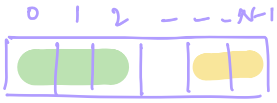
Today's content

- → Revise subarray
- -> Subsequences vs subsets
- → Check subset with given sum
- → sum of all subsets
- → sum of max of all subsets.

Subarray basics.



Continuous part of the array.

// Subarray: [s e].

Total subarrays = $\frac{N(N+1)}{2}$.

Subsequence:

Sequence generated by deleting zero or more elements from the array.

0	1	2	3	4	5	6	7		
3	-2	0	1	8	7	4	9		
×	✓	✓	×	✓	×	✓	×	✓	[-2,0,8,4]
✓	✓	✓	✓	×	×	×	×	✓	[3,-2, 0,1]
✓	✓	✓	✓	✓	✓	✓	✓	✓	[All elements]
×	✓	×	×	✓	✓	×	✓	✓	[-2,8,7,9]
×	×	×	×	×	×	×	×	✓	[Empty sequence]
×	×	✓	✓	✓	×	✓	✓	✓	[0,1,8,4,9]
									[1,0,8,4,9].

Subarrays vs subsequences.

Every subarray is a subsequence.

Every subsequence is not a subarray.

$$ar[5] : \{\overset{0}{-3}\ \overset{1}{0}\ \overset{2}{1}\ \overset{3}{2}\ \overset{4}{6}\}$$

	Subarray	Subsequence
[1, 2, 6]	✓	✓
[-3, 1, 2]	×	✓
[0, 1, 2]	✓	✓
[-3, 1, 6]	×	✓
[6, 1, 0]	×	×

Sorting in subsequence.

$$arr[3] : \{\overset{0}{3}, \overset{1}{-2}, \overset{2}{1}\} \xrightarrow{Sort} \{-2, 1, 3\}.$$

All subsequences.	All subsequences.
{ }	{ }.
{3}	{3}.
{-2}	{-2}
{1}	{1}
{3, -2}	{-2, 3}.
{-2, 1}	{-2, 1}
{3, 1}	{1, 3}
{3, -2, 1}	{-2, 1, 3}.

order is important

If we sort, **subsequences will change!.**

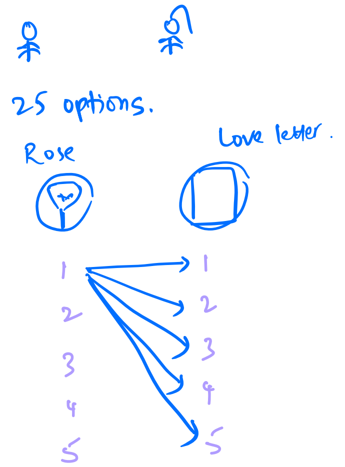
Subsets: Exactly same as subsequence, order doesn't matter.

$$\{\overset{0}{3}, \overset{1}{-2}, \overset{2}{1}\} \xrightarrow{\text{sort}} \{-2, 1, 3\}.$$

All subsets.	All subsets.
{ }	{ }
{3}	{3}
{-2}	{-2}
{1}	{1}
{3,-2}	{-2,3}
{3,1}	{1,3}
{-2,1}	{1,-2}
{3,-2,1}	{-2,1,3}.

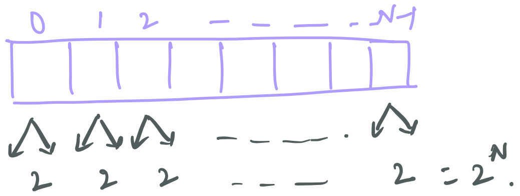
If you sort, subsets won't change.

Valentine day.



Count no. of subsequences.

Given N Elements?



No. of subsequences = 2^N. [{ } is considered].

No. of subsets = 2^N ⟶ [No duplicates].

$$\begin{array}{ccc} 0 & 1 & 2 \\ [1, & 2, & 2]. \end{array}$$

Subsequences. = 8.

{ } {1} {2} {2}

{1,2} {2,2} {1,2}

{1,2,2}.

$$\begin{array}{ccc} 0 & 1 & 2 \\ [1, & 2, & 2] \end{array}$$

Subsets. = 4.

{ } {1} {2} {2}

{1,2} {2,2} {1,2}

{1,2,2}.

// Subsequences → order matters. → 2^N.

// Subset → order doesn't matter → 2^N (distinct).

Q: Given N distinct elements, check if there exists (→ true/false) a subset with sum=k.

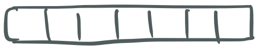
Ex1:

$$\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ [3 & -1 & 0 & 6 & 2 & -3 & 5] \end{array}$$

k=10, [3,2,5].
[-1,6,5]
[6,2,-3,5]
True.

k=20, return false.

ideas:

(i) Prefix sum →  won't help.

(ii) dictionary → ×

(iii) 2 loops → × →

```
for i in range(0,N)
    for j in range(i,N).
        if (ar(i)+ar(j)==k)
```

(iv) carry forward → → ←

(v) Sorting → ×

(vi) Sliding window → ×

Bit manipulation:

$$\begin{array}{ccc} 0 & 1 & 2 \\ [\,3, & -2, & 1\,]. \end{array} \quad 2^N = 2^3 = 8. \text{ subsets.} \qquad [0, 7].$$

map each 8 subsets to a number from [0,7].

		2	1	0	subsets.	subset sum.
0	:	0	0	0	[ ]	0
1	:	0	0	1	[3]	3
2	:	0	1	0	[-2]	-2
3	:	0	1	1	[3, -2]	1
4	:	1	0	0	[1]	1
5	:	1	0	1	[3, 1]	4
6	:	1	1	0	[-2, 1]	-1
7	:	1	1	1	[3, -2, 1]	2

Break:
10: 11:00
10: 19:00
20:00.

```
def checkForSubsetSumk(arr, k, n)
{
    for i in range(0, $2^N$).
        // for every i, check all N bit positions
        sum=0. // each subset sum.
        for j in range(0, N)
            if (checkBit(i,j))
                sum = sum + arr[j]
        if(sum==k) return true.
    return false
}
```

TC: $O(N*2^N)$.

SC: $O(1)$.

Advanced content.

(i) using backtracking $O(2^N)$.

(ii) using recursion + dp [$O(n*k)$].

Q2: Given N distinct elements, sum of subset sums.

{ 3, 1, 4}.

Subsets:	sum.
{ }	0
{3} ✓	3
{1} ✓	1
✓{4}	4
{3,1} ✓✓	4
✓ {3,4} ✓	7
✓ {1,4} ✓	5
✓ {3,1,4} ✓✓	8

O/P = 32

idea1: for every subset iterate & get the sum.

TC : $O(2^N * N)$.

SC: $O(1)$.

idea2: Contribution technique.

$3*4 + 1*4 + 4*4$

$= 12 + 4 + 16 = 32$

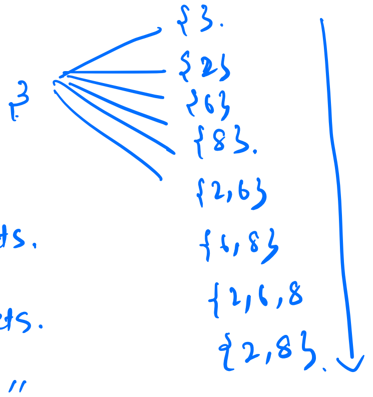
Sum = sum + ar[i] * [No. of times it repeats].

No. of times each element is repeating?.

ar : [3 2 6 8], n=4.

[2 6 8].

for (N-1) elements → 2^{N-1} subsets.



ar[0] appearing 2^{N-1}. times in all subsets.

ar[1] appearing 2^{N-1} times in all subsets.

ar[2] " 2^{N-1} " " " ".

Each ar[i] appears 2^{N-1} times.

[6,8]. ⟹ [ ]
[6]
[8]
[6,8].

2.

Q: Given an array, find the sum of [max of every subsequence]

[3, 1, -4] → [-4, 1, 3]

Subsequences.	max.
[ ]	0
[3]	3 ✓
[1]	✓ 1
[-4]	-4 →
[3 1]	3 ✓
[3 -4]	3 ✓
[1 -4]	✓ 1
[3 1 -4]	3 ✓

Sum = 10.

idea: Generate all subsequences.

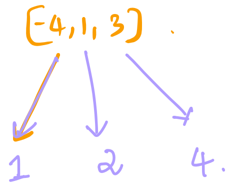
⇒ $O(2^N * N)$.

idea 2: Contribution technique.

$3*4 + 1*2 + (-4)*1$

⇒ $12 + 2 - 4$

⇒ 10.



(i) Sort the array.

[3 2 6 4 5].

$$\begin{array}{lcccccc} & & 0 & 1 & 2 & 3 & 4 \\ \text{Sort} \rightarrow & [& 2 & 3 & 4 & 5 & 6] \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{No. of times} \rightarrow & & 1 & 2 & 4 & 8 & 16. \\ & & 2^0 & 2^1 & 2^2 & 2^3 & 2^4. \end{array}$$

```
def    sumOfMaxOfEverySubsequence(arr, n)
{
    // sort the array in ascending. → NlogN.

    Sum=0.

    for i in range(0,n)
        sum = sum + ar[i] * (1<<i)
                                ↓
                              (2^i).
    return sum
}
```

TC: O(NlogN+N)=O(NlogN)

SC: O(1).

TODO:

Sum of (max of every subsequence) ✓

sum of (min of every subsequence)

Sum of ((max-min) in every subsequence). → Google.

END OF INTERMEDIATE MODULE.