Todays Content

1. Subsets/Subsequences
2. Check if there exists a subsequence with sum=0

#Subarray: Continous part f an array.

#Subsequence: Take any element in arr[].
Arrage them in increasing order f index

0 1 2 3 4 5
Ex: arr[] = { 7 2 6 9 10 8 }

Subsequence:

{2 9 10 8} #subsequence   {6 10 7 8} #Not subseque

{7 6 9 8} #subsequence

{2 7 10 8} #Not subseque

Subset : Same as sequence No need to maintain order
We identify purely based on data it has

0 1 2 3 4 5
Ex: arr[] = { 7 2 6 9 10 8 }

Subset

{2 9 10 8} {2 8 10 9} # both are same

Note: An { } subset/subsequce considered

0 1 2
arr[] = {3 2 9}

All Subsets: # 8 = 2^3

{ }

{3} {2} {9}

{2 3} {9 2} {3 9}

{9 3 2}

0 1 2
arr[] = {3 2 9}

All Subsequence # 8 = 2^3

{ }

{3} {2} {9}

{3 2} {2 9} {3 9}

{3 2 9}

	Continous	Order Index
Subarray	Yes	Yes
Subsequence	No	Yes
Subset	No	No

Given an arr[N] check if there exists a subset with sum = k.

Note: Cannot use any kind of extra space. ↳ Any elements/Need not be continous

Note: # Empty set is allowed.

Ex: 0 1 2 3 4 5 6

arr[] = { 2 -3 6 11 4 -5 6 }

k = 14 : { 2 6 6 } ✓

k = 16 : { 6 4 6 } ✓

Constraints

$1 <= N <= 20;$

$-10^6 <= ar[i] <= 10^6$

Idea:

```
for( — ) {    # It will generate pairs, but we need subsets *
    for( — ) {
        if( — ) {
        }  return True
    }
}
```

Idea2: Generate all subsets, calculate sum & cmp == k.

0 1 2

Ex: arr[] = { 2 3 -6 }

i:	2	1	0	
0:	0	0	0	{ }
1:	0	0	1	{2}
2:	0	1	0	{3}
3:	0	1	1	{2 3}
4:	1	0	0	{-6}
5:	1	0	1	{2 -6}
6:	1	1	0	{3 -6}
7:	1	1	1	{2 3 -6}

#obs: N = 3 # 2^3 subsets = 8

Bits : 3 ⇒ # 3 ele & each ele

Numbers 2 1 0 maps to 1 bit

0

1

2

⋮

7

$$\begin{array}{lcccc} & 0 & 1 & 2 & 3 \\ \text{Eg: arr[] = \{} & 2 & 3 & -6 & 3\ \} \end{array}$$

Numbers

#obs: N = 4 # 2^4 subsets = 16

Numbers		3	2	1	0		Sum
0	:	*0	*0	*0	*0	{ }	0
1	:	*0	*0	*0	✓1	ar[0]	2
2	:	*0	*0	✓1	*0	ar[1]	3
3	:	*0	*0	✓1	✓1	ar[0] + ar[1]	5
4	:	*0	✓1	*0	*0	ar[2]	-6
5	:	*0	✓1	*0	✓1	ar[0] + ar[2]	-4
6	:	*0	✓1	✓1	*0	ar[1] + ar[2]	-3
7	:	*0	✓1	✓1	✓1	ar[0] + ar[1] + ar[2]	-1
8	:	✓1	*0	*0	*0	ar[3]	3
⋮							
15	:	TODO					

Bits : 4

Numbers 3 2 1 0

0

1

2

⋮

15

Generalize

Given ar[N] # SubSet = 2^N

Bits : N

Numbers N-1 ... 2 1 0

0

1

⋮

2^N-1

#PseudoCode:

```
for every number 0 .. 2^N-1 :
    generate N bits & map with subset & get subset sum.
    if (sum == target) {
        return true;
    }
}
return false;
```

Constraints

$1 <= N <= 20;$

$-10^6 <= ar[i] <= 10^6$

```
boolean checkSum(vector<int> &ar, int k){ TC: O(2^N * N)  SC: O(1)
    int N= ar.size();
    for(int i=0; i<2^N; i++){
        # i: Generate N Bits: {0.. N-1} → subset & cal sum.
        int sum=0;
        for(int j=0; j<N; j++){
            if((i>>j)&1==1){ # i: j^th Bit set ⇒ ar[j] selected
                sum=sum+ar[j];
            }
        }
        if(sum==k){
            return true;
        }
    }
}   return false;
```

Dry Run:

	0	1	2	3
Ex: ar[] = {	2	3	-6	3 }

Numbers

i:	3	2	1	← 0
0 ↓				
	✓	✗	✓	✓
i = 11	1	0	1	1
↓ 15				

sum = sum + ar[0] + ar[1] + ar[3] = 8

if(sum==target){return True}

2B Given ar[N] it contains all elements from 1.. N.

1 element from 1 to N repeats

1 element from 1 to N missing

Return both repeat & missing element

Note: No Extra space, No modifying array.

Constraints:

$1 <= N <= 10^6$

$1 <= ar[i] <= N.$

Ex: missing repeat

ar[5] = { 2 2 1 4 5 }

ar[7] = { 1 3 6 5 4 6 7 }

Idea: