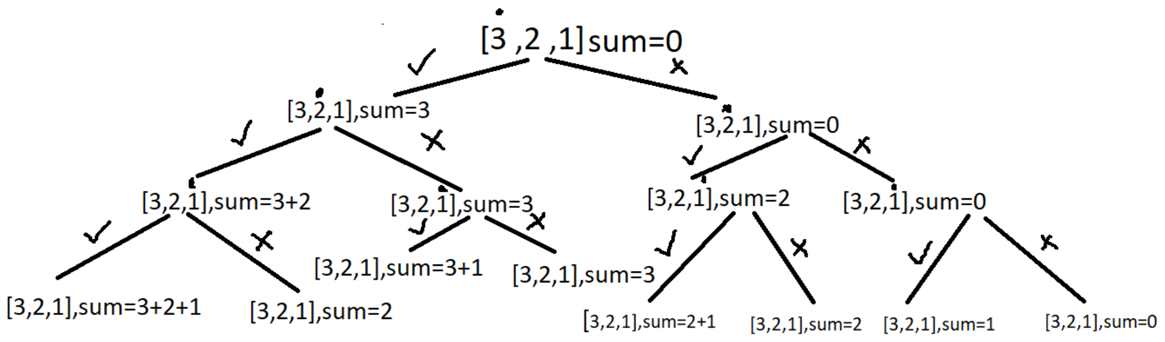
Input: N=3,arr[]= {3,1,2}

Output: 0,1,2,3,3,4,5,6

Explanation: We have to find all the subset's sum and print them.in this case the generated subsets are [[], [1], [2], [2,1], [3], [3,1], [3,2]. [3,2,1],so the sums we get will be 0,1,2,3,3,4,5,6

Solution

Disclaimer*: Don't jump directly to the solution, try it out yourself first.*
Solution 1: Using recursion
Intuition: The main idea is that on every index you have two options either to select the element to add it to your subset(pick) or not select the element at that index and move to the next index(non-pick).
Approach: Traverse through the array and for each index solve for two arrays, one where you pick the element,i.e add the element to the sum or don't pick and move to the next element, recursively, until the base condition. Here when you reach the end of the array is the base condition.



Code:

- C++ Code
- Java Code
- Python Code

```cpp
#include<bits/stdc++.h>

using namespace std;
class Solution {
  public:
    void subsetSumsHelper(int ind, vector < int > & arr, int n, vector < int > & ans, int sum) {
      if (ind == n) {
        ans.push_back(sum);
        return;
      }
      //element is picked
      subsetSumsHelper(ind + 1, arr, n, ans, sum + arr[ind]);
      //element is not picked
      subsetSumsHelper(ind + 1, arr, n, ans, sum);
    }
  vector < int > subsetSums(vector < int > arr, int n) {
    vector < int > ans;
    subsetSumsHelper(0, arr, n, ans, 0);
    sort(ans.begin(), ans.end());
    return ans;
    }
};


int main() {
    vector < int > arr{3,1,2};
    Solution ob;
    vector < int > ans = ob.subsetSums(arr, arr.size());
    sort(ans.begin(), ans.end());
    cout<<"The sum of each subset is "<<endl;
    for (auto sum: ans) {
        cout << sum << " ";
    }
    cout << endl;

    return 0;
}
```

Output:

```
The sum of each subset is
0 1 2 3 3 4 5 6
```

Time Complexity: O(2^n)+O(2^n log(2^n)). Each index has two ways. You can either pick it up or not pick it. So for n index time complexity for

O(2^n) and for sorting it will take (2^n log(2^n)).

Space Complexity: O(2^n) for storing subset sums, since 2^n subsets can be generated for an array of size n.