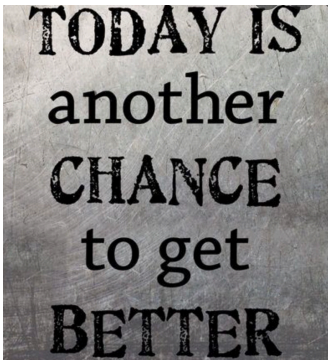
Today's Quote →



→ fill out the form.

Today's content.

→ Properties of sub-arrays

→ Questions

3	2	5	7	6	11
0	1	2	3	4	5

Recap.

Subarray → continuous part of an array.

→ single element / complete array → subarray

→ Empty array is not a subarray.

→ i_j : length = j - i + 1.

Properties of Subarrays

arr[4] : { 2 (0) 6 (1) 3 (2) 9 (3) }

count : 4 + 3 + 2 + 1 = 10.

subarrays:

[0, 0] : { 2 }

[0, 1] : { 2, 6 }

[0, 2] : { 2, 6, 3 }

[0, 3] : { 2, 6, 3, 9 }

[1, 1] : { 6 }

[1, 2] : { 6, 3 }

[1, 3] : { 6, 3, 9 }

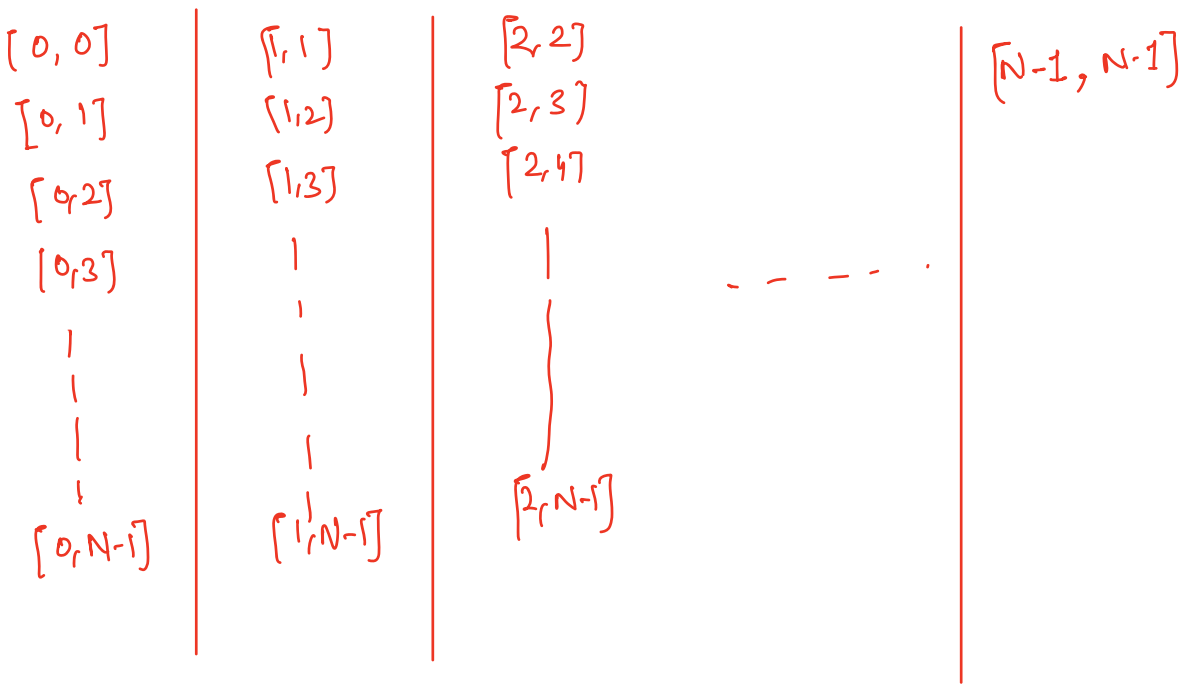
[2, 2] : { 3 }

[2, 3] : { 3, 9 }

[3, 3] : { 9 }

arr[N] : a_0 a_1 a_2 a_3 — — — — — — — a_{N-2} a_{N-1}

Subarrays :



$N + (N-1) + (N-2) + \cdots + 1$

Total no. of subarrays = $\frac{N(N+1)}{2}$ $\approx N^2$.

Problems :

① Given arr[N], [s, e] print subarray from s to e

(s → start-index, e → end-index)

```
void printSubarray(arr, s, e){
    for(i = si; i <= ei; i++){
        print(arr[i])
    }
}
```

T.C → O(N)

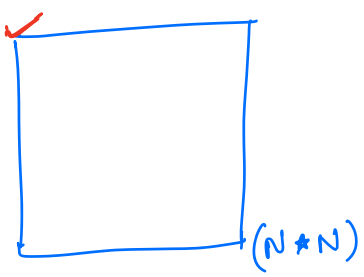
S.C → O(1)

Eg → { 2 1 3 5 7 6 11 4 }

indices: 0 1 2 3 4 5 6 7

s = 3 ✓

e = 6 ✓



→ print all Elements of the matrix.

N^2 elements.

T.C → $O(N^2)$

→ print all Elements of the array. T.C → O(N)

printing 1 subarray → N iterations

printing N^2 subarrays → $N^2 * N$ iterations

→ N^3.

Q) Given N array elements. print all the subarrays.

arr[4] : { 6 8 -1 7 }
0 1 2 3

T.C → $O(N^3)$

Subarray e/g.

[0,0] : {6}
[0,1] : {6, 8}
[0,2] : {6, 8, -1}
[0,3] : {6, 8, -1, 7}
[1,1] : {8}
[1,2] : {8, -1}
[1,3] : {8, -1, 7}
[2,2] : {-1}
[2,3] : {-1, 7}
[3,3] : {7}

[0,0] [0,1] [0,2] [0,3]
[1,1] [1,2] [1,3]
[2,2] [2,3]
[3,3].

Pseudo code:

```
void printAllSubarrays(arr, N){
    for(i=0; i<N; i++){          // starting index of the subarray
        for(j=i; j<N; j++){      // ending index of the subarray
            // print subarray from s to e
            for(k=i; k<=j; k++){print arr[k]}
            print(newLine)
        }
    }
}
```

T.C → $O(N^3)$
S.C → $O(1)$

Q) Given N array elements print each sub-array sum.

arr[4] : { 6 8 -1 7 }
0 1 2 3

Sub-array	sum
[0 0] :	6
[0 1] :	14
[0 2] :	13
[0 3] :	20
[1 1] :	8
[1 2] :	7
[1 3] :	14
[2 2] :	-1
[2 3] :	6
[3 3] :	7

Idea-1.

```
void printSubarraySums(arr, N){
    // starting index of the subarray
    for(i = 0; i < N; i++){
        // ending index of the subarray
        for(j = i; j < N; j++){
            // print sum of subarray from s,e
            sum = 0
            for(k -> i to j){ sum += arr[k] }
            print(sum)
        }
    }
}
```

TL → O(N^3)
S.L → O(1)

idea-2 → prefix sum.

```
void printSubarraySums(arr, N){
    // Create pSum[N]   [TODO]

    for(i=0; i<N; i++){        // starting index of the subarray
        for(j=i; j<N; j++){    // ending index of the subarray
            // sum of subarray [i,j]
            if(i==0)
                print(pSum[j])
            else
                print(pSum[j]-pSum[i-1])
        }
    }
}
```

T.C → $O(N^2)$

S.C → $O(N)$

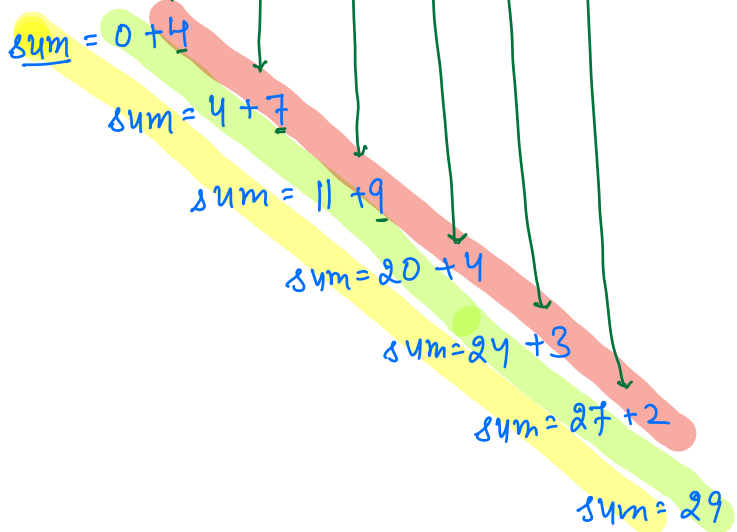
You can't modify the given array.

Q) Given arr[N]. Print all sub-array sums starting at index 3.

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

subarray.

[3,3] : 4
[3,4] : 11
[3,5] : 20
[3,6] : 24
[3,7] : 27
[3,8] : 29



Pseudo-code.

```
void printSums (arr, N) {
    sum = 0
    for (j = 3 ; j < N ; j++) {
        sum += arr[j]
        print (sum)
    }
}
```

T.C → O(N)
S.C → O(1)

[print all sub-array sums starting with index i -]

final pseudo-code.

Printing all sub-array sums using carry forward.

```
void printSums ( arr, N ){
    for( i = 0 ;  i < N  ; i++) {        // starting index of the subarray
        sum = 0
        for( j = i ; j < N ; j++){
            sum += arr[j]
            print (sum)
        }
    }
}
```

print all sub-array sums starting with ith index.

T.C → $O(N^2)$
S.C → $O(1)$

i = 0 : print all sub-array sums starting with idx 0

i = 1 : print all sub-array sums starting with idx 1

i = 2 : print all sub-array sums starting with idx 2

|

i = N-1 : print all sub-array sums starting with idx N-1

All sub-array sums are printed.

Q) Given N array elements, return sum of all sub-array sums.

arr[4] : { 6 8 -1 7 }
0 1 2 3

subarray	sum
[0,0]	6
[0,1]	14
[0,2]	13
[0,3]	20
[1,1]	8
[1,2]	7
[1,3]	14
[2,2]	-1
[2,3]	6
[3,3]	7

Sum of subarray sums = 94

arr[3] : { 4 3 7 }
0 1 2

subarray	sum
[0,0]	4
[0,1]	7
[0,2]	14
[1,1]	3
[1,2]	10
[2,2]	7

Sum of all sub-array sums → 45

idea : For every subarray, get sum & add it to total sum.

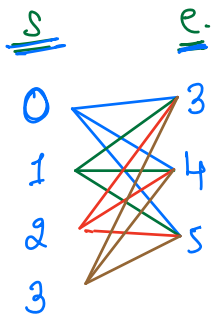
Approach-1.
→ 3 nested loops
→ TC : O(N3), SC : O(1)

Approach-2
→ prefix sum
→ TC : O(N²), SC : O(N)

Approach-3
→ carry forward
→ TC : O(N²), SC : O(1)

arr[6] : { 3 (0) -2 (1) 4 (2) -1 (3) 2 (4) 6 (5) }

In how many sub-arrays index - 3 is present?

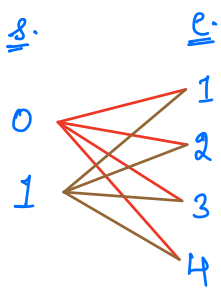


[4 * 3 = 12]

Quiz

arr[5] : { 3 (0) -2 (1) 4 (2) -1 (3) 2 (4) }

In how many sub-arrays index-1 is present?



[2 * 4 = 8]

Generalize- Given n elements, # subarrays ith index is present.

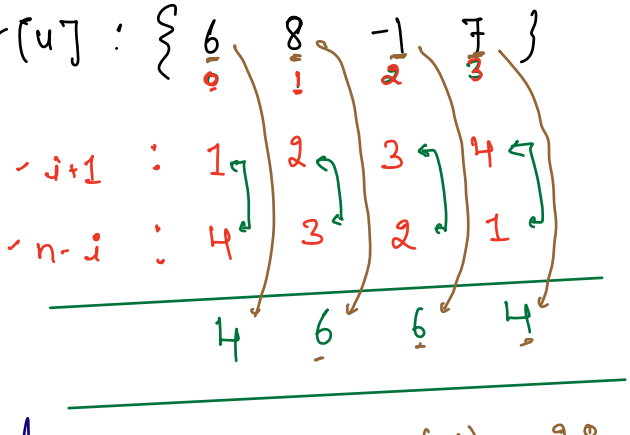
arr[N] : a_0 a_1 a_2 --- a_{i-1} a_i a_{i+1} ---- a_{N-2} a_{N-1}

s : [0, i] e : [i, N-1]

[i+1 × n-i]

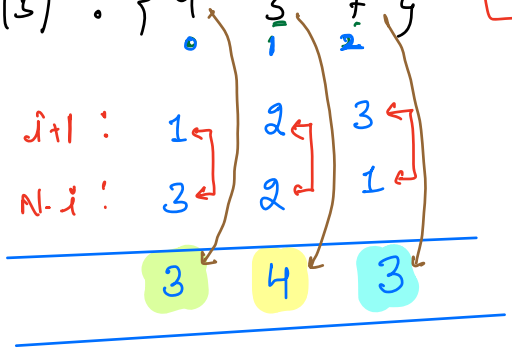
[a, b] = b - a + 1 [i, N-1] = N - 1 - i + 1 = N - i

Eg: arr[4] : { 6, 8, -1, 7 } N = 4



individual contribution = 24 + 48 + (-6) + 28 = 94

Eg: arr[3] : { 4, 3, 7 } N = 3



individual contribution: (4*3) + (3*4) + (7*3)

= 45.

sub-arrays:

[0,0] : { 4 }
[0,1] : { 4, 3 }
[0,2] : { 4, 3, 7 }
[1,1] : { 3 }
[1,2] : { 3, 7 }
[2,2] : { 7 }

(4*3) + (3*4) + (7*3)

= 45.

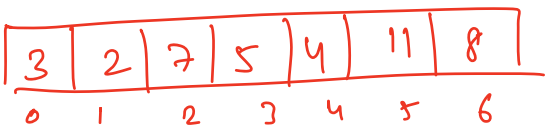
pseudo-code:

```
int totalSum( arr , N ) {
    totalSum = 0
    for( i = 0 ; i < N ; i++) {
        //How many times idx i is present in all sub-arrays
        int contri = arr[i] * [(i+1) * (N-i)]
        totalSum += contri
    }
    return totalSum
}
```

T.C → O(N)
S.C → O(1)

[Contribution technique]

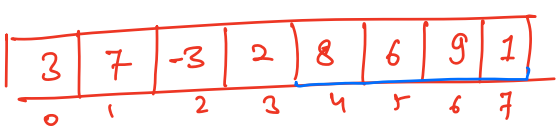
Doubts :




```
for( i = 0 ; i < N ; i++){
    for( j = i ; j < N ; j++){
        sb.append( arr[j]);
        print (sb)
    }
}
```

Pick from both sides

arr =



B = 4

left	right	
0	4	→ 24.
1	3	→ 19
2	2	→ 20
3	1	→ 8
4	0	→ 9

maxSum = 24.

Immutable.

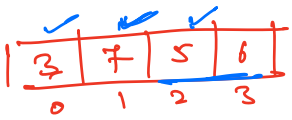
str → "abcd"

str + e = "abcde"

O(1)

Actually, it takes O(N)

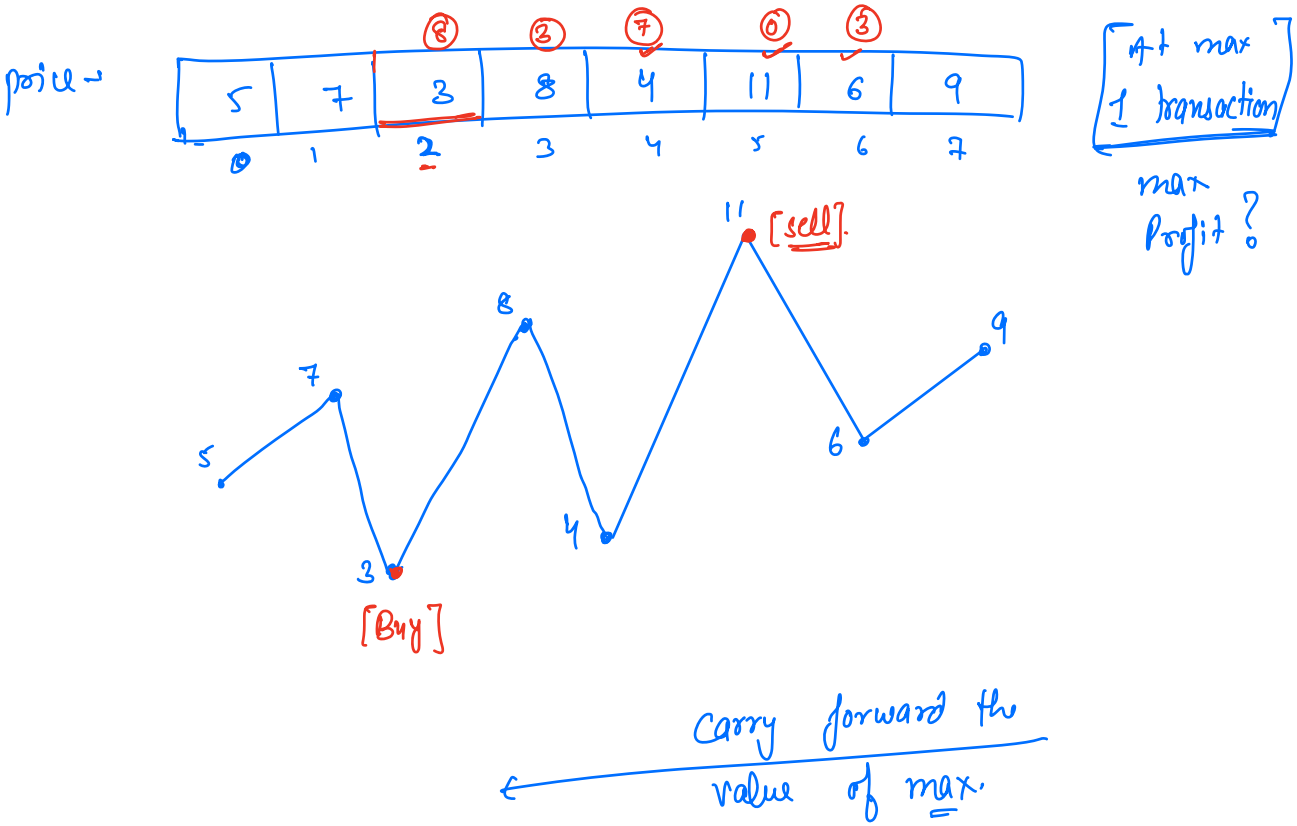
```
String res = " "
for (i = N-1 to 0) {
    res = A[i] + res;
    print(res)
}
```



res = 7 5 6.

{6}
{5, 6}
{7, 5, 6}
{3, 7, 5, 6}

{7, 5
5, 6
7, 5, 6
7
5
6}



max , second max.

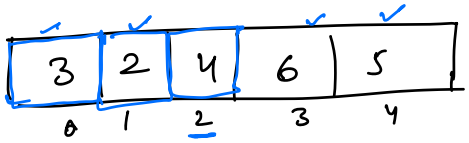
arr[] = { 1, 3, 6, 8, 5, 8, 1, 8 }.

smax →
Smax →

```
if ( arr[i] > max) {
    smax = max
    max = arr[i]
}
else if ( arr[i] > smax && arr[i] != max){
    smax = arr[i]
}
```

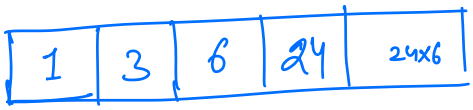
Product array.

arr →

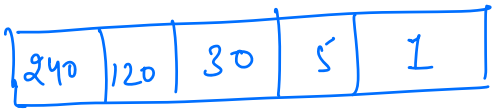


{Product of all no's. / arr[i].}

prefix →



suffix →



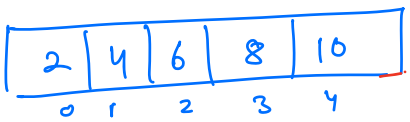
$\Rightarrow \frac{3 \times 2 \times 4 \times 6 \times 5}{4}$

$\Rightarrow 3 \times 2 \times 6 \times 5$

prefix[i] → product of all elements [0, i-1]

suffix[i] → product of all elements [i+1, N-1]

Q.1



Q.2