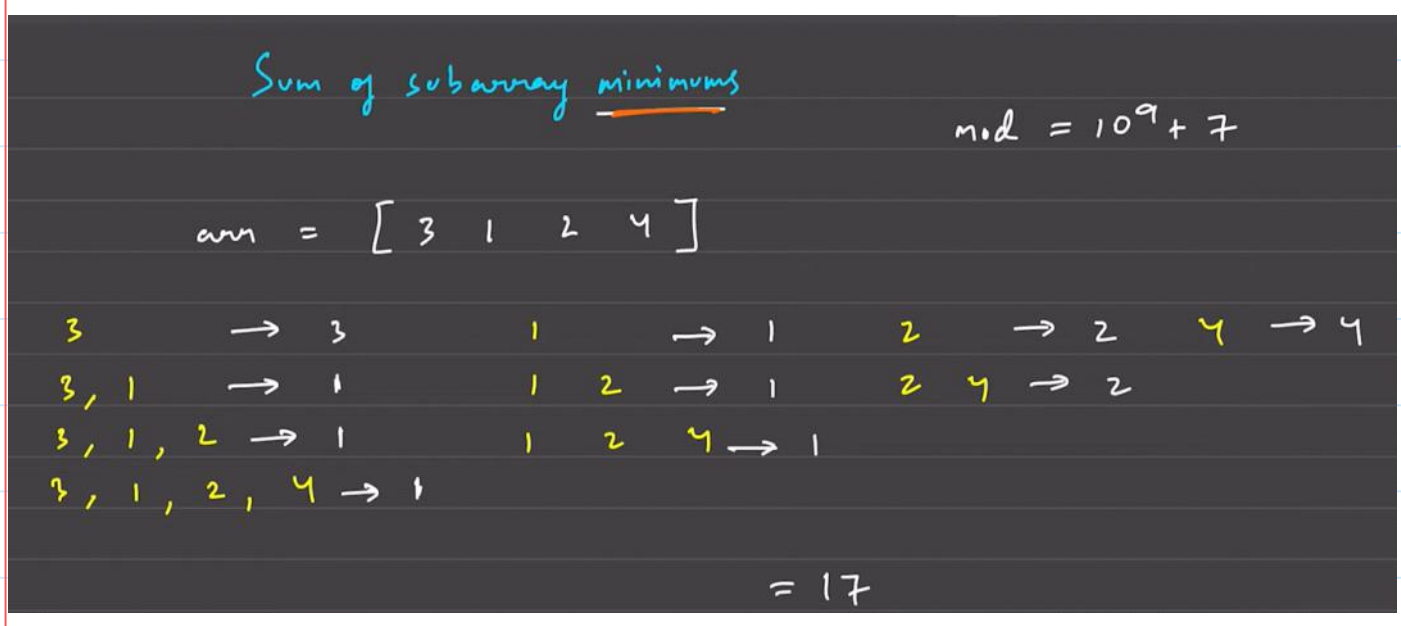
Question –



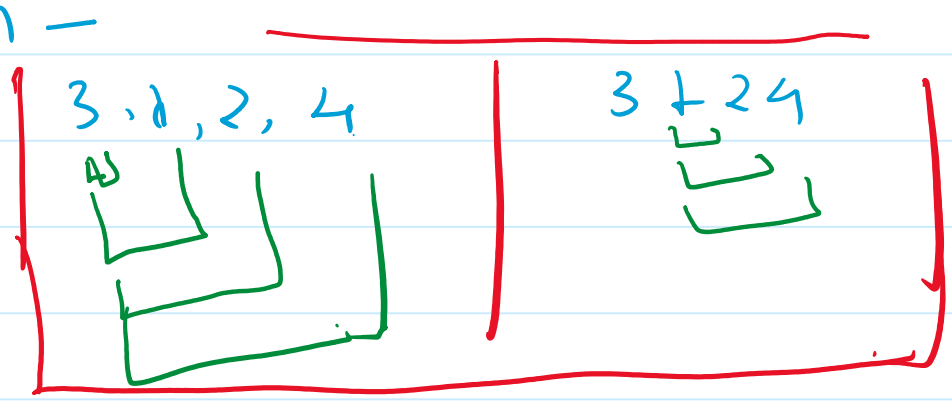
Method – 1. Brute force approach

1.1. Generate all subarray (subsequence)

1.2. find minimum from an subsequence.

1.3. sum all subsequence and return it.

Dry Run –



Code –

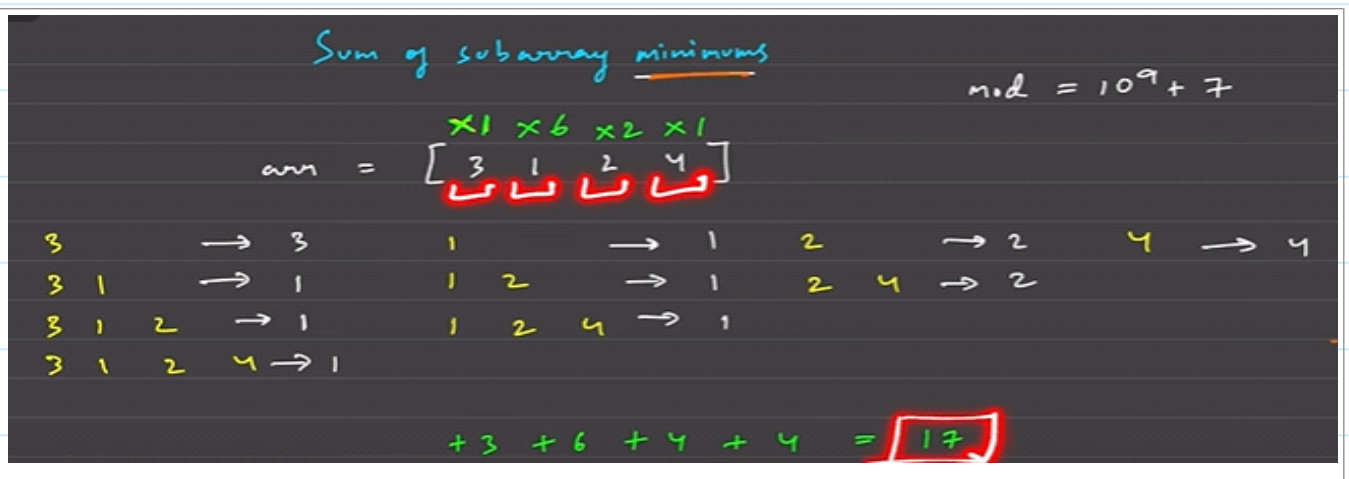
```
sum = 0                    mod = (int)(1e9 + 7)
for (i = 0 → n-1)                          TC → O(N²)
{                                          SC → O(1)
    mini = arr[i]
    for (j = i → n-1)
    {
        mini = min(mini, arr[j]);
        sum = (sum + min) % mod
    }
}
return sum;
```

TC – $O(N^2)$

SC → $O(1)$

2nd optimal solution" –

Approach



- Some points

  + Handing of mod –.

    (10^9+7) ⇒ long MOD = 1_000_000_007

  + Counting the Number of time element is smaller

    • use knowledge of NGE, NSE arr.

  + final result

    • No. of subarr. * (element)

Here 3*1 + 1*6 + 2*2 + 4*1 = (17) Answer

21 February 2025 10:11

Dry Run — & Understanding & Approach.

arr = {1, 4, 6, 2, 3, 2, 8, 1}

What are the contribution of 3, in given the smaller element

(this can be)

{3}

{3, 2}

{3, 2, 8}

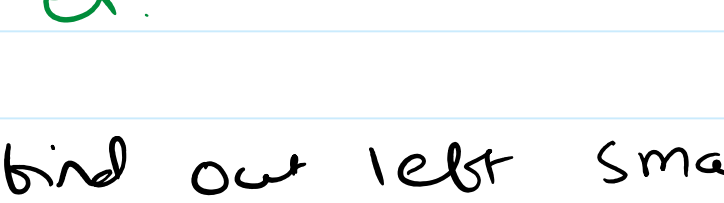
{3, 2, 8, 1} ✗

✗ ↳ 1 smallest

Similar

{4, 6, 2, 3} } in all case

3 is smaller



Total No. of subarray generate in which '3' is smallest. => (4 x 3) How?

{4, 6, 2, 3}	{6, 2, 3}	{2, 3}	{3}
{4, 6, 2, 3, 2}	{6, 2, 3, 2}	{2, 3, 2}	{3, 2}
{4, 6, 2, 3, 2, 8}	{6, 2, 3, 2, 8}	{2, 3, 8}	{3, 2, 8}

this is 12 subarray generated by taking an element '3'

Formula for it = left subarr * right subarr

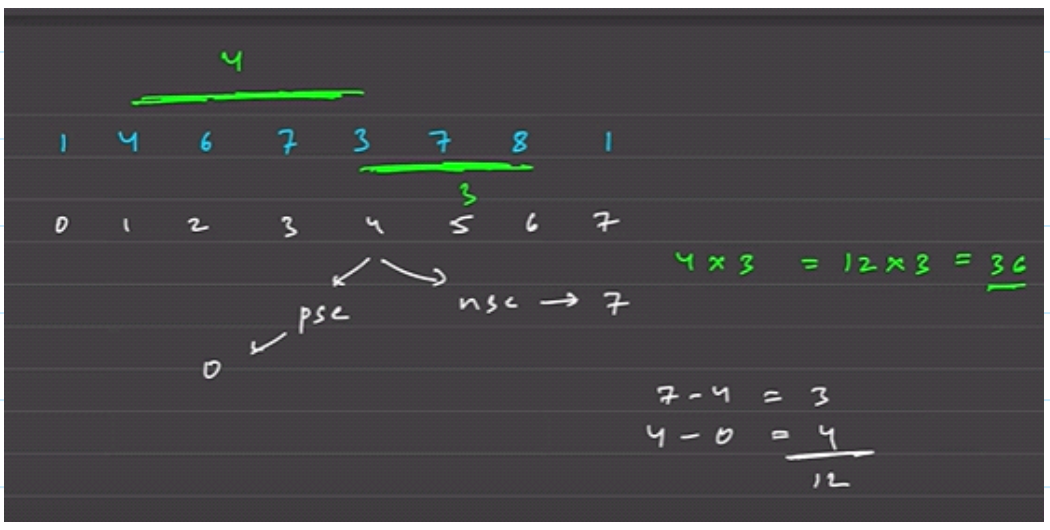
Sum of smallest in subarray taking '3'

= 3 x (4 x 3)

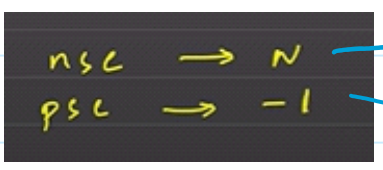
el → left → Right

How to find out left smallest, Right Smallest

- Use concept of NGE, PreviSE
  - → Instance of storing Value store idx, (Why??)



Use the next smallest element and previous smallest element ...



→ instance of Not found store 'N'

→ Previous smaldest NOT found Store '-1'

code

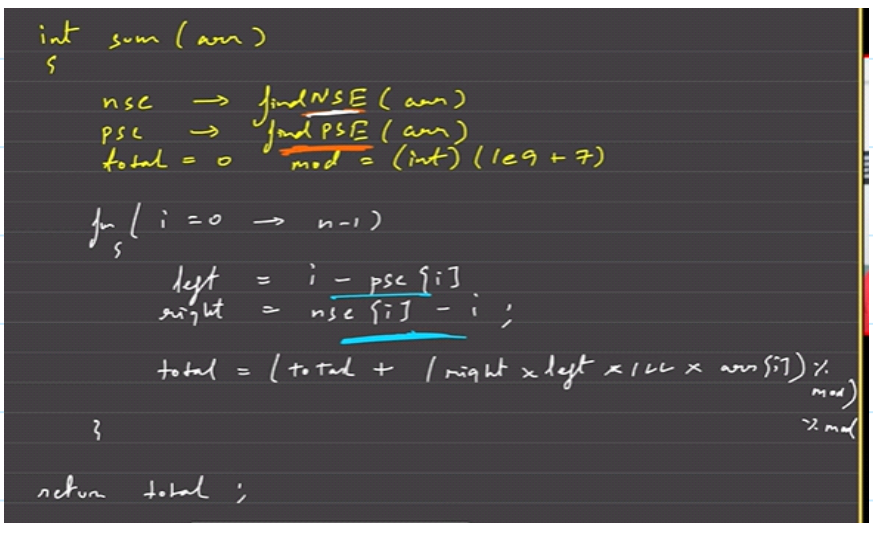
21 February 2025 10:41

```
import java.util.*;
class Solution {
```

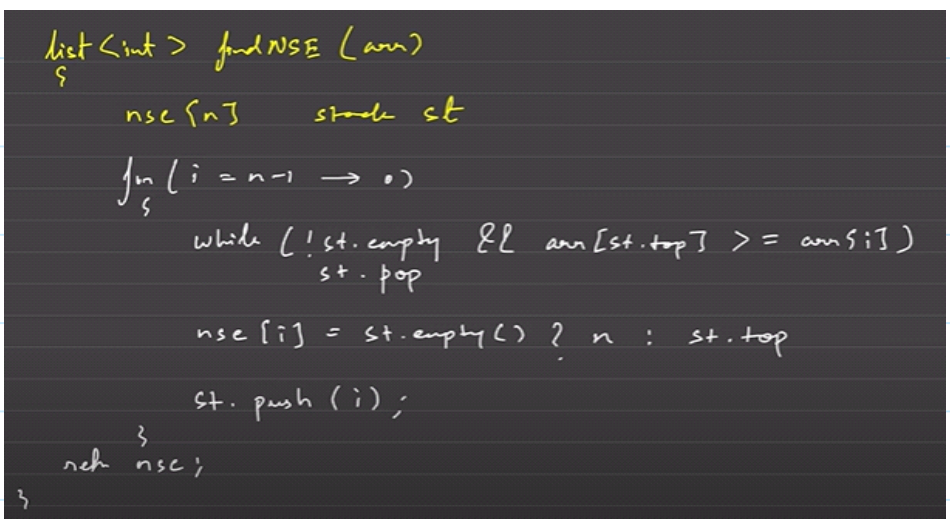
```
    public int sumSubarrayMins(int[] arr) {
        int n = arr.length;
        int[] nextSmaller = nextSmaller(arr);
        int[] prevSmaller = prevSmaller(arr);
        int MOD = 1_000_000_007;
        long ans = 0;
        for (int i = 0; i < n; i++) {
            long leftDist = (i - prevSmaller[i]);
            long rightDist = (nextSmaller[i] - i);
            long count = leftDist * rightDist;
            ans = (ans + ((count * arr[i]) % MOD)) % MOD;
        }
        return (int) ans;
    }
```

TC - O(2N)
SC - O(2N)
— 1 —

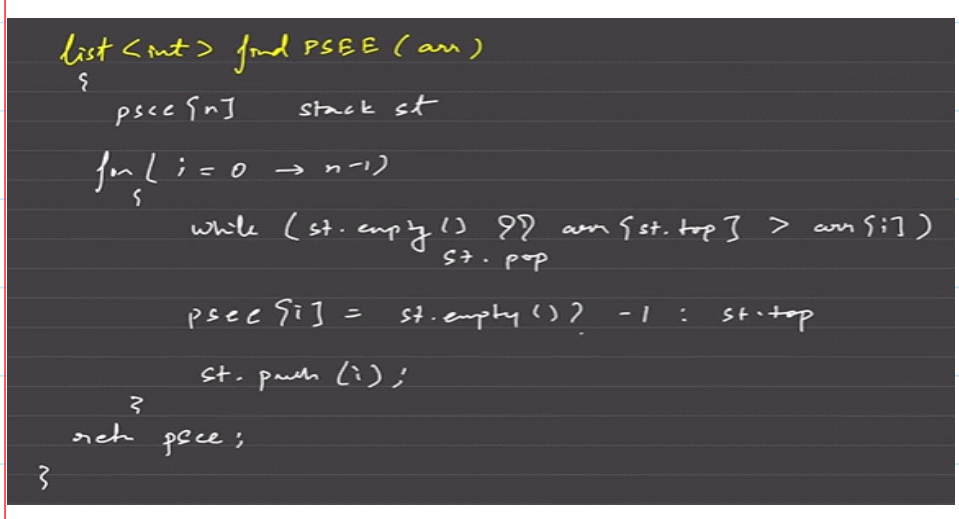
→ O(N), O(N)



```
    public static int[] nextSmaller(int[] arr) {
        int[] ans = new int[arr.length];
        Stack<Integer> stack = new Stack<>();
        int n = arr.length;
        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && arr[i] <= arr[stack.peek()])
                stack.pop();
            ans[i] = stack.isEmpty() ? n : stack.peek();
            stack.push(i);
        }
        return ans;
    }
```



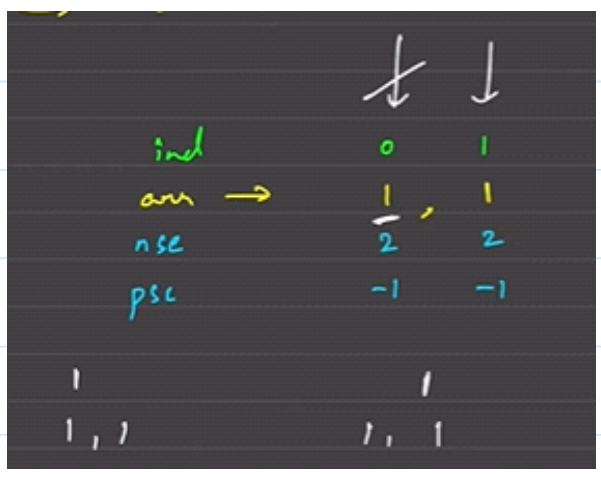
```
    public static int[] prevSmaller(int[] arr) {
        int[] ans = new int[arr.length];
        Stack<Integer> stack = new Stack<>();
        int n = arr.length;
        for (int i = 0; i < n; i++) {
            while (!stack.isEmpty() && arr[i] < arr[stack.peek()])
                stack.pop();
            ans[i] = stack.isEmpty() ? -1 : stack.peek();
            stack.push(i);
        }
        return ans;
    }
```




```
}
```

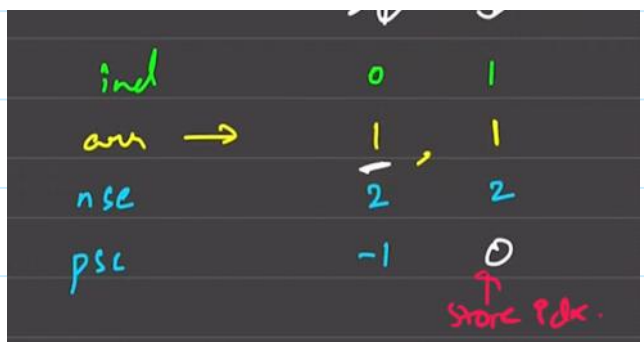
Edge case :
suppose arr={1,1}

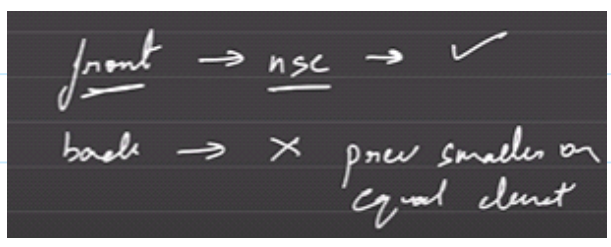
then sub array



problem :Is that … two time consider … {1,1} :

For that problem resolution …





Front store the "next smallest "
But .. Previous store … only …
Previous smaller or equal element

21 February 2025 11:05

Overall time complexity : is : … O(5N)
Space complexity is : O(5N) ….

Space complexity can we reduce but ……keep the same space complexity .. Because of the ..easy to understand format …