Subset Sum problem

I/p : arr[] = { 3, 34, 4, 12, 5, 2 }
sum = 9

n = 6

o/p : True
(4 + 3 + 2)

False ← n = 0

I/p : arr[] = { }
sum = 7

o/p : False

{ } { }
n = 0
10

I/p : arr[] = { 4, 3, 2 }
sum = 5

o/p : ~~false~~ True

I/p : arr[] = { 4, 3, 2 }
sum = 0

o/p : True ;

sum = 0
o/p - True

{ }

Subset sum Problem

I/p : arr[] = { 3, 34, 4, 12, 5, 2 }

sum = 9

n = 6

O/p : True

(4 + 3 + 2)

n = 3

I/p : arr[] = { 4, 3, 2 }

sum = 6

O/p : True

$\boxed{n = 0}$

I/p : arr[] = { }

sum = 7

O/p : false

I/p : arr[] = { 4, 5, 1 }

sum = 0

O/p : True

{ }

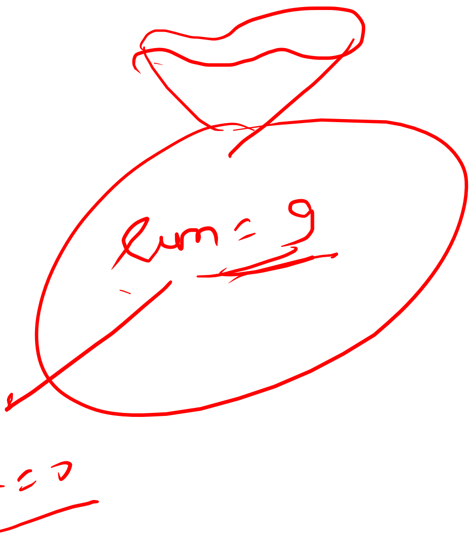
Sum = 0

Last lecture, we have seen that, 0/1 knapsack problem,
there is some similarities in both problem

I/p : arr[] = { 3, 34, 4, 12, 5, 2 }
sum = 9

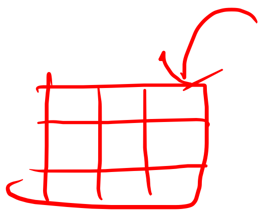
o/p : True
(4+3+2)

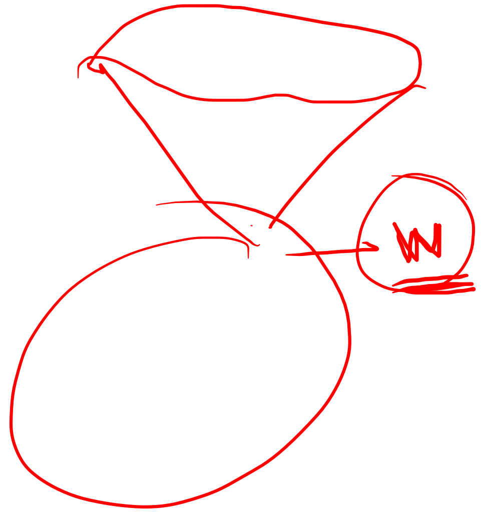
false



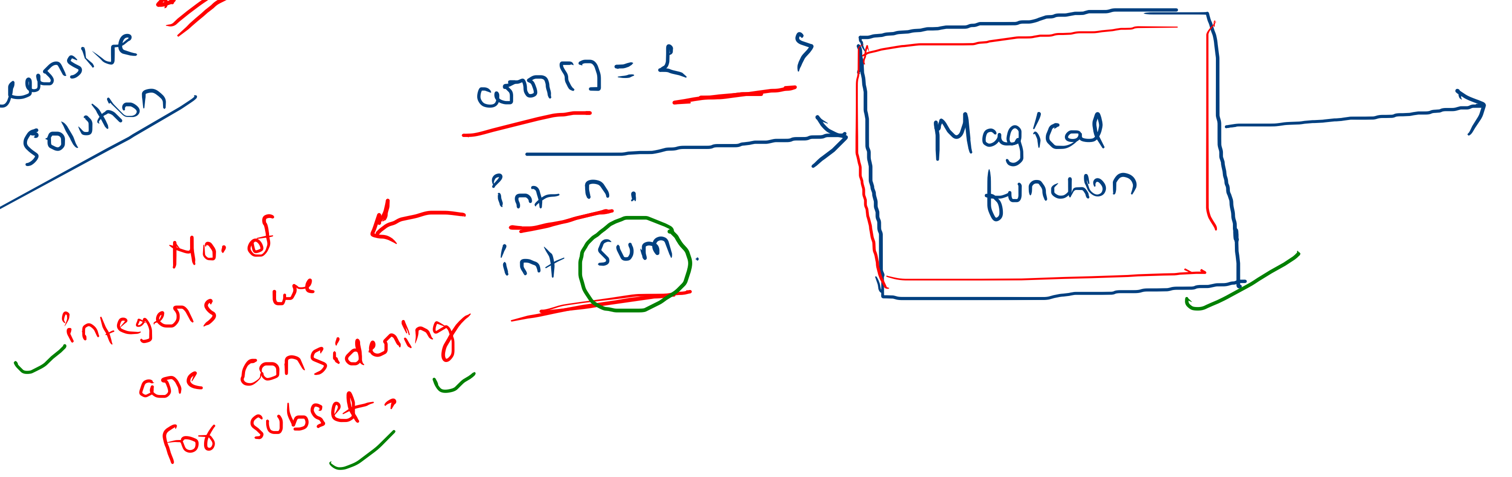
⟹ weight of items

⟹ Capacity of bags

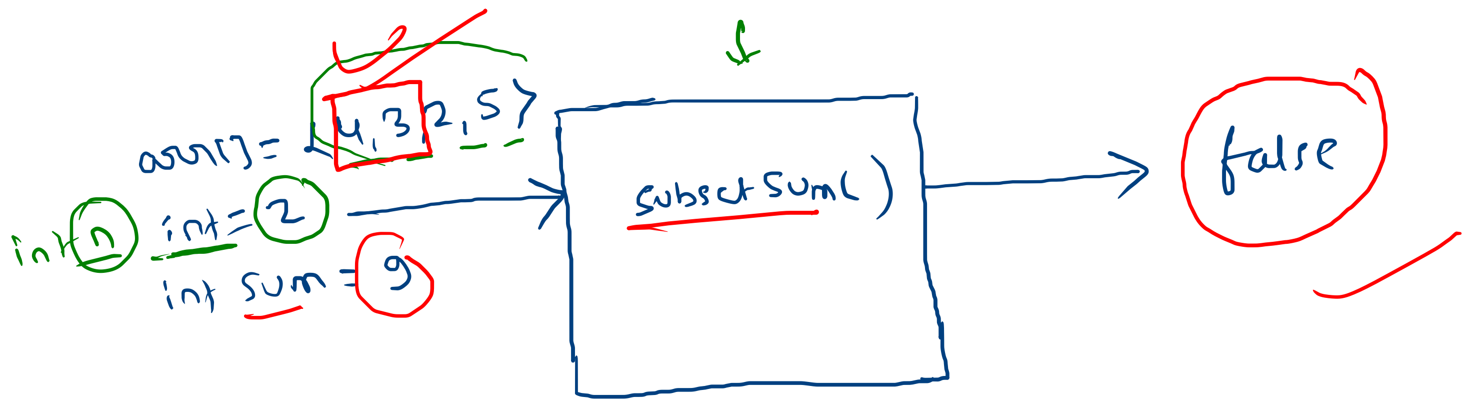




Recursive solution



is it Possible to give a targeted sum from the integers set (True/False)

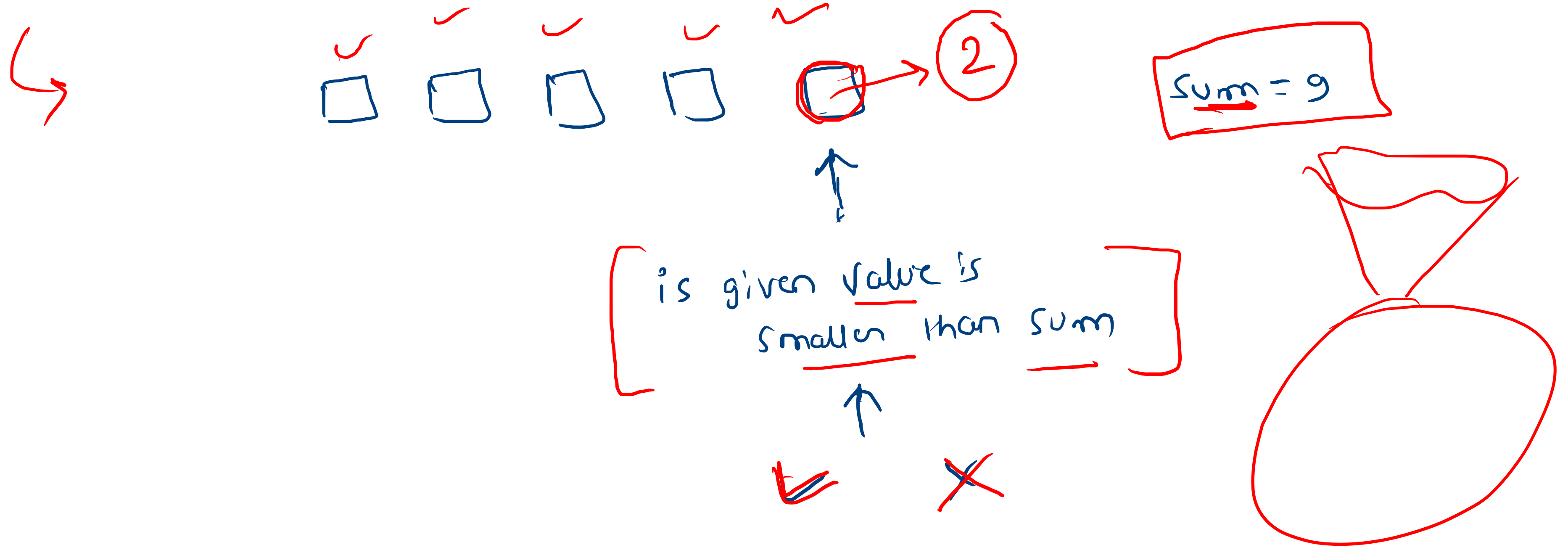


Recursive Solution

I/p : arr[] = { 3, 34, 4, 12, 5, 2 }
sum = 9

O/p : $\boxed{\text{True}}$
(4 + 3 + 2)

** All questions have almost same type of approach



Recursive solution

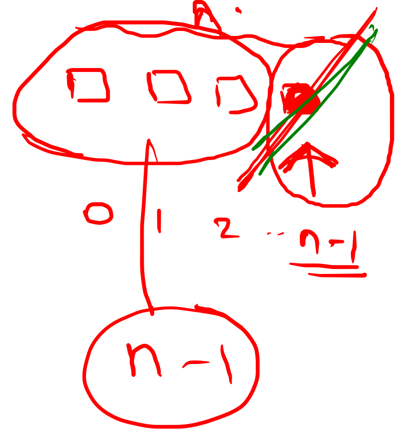
Logic ✓

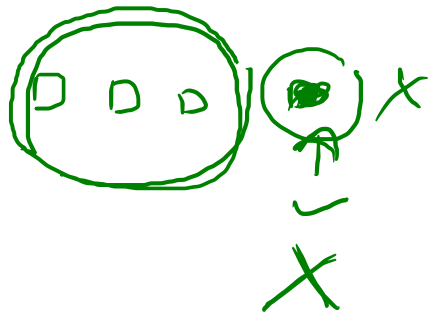
```
bool SubsetSum (vector<int> arr, int n, int sum)
{
    // Base condition
    if (arr[n-1] <= sum)
    {  return
          SubsetSum (arr, n-1, sum - arr[n-1])   ||   OR
          SubsetSum (arr, n-1, sum);
    }
    else
        return SubsetSum (arr, n-1, sum);
}
```

True

T

T





len = 9

({4, 3, 2}, 7)

Sum

Base condition

↓

```
if (n == 0) return false ✓
if ( sum == 0) return true ;
```

bool SubsetSum (vector <int> arr , $\boxed{\text{int } n}$, $\boxed{\text{int sum}}$)

int n ↓ 0

int sum ↓ 0

arr[] = { }

sum = 7

false

arr = {4, 1} { }

sum = 0

True

Recursive solution

Logic

Base condition

```
bool SubsetSum (vector<int> arr, int n, int sum)
{
    if (n==0) return false
    if ( sum ==0) return true ;
    if ( arr[n-1] <= sum)
    {   return
            SubsetSum( arr, n-1 , sum - arr[n-1] )   ||
            SubsetSum ( arr, n-1 , sum) ;
    }
    else
        return SubsetSum ( arr, n-1 , sum) ;
}
```

$$\begin{tikzcd}
\left(\text{Recursive solution}\right) \arrow[rr] \arrow[dr, bend right] & & \left(\text{DP solution}\right) \\
& \text{Memoization}^{*} \arrow[ur, bend right] &
\end{tikzcd}$$

Top-down

```
vector<vector<int>> dp(n+1, vector<int>(sum+1, -1));
```

2D vec

```
vector< vector<int>> dp(n+1, vector<int>(sum+1, -1));
```

	0	1	2	3	4	5	6	7
0	T	F	F	F	F	F	F	F
1	T							
2	T							
3	T							
4	T							

n (i)

→ Do initialisation

if (n==0) → F

if (sum==0) → T

Base Condition

DP Code

```
bool SubsetSum (vector<int> arr, int n, int sum)
{
    if(dp[n][sum] != -1) return dp[n][sum];

    if ( arr[n-1] <= sum)
    {   return dp[n][sum] =
              SubsetSum( arr, n-1, sum - arr[n-1])  ||
              SubsetSum ( arr, n-1, sum) ;
    }
    else
        return dp[n][sum] = SubsetSum ( arr, n-1, sum) ;
}
```