# Day 87/180

## Recursion on Subset 📑

## 1. Subset Sums

```
class Solution {
   public:
    void print(vector<int> arr, int index, int n, int sum, vector<int>&
ans) {
        if (index == n) {
            ans.push_back(sum);
            return;
        }
        print(arr, index + 1, n, sum, ans);
        print(arr, index + 1, n, sum + arr[index], ans);
    }
    vector<int> subsetSums(vector<int> arr, int N) {
        vector<int> ans;
        int index = 0, n = arr.size(), sum = 0;
        print(arr, 0, n, 0, ans);
        return ans;
    }
};
```

### Code Explanation and Complexity

The code defines a function print and another function subsetSums.

1. print function:
    - This is a recursive helper function that generates all possible subset sums.
    - It takes the input array arr, the current index index, the total number of elements n, the current sum sum, and a vector 'ans' to store the subset sums.
    - It recursively explores two possibilities for each element:
        - Exclude the current element by calling print with the same index and sum.
        - Include the current element by calling print with the incremented index and the updated sum.
    - The base case for the recursion is when the index reaches the size of the array (index == n), and at this point, the current sum is added to the ans vector.

2. subsetSums function:
    - This is the main function that initializes the process by calling the print method with the initial parameters.
    - It returns the vector ans containing all subset sums.
    - The subsetSums method initializes the necessary variables (index, n, sum) and then calls the print function with the initial values.

Time Complexity:

The time complexity is O(2^N), where N is the number of elements in the input array. This is because for each element, there are two recursive calls (include/exclude), leading to an exponential time complexity.

Space Complexity:

The space complexity is also O(2^N) as the recursion depth corresponds to the number of elements in the power set of the input array. The auxiliary space is used for the recursive call stack.

## 2. Given an array of size N, Print subset sum and corresponding to it, print the given subset also.

Ex: arr[1,2]

output:

```
Sum Subset
1      1
2      2
3     1 2
```

```
#include <iostream>
#include <vector>
using namespace std;

class Solution {
   public:
    void printSubsets(vector<int> &arr, int index, int n, int sum,
                      vector<int> &ans, vector<int> currentSubset) {
        if (index == n) {
            ans.push_back(sum);
            // Print the sum and the subset
            if (sum == 0)
```

```
                cout << "Sum: " << sum << "  Subset: ∅";
            else {
                cout << "Sum: " << sum << "  Subset:";
                for (int num : currentSubset) {
                    cout << " " << num;
                }
            }
            cout << endl;
            return;
        }
        // Exclude element
        printSubsets(arr, index + 1, n, sum, ans, currentSubset);
        // Include element
        currentSubset.push_back(arr[index]);
        printSubsets(arr, index + 1, n, sum + arr[index], ans,
currentSubset);
    }

    vector<int> subsetSums(vector<int> arr, int N) {
        vector<int> ans;
        vector<int> currentSubset;
        printSubsets(arr, 0, N, 0, ans, currentSubset);
        return ans;
    }
};

int main() {
    Solution solution;
    vector<int> arr = {1, 2, 5};
    solution.subsetSums(arr, arr.size());
    return 0;
}
```

## Code Explanation and Complexity

The code defines a class Solution with two methods: printSubsets and subsetSums.

1. printSubsets function:

    - This is a recursive helper function that generates all possible subset sums and prints the corresponding subsets.

- It takes the input array arr, the current index index, the total number of elements n, the current sum sum, a vector ans to store the subset sums, and a currentSubset vector to store the current subset being considered.
- If the index reaches the size of the array (index == n), it means a subset has been generated, and the sum is added to the ans vector. The function then prints the sum and the corresponding subset.
- The subset is printed as an empty set (∅) if the sum is 0. Otherwise, it prints each element of the subset.

2. subsetSums function:

   - This is the main function that initializes the process by calling the printSubsets method with the initial parameters.
   - It returns the vector ans containing all subset sums.
   - The subsetSums method initializes the necessary variables (index, n, sum, currentSubset) and then calls the printSubsets function with the initial values.

3. In the main function, an instance of the Solution class is created, and the subsetSums method is called with a sample array {1, 2, 5}.

Time Complexity:
The time complexity is O(2^N), where N is the number of elements in the input array. This is because for each element, there are two recursive calls (include/exclude), leading to an exponential time complexity.

Space Complexity:
The space complexity is O(N) due to the recursive call stack. The auxiliary space is used for the recursive call stack and the currentSubset vector. The space required for the ans vector is not counted in the space complexity analysis.

## 3. Perfect Sums

**Recursive Code**, it **will give the correct output but will give TLE** on GFG at the last because of the time complexity

```
class Solution {
public:
    void calculateSubsetSums(vector<int>& arr, int index, int n, int sum,
unordered_map<int, int>& subsetSumsCount) {
        if (index == n) {
            subsetSumsCount[sum]++;
            return;
        }

        calculateSubsetSums(arr, index + 1, n, sum, subsetSumsCount);
        calculateSubsetSums(arr, index + 1, n, sum + arr[index],
subsetSumsCount);
    }

    int perfectSum(int arr[], int N, int sum) {
        vector<int> temp(arr, arr + N);
        unordered_map<int, int> subsetSumsCount;
        calculateSubsetSums(temp, 0, N, 0, subsetSumsCount);

        return subsetSumsCount[sum];
    }
};
```

## Code Explanation and Complexity

- The calculateSubsetSums method recursively calculates the sums of all subsets of the given array and stores the count of each sum in the subsetSumsCount unordered_map.
- The perfectSum method initializes the subsetSumsCount map, calls the calculateSubsetSums method, and then returns the count of subsets with the given sum.

The time complexity of this approach is O(2^N) and the space complexity is O(2^N), where 'N' is the number of elements in the array. This approach efficiently counts all the subsets of the given array with a sum equal to the given sum, considering the modulo operation to handle large outputs.

This approach utilizes the same recursive strategy as the "Subset Sums" code provided earlier, and can effectively count all subsets with a sum equal to the given sum.

END