Subset Sum

Given a list of numbers.
Figure out if it is possible to have the given sum S. Using any subset of given numbers. Use a number from the subset only once.

Numbers: [2,4,6,4]
S: 10

```Python
def isSubsetSumPossible(numbers, s):
    pass
```

Recursive Brute Force

Java
```Java
 public boolean rec(int arr[] , int sum , int curr){
        if(sum < 0) return false;

        if(sum == 0) return true;
        if(curr == arr.length) return false;
        return rec(arr , sum , curr+1 ) || rec(arr,sum-arr[curr],curr+1);
    }
```

Java
```Java
static void sumSubet(int[] numbers, int i, int currSum) {
        if (currSum == sum) {
            count++;
            return;
        }
        if (currSum < sum && i < n) {
            sumSubet(numbers, i+1, currSum + numbers[i]);
            sumSubet(numbers, i+1, currSum);
        }
    }
```

Python
```Python
def isSubsetSumPossible(numbers, s):
    isSubsetSumPossibleUtil(numbers, s, len(numbers)-1)

def isSubsetSumPossibleUtil(numbers, s, i):
    if i < 0:
        return False

    if s == 0:
        return True
    if s < 0:
        return False
```

```
    return isSubsetSumPossibleUtil(numbers, s-numbers[i], i-1) or
isSubsetSumPossibleUtil(numbers, s, i-1)
```

TC: O(2^n)
SC: O(n)

2 4 6 3
0 1 2 3

Memoization

Java

```
    // x x x  x-> null
    // x x x

    // F T F
    // F x x

    //         sum  subset index   answer
    // map <  (int,          int),  bool >

    public boolean topDown(int arr[] , int sum , int curr , Boolean [][]
    dp){
            if(sum < 0) return false;

            if(sum == 0) return true;
            if(curr == arr.length) return false;
            if(dp[sum][curr] != null)
                return dp[sum][curr];
            return dp[sum][curr] = topDown(arr , sum , curr+1,dp ) || to
    pDown(arr,sum-arr[curr],curr+1,dp);
        }

    TC: O(n*sum)
    SC: O(n*sum)
```

Tabulation

```
i  j=0 1 2 3 4 5 6 7 8 9 10  S
--------------------------
0 -  T F F F F F F F F F F
1 2  T F T F F F F F F F F
2 4  T F T F T F T F F F F
3 6
4 4
```

Each position = is sum of j possible by including elements till inde
x i

Java

```java
public boolean bottomUp(int [] arr , int sum ){
        int n = arr.length;
        Boolean [][] dp = new Boolean [n+1][sum+1];

        for(int i= 0; i <= sum ; i++){
            dp[0][i] = false;
        }

        for(int i =0 ; i <= n ; i++){
            dp[i][0] = true;
        }

        for(int i = 1; i <= n ;i++){
            for(int j = 1 ;j <= sum ; j++){
                    if(j < arr[i-1]) dp[i][j] = dp[i-1][j];
                    else
                    dp[i][j] = dp[i-1][j] || dp[i-1][j-arr[i-1]];
            }
        }
        return dp[n][sum];
    }
```

C++

```
bool isSubsetSum(vector<int>arr, int sum){
        int n = arr.size();

        bool dp[sum+1][n+1];

        for(int i= 0; i <= sum ; i++){
            dp[0][i] = false;
        }

        for(int i =0 ; i <= n ; i++){
            dp[i][0] = true;
        }

        for(int i = 1; i <= n ;i++){
            for(int j = 1 ;j <= sum ; j++){
```

In [ ]:

In [ ]:

https://leetcode.com/problems/coin-change/

Brute Force

Java

```java
package org.example.dp;

public class CoinChain {

  class Solution {
    public int coinChange(int[] coins, int amount) {
      int n = coins.length;
      int result = (int) coinChangehelper(coins, n, amount);
      if(result == Integer.MAX_VALUE - 1) return -1;
      else return result;
    }

    private long coinChangehelper(int[] coins, int n, int amount){
      if(amount == 0){
        return 0;
      }
      if(n == 0){
        return Integer.MAX_VALUE;
      }
      if(amount >= coins[n - 1]){
        int in = (int) (1 + coinChangehelper(coins, n, amount - coin
s[n - 1]));
        int ex = (int) coinChangehelper(coins, n - 1, amount);
        return Math.min(in, ex);
      }
      else{
        int ex = (int) coinChangehelper(coins, n - 1, amount);
        return ex;
      }
    }
  }
}
```

Java

```
class Solution {
    public int rec(int [] coins , int amount , int curr){
        if(amount == 0) return 0;
        if(amount < 0 ) return Integer.MAX_VALUE-1;
        if(curr == coins.length) return Integer.MAX_VALUE-1;

        return Math.min( rec(coins,amount,curr+1) , 1+rec(coins,amou
nt-coins[curr],curr));
```

In [ ]:

Memoization

Java

```
class Solution {
    public int rec(int [] coins , int amount , int curr,Integer[][]d
p){
        if(amount == 0) return 0;
        if(amount < 0 ) return Integer.MAX_VALUE-1;
        if(curr == coins.length) return Integer.MAX_VALUE-1;
        if(dp[amount][curr] != null)
            return dp[amount][curr];
        return dp[amount][curr] = Math.min( rec(coins,amount,curr+1,
dp) , 1+rec(coins,amount-coins[curr],curr,dp));
    }
    public int coinChange(int[] coins, int amount) {
        int n =coins.length;
        Integer [][] dp = new Integer[amount+1][n+1];
        int ans = rec(coins,amount , 0,dp);
        return (ans == Integer.MAX_VALUE || ans==Integer.MAX_VALUE-
1) ? -1 : ans;
    }
}
```

C++

```
class Solution {
public:
    int coinChangeUtil(vector<int>& coins, int amount,int curr)
    {
        if(amount < 0)
           return INT_MAX-1;

        if(amount == 0){
            return 0;
        }

        if(curr == coins.size()) return INT_MAX-1;

        int in = 1 + coinChangeUtil(coins, amount - coins[curr],cur
r);
        int ex =  coinChangeUtil(coins,  amount,curr+1);
```

In [ ]:

Java

```java
public int coinChange(int[] coins, int amount) {
      int n = coins.length;
      int[][] dp = new int[n + 1][amount + 1];
      for(int j = 1; j < amount + 1; j++){
        dp[0][j] = Integer.MAX_VALUE - 1;
      }
      // int result = (int) coinChangehelperR(coins, n, amount);
      int result = (int) dP(coins,amount);
      if(result == Integer.MAX_VALUE - 1) return -1;
      else return result;
    }
```

In [ ]:

Greedy Fails for [1,2,5,10]

```javascript
function coinChange(coins: number[], amount: number): number {
    let result = 0;
    coins.sort();
    // Traverse through all denomination
    for (let i = coins.length - 1; i >= 0; i--)
    {
        // Find denominations
        while (amount >= coins[i])
        {
            amount -= coins[i];
            result+=1;
        }
    }
    if(amount==0){
        return result;
    } else {
        return -1;
    }
};
```

```
class Solution {
    public int coinChange(int[] coins, int amount) {
        int len=coins.length;
        Arrays.sort(coins);
        int amt=amount;
        int coin=0;;
        int result;
        int count=0;
        for(int i=len-1;i>=0;i--)
```