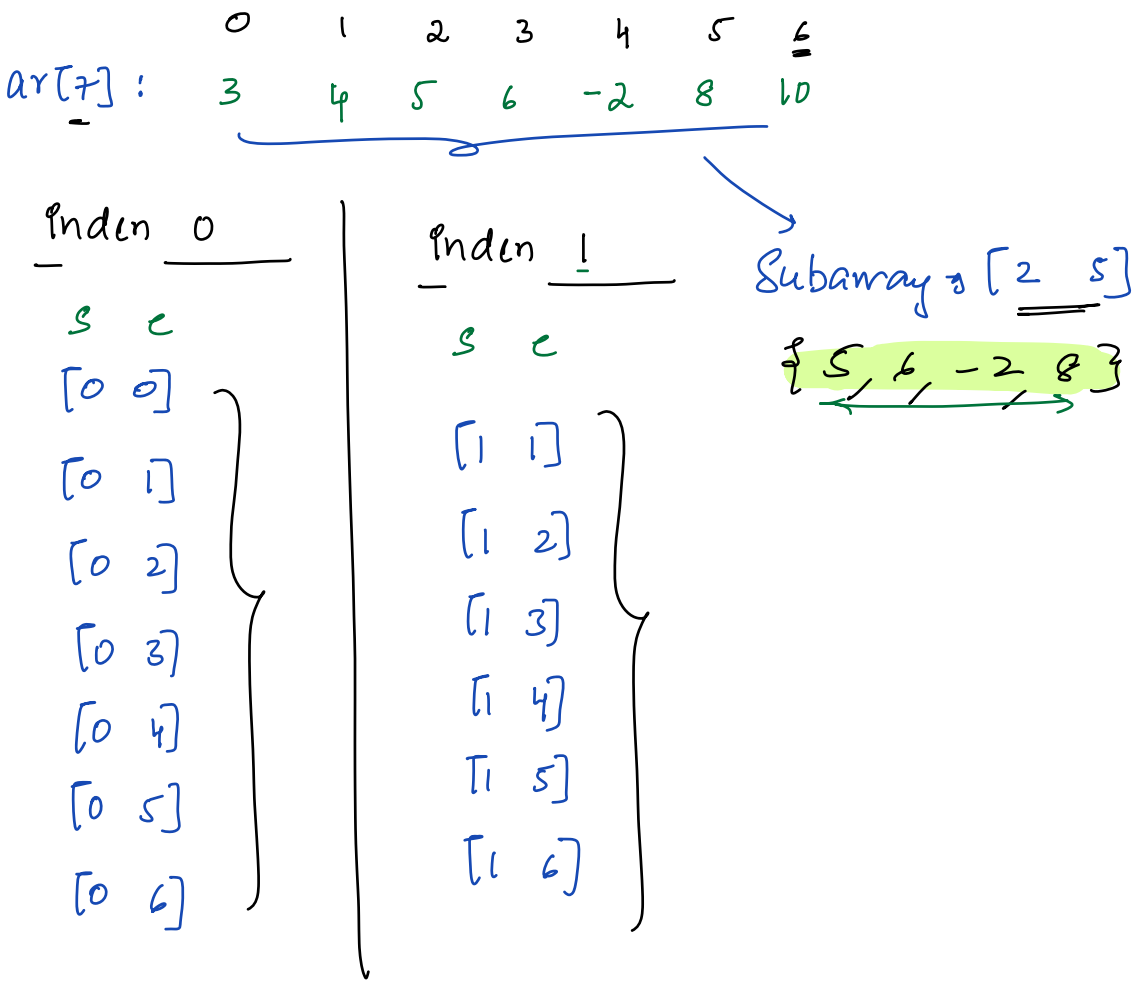
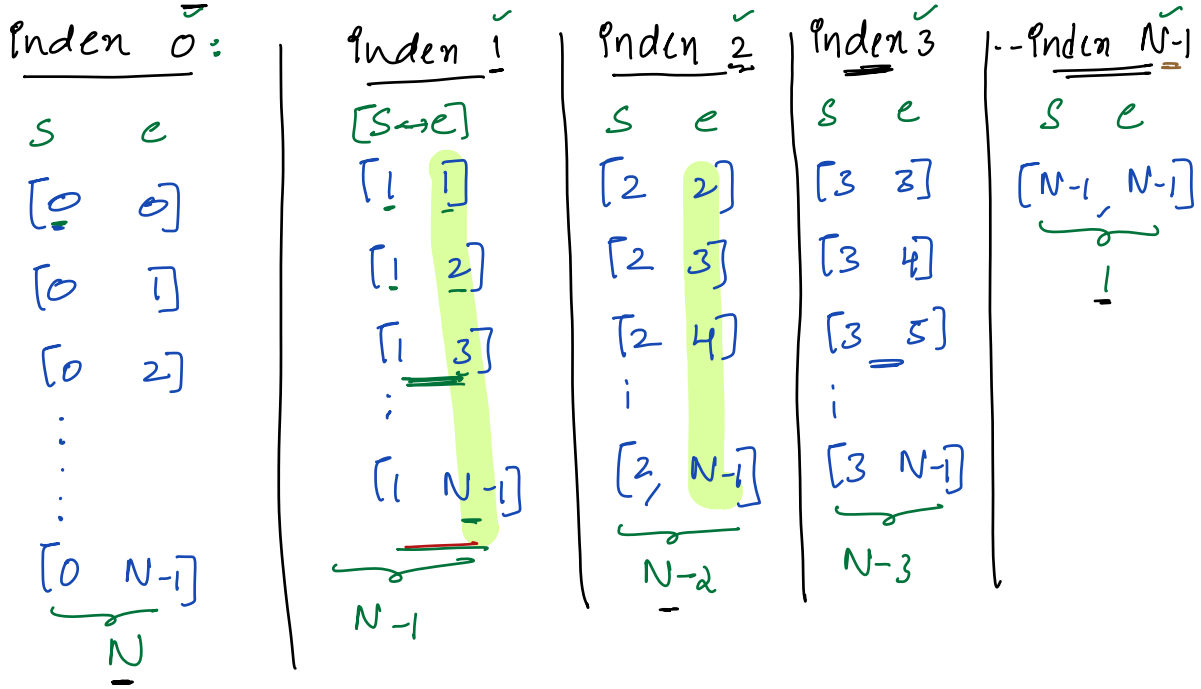
Subarray Basics

- Continous part of an array is called subarray
- Single element is an subarray
- Complete array is subarray
- [s, e] All elements from s to e is Subarray

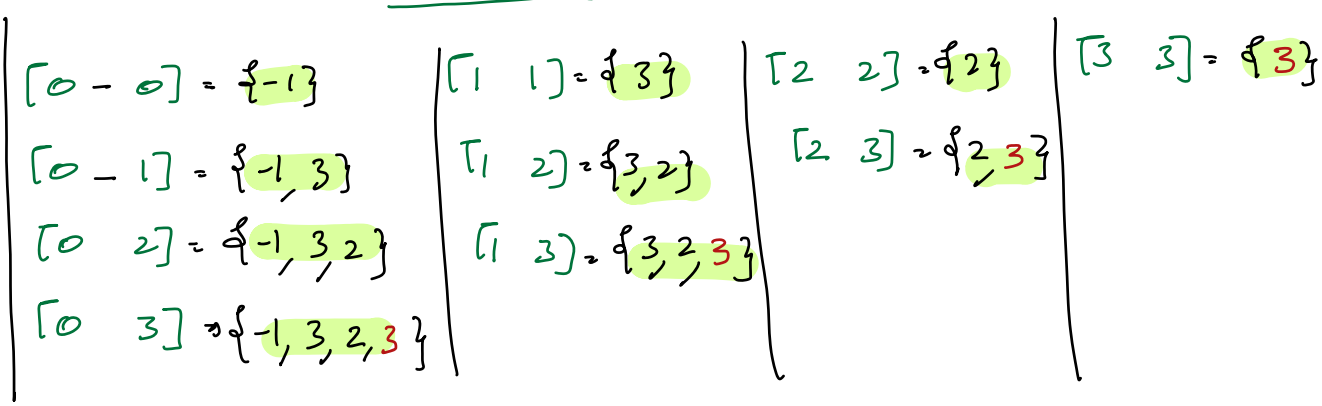


// Say N Elements



Total subarrays = $N + N-1 + N-2 + N-3 + \cdots 1 = \frac{(N)(N+1)}{2}$

Ex: ar[4] : -1 3 2 3 →



// Print all Subarrays.

ar[3] = { 2, 8, 9 } Given N array elements print all subarrays?

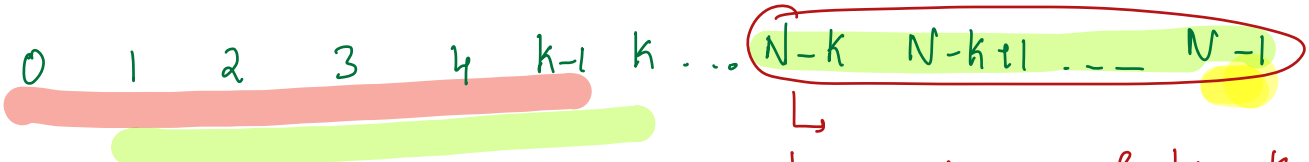
S	e	output
[0	0]	2
[0	1]	2 8
[0↔2] : 3		2 8 9
[1	1]	8
[1↔2]		8 9
[2	2]	9

TC : O(N3)

SC : O(1)

```
Start index of subarray
i = 0 ; i < N ; i++) {
    End index of Subarray
    j = i ; j < N ; j++) { a[i j]
        [i j] is your subarray
        k = i ; k <= j ; k++) {
            print(ar[k])
        }
        println()
    }
}
```

// N Elements



28) Print start and end index of all subarrays of len = k

[a b] = b - a + 1

	0	1	2	3	4	5	6	7	8	9	10	11
ar[12] :	3	4	2	-1	6	7	8	9	3	2	-1	4

k = 2

s	e
✓ [0	1]
✓ [1	2]
✓ [2	3]
✓ [3	4]
.	.
.	.
.	.
✓ [10	11]

// 11 Subs

k = 6

s	e
[0	5]
[1	6]
[2	7]
[3	8]
[4	9]
[5	10]
[6	11]

// 7 Subs

// N array elements, s, e of subarray len = k

s	e
✓ [0	k-1]
✓ [1	k]
✓ [2	k+1]
✓ [3	k+2]
✓ [4	k+3]
[x	N-1]
✓ [N-k	N-1]

} Number of subarrays of len k = N-k+1

} → N-1-x+1 = k

x = N-k

```
{ {S = 0, e = k-1}   // 1st subarray len = k
  ; S < = N-k ; S++, e++) {
    print (S, e)
  }
```

Iterations = (N-k+1)

38) Find max subarray sum with len = (k) } N & k are input

↳ Sum of all elements in subarray.

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

k = 5

S	e	sum
0	4	7
1	5	8
2	6	12
3	7	16
4	8	10
5	9	11

Max : 16

Solutions

1) For every subarray of len = k
   get sum.

```
S = 0, e = k-1, ans = -∞ / INT_MIN
for( ; s <= N-k ; s++, e++ )   } → (N-k+1)
    // sum of all elements [s, e] → k
    sum = 0
    for(i = s; i <= e; i++){
        sum = sum + ar[i]
    }
    ans = max(ans, sum)
}
```

Exact iterations = $k * (N-k+1)$ }

min max

k : { 1, N } →

- $k = N$ → $(N)(1)$
- $k = 1$ → $(1)(N)$
- $k \approx N/2$ → $(N/2)(N/2+1)$

$\approx O(N^2)$

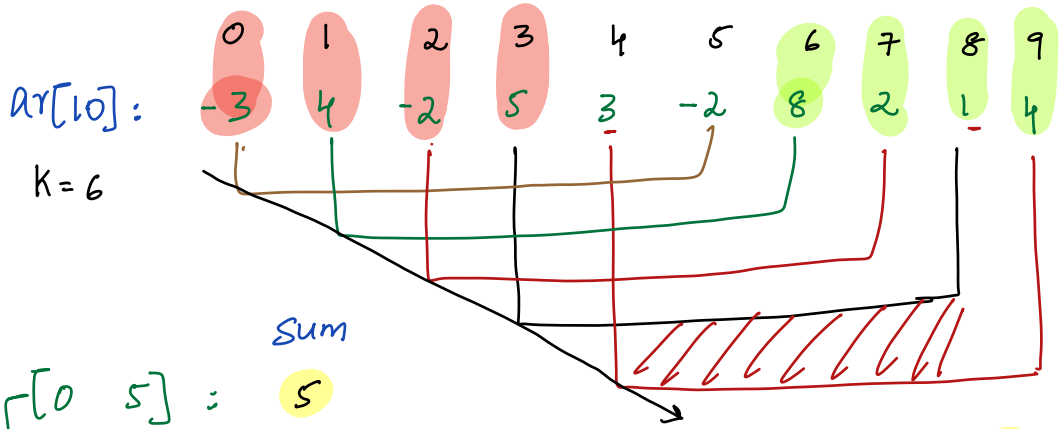
Sol 2:

1) For every subarray of len = k
   get sum using pf[]

1) Create pf[] for entire arr[] : O(N)
2) s = 0, e = k-1, ans = $-\infty$ / INT_MIN
3) for(; s < N-k ; s++, e++) → N-k+1

```
|  // sum of all elements [s, e]
|  {sum = [s e] using pf[]}
|  ans = max(ans, sum)
}
```

TC : O(N) SC : O(N) : 10:38PM



[0 5] : 5

[1 6] : 5 - ar[0] + ar[6] = 5 + 3 + 8 = 16

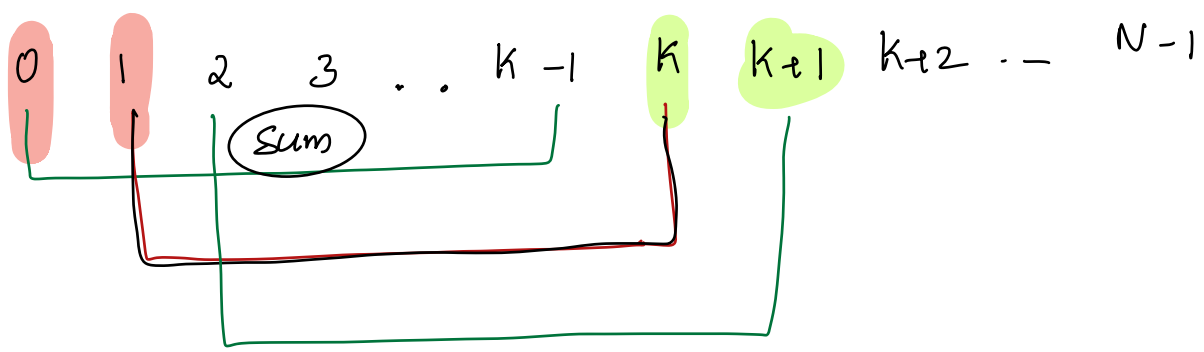
[2 7] : 16 - ar[1] + ar[7] = 16 - 4 + 2 = 14

[3 8] : 14 - ar[2] + ar[8] = 14 + 2 + 1 = 17 ✓

[4 9] : 17 - ar[3] + ar[9] = 17 - 5 + 4 = 16

// using previous window sum - we are getting ans for current window

// N Indexed elements



—

// N array elements, window size k

1st subarray : [0, k-1] = Iterate & get sum = sum

(s) (e)

2nd subarray : [1, k] sum = sum - ar[0] + ar[k]

3rd subarray : [2, k+1] sum = sum - ar[1] + ar[k+1]

4th subarray : [3, k+2] sum = sum - ar[2] + ar[k+2]

5 subarray : [4, k+3] sum = sum - ar[3] + ar[k+3]

.

last subAr : [N-k, N-1] sum = sum - ar[N-k-1] + ar[N-1]

Pseudocode

```
sum = 0
for(i = 0; i < k; i++){
    sum = sum + ar[i]
}
ans = sum.
s = 1, e = k;
for( ; s <= N-k; s++, e++){
    sum = sum - ar[s-1] + ar[e]
    ans = max(sum, ans)
}
return ans;
```

k

TC: O(N)

SC: O(1)

N-k

48) Given N array elements, print all subarray sums

	0	1	2	3
ar[4] :	3	-1	0	2

[0 0] : 3
[0 1] : 2
[0 2] : 2
[0 3] : 4

[1 1] : -1
[1 2] : -1
[1 3] : 1

[2 2] : 0
[2 3] : 2

[3 3] : 2

Total Sum = 14
↳ Sum of all Subarray Sums

Idea: For every subarray calculate & print sum.

```
for(i = 0; i < N; i++){          // Start index of subarray
    for(j = i; j < N; j++){      // End index of Subarray
        // [i j] is your subarray
        sum = 0
        for(k = i; k <= j; k++){
            sum = sum + ar[k]
        }
        print(sum)
    }
}
```

TC: $O(N^3)$ SC: $O(1)$

2nd Sol: Sum of Subarray with prefix sum

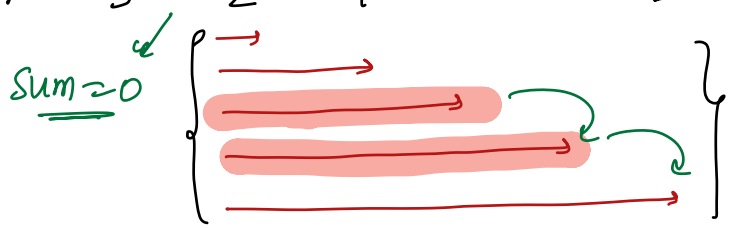
TC: $O(N^2)$ SC: $O(N)$

TC: $O(N^2)$ SC: $O(1)$
using carry forward

→ Qu) Print all subarray sums starting at index 2

Ex: ar[7] :

0	1	2	3	4	5	6
7	3	2	-1	6	8	2



[2 2] : 2 ✓

[2 3] : 1

[2 4] : 7

[2 5] : 15

[2 6] : 17

Sum = 0

```
Sum = 0
i = 2 ; i < N ; i++){
    Sum = Sum + ar[i]
    print(Sum)
}
```

// Sum of Each Subarray Sum.

→ TC: O(N²) SC: O(1)

```
maxSum = ar[0]
TotalSum = 0
i = 0; i < N; i++) {
    Sum = 0
    j = i ; j < N ; j++){ // Sum of Subarrays starting
        Sum = Sum + ar[j]   at index i
        print(Sum) // printing all subarray sums
        maxSum = max(maxSum, Sum) } Max subarray Sum
    TotalSum = TotalSum + Sum //
}
```

TC: O(N²) O(1)

↳ Sum of all subarray sums

// In how many subarrays index 3 is present?

	0	1	2	3	4	5
ar[] :	3	-2	4	-1	2	6

left | right

left: 0, 1, 2, 3 | right: 3, 4, 5

// Subarray index 3 present = 4 * 3 = 12

left | right

[0 3] [0 4] [0 5] [1 3] [1 4] [1 5]
[2 3] [2 4] [2 5] [3 3] [3 4] [3 5]

In how many subarrays index 1 is present?

	0	1	2	3	4	5
ar[] :	3	-2	4	-1	2	6

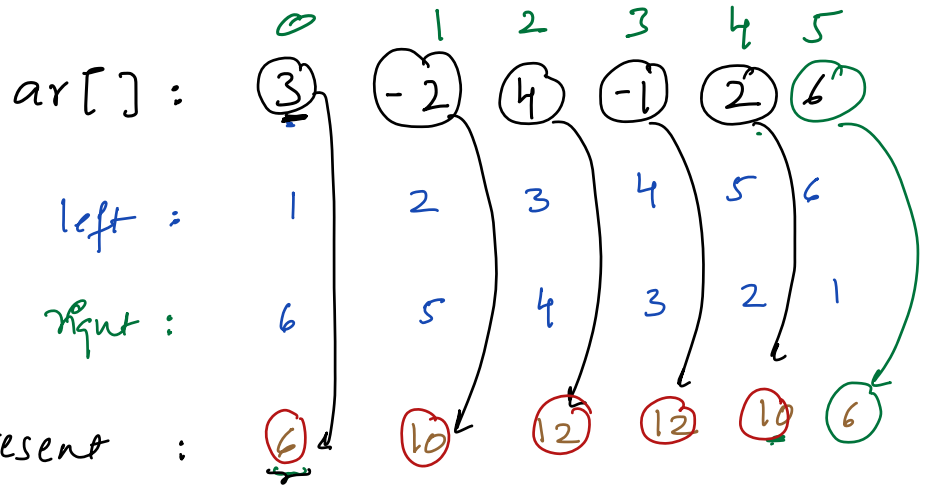
left: 0, 1 | right: 1, 2, 3, 4, 5

// Subarrays = 10

= left * right

= 2 * 5 = 10

Contribution Technique



Sum of All Subarray sums : $18 - 20 + 48 - 12 + 20 + 36$

$\Rightarrow 90$

Given N Elements In how many subarrays index i is present?

Index: 0 1 2 3 - i $i+1$ $i+2$ - — $N-1$

left = $[0\ i] = i+1$ right = $[i, N-1] = N-i$

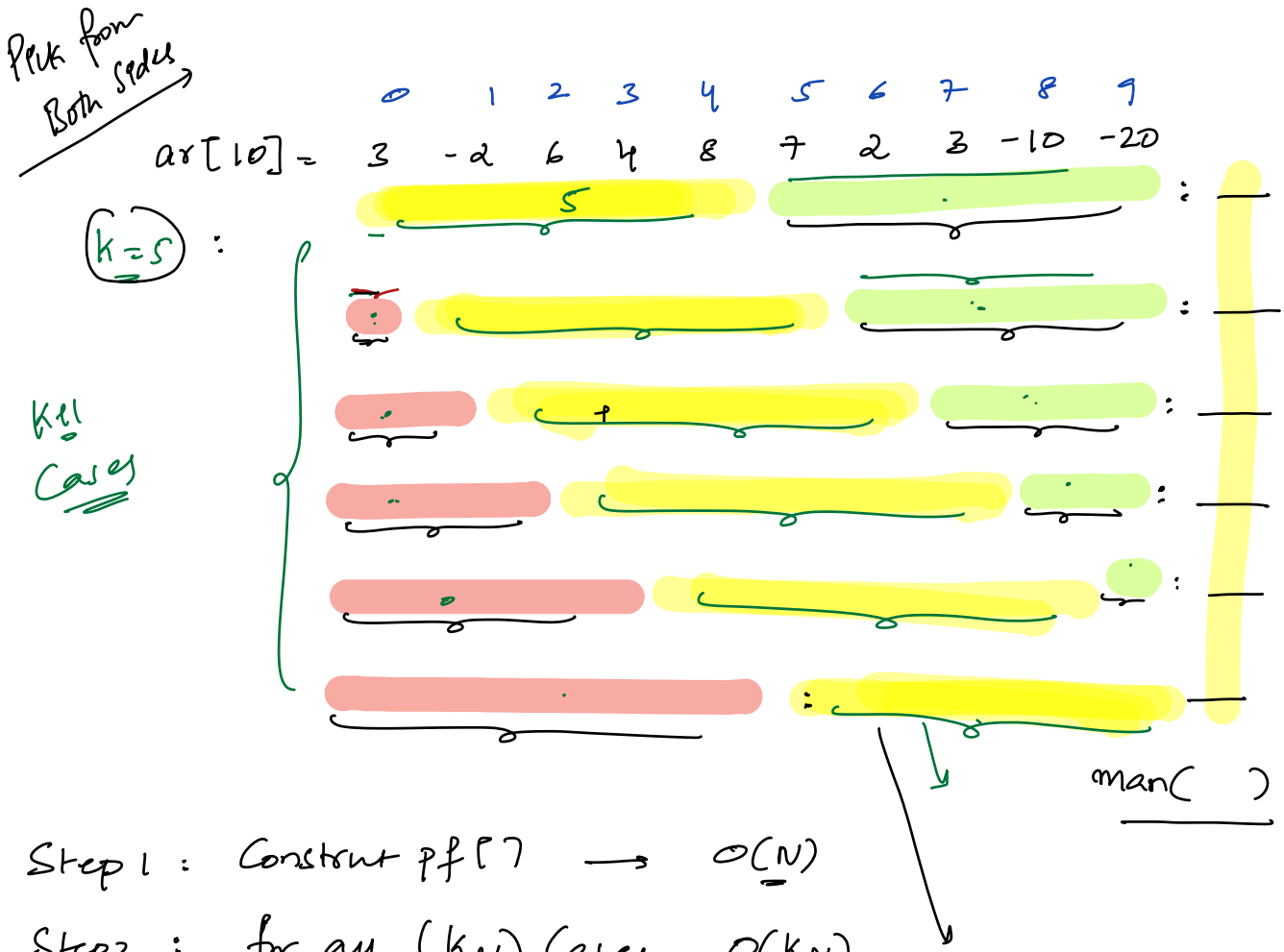
Number subarray = left * right = $(i+1)*(N-i)$

Brudfod

```
sum = 0
for(i = 0; i < N; i++){
    left = i+1   right = N-i
    c = left * right
    sum = sum + c * ar[i]
}
```

TC: O(N)

SC: O(1)



Step 1 : Construct pf[] → O(N)

Step 2 : for all (k+1) Cases → O(k+1)

TC : O(N) SC : O(N)

+ = Plan

+ + = Total Array Sum

= Total Array - Red & green

→ find minimum subarray sum with size N-k

→ Final ans = Sum of array -

0 1 2 3 4 → } ans = false

[10 9 8 9 4]

0 1 2 3 4 5

[3 2 9 6 10 7] → If one of the corner odd gone.

{ Array size even
Both corner even } → ans = True

→ {[2 5 5 5 5 2]}

→ Ex: [2 5 5 5 5 2]

→ Pf [ ] ⟶
↳ Carry for
Even → Carry —
Size — Carry for