Today's agenda

- Subarrays recap.
- Subarrays questions.
- Contribution technique
- Number of subarray with bound
- Kadane's algo

Subarrays:

↳ Continuous Part of an array is Subarray.

(i) single element of an array? ✔

(ii) Complete array is also a subarray? ✔

(iii) empty array? ✗✗

(iv) {1, 2, 3, 4} → {3, 2} ??

↳ No, you can't reverse the order in subarray.

* Total no. of subarray in an array of length N.

↳ $\frac{N*(N+1)}{2}$

→ arr[5] = { 10 (0) 20 (1) 30 (2) 40 (3) 50 (4) }

Start 0th idx	Start 1st idx	Start 2nd idx	Start at last idx
10	20	30	
10 20	20 30	30 40	50
10 20 30	20 30 40	30 40 50	↓
10 20 30 40	20 30 40 50	3 : N-2	!
10 20 30 40 50	4 : N-1		...
5 : N			

Total Count : $N + (N-1) + (N-2) \ldots\ldots\ldots + 1$

$\frac{N*(N+1)}{2}$

Q) Given an arr[N], Print subarray from [s, e].

ex: arr[4] : { 5, 3, -1, 8 }
0 1 2 3

s = 1 e = 3

O/P 3 -1 8

```
for (int i=s; i<=e; i++){
    Print(arr[i]);
}
```

T.C: O(N)

Q) Given N array elements Print each and every subarray.

Ex: arr[4]: { 5, 3, -1, 8 }
(0, 1, 2, 3)

S=0		S=1		S=2		S=3
5	{0,0}	3	{1,1}	-1	{2,2}	8
5 3	{0,1}	3 -1	{1,2}	-1 8	{2,3}	
5 3 -1	{0,2}	3 -1 8	{1,3}			
5 3 -1 8	{0,3}					

```
void PrintSubArrays (int arr[N]){

    for(int s=0; s<N; s++){
        for(int e=s; e<N; e++){
            //[s,e] subarray
            for(int i=s; i<=e; i++){
                Print(arr[i]);
            }
        }
    }

}
```

T.C: O(N^3)

S.C: O(1)

Tracing

```
for (int s=0; s<N; s++){
    for (int e=s; e<N; e++){
        //[s,e] Sub array
        for (int i=s; i<=e; i++){
            Print(arr[i]);
        }
    }
}
```

arr[4]: { 5, 3, -1, 8 }
(0, 1, 2, 3)

s→0 → e=0 ⇒ 5
↳ e=1 ⇒ 5,3
↳ e=2 ⇒ 5,3,-1
↳ e=3 ⇒ 5,3,-1,8

s→1 → e=1 ⇒ 3

Q) Given N array elements, Print each subarray sum

Ex: arr[4] : { 5, 3, -1, 8 }

S=0	S=1	S=2	S=3
5 {0,0} → 5	3 {1,1} → 3	-1 {2,2} → -1	8 {3,3} → 8
5 3 {0,1} → 8	3 -1 {1,2} → 2	-1 8 {2,3} → 7	
5 3 -1 {0,2} → 7	3 -1 8 {1,3} → 10		
5 3 -1 8 {0,3} → 15			

```
void PrintSubArrays (int arr[N]){
    for(int s=0; s<N; s++){
        for(int e=s; e<N; e++){
            //[s,e] subarray
            int sum=0;
            for(int i=s; i<=e; i++){
                sum = sum+arr[i];
            }
            Print(sum);
        }
    }
}
```

T.C: O(N^3)

S.C: O(1)

//idea2

↳ Prefix Sum idea

Ex: arr[4] : { 5, 3, -1, 8 }

S=0	S=1	S=2	S=3
5 {0,0} → 5	3 {1,1} → 3	-1 {2,2} → -1	8 {3,3} → 8
5 3 {0,1} → 8	3 -1 {1,2} → 2	-1 8 {2,3} → 7	
5 3 -1 {0,2} → 7	3 -1 8 {1,3} → 10		
5 3 -1 8 {0,3} → 15			

Pf[4] : { 5, 8, 7, 15 }

Sum(i,j) : Psum[j] - Psum[i-1]

//Pseudo code

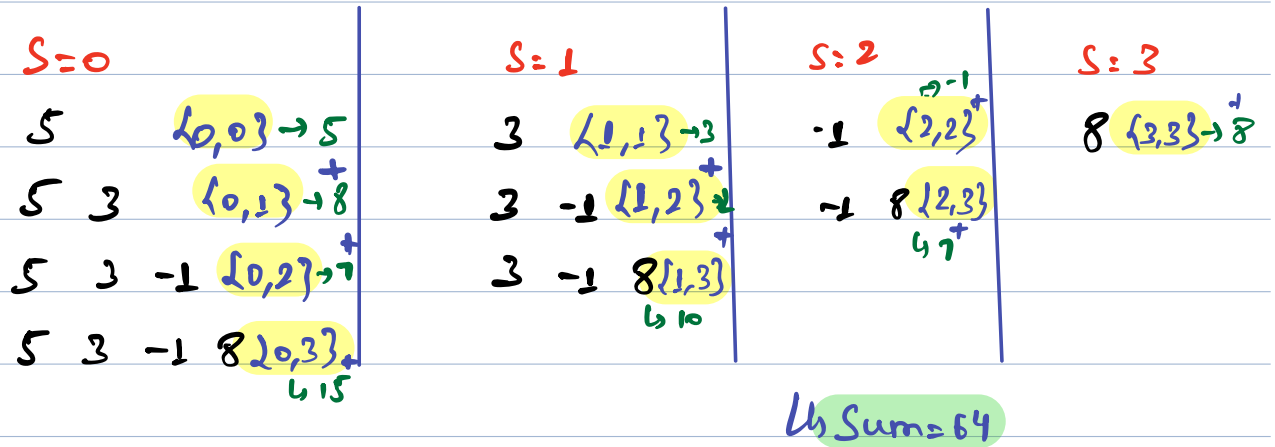
```
void PrintSubArrays(int arr[N]){
    int Psum[] = PrefixSum(arr);

    for(int s=0; s<N; s++){
        for(int e=s; e<N; e++){
            //[s,e] subarray
            if(s==0){ Print(Psum[e]); }

            else{ Print(Psum[e] - Psum[s-1]); }
        }
    }
}
```

T.C : O(N) + O(N²) = O(N²)

S.C = O(N)

Q) Given arr[N] elements, return Sum of all subarray sums

ex: arr[4] : {5 3 -1 8} (0 1 2 3)



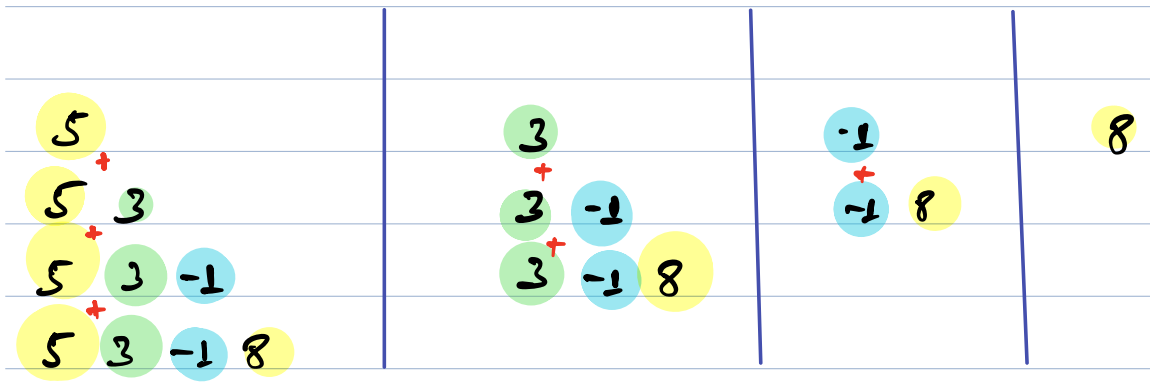

```
void PrintSubArrays(int arr[N]){
    int Psum[] = PrefixSum(arr);
    int ans=0;
    for(int s=0; s<N; s++){
        for(int e=s; e<N; e++){
            //[s,e] subarray
            if(s==0){ans=ans+Psum[e];}
            else{ans= ans+Psum[e]-Psum[s-1];}
        }
    }
    Print(ans);
}
```

T.C: O(N) + O(N²) = O(N²)

S.C = O(N)

//idea2

arr[4] : { 5 (0) 3 (1) -1 (2) 8 (3) }



↳ ans : 0

occ.

↳ arr[0] : 5 ⟶ 4 times

↳ 5 * 4 = 20

↳ arr[1] : 3 ⟶ 6 times

↳ 3 * 6 = 18

↳ arr[2] : -1 ⟶ 6 times

↳ -1 * 6 = -6

↳ arr[3] : 8 ⟶ 4 times

↳ 8 * 4 = 32

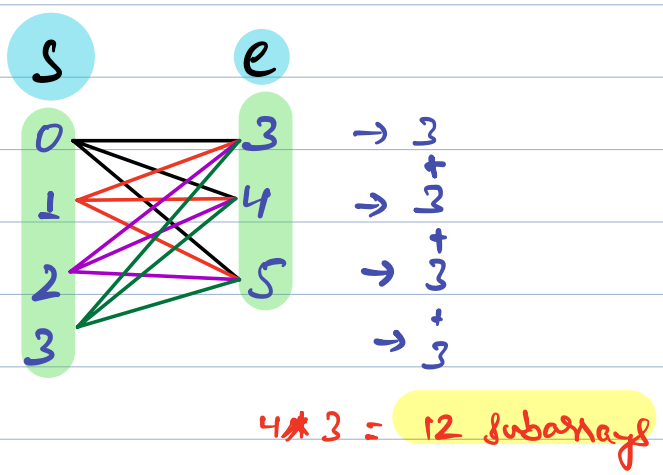
ans : 64

arr[4] : { 5 3 -1 8 }
0 1 2 3
↓ ↓ ↓ ↓
n0 n1 n2 n3

ans = (arr[0] * n0) + (arr[1] * n1) + (arr[2] * n2)
+ (arr[3] * n3)

→ Finding occurence

ex: { 3 -2 4 -1 2 6 }
0 1 2 3 4 5



4 * 3 = 12 subarrays

ex: [3 (0) | -2 (1) | 4 (2) | -1 (3) | 2 (4) | 6 (5)]

S	e
0	1
1	2
	3
	4
	5

⇒ 2 * 5 = 10 subarrays

general

ex: [3 (0) | -2 (1) | 4 (2) ← i | -1 (3) | 2 (4) | 6 (5)]

No. of occ = (No. of valid S) * (No. of valid e)

(No. of valid S) → [0, i] → (i+1) elements

(No. of valid e) → [i, N-1] → N-1-i+1 = (N-i) elements

Total no. of occ of ith index = (i+1) * (N-i)

//Pseudo code

```
int TotalSum(int arr[N]){
    int ans=0;
    for(int i=0; i<N; i++){
        int occ=(i+1)*(N-i);
        ans = ans + (occ*arr[i]);
    }
    return ans;
}
```

T.C: O(N)

S.C: O(1)

Tracing

arr[4]: {5 3 -1 8} (0 1 2 3)

```
int ans=0;
for(int i=0; i<N; i++){
    int occ=(i+1)*(N-i);
    ans = ans + (occ*arr[i]);
}
return ans;
```

occ = 4*1

ans = 0 + (5*4) + (3*6) + (-1*6) + 8*4

= 0 + 20 + 18 - 6 + 32

= 64

II/ Contribution technique

Q) number of subarrays with bounded maximum

Ls Given arr[N] and two integers left and right. Return the number of subarrays such that value of the maximum array element in that subarray is in the range {left, right}.

Ex1: arr[4]: { 2 1 4 3 } (0 1 2 3) left=2 Right=3

2
2 1
2 1 4
2 1 4 3

1
1 4
1 4 3

4
4 3

3

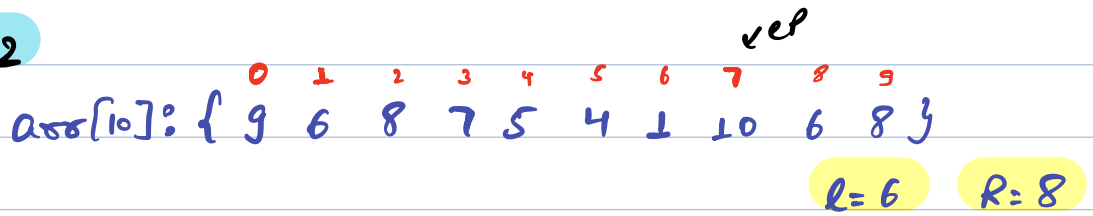
Ls ans=3

//Idea1

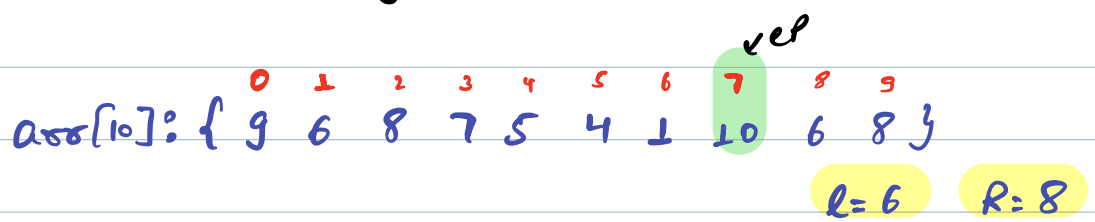
Ls create all the subarrays, Calculate the max and Pick the valid once.

T.C : $O(N^2) * O(N) \approx O(N^3)$

//idea2

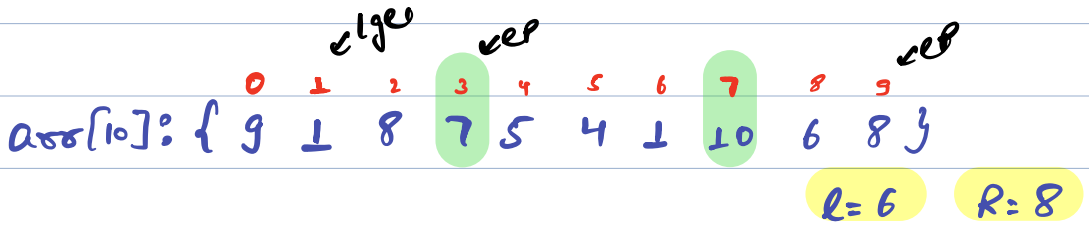


Case 1: arr[ep] > R



validStartingPoints = 0

Case 2: arr[ep] >= L && arr[ep] <= R

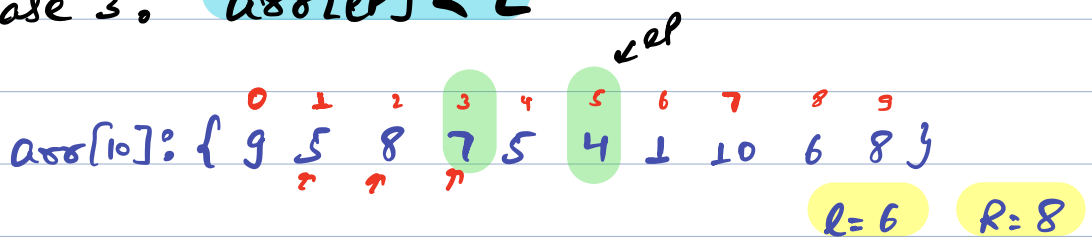


validStartingPoints = [lgei, ep]

↓ last greater than Right element index + 1.

ep - lgei + 1

Case 3: arr[ep] < L



validStartingPoints: no. of valid Starting Points for the element in the range of [L,R] just before the current element.

//Pseudocode

```
int validSubarray(arr[N], int left, int right){
    int ans = 0;
    int lgei = 0;
    int PervalidCount = 0;
    for(int ep=0; ep<N; ep++){
        if(arr[ep] > R){
            ans = ans + 0;        PervalidCount=0;
            lgei = ep+1;
        }
        else if(arr[ep] >=L && arr[ep]<=R){
            ans = ans + (ep - lgei + 1);
            PervalidCount = ep - lgei + 1;
        }
        else{ // arr[ep] < L
            ans = ans + PervalidCount;
        }
    }
}
```

T.C: O(N)

S.C: O(1)

Tracing

```
int validSubArray (arr[N], int left, int right){
    int ans = 0;
    int lgei = 0;
    int PervalidCount = 0;
    for (int ep=0; ep<N; ep++){
        if (arr[ep] > R){
            ans = ans + 0;        PervalidCount=0;
            lgei = ep+1;
        }
        else if (arr[ep] >= L && arr[ep]<=R){
            ans = ans + (ep - lgei + 1);
            PervalidCount = ep - lgei + 1;
        }
        else{ // arr[ep] < L
            ans = ans + PervalidCount;
        }
    }
```

arr[10]: { 9 6 8 7 5 4 1 10 6 8 }

indices: 0 1 2 3 4 5 6 7 8 9

ep → 3 (value 7)

lgei → 7 (value 10)

L = 6 R = 8

ans = 0+0+1+2+3+3+3+3+0+1+2

PerValidCount = 0 1 2 3 0 1 2

Q) max subarray sum → {Kadane's algo}

↳ Given N array elements, calculate max subarray sum.

Ex1: arr[7]: {3 2 -6 8 2 9 4}

↳ ans: 23

Ex2: arr[7]: {-3 2 4 -1 3 -4 3}

↳ ans: 8

//idea1

↳ generate all subarray sum and find max out of them.

T.C: $O(N^2)$ S.C: $O(N)$

//idea2 → {Kadane's algo}

{ +ve +ve +ve +ve +ve } → Sum of all these

{ -ve -ve +ve +ve +ve } → Sum of last three

{ +ve -ve +ve +ve +ve }

(-ve: < 0)

	0	1	2	3	4	5	6
arr[7] :	3	4	2	-14	16	-20	5
Sum : 0	3	7	9	-5	16	-4	5
ans = -∞	3	7	9	9	16	16	16

//Pseudo code

```
int Kadanes (int arr[N]) {
    int sum = 0;
    int ans = Integer.MIN_VALUE;

    for (int i=0; i<N; i++) {
        if (sum >= 0) {
            sum = sum + arr[i];
        }
        else {
            sum = arr[i];
        }
        ans = math.max (ans, sum);
    }
    return ans;
}
```

T.C : O(N)

S.C : O(1)

```
for (int i=0; i<N; i++) {
    if (sum >= 0) {
        sum = sum + arr[i];
    }
    else {
        sum = arr[i];
    }
    ans = Math.max(ans, sum);
}
return ans;
```

arr[7]: 3 4 2 -14 16 -20 5 (i: 0 1 2 3 4 5 6)

Sum = ~~0~~ 3 ~~7~~ ~~9~~ ~~-5~~ 16 ~~-4~~ 5

ans = ~~-∞~~ ~~3~~ ~~7~~ ~~9~~ ~~9~~ ~~16~~ ~~16~~ 16

Sum: 0 3 7 9 -5 16 -4 5

ans: -∞ 3 7 9 9 16 16 16

arr[7]: 3 4 2 -14 16 -20 5 (i: 0 1 2 3 4 5 6)