Todays Content:

→ Subset sum

- Case:1 element picked 0/1 time
  - a) Sum possible
  - b) #ways to get Sum
  - c) Min elements
- Case:1 element picked 0/any time
  - a) Sum possible
  - b) #ways to get Sum
  - c) Min elements

→

→ N elements divide into 2 parts such that both have equal sum

Ig) Given ar[N] elements

→ Check if There exists a subset with sum = k : all ele +ve

Note: Every element can be picked only once atmax

	0	1	2	3	4	5	6	7
ar[] =	7	4	9	6	10	13	14	11

k=22 : {9 13} {7 4 11} {9 6 7} .. return True

k=26 : {4 9 13} {7 4 9 6} ... return True

Idea1: For all subsets check if subset sum = k

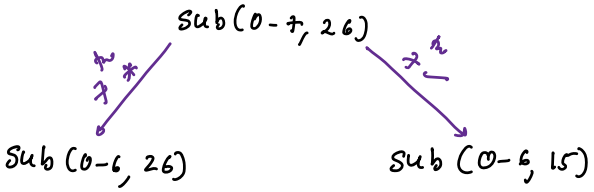
- TC1: $2^n * n$ {Using bit manipulations}
- TC2: 2^n {Using backtracking}

1. Subproblems ✓
2. Stay on it old assume it does

Idea2: k=26 :

	0	1	2	3	4	5	6	7
	7	4	9	6	10	13	14	11

// Using elements from [0-7] check if we can get sum = 26



Dp Steps:

1. dpState: dp[i,j] (i: index, j: rem sum): Using ind from [0-i] check if we get sum = j

2. dpExpression: dp[i,j]: 0 1 2 ... i-1 i

leave / pick

dp[i,j] = dp[i-1, j] or dp[i-1, j - A[i]]

: if j - A[i] >= 0, only then above state exists.

Because: -ve sum is not possible hence we keep above condition

3. final ans: Using elements (0, n-1) check if we can get sum k : dp(n-1, k)

4. Dp table: dp[n][k+1]

5. TC: # States * TC for each state SC: O(N*k) + StackSize: n

O(N*k) * O(1), TC: dp < TC: BT suggests overlapping subProblems

Code

```
int dp[n][Ksi] = INVALID [-1/...   or We can only Hashmap.

int  subsum ( int i, int j, int ar[] ){
    if (i<0){ // no elements to pick
        if(j==0) {return 1 } // Since no ele, are we can get sum 0
                    True
        else {return 0 }
                    False
    }
    if( dp[i][j] == -1) {
        a = subsum( i-1, j, ar)
        if( j - ar[i] >= 0){
            b = subsum( i-1, j - ar[i], ar)
        }
        dp[i][j] = a | b
    }
    return dp[i][j]
}
```

1Q) Given ar[N] elements no: of subset with sum = k : all elements +ve

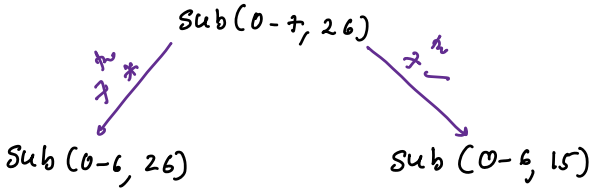
Note: Every element can be picked only once atmost

Idea:

1. Subproblems ✓
2. Stay on it old assume it does

	0	1	2	3	4	5	6	7
k=26 :	7	4	9	6	10	13	14	11

// Using elements from [0-7] count no: of subsets sum = 26



Dp steps:

1. dpState: dp[i,j] (i: index, j: rem sum) : Using ind from [0-i] count no: of subsets with sum=k

2. dpExpression: dp[i,j]: 0 1 2 ... i-1 i

dp[i,j] = dp[i-1, j] + dp[i-1, j-A[i]]

: if j - A[i] >= 0, only then above state exists.

Because: -ve sum is not possible hence we keep above condition

3. final ans: Using elements [0, n-1) check if we can get sum k : dp[n-1, k]

4. Dp table: dp[n][k+1]

5. TC: #States * TC for each state SC: O(N*k) + StackSpace: n

O(N*k) * O(1), TC: dp < TC: BT suggests overlapping subProblems

int dp[n][Ksi] = INVALID /-1/... or We can only hashmap.

```
int subsum(int i, int j, int ar[]){
    if(i<0){ // no elements to pick
        if(j==0){return 1 } // True Since no elle, are we can get sum 0
        else{return 0 // not passible to get target sum} // False
    }
    if(dp[i][j]==-1){
        a = subsum(i-1, j, ar)
        if(j - ar[i]>=0){
            b = subsum(i-1, j-ar[i], ar)
        }
        dp[i][j] = a+b
    }
    return dp[i][j]
}
```

TODO: Count no: of time we return 1 ?

: Check correctness of it

1Q) Given ar[N] elements

min ele req to get subset sum = k : all ele +ve

Note: Every element can be picked only once atmax

Idea:

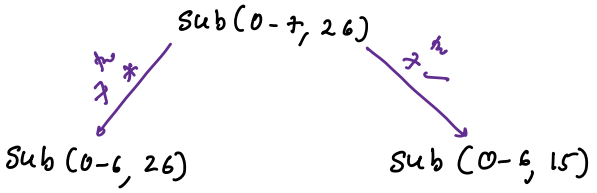
	0	1	2	3	4	5	6	7	
k=26:	7	4	9	6	10	13	14	11	ans=3

↳ {4 9 13} {7 4 9 6}
3ele 4ele

1. Subproblems
2. Stay on Hold assume it does

	0	1	2	3	4	5	6	7
k=26:	7	4	9	6	10	13	14	11

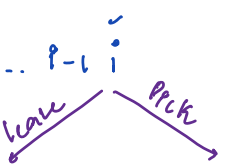
// Using elements from [0-7] min no: of ele to get sum = 26



Dp steps:

1. dpState: dp(i,j): Using ind from [0-i] min ele req to sub sum = k (ind, rem sum)

2. dpExpression: dp(i,j): 0 1 2 ... i-1 i



dp(i,j) = min(dp(i-1, j) , dp(i-1, j-A[i]) +1)

Because we are keeping count of we picked

: if j - A[i] >= 0, only than above state exists.

Because: -ve sum is not possible hence we keep above condition

3. Final ans: Using elements (0, n-1) check if we can get sum k
: dp(n-1, k)

4. Dp table: dp[n][k+1]

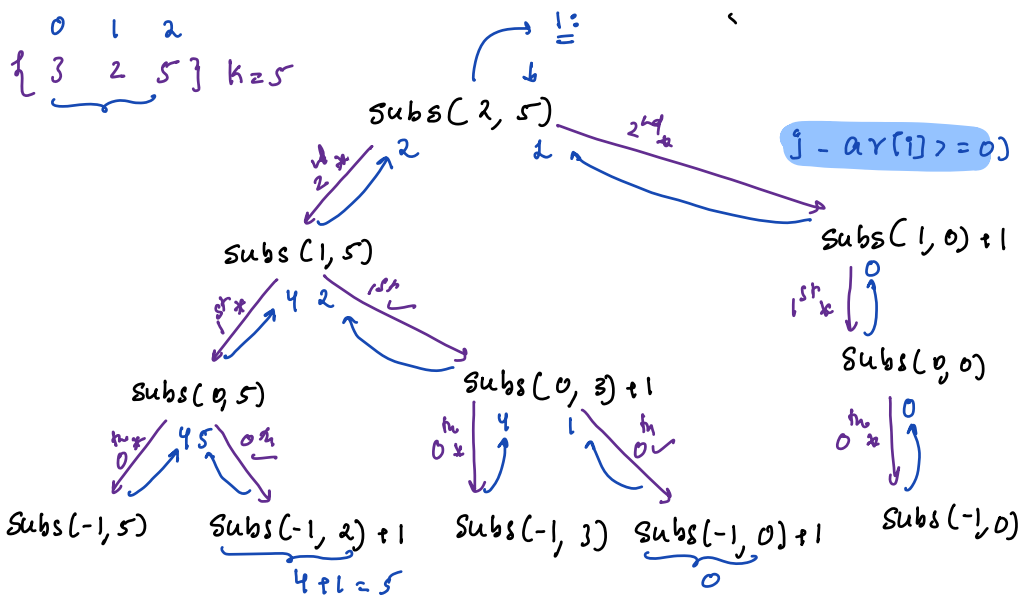
5. TC: # States * TC for each state SC: O(N*k) + StackSize: n
O(N*k) * O(1), TC: dp < TC: BT suggests overlapping subProblems

int dp[n][k+1] = INVALID /-1/... or We can only Hashmap.

```
int subsum(int i, int j, int ar[]){
    if(i<0){ // no elements to pick
        if(j==0){return 0} // Sum is possible, We need 0 ele to get Target sum.
        else{return n+1} // n+1: Impossible, because we can never pick n+1
                         //  elements // n=0, -1, INT_MAX Invalid return
    }
    if(dp[i][j]==-1){
        a = subsum(i-1, j, ar) // not pick
        if(j - ar[i]>=0){
            b = subsum(i-1, j-ar[i], ar) +1 // pick
        }
        dp[i][j] = min(a, b)
    }
    return dp[i][j]
}
```

Dry run:

ar[3] = { 3 2 5 } k=5
N=3



4Q) Given ar[N] elements TODO

Check if There exists a subset with sum = k : all ele +ve

Note: Every element can be picked as many time

5Q) Given ar[N] elements TODO

ways exists a subset with sum = k : all ele +ve

Note: Every element can be picked as many times

6Q) Given ar[N] elements TODO

min ele req to get subset sm sum = k : all ele +ve

Note: Every element can be picked as many times as required

Ques) Given N elements, Check if we can divide all elements into part1 or part2 such that both parts have equal sum.

Eg: ar[6] = { 1 5 3 6 9 2 } return true

{ 1 9 3 } = 13 = { 5 6 2 }

Eg: ar[5] = { 3 5 2 100 90 } return true

{ 3 5 2 90 } = 100 = { 100 }

Eg: ar[5] = { 6 2 1 4 10 }

Obs: If Total sum of all ele is odd : return false

Eg: ar[3] = { 6 2 10 } return false

Obs1: If sum of all ele is odd : return false.

else {

Every ar[i] either 1 or 2 but not both

{ Part A } { Part B }

Sum(A) + Sum(B) = Sum of all ar[]

↳ Sum(A) = Sum(B)

Sum(A) + Sum(A) = Sum of ar[]

2 x Sum(A) = Sum of ar[]

$$Sum(A) = \frac{\text{Sum of ar[]}}{2}$$

}

Obs: If there exists a subset with Sum = TS/2 TS: Sum of all ar[]

In that case remaing ele Sum = TS/2

8Q) Given ar[N] ele, divide all elements into 2 subsets

Divide in such a way that, abs diff between both is minimum

ar[N] → S_1, S_2 → $|sum(S_1) - sum(S_2)|$ is minimized

Eg: ar[] = {1 5 7}

Case1: {1 7} {5} Diff: |8-5| = 3

Case2: {1 5} {7} Diff: |6-7| = 1

Eg: ar[] = {3 2 4 7 6 3}

Case1: {3 2 4} {7 6 3} Diff: |9-16| = 7

Case2: {7 6} {3 2 4 3} Diff: |13-12| = 1

Eg: ar[] = {4 10 2} TS = 16

Obs: If we do equal distribution, diff between them minimum

TS/2

ar[] = {4 10 2} = 8 *

{4 10 2} = 7 *

{4 10 2} = 6 ✓

S_1	S_2
8	8
↓-1	↓+1
7	9
↓-1	↓+1
6	10

= return |6-10| = 4

TS/2

Eg: ar[5] = {1 3 11} = 7 *

{1 3 11} = 6 *

{1 3 11} = 5 *

{1 3 11} = 4 ✓

S_1	S_2
7	8
↓	↓
6	9
↓	↓
5	10
↓	↓
4	11

= return |4-11| = 7

Eg: ar[5] = { 1 (0) 3 (1) 11 (2) } k = 15/2 = 7

: Try to do equal distribution

S_1	S_2
7	8 *
6	9
5	10
4	11

: dp[3][8]:

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

: Using 0-2 can we get sum 7: dp[2][7] = false

: Using 0-2 can we get sum 6: dp[2][6] = false

: Using 0-2 can we get sum 5: dp[2][5] = false

: Using 0-2 can we get sum 4: dp[2][4] = True

Idea: Given ar[N] & Say Sum of all of all ar[] = S

↳: If say we do equal distribution $\frac{S_1}{S/2}$ $\frac{S_2}{}$

: Check if we get subset with sum $k = S/2$

dp[N][S/2+1] → (1)

Fill dp table

$S_1 = S/2$ $S_2 = S - S_1$

```
while (dp[N-1][S1] == False)
    S1 = S1 - 1   S2 ++
return |S1 - S2|
```

TC: $O(N * S/2)$ // S = sum of all ar[] elements.