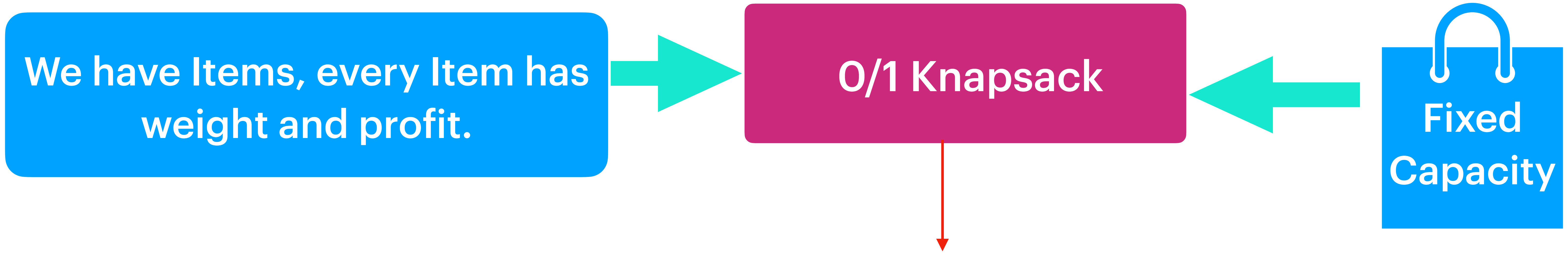


We have Items, every Item has weight and profit.

0/1 Knapsack

Fixed Capacity

Minimum Subset Sum Difference to make equal partition:

Problem Statement : Given a set of positive numbers, partition the set into two subsets with a minimum difference between their subset sum

Input: {1, 2, 7}
Expected Output : 4
The possible subsets are {1,2} & {7} so we need ‘4’ to make minimal possible two subsets.

Input: {1, 2, 3, 9}
Expected Output : 3
The possible subsets are {1,2,3} & {9} so we need ‘3’ to make minimal possible two subsets.

Input: {2,7,8}
Expected Output : 1
The possible subsets are {2,7} & {8} so we need ‘1’ to make minimal possible two subsets.

Input: {1, 3, 100,4}
Expected Output : 92
The possible subsets are {1,3,4} & {100} so we need ’92’ to make minimal possible two subsets.

Input: {1, 2,3,4}
Expected Output : 0
We can partition the given array into two subsets

Minimum Subset Sum Difference

int[] arr = {1,2,7}
totalSum = 10
sum = totalSum/2 = 5

Tabulation : dp[n][sum/2]
TimeComplexity : O(n*sum)
SpaceComplexity : O(n*sum)

Expected output {4}

Value	Index
1	0
2	1
7	2

Target Sum

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

leftSum = 3
targetSum=3 is possible without including last element.(i.e {1,2})

Algorithm:

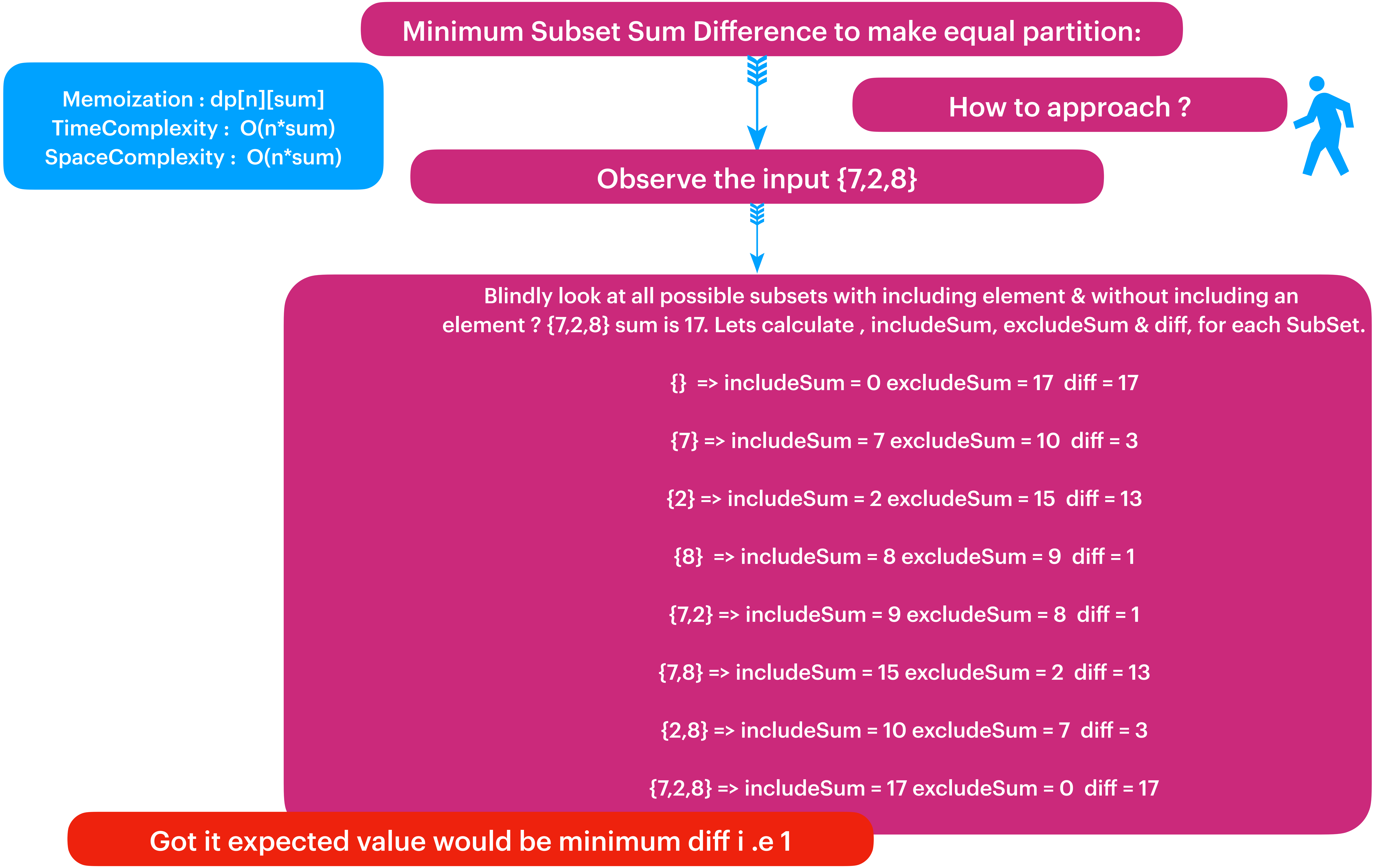
=> Derive tabulation by just following equal subset sum partition Algorithm.

=> Find out what the max subset can be possible with out including last element by moving left to right in n-1 row.

=> The true value talks about possible leftSum, then calculate the rightSum find out the difference.

totalSum = 10
leftSum = 3
rightSum = 10-3 = 7
expectedMinValue = rightSum - leftSum = 7-3 = 4

If we include 4 in the input array, we could make equal subset sum partition.
{1,2,4} {7}



Minimum Subset Sum Difference to make equal partition:

Memoization : dp[n][sum]
TimeComplexity : O(n*sum)
SpaceComplexity : O(n*sum)

How to approach ?

Observe the input {7,2,8}

Blindly look at all possible subsets with including element & without including an element ? {7,2,8} sum is 17. Lets calculate , includeSum, excludeSum & diff, for each SubSet.

{} => includeSum = 0 excludeSum = 17 diff = 17

{7} => includeSum = 7 excludeSum = 10 diff = 3

{2} => includeSum = 2 excludeSum = 15 diff = 13

{8} => includeSum = 8 excludeSum = 9 diff = 1

{7,2} => includeSum = 9 excludeSum = 8 diff = 1

{7,8} => includeSum = 15 excludeSum = 2 diff = 13

{2,8} => includeSum = 10 excludeSum = 7 diff = 3

{7,2,8} => includeSum = 17 excludeSum = 0 diff = 17

Got it expected value would be minimum diff i .e 1

Memoization : dp[n][sum]
TimeComplexity : O(n*sum)
SpaceComplexity : O(n*sum)

Items : {1,2,7}
sum: 10

✓ Indicates includeSum

✓ Indicates excludeSum

includeSum : 0
excludeSum : 0
diff : 0

1

Step1

Step2

includeSum : 1
excludeSum : 0
diff : 1

2

input : {2,7}

Step3

includeSum : 3
excludeSum : 0
diff: 3

7

input : {7}

Step4

includeSum : 10
excludeSum : 0
diff: 10

Step5

includeSum : 3
excludeSum : 7
diff:4

includeSum : 1
excludeSum : 2
diff: 1

input : {7}

7

Step6

Step7

includeSum : 8
excludeSum : 2
diff:6

Step8

includeSum : 1
excludeSum : 9
diff: 8

Step9

includeSum : 0
excludeSum : 1
diff : 1

2

input : {2,7}

includeSum : 2
excludeSum : 1
diff: 1

input : {7}

7

Step10

Step11

includeSum : 9
excludeSum : 1
diff: 8

Step12

includeSum : 2
excludeSum : 8
diff: 6

includeSum : 0
excludeSum : 3
diff : 3

input : {3}

7

Step13

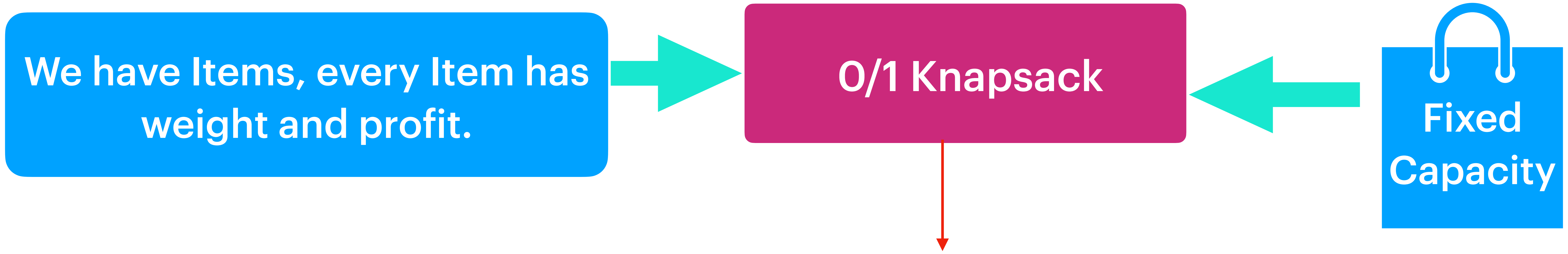
Step14

includeSum : 7
excludeSum : 3
diff: 4

Step15

includeSum : 0
excludeSum : 10
diff :10

return minimum diff i.e 4



We have Items, every Item has weight and profit.

0/1 Knapsack

Fixed Capacity

Count of Subset Sum

Problem Statement : Given a set of positive numbers, find the total number of subsets whose sum is equal to a given number ‘S’.

Input: {2,3,5}, S=5
Output: 2
The given set has '2' subsets whose sum is '5': {2,3}, {5}

Input: {1, 1, 2, 3}, S=4
Output: 3
The given set has '3' subsets whose sum is '4': {1, 1, 2}, {1, 3}, {1, 3}
Note that we have two similar sets {1, 3}, because we have two '1' in our input.

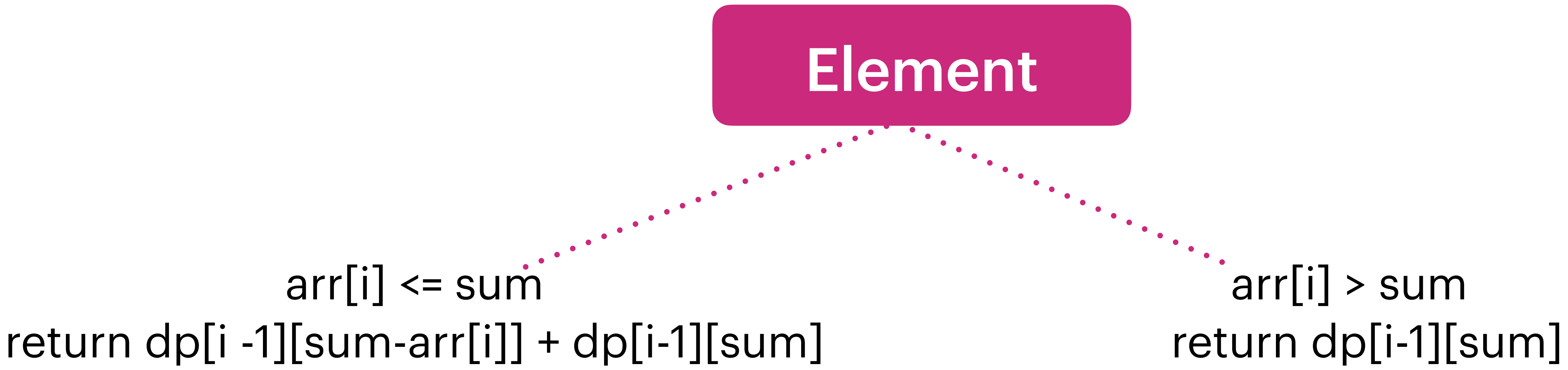
int[] arr = {2,3,5}
targetSum = 5
output : 2

Count of Subset Sum

Tabulation : dp[n][sum]
TimeComplexity : O(n*sum)
SpaceComplexity : O(n*sum)

Value	Index
2	0
3	1
5	2

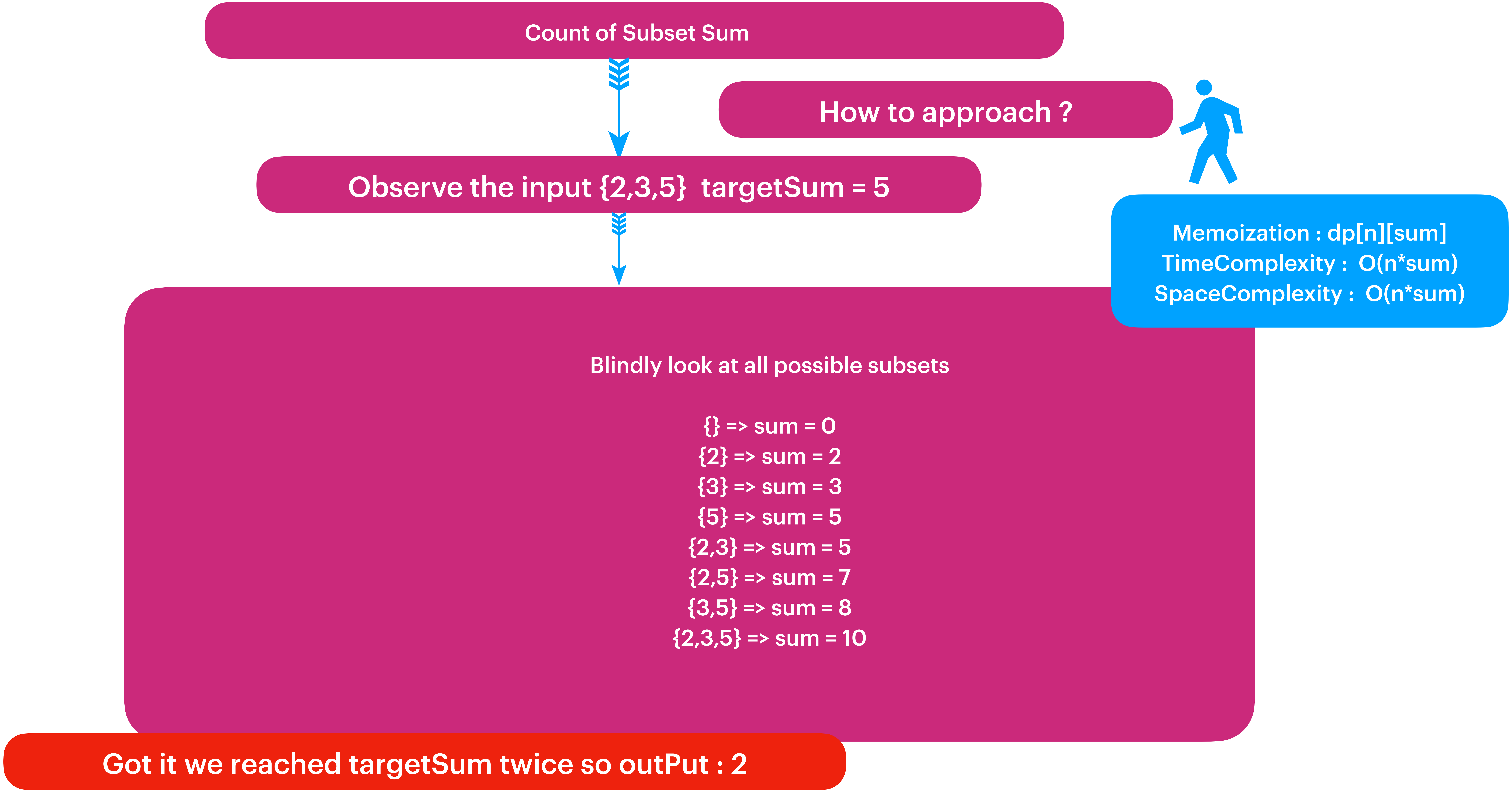
Target Sum					
0	1	2	3	4	5
1	0	1	0	0	0
1	0	1	1	0	1
1	0	1	1	0	2



Element

arr[i] <= sum
return dp[i -1][sum-arr[i]] + dp[i-1][sum]

arr[i] > sum
return dp[i-1][sum]



Count of Subset Sum

How to approach ?

Observe the input {2,3,5} targetSum = 5

Memoization : dp[n][sum]
TimeComplexity : O(n*sum)
SpaceComplexity : O(n*sum)

Blindly look at all possible subsets

{} => sum = 0
{2} => sum = 2
{3} => sum = 3
{5} => sum = 5
{2,3} => sum = 5
{2,5} => sum = 7
{3,5} => sum = 8
{2,3,5} => sum = 10

Got it we reached targetSum twice so outPut : 2

Memoization : dp[n][sum]
TimeComplexity : O(n*sum)
SpaceComplexity : O(n*sum)

Items : {2,3,5}
sum : 5

Step1
2
sumLeft = 5

Step2
3
sumLeft = 3
input : {2,5}

Step3
5
sumLeft = 0
input : {5}

7 > sumLeft
Don't need to derive,
move right

Step4

Step5
sumLeft = 0

Step6
5
sumLeft = 3
input : {5}

Step7
5 > sumLeft
Don't need to derive,
move right

Step8
sumLeft = 3

Step9
3
sumLeft = 5
input : {3,5}

Step10
5
sumLeft = 2
input : {5}

Step11
5 > sumLeft
Don't need to derive,
move right

Step12
sumLeft = 2

Step13
5
sumLeft = 5
input : {5}

Step14
sumLeft = 0

Step15
sumLeft = 5

Possible subSet count is equals to total sumLeft = 0 at subProblem .
Here it is 2.