Soumya Sambit Mangaraj

Senior Software Engineer @Goldman Sachs

Ex : SDE 2 @PayPal | GATE AIR 485

5+ years of Industry Experience

2+ years of Teaching Experience

Today's Contest :

1. Print Zig-Zag
2. Numbers smaller than current number
3. Longest subarray with sum K

Print Zig Zag

Problem Description

Given a positive integer **N**, print a pattern of numbers based on observation of following examples:

N -> 1

O/P 1 -> 1 1 1

N -> 2

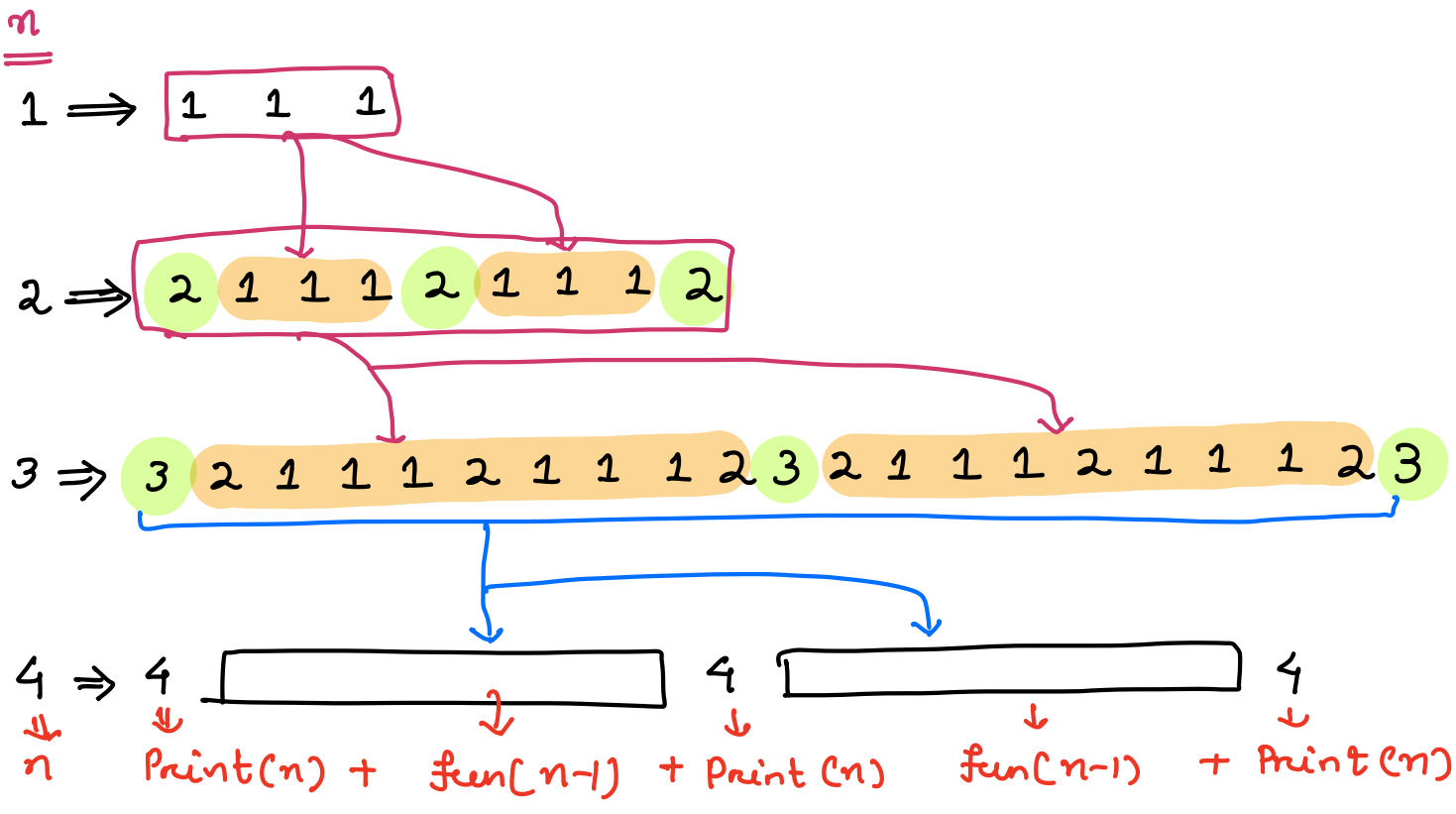
O/P 2 -> 2 1 1 1 2 1 1 1 2

N -> 3

O/P 3 -> 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

Problem Constraints

1 <= N <= 10



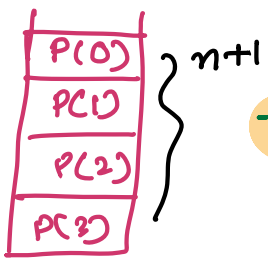
A : fun(n) ⟶ for any given n

E : Print(n) + fun(n-1) + Print(n) + fun(n-1) + Print(n)

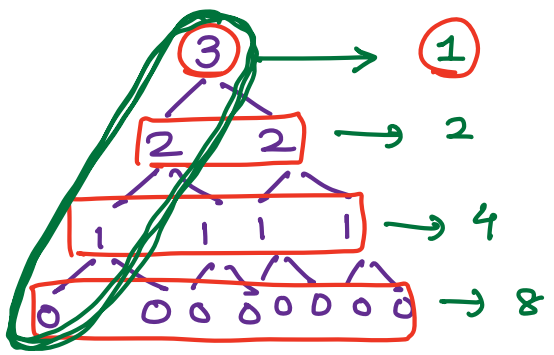
Print zig zag

B : n = 0 ; ⇒ return ;

```
void PrintZigZag(int n) {
    if (n == 0) { return; } // Base Case
    System.out.print(n + " ");
2 ← printZigZag(n-1); // 1st Recursion Call
    System.out.print(n + " ");
2 ← printZigZag(n-1); // 2nd Recursion Call
    System.out.print(n + " ");
}
```



T.C. = O(2^n)



S.C. = O(n)

1 + 2 + 4 + 8 + ... n
= 2^n
Sum of G.P.

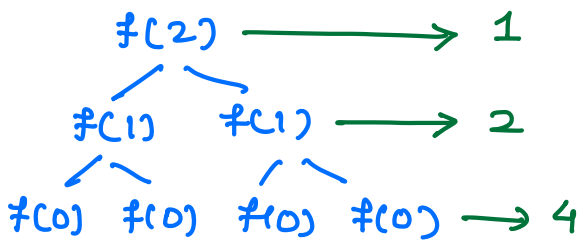
# of function calls

T(n) = T(n-1) + T(n-2)

⇓

T.C. = O(2^n)

S.C = O(n)



T.C.

1 + 2 + 4 = 7 = 2^3 - 1

n ⟶ 2^(n+1) - 1 = O(2^n)

Que// Given the integer array A of size N, find out how many no. in the array are smaller than it. That is for each A[i] you have to count the no. of valid j such that

$A[j] < A[i]$

Constraints

$1 \leq N \leq 10^5$

$0 \leq A[i] \leq 10^3$

In simple words, they are asking you to calculate no. of ele smaller than the current ele

Ex 1: ar[5] = {8 1 2 2 3}

	0	1	2	3	4
ar	8	1	2	2	3

ans[5] = {4 0 1 1 3}

	0	1	2	3	4
ans	4	0	1	1	3

Ex 2: ar[8] =

	0	1	2	3	4	5	6	7
ar	5	1	6	2	8	9	6	11

ans[8] =

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

Ex 3: ar[4] =

	0	1	2	3
ar	5	5	5	5

ans[4] =

	0	1	2	3
ans	0	0	0	0

Idea 1 (Brute force): For every element; Get count of no of elements lesser than current elements.

T.C = $O(n * n) = O(n^2)$

S.C = $O(1)$

Optimised Idea –

	0	1	2	3	4	5	6	7	8	9
arr[10] =	3	4	2	4	2	7	1	7	2	7

Step 1 : Iterate on this Arr; find max = 7

1. All elements ⇒ [0 ... 7]
2. Need frequency.

Step 2 : Iterate on Arr; Store frequency (cnt[max+1])

	0	1	2	3	4	5	6	7	8	9
arr[10] =	3	4	2	4	2	7	1	7	2	7

max = 7

cnt[8] =

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

How many elements are smaller than 4 ?

↳ cnt[0] + cnt[1] + cnt[2] + cnt[3]

= cnt[0 ... 3] = 0 + 1 + 3 + 1 = 5

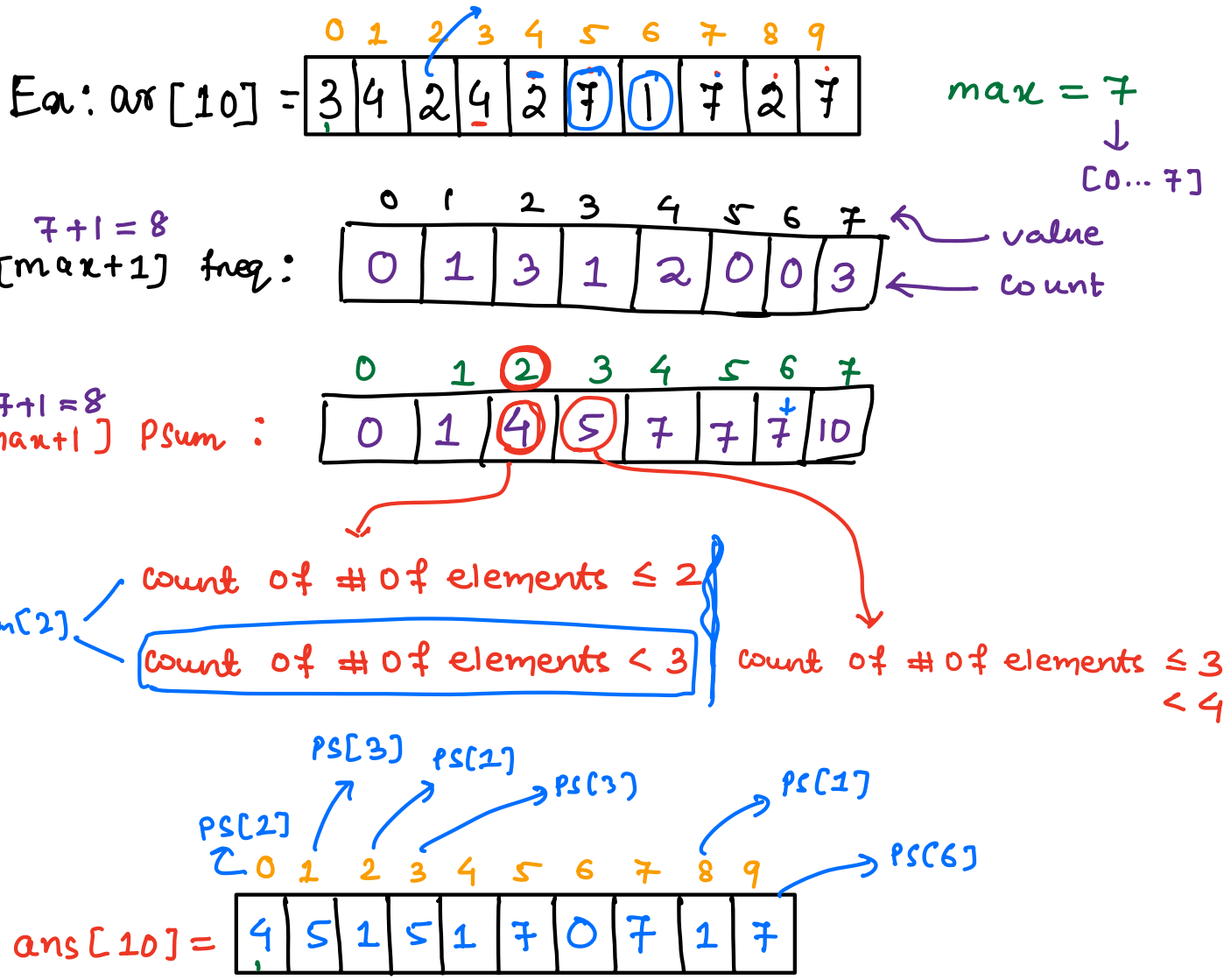
↳ Prefix Sum

#Of elements smaller than 5 = cnt[0 ... 4]

3 = cnt[0 ... 2]

7 = cnt[0 ... 6]

} Prefix Sum


Ex: arr[10] =
0 1 2 3 4 5 6 7 8 9
3 4 2 4 2 7 1 7 2 7
max = 7
[0... 7]
7+1 = 8
int [max+1] freq :
0 1 2 3 4 5 6 7
0 1 3 1 2 0 0 3
value
count
7+1 = 8
int[max+1] Psum :
0 1 2 3 4 5 6 7
0 1 4 5 7 7 7 10
Psum[2]
count of # of elements ≤ 2
count of # of elements < 3
count of # of elements ≤ 3
< 4
PS[3]
PS[1]
PS[2]
PS[3]
PS[1]
PS[6]
0 1 2 3 4 5 6 7 8 9
ans[10] =
4 5 1 5 1 7 0 7 1 7

Ex: arr[n] = | 5 | 2 | 5 | 1 | 0 | 5 | 0 | 1 |

max = 5

5+1 = 6

[max+1] Freq Arr =

0	1	2	3	4	5
2	2	1	0	0	3

[max+1] Psum =

0	1	2	3	4	5
2	4	5	5	5	8

$0 \leq A[i] \leq 1000$

A [n] =

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

ans [n] =

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

PS[4]

PS[1]

PS[0]

Edge case

A[i] == 0

ans[i] = 0

```
int[]  lesser(int[] A) {
    int max = getMax(A); // Todo
    // Create frequency Array
    int[] freq = new int[max+1];
    for (i=0; i<n; i++) {
        int val = A[i];
        freq[val]++;
    }
    int[] Psum = new int[max+1]; // Todo
    int[] ans = new int[n];
    for (i=0; i<n; i++) {
        int val = A[i];
        if (val==0) {
            ans[i] = 0;
        } else {
            ans[i] = Psum[val-1];
        }
    return ans;
}
```

T.C. = O(n)

S.C. = O(max)

S.C. = O(1000)

Longest Subarray with Sum = k

→ Given an array of N integers, find the length of longest subarray such that sum of subarray = k

$1 \le N \le 10^5$

	0	1	2	3	4	5	6	7	8	9
arr[10] :	8	2	1	-3	4	3	1	-2	-3	2

$[a-b] = b-a+1$

k = 6

SubArr [2-6] = 6 ; len = 6-2+1 = 5

SubArr [4-7] = 6 ; len = 7-4+1 = 4

SubArr [1-7] = 6 ; len = 7-1+1 = 7 → Ans → MaxLen.

Idea For all the subArrays

Get sum = k ;

Find the maximum length among them.

T.C. = $O(n^2 * n) = O(n^3)$

↓

$O(n^2 * 1)$ with Prefix Sum

↳ $O(n^2)$

↓

$n = 10^5$

$O(10^{10})$ ⟹ Higher

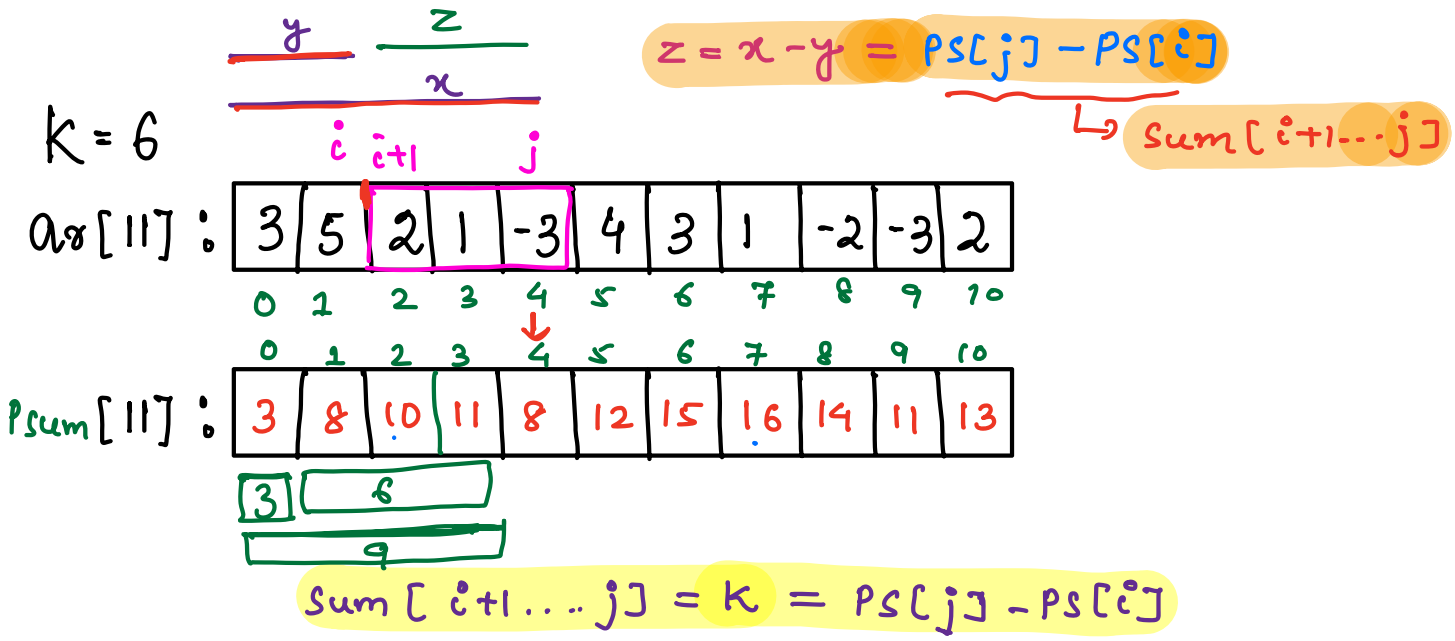
↓

Reduce T.C.

Sum of subarray = k

↳ Continuous Sum

↳ Prefix Sum



len = 7-2 = 5 ; Sum[3...7] = PS[7] - PS[2] = 16 - 10 = 6

len = 8-1 = 7 ; Sum[2..8] = PS[8] - PS[1] = 14 - 8 = 6

len = j - (i+1) + 1 = j - i - 1 + 1 = j - i → Len

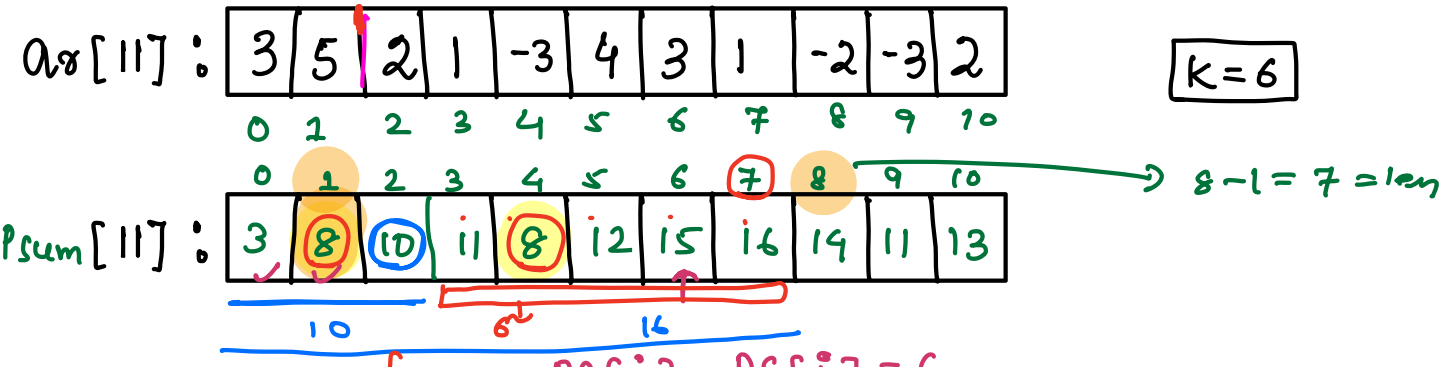
Sum[i+1....j] = PS[j] - PS[i] = K

{j > i} ⇒ PS[j] - K = PS[i]

By fixing ; PS[j] ; we need to look for PS[i]

⇓

PS[j]-K



$15 - \underline{9} = 6$

len = 7-2 = 5

$16 - \underline{10} = 6$

$14 - \underline{8} = 6$

PS[j] - PS[i] = 6

$3 - \underline{x} = 6$

$3 - x = 6 \Rightarrow x = -3$

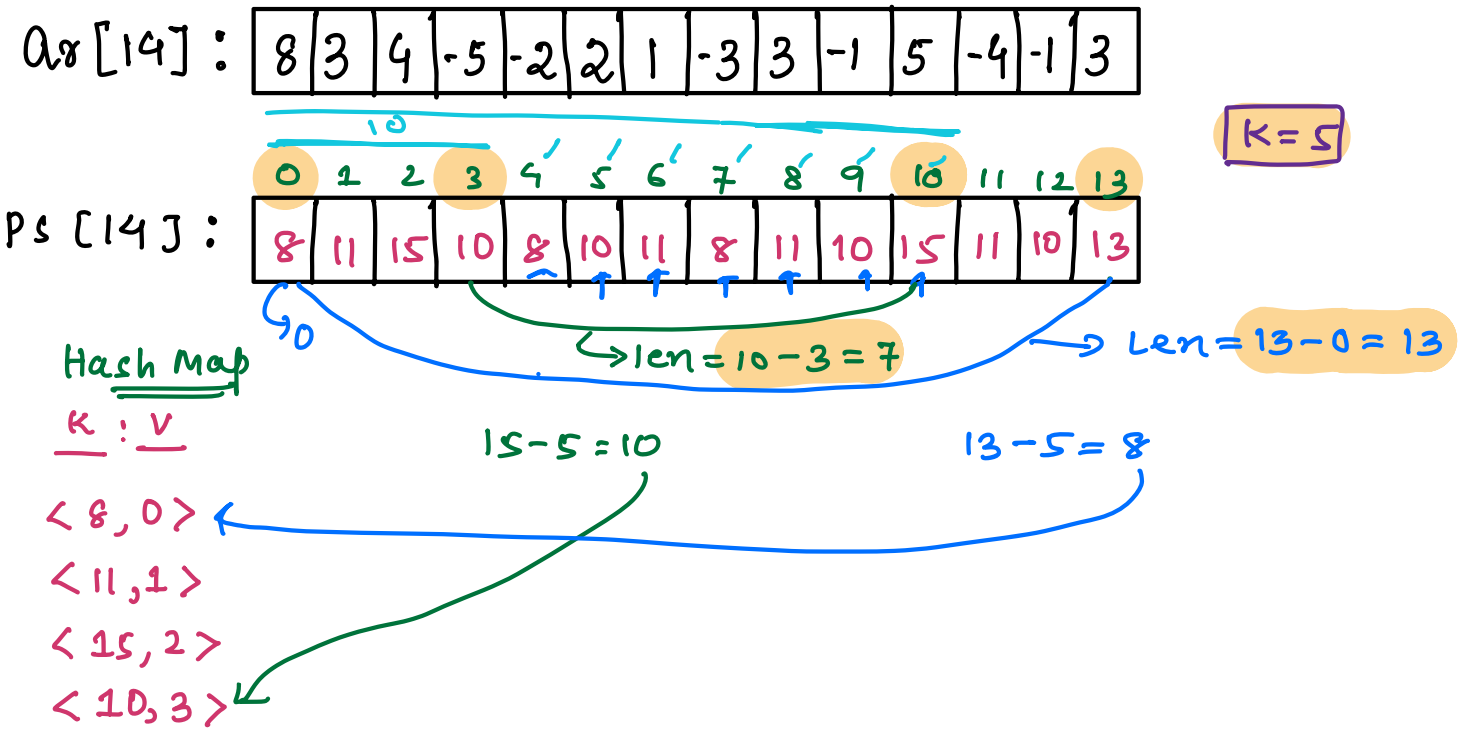
$8 - \underline{2} = 6$

$10 - \underline{4} = 6$

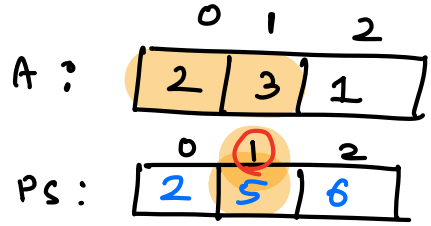
$11 - \underline{5} = 6$

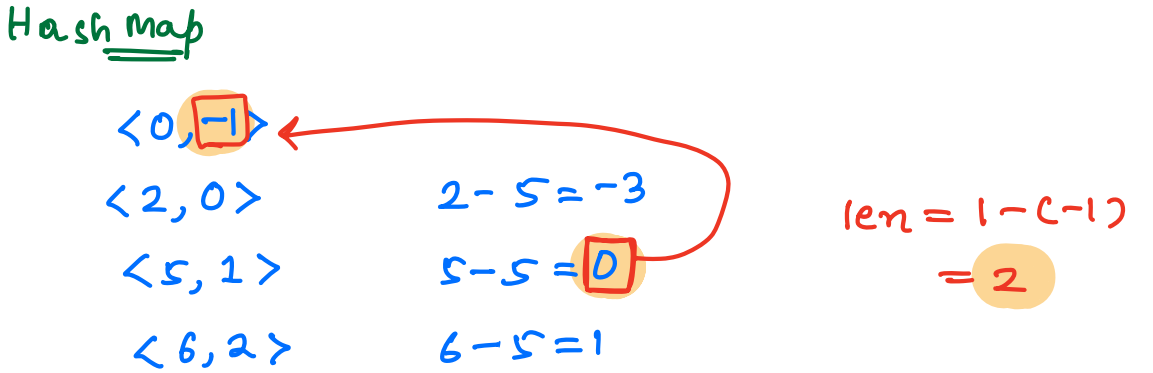
Obs : Don't update the value if Psum[i] is in hashmap ; if it is present.

(1) Since ; we have to find out longest



Edge Case ⇒ hmap.put(0,-1);





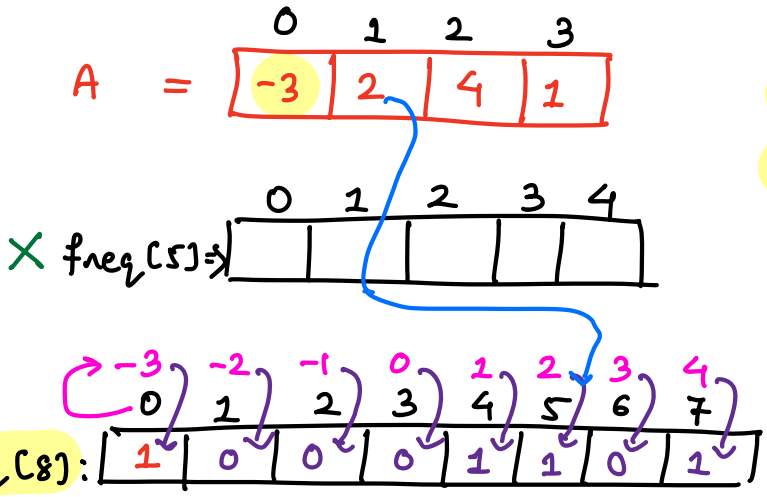
```
int longest(int[] A) {
    int psum[n]; // Todo
    Hashmap<Integer, Integer> hmap = new HashMap<>();
    int ans = 0;
    hmap.put(0, -1); // Edge Case
    for(i=0; i<n; i++) {
        int ele = psum[i];
        int target = ele - k;
        if(hmap.containskey(target)){
            ans = max(ans, i - hmap.get(target));
        }
        else {
            if(hmap.containskey(ele) == false) {
                hmap.put(ele, i);
            }
        }
    }
    return ans;
}
```

In Order to find longest subArray.

T.C = O(n)

S.C = O(n)

Doubt session



max = 4 ✓

min = -3 ✓

max - min + 1

Size freq = 4 - (-3) + 1

= 4 + 3 + 1 = 8

-3 + |min| = -3 + |-3| = -3 + 3 = 0

2 + |min| = 2 + |-3| = 2 + 3 = 5

4 + |min| = 4 + |-3| = 4 + 3 = 7

1 + |min| = 1 + |-3| = 1 + 3 = 4