Subset Sum

BRUTE FORCE : BIT MANUPULATION

Using power set. In power set the basic idea is to see which ever bit is set add to the answer. there are 2^n subsets possible so we write down binary notation for all numbers from 0 to 2^n-1 and run a loop again to check whether the ith bit is set or not if it is set it adds up to the answer.

for example when N=3

possible subsets 2^3 = 8 so run loop from 0 to 7

number of bits = 3 so inner loop from 0 to 2

for each bit left shift 1 to check a bit is set or not for example 2nd bit then 1<<2 will check for 2nd bit.

num&(1<<2) ≠0 checks that 2 bit is set.

```
class Solution
{
public:
    vector<int> subsetSums(vector<int> arr, int N)
    {
        // Write Your Code here
        vector<int> ans;
        for(int i=0;i<pow(2,N);i++)
        {
            int sum=0;
            for(int j=0;j<N;j++)
            {
                if((i&(1<<j))!=0)
                {
                    sum+=arr[j];
                }
            }
            ans.push_back(sum);

        }
        return ans;
    }
};
```

- Time Complexity : O(2^n * n)

- Space Complexity : O(1)

Optimal Approach :

Using recursion. pick or not pick and add to answer maintain an index and when index == n then a subset is found which is the lowermost leaf level of the recursive tree.

```
class Solution
{
public:
    void subSum(int ind,vector<int> arr,int N,int sum,vector<int>& ans)
    {
        if(ind==N)
        {
            ans.push_back(sum);
            return;
        }
        subSum(ind+1,arr,N,arr[ind]+sum,ans);
        subSum(ind+1,arr,N,sum,ans);
    }
    vector<int> subsetSums(vector<int> arr, int N)
    {
        // Write Your Code here
        vector<int> ans;
        subSum(0,arr,N,0,ans);
        return ans;
    }
};
```

- Time Complexity : O(2^N)
- Space Complexity : O(2^N) recursive calls stack space.