Subarrays

Subarray

↳ Continuous part of an array

↳ Complete Array / Single Element are also subarray

↳ Empty [] → NOT a Subarray

$$Arr = \left[\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 3, & 4, & 5, & 6, & -2, & 8, & 10 \end{array}\right]$$

(5, 6, -2)   Yes

Subseq → (6, 8, 10)   Not Subarray → indices 3, 4, 5 6   (i - - - - j)

↑ Missing

(6)   Yes

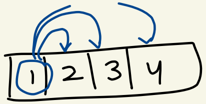
(8, 10)   Yes

(3, 4)   Yes

(5, 6)   Yes

(3, 4, 5, 6)   Yes

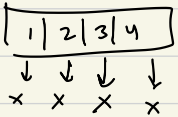
Subsequence (subsets)



↳ subset of the array which may or may not be continuouos

len-0

[ ]



{ }

Len-1

1

2

3

4

Len 2 $\quad {}^4C_2 = \frac{4 \times 3}{2} = 6$ subseq

→ (1,2)

(1,3)

(1,4)

(2,3)

(2,4)

(3,4)

$6 \left\{ \right.$

Length-3 $\quad {}^4C_3 = 4$ ways

(1,2,3)

(1,2,4)

(1,3,4)

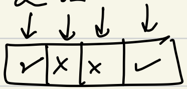
(2,3,4)

Length-4 $\quad {}^4C_4 = 1$

{1,2,3,4}

$2 \cdot 2 \cdot 2 \cdot 2 = 2^4$

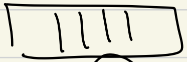
$= 16$ ways



No of subsequences

Subsequences = 1 + 4 + 6 + 4 + 1 = 16 options for len-4

For a array of len →



Subseq → $\boxed{2^N}$

Ex

$$\begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ [\,4, & 2, & 10, & 3, & 12, & -2, & 15\,] \end{array}$$

Count subarrays starting from 0th idx => (0,0)

↑ include

$\boxed{4}$

$\boxed{4,2}$

$\boxed{4,2,10}$

$\boxed{4,2,10,3}$

⋮

(0,0)
(0,1)
(0,2)
(0,3)
(0,4)
(0,5)
(0,6)

} 7 subarrays

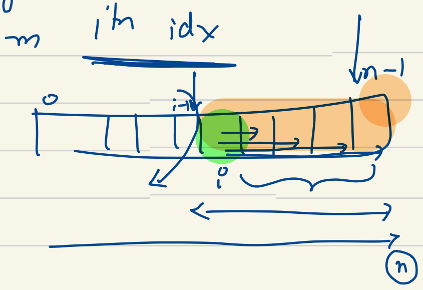
Subarrays starting from 1st idx →

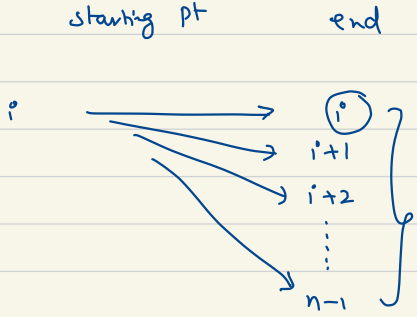
=

(1,1)
(1,2)
(1,3)
(1,4)
(1,5)
(1,6)

} 6 Subarrays

Subarrays starting from i'th idx





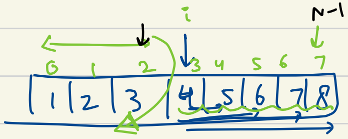
one way

$\boxed{N-i}$

$$N-1-(i-1)$$

$$= N-\cancel{1}+\cancel{1}-i$$

$$= \boxed{N-i}$$ Subarrays starting from index i



(4) ✓

(4,5) ✓

(4,5,6) ✓

(4,5,6,7) ✓

(4,5,6,7,8) ✓

5 Subarrays

7−2

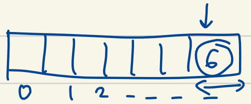
$= \boxed{5}$

$\Rightarrow N-i$

$= 8-3$

$= \boxed{5}$

→ Given N array elements, how to find the total count of subarrays?



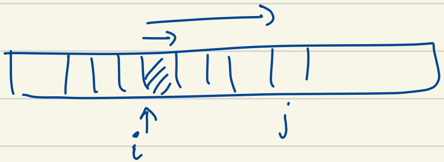
0th indx	1st
[0 0]	[1,1]
[0,1]	[1,2]
[0,2]	⋮
⋮	[1,N-1]
[0,N-1]	

Exact

$$N \text{ subarray} + (N-1) + (N-2) + \text{-----} + 1 = \frac{N(N+1)}{2}$$

$$= O(N^2) \text{ subarray}$$

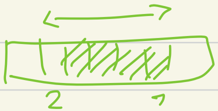
Q. Print all "subarrays". (including the values of subarrays)



$j >= i$

$\frac{N(N+1)}{2}$ Subarrays × N

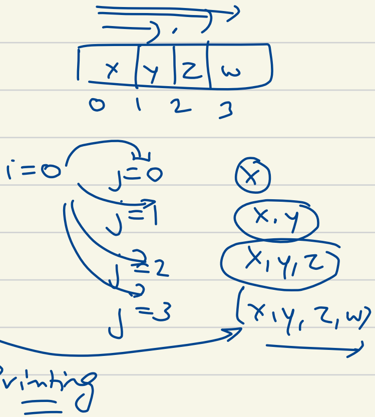
↑ Printing



```
for (i=0 ; i<= n-1 ; i++ ) {

    for (j=i ; j<=n-1 ; j++) {

        // Array[i---j] → subarrays
 N ←    for ( k=i ----j ){
               print(a[k]);
        }
        print(newline)
    }
}
```



Overall Time Complexity → $O(N^3)$

Space complexity → $O(1)$

② Print the sum of all Subarrays starting from index ~~2~~ ⓘ

	0	1	2	3	4	5	6	7
arr[7] =	7	3	2	-1	6	8	2	5

Basic

```
for( j = ~~2~~ i ; j ≤ n-1 ; j++ ){
    sum = 0
    for( K = 2;  K ≤ j ; K++ ){
        sum = sum + a(K);
    }
    print(sum);
}
```

j=2

[2,2] [2] ②

[2,3] (2,-1) ①

j=4 (2,-1,6) ⑦

[0] — 2
1
7

Carry Forward / Prefix Sum

prev Sum

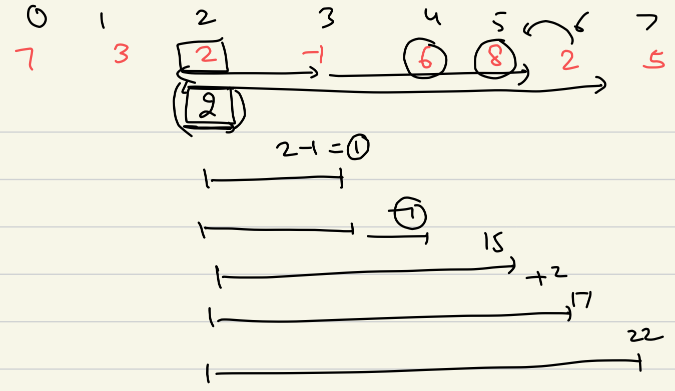
```
sum=0

for( j = ②   j<=n-1 , j++ ){
                        index [2 --- j-1]
    sum = sum + a(j);
    print(sum);
}
```

$\to (2, j-1) + a(j)$

$= \text{sum}(2, j)$

sum = 0

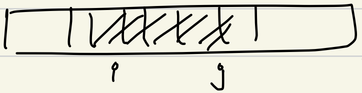


→ [2,2]
[2,3]
[2,4]
[2,5]
[2,6]
[2,7]

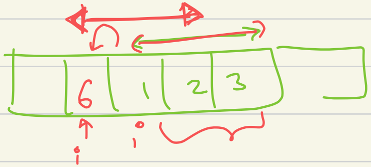
TC = O(N)
SC = O(1)

Doubt Q

Sum of sub of Len 3

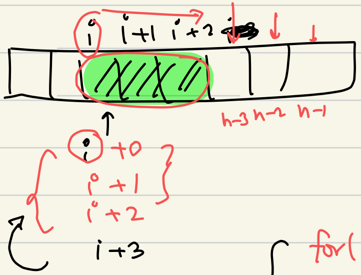


sum=0



for(k=i , k≤j ; k++)
sum += a(k)

for every i → All cases would be covered



for (k=i ; k ≤ i+2, k++){
sum = sum + a(k)

for(i = 0 , i≤n-3; i++){
sum = 0,

= 3N - ~~X~~
= 3 f(N-3)

```
[ for (k=i; k<i+3 ; k++){
        sum = sum + a(k);
  }

  print (sum);
```

$\boxed{3 * N}$

$= O(N)$

3

Q How many subarrays from an Array of len N, would contain index 3 inside them? anywhere inside subarray.

s=3, e=3

Indices	0	1	2	3	4	5
arr [ ]	3	-2	4	(-1)	2	6

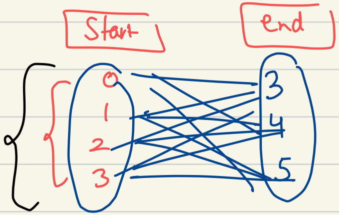
S = 0   [0,3] [0,4] [0,5]

S = 1   [1,3] [1,4] , [1,5]

S = 2   [2,3] [2,4], [2,5]

S = 3   [3,3] [3,4] [3,5]

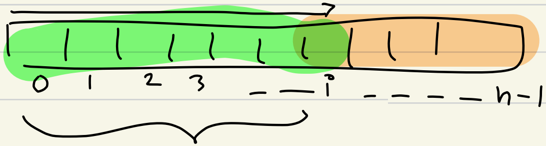
4 x 3 = 12



$\boxed{4} * \boxed{3}$

= 12 ways

Q ———— index i would be present ?



start (0 to i)

i+1 ways

↑

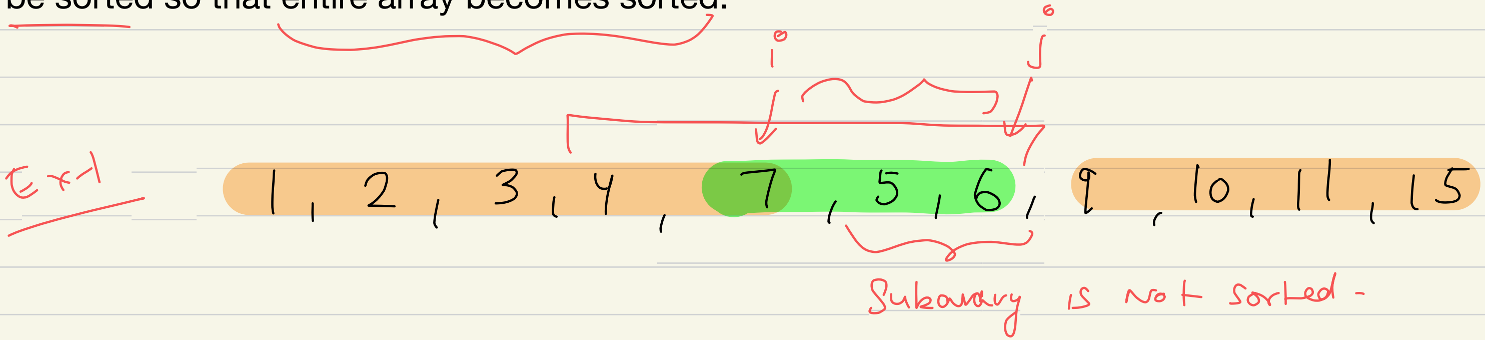
End (i, – – – – n-1)

(N-i) ways

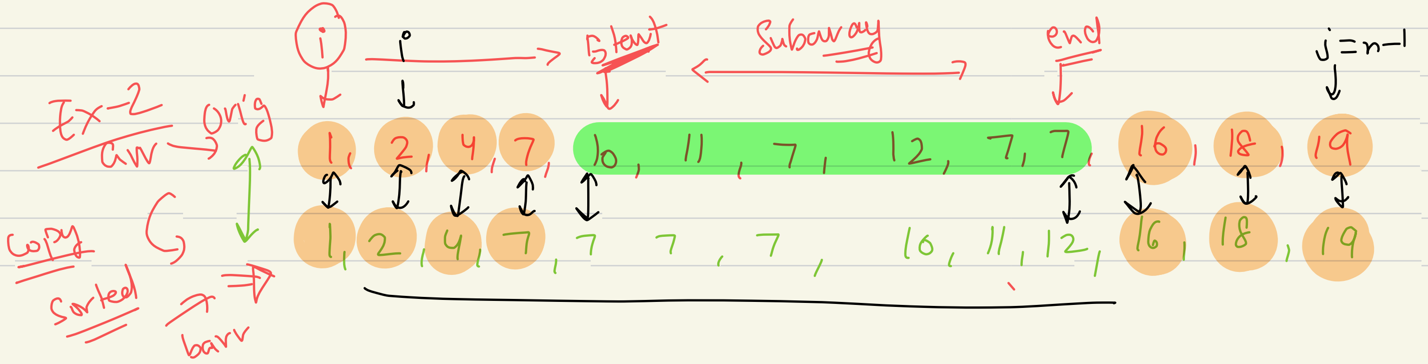
↑

Total ways → $(i+1) \times (N-i)$ ✓

Given an array that contains at least 2 integers, one subpart of the array is unsorted and rest array is sorted in increasing order. You need to the indices of the smallest subarray that needs to be sorted so that entire array becomes sorted.



1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 15



Array. sort (arr Copy) → N log N

Time

= O(N log N + N)

= O(N log N)

```
i=0          10 == 7
while (arr(i) == barr(i)){
        i++;
}

j=n-1
while (arr(j) == barr(j)) {
        j--;
}
```

Stop

Start

i ⇒ N

end

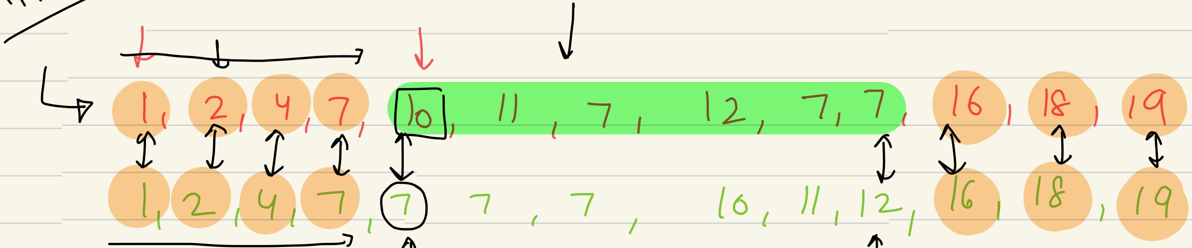
j ⇒ N

Space

O(N)

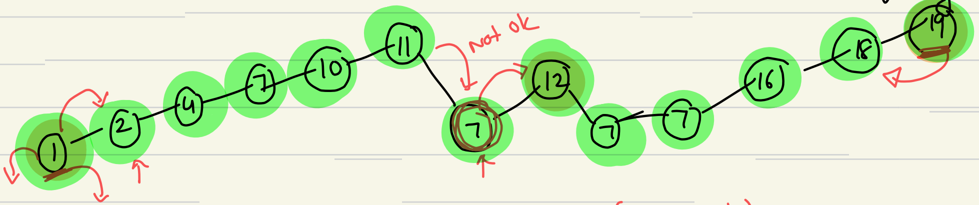
~~Return~~ (i, j)

Approach-2



Smallest element in an array that is not in its correct position in original array

Largest element in array that is not in the correct position in the original array,



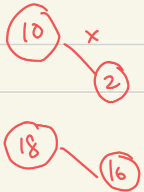
out of order

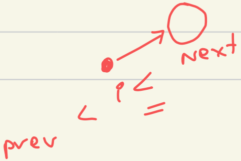
11, 7, 12, 7, 7,

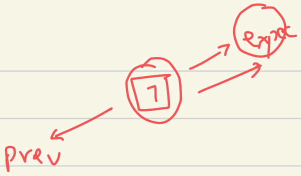
Min → 7

Max → 12

```
boolean   isOutOfOrder (arr, i)
                              T/F
    if (i==0)
        return  a[i] > a[i+1];
                              T/F
    else if  (i==n-1)
        return  a[i] < a[i-1],
```







else ~~if~~ {

return a(i) > a(i+1) || a(i) < a(i-1),

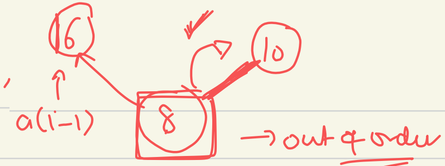
F ✓ or F

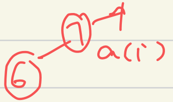
T or T

T/F

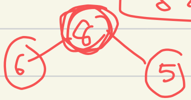
3

3



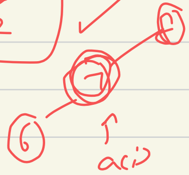


S > 5



Min = ~~11~~ 7

Max = ~~11~~ 12



7 ↓ 12 ↓ j ↓

1, 2, 4, 7, 10, 11, ~~7~~, 12, 7, 7, 16, 18, 19

just larger than 7

TC => O(N)

SC => O(1)

```
for( i=0, i<=n-1; i++ ){
        if(arr,i is out of order){
            min = ✓
            max = ✓
        }
}    }
```

O(N)

element

```
i=0
while(arr(i) <=min) {
        i++;
    }

j=n-1;
while (arr(j) >= max){
        j--,
    }
```

worst case

Positions

O(N)

=> print(i, j) → smallest subarray indices

Good Night :)