Todays Content:

- Subset sum = k

- Check, if there exists s_1 & s_2 such that

  $sum(s_1) - sum(s_2) = k$

1Q Given ar[N] elements, # Calulate no: of subsets with sum == k

	0	1	2	3	4	5	6	7
ar[8] =	{7	4	9	6	10	13	11	14}

All elements >= 0 k >= 0

If ele == 0 take care of Edge Case → dp[0][0] = 2

k = 22 → {7 4 11}
{7 9 6}
{13 9}

Ideas:

1) Generate all Subsetsums, & compare == k

- a) Bit Manipulation = TC ⇒ $O(2^N * N)$
- b) Back Tracking = TC ⇒ $O(2^N)$

// using elements from index [0-7] get no: of subsets with sum = 22

Sbs(0 - 7, 22)

leave it → Sbs(0 - 6, 22)

pick it → 7th index element → Sbs(0 - 6, 8)

Sbs(0 - 6, 22) + Sbs(0 - 6, 8)

$\underbrace{Sbs(0-6,\ 22)}_{\#\ subsets}$: leave it → Sbs(0-5, 22); pick it → Sbs(0-5, 11)

$\underbrace{Sbs(0-6,\ 8)}_{\#\ Subsets}$: leave it → Sbs(0-5, 8); pick it → * not possible → Sbs(0-5, -3)

Ex: _ _ _ _ _ _ | 6 10 14 2 8   k = 45

using 6 elem, 25 | ✓ * ✓ * *

using 6 elem, 25 | * ✓ * ✓ ✓

// overlapping Subproblems

$dp(i, j)$ = # Number of subsets with sum = j using index from [0, i]

$$dp(i,j) = \left\{ \underbrace{dp(i-1, j)}_{\text{// leave } i^{th} \text{ index}} + \underbrace{dp(i-1, j-ar[i])}_{\text{// pick } i^{th} \text{ index}} \quad j >= ar[i] \right\}$$

using [0, i-1] subsets with sum = j

using [0, i-1] subsets with sum = j - ar[i]

dpTable: $dp[N][k+1]$

Base Conditions:

$dp(i,j) = dp(i-1, j) + dp(i-1, j-ar[i])$ if $j >= ar[i]$

// if i == 0 expression fails.

Ex: $ar[4] = \{4\ 2\ 1\ 3\}$ (indices 0 1 2 3), $k = 7$

$dp[4][8]$

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

$dp[0, i]$ = using all indices from [0-0]
no: of subsets with sum = i

$dp[0, 0]$ = using all indices from [0-0]
no: of subsets with sum = 0
{4}, empty subset { } = 0

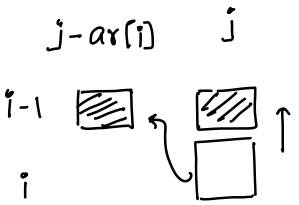
Base Conditions:

```
for( j = 0; j <= k; j++){ dp[0, j] = 0}
dp[0][0] = 1,
if (ar[0] <= k)
{ dp[0][ar[0]] += 1
}
```

$ar[3] = \{7\ 4\ 2\}$

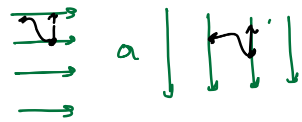
$k = 4$ ← $ar[0] > 4$

Pseudo Code:

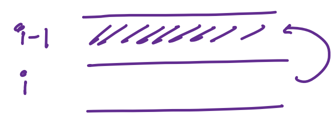


Iterate top-down

Iterate left-right




```
i = 1 ; i < N ; i++) {
    j=0 ; j <= k ; j++) {
        dp(i, j) = dp(i-1, j)
        if ( j >= ar[i]) {
            dp(i, j) += dp( i-1, j- ar[i])
        }
    }
}
return dp[N-1][k]
```



[At any given point we only 2 rows]

TC: $\underline{\# \text{States}}$ * # $\underline{\text{TC for each state}}$ ⇒ $O(N^*k)$

(N*k) $O(1)$

SC: $O(N^*k)$

→ Space Optimization ⇒ $O(2^*k)$ ⇒ $O(N)$ → TODO

// ar[3] = {0, 0, 0} k = 0

dp[3][1]

	0
0	2
1	4
2	8

return 8

dp(i, j) = dp (i-1, j)

if (j >= ar[i]) {
0 >= 0

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

}

dp[1, 0] = dp[0, 0] + dp[0, 0] = 2*dp[0, 0]

dp[2, 0] = dp[1, 0] + dp [1, 0] = 2*dp[1, 0]

// ar[4] = { 2 0 4 0 }   k = 0

dp[4][1] =

	0
0	1
1	2
2	2
3	4

dp(i, j) = dp(i-1, j)
if(j >= ar[i]){
0 >= 4
dp(i, j) += dp(i-1, j - ar[i])
}

dp[1, 0] = dp[0, 0] + dp[0, 0] = 2 * dp[0, 0]

dp[2, 0] = dp[1, 0] = 2

dp[3, 0] = dp[2, 0] + dp[2, 0] = 2 * dp[2, 0] = 4

// Given N Elements, find length of smallest subset with sum = k

dp(i, j) = { Using [0, -i] find length of smallest subset with sum = j }

$$dp(i,j) = \min \left\{ \underbrace{dp(i-1, j)}_{\text{// leave } i^{th} \text{ index}},\ \underbrace{dp(i-1, j-ar[i]) + 1}_{\text{// pick } i^{th} \text{ index}} \right\}$$

j >= ar[i]

[Since we are pick i^{th} element length increases]

dp[N][k+1] ⇒ Table Size

Base Conditions : i = 0, Expression fails

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

dp[0, 6] = 0 ~~*~~ ∞ ~~*~~, $\boxed{N+1}$ ✓

dp[4][8]

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

[Min number of elements required to get sum = 6 for
This way of initialization *]

→ dp[0, 0] = 0
[min elem, to get sum = 0]

→ Base Conditions

```
for( j = 0 ; j <= k ; j++ ) { dp[0, j] = N+1 }
dp[0, 0] = 0,
if ( ar[0] <= k ) {
    dp[0, ar[0]] = 1
}
```

⇒ Code:

```
i = 1 ; i < N ; i++ ) {
    j = 0 ; j <= k ; j++ ) {
        dp(i, j) = dp(i-1, j)
        if ( j >= ar[i] ) {
            dp(i, j) = min(dp(i, j), dp(i-1, j- ar[i]) +1)
        }
    }
return dp[N-1][k]
```

TC: O(N)(k)   SC: O(N)(k)   optimize : O(2*k) ≈ O(k)

3rd) Check if there exist a subset with sum = k

$dp(i, j)$ = { Using [0, i] check, if there exists a subset with = j }

// not pick   pick

$dp(i, j) = dp(i-1, j) \;||\; dp(i-1, \underbrace{j - ar[i]}_{\text{if } (j >= ar[i])})$

bool dp[N][k+1];

Base Conditions:

$\underline{dp(i, j) = dp(i-1, j)} \;||\; dp(i-1, \underbrace{j - ar[i]}_{\text{if } (j >= ar[i])})$

↳ if i == 0, fails

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

dp[4][8]

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

Base Code:

j = 0; j <= k; j++) { dp[0, j] = False }

dp[0, 0] = True

if (ar[0] <= k) {
  dp[0, ar[0]] = True
}

Code // TODO

3Q // Given ar[N], divide all elements into 2 Subsets

Check, if we can divide all elements in 2 Subsets, such that both subsets have equal sum

[Note: ar[i] can only goto single subset]

Ex1: ar[6] = { 1 5 3 6 9 2 } = { True }

→ { S_1 } { S_2 }

→ {5, 6, 2} {1, 9, 3}

Ex2: ar[4] = { 1 3 4 2 } return True

S_1 S_2

{1 4} {2 3}

Ex3: ar[5] = { 6 2 1 4 10 } return false

S_1 S_2

Ex4: ar[3] = { 10 2 6 } return false

S_1 S_2

//obs:

1) If sum of all elements is odd = { return false }

2) All Elements → S_1, S_2] Sum(S_1) + Sum(S_2) = [Sum of all arr[] elements]

Sum(S_1) = Sum(S_2)

3) Sum(S_1) = { Sum of all array elements } / 2

// Check if there exist a subset with sum = { Sum of arr[] } / 2

//Q) Given ar[N] elements, divide all elements in 2 Subsets

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

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

→ abs(-3) = abs(3)

Ex: ar[ ] = {1, 5, 7} → S_1 : {1, 5}, S_2 = {7}

$|sum(S_1) - sum(S_2)|$ → 7 - 6 = 1

Ex2: ar[ ] = { 3 2 4 7 6 3 } → S_1 : { 7 4 2 }, S_2 : { 3 3 6 }

$|sum(S_1) - sum(S_2)|$ → 13 - 12 = 1

Ex3: ar[ ] = { 4 10 2 } → S_1 : {10}, S_2 : {4, 2}

$sum(S_1) - sum(S_2)$ → 4

TS = 16, We can get min, diff if equally distributed?

ar[4, 10, 2]	k/subset sum	S_1	S_2
*	8	8	8
*	7	7	9
✓	6	6	10 ⇒ 4

16 → S_1, S_2

Ex4: ar[4] = { 3 7 5 10 }

TS = 25:

ar[4] = { 3 7 5 10 } k = 12 ✓

25 → S_1, S_2

12 ⟷ 13 = 1

Ens: ar[5] = { 1 3 11 }

TS = 15

15 → S_1, S_2

S_1	S_2
7	8
6	9
5	10
4	11

= 7

ar[5] = { 1 3 11 } k

will we apply dp, 4 times *

- * 7, dp[2, 7]
- * 6, dp[2, 6]
- * 5 dp[2, 5]
- ✓ 4 dp[2, 4]

→ N=3, k=7 { 1 3 11 } → We will fill Entire dp table

bool dp[3][8]

	0	1	2	3	4	5	6	7
0								
1								
2								

dp[2, 7] =
↳ using [0, 2] check if we can get 7 or not?

dp[2, 4] dp[2, 5] dp[2, 6] → already filled
↳ Re-un filled state

// Given Elements, k = (Total Sum)/2 → TC: $\left[N * \frac{[\text{Total Sum arr}[\,]]}{2} \right]$

// Fill dp table dp[N][k+1] → TODO

SC: $N * \frac{[\text{Total sum arr}[\,]]}{2}$

SC: O(Total sum arr[])

```
S1 = Total Sum/2      S2 = Total Sum - S1
while( dp[N-1][S1] == false ){
    S1 = S1-1, S2 = S2+1
}
return S2 - S1
```

↳ Subset sum S_1 is not possible continue process