Arrays Technique

Hello Everyone

A very special Good Evening to All of you

We will start the session at 9:05 PM

Agenda

1. Reverse the Entire Array } Array Intro
2. Reverse the Subarray } Reversal within Sub Array
3. Range Sum Query } Prefix Sum
4. Prefix Sum Array
5. Equilibrium Index
6. Start and End of Subarray } Sliding Window
7. Max Subarray Sum with length K
8. Sum of all Subarrays Sum } Contribution Technique.

Reverse the Entire Array

Given an array 'arr' of size 'N'. Reverse the entire array.

arr: [10 20 30 40 50]
0 1 2 3 4

o/p: [50 40 30 20 10]
0 1 2 3 4

Approach:

arr[]: [10 20 30 40 50 60 70 80]
0 1 2 3 4 5 6 7

hi lo

ptr = n/2

T.C: O(n)

S.C: O(1)

```
int lo = 0;
int hi = n-1;
while( lo < hi) {
    //swap arr[lo], arr[hi]
    int temp = arr[lo];
    arr[lo] = arr[hi];
    arr[hi] = temp;
    //move variables
    lo++;
    hi--;
}
```

Reverse the Subarray

What is Subarray ?

(start index, end index)

* Subarray is contiguous part of an array
* complete array can be Subarray OR Single elmt can be also subarray.

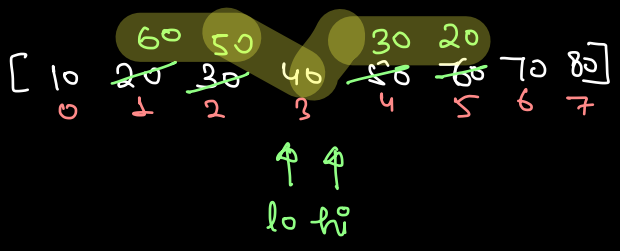
arr[]: [1 2 3 4 5 6 7 8]
0 1 2 3 4 5 6 7

start : 2
end : 7

o/p: [1 2 8 7 6 5 4 3]
0 1 2 3 4 5 6 7

```
function reverseSubarray(int[] arr, int start, int end){
    int lo = start;
    int hi = end;
    while( lo < hi){
        //swap arr[lo], arr[hi]
        int temp = arr[lo];
        arr[lo] = arr[hi];
        arr[hi] = temp;
        //move variables.
        lo++;
        hi--;
    }
}
```

Dry Run:



start = 1
end = 5

lo = 1 2 3
hi = 5 4 3

lo < hi
false
loop stop.

T.C: O(n)
S.C: O(1)

Range Sum Query

Given N elements and Q queries. For each query, calculate sum of all elements from L to R [0 based index].

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

query:

	L	R		sum b/w [L to R]
	4	8	→	9
	3	7	→	10
	1	3	→	12
	0	4	→	14
	7	7	→	-9

Approach - 0: [Brute force Approach]

* make an answer array of size q [no. of queries]
* Iterate on queries & for each query L to R
  → Iterate on array from L to R and calcute sum
  → place that sum in answer array.

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

L = 4 3 1 0 7
R = 8 7 3 4 7

query:

i	L	R	
0	4	8	→ 9
1	3	7	→ 10
2	1	3	→ 12
3	0	4	→ 14
4	7	7	→ -9

ans:

9	10	12	14	-9
0	1	2	3	4

```
ans[q];
for(q=0; q<totalQueries; q++) {        // q times; no. of queries
    int l = query[q][0];
    int r = query[q][1];
    int sum=0;
    for(int i=l; i<=r; i++) {          // in worst case ⇒ n iterations
        sum += arr[i];
    }
    ans[q] = sum;
}
```

T.C: O(q*n)
S.C: O(1)

Quiz1:

Given the scores of the 10 overs of a cricket match

2, 8, 14, 29, 31, 49, 65, 79, 88, 97

How many runs were scored in just 7th over?

2	8	14	29	31	49	65	79	88	97
1	2	3	4	5	6	7	8	9	10

arr[i] → total score after i^{th} over

How many Runs scored in 7^{th} over

= Score after 7^{th} over − Score after 6^{th} over.

= score[7] − score[6]

= 65 − 49 = (16) Ans

Quiz 2:

Given the scores of the 10 overs of a cricket match

2, 8, 14, 29, 31, 49, 65, 79, 88, 97

How many runs were scored from 6th to 10th over(both included)?

2	8	14	29	31	49	65	79	88	97
1	2	3	4	5	6	7	8	9	10

Score Scored b/w [6 to 10]th over = Runs after 10th over − Runs after 5th over

= score[10] − score[5]

= 97 − 31

= 66 Ans

Quiz 3:

Given the scores of the 10 overs of a cricket match

2, 8, 14, 29, 31, 49, 65, 79, 88, 97

How many runs were scored in just 10th over?

2	8	14	29	31	49	65	79	88	97
1	2	3	4	5	6	7	8	9	10

Score in 10^{th} over only = Score after 10th over − Score after 9th over = 97 − 88 = (9) Ans

Quiz 4:

Given the scores of the 10 overs of a cricket match
2, 8, 14, 29, 31, 49, 65, 79, 88, 97
How many runs were scored from 3rd to 6th over(both included)?

2	8	14	29	31	49	65	79	88	97
1	2	3	4	5	6	7	8	9	10

Score from 3rd to 6th over = score[6] − score[2]
= 49 − 8
= 41 Ans

Quiz 5:

Given the scores of the 10 overs of a cricket match
2, 8, 14, 29, 31, 49, 65, 79, 88, 97
How many runs were scored from 4th to 9th over(both included)?

Score [4 to 9] = score[9] − score[3]
= 88 − 14
= 74 Ans

overs →	1	2	3	4	5	6	7	8	9	10
Runs score in ith over →	2	7	4	9	16	3	0	7	5	6
Score board → Runs after ith over	2	9	13	22	38	41	41	48	53	59

→ Preparing Prefix Sum

How to create prefix sum:

arr[]: [2 6 −1 7 1]

psum[0] = arr[0]

psum[1] = arr[0] + arr[1]

psum[2] = arr[0] + arr[1] + arr[2] ⇒ psum[2] = psum[1] + arr[2]

(arr[0] + arr[1] = psum[1])

psum[3] = arr[0] + arr[1] + arr[2] + arr[3] ⇒ psum[3] = psum[2] + arr[3]

psum[2]

⇒ **psum[i] = psum[i-1] + arr[i]**, $i \neq 0$

arr[]: [2 5 -1 7 1]
0 1 2 3 4

psum[0] = arr[0]

psum: | 2 | 7 | 6 | 13 | 14 | ⟶ prefix sum array.
0 1 2 3 4

Quiz6:

Calculate the prefix sum array for following array:-

10 32 6 12 20 1

0 1 2 3 4 5

psum: | 10 | 42 | 48 | 60 | 80 | 81 |
0 1 2 3 4 5

Ans

```
function calculatePrefixSum(int[] arr){
    int psum[] = new int[n];
    psum[0] = arr[0];
    for(int i=1; i<n; i++){
        psum[i] = psum[i-1] + arr[i];
    }
    return psum;
}
```

T.C: O(n)

S.C: O(1) : If user is asking for Prefix Sum array.
→ O(1)

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

query:

psum:

-3	3	5	9	14	16	24	15	18
0	1	2	3	4	5	6	7	8

L R
(4) (8) → psum[8] - psum[3] = 18 - 9 = (9)
(3) (7) → psum[7] - psum[2] = 15 - 5 = (10)
(1) (3) → psum[3] - psum[0] = 9 - (-3) = (12)
★ (0) (4) → if left == 0 → psum[r] ⇒ psum[4] = (14)
(7) (7) → psum[7] - psum[6] = 15 - 24 = (-9)

```
function rangeSumQuery(int[] arr, int[][] query) {
    int n = arr.length;
    int Q = query.length;

    int[] ans[q];
    // prepare prefix sum array
    int[] psum = calculatePrefixSum(arr);   → function will return prefix sum

    // iterate on query & fill answer
    for(int q=0; q<Q; q++) {                → q times
        int l = query[q][0];
        int r = query[q][1];
        if(l==0) {
            ans[q] = psum[r];
        } else {
            ans[q] = psum[r] - psum[l-1];
        }
    }
    return ans;
}
```

n times
n space → for prefix sum
↳ We can use same array for prefix sum.

Total qtr = (n+q)

T.C: O(n+q)
S.C: O(n);
↳ of prefix sum;

if we use same array for prefix sum: O(1)

Equilibrium Index

Find the number of equilibrium index in the give array of size N where equilbrium index is defined as :-
sum of all elements to the left of index = sum of all elements to the right of index

Note:- for index = 0 then left_sum = 0
for index = n - 1 then right_sum = 0

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

	leftsum	rightsum	
i=0	0	5	
i=1	-3	3	
★ i=2	-1	-1	→ equilibrium index
i=3	3	0	

Quiz 7:

Which one is an equilibrium index in this array?

-7, 1, 5, 4, 2, -4, 3, 0 ?
0 1 2 3 4 5 6 7

i	left sum	right sum
0	0	11
1	-7	10
2	-6	5
3	-1	1
4	3	-1
5	5	3
6	1	0
7	4	0

0 : → 0 Equilibrium point

→ wrong option:

Brute force Approach:

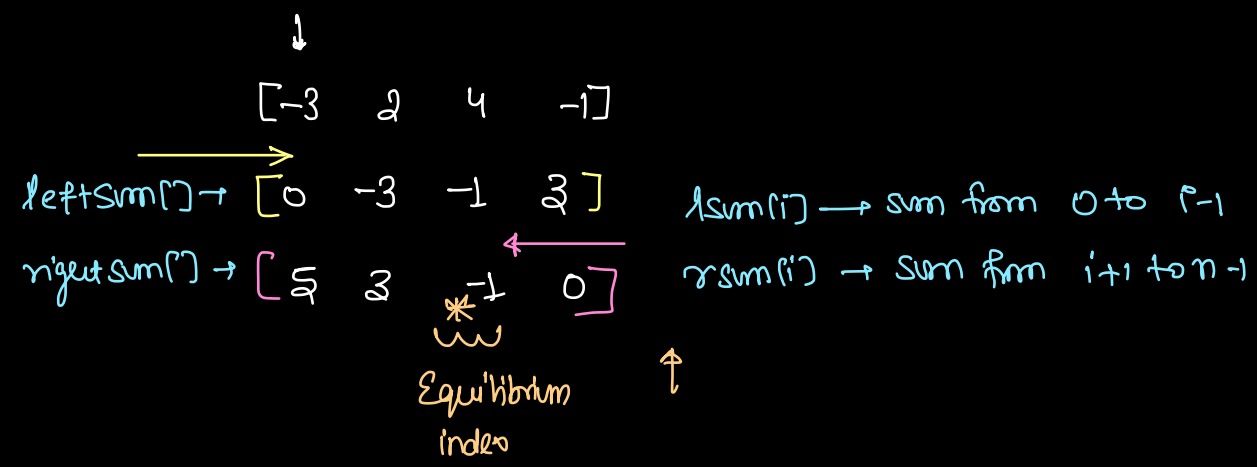
```
int cnt=0;
for(int i=0; i<n; i++){ ──────────→ n iterations
    // calculate left sum from 0 to i-1
    // calculate right sum from i+1 to n-1 ──→ n times iterations
    if( lsum == rsum){
        cnt++;
    }
}
return cnt;
```

total itr = n*n

⇒ T.C: O(n²)

S.C: O(1)

Optimised Approach:



TODO: Write code part

10:31 – 10:41 → break

Total number of subarrays having length is K ?

arr → N → length

Subarrays with length ⇒ K

arr [0 1 2 3 4 5 6 7] , K=4

0 → 3
1 → 4
2 → 5
⋮
4 : 7 →

[3 7] = 7-3+1
= 5

generic example

first window ⇒ 0 → K-1
2nd " 1 → K
3rd " 2 → K+1
⋮
last : 0 → n-1

[K-1 to n-1]
⇒ [n-1-(K-1)+1]
= n-K+1 ⇒ n-K+1 Ans.

Quiz: 8

Given N = 7, K = 4, what will be the total number of subarrays of len K? → Ans = n-K+1 = 7-4+1 = 4

~~~~~~~~~~~~~~~~~~~~~~~~~~

Start and End of Subarray

~~~~~~~~~~~~~~~~~~~~~~~~~~

Given an array of size N, print start and end indices of subarrays of length K.

N = 8, K = 3

	Start	end	
O/P:	0	2	window 1
	1	3	2nd
	2	4	3rd
	3	5	4th
	4	6	5th
	5	7	6th

```
i=0;
j=K-1;
while(j<n){
    print(i j); → window of size K
    i++;
    j++;
}
```

Max Subarray Sum with length K

Given an array of N elements. Print maximum subarray sum for subarrays with length = K.

arr[]: [-3 4 -2 5 3 -2 8 2 -1 4] , N=10 , K=5
indices: 0 1 2 3 4 5 6 7 8 9

all windows →

Start	end	Sum in window
0	4	7
1	5	8
2	6	12
3	7	16
4	8	10
5	9	11

max subarray sum

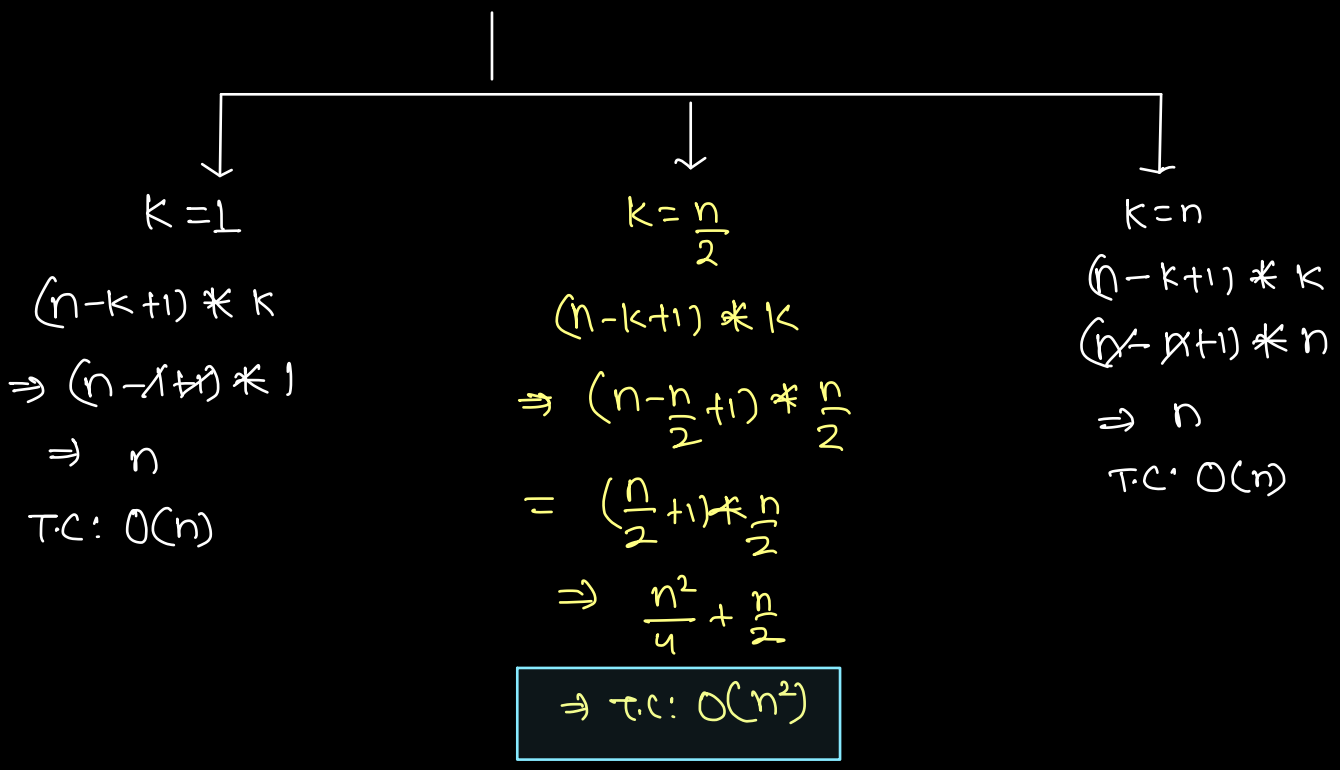
max sum = 16

Bruteforce Approach:

* Iterate on all window of size K
* for each window calculate the sum of that window.
* maximise that sum.

```
int ans = (-∞);  → Smallest possible value.
int i=0;
int j=K-1;
while(j<n){                                   → All subarray having length K
                                                 ctr = n-K+1
    // i & j are index of window
    int sum=0;
    for(int indx=i; indx<=j; indx++){         → K times
        sum += arr[indx];
    }
    ans=Max(ans, sum);
    i++;
    j++;
}
return ans;
```

total ops = $(n-k+1) * k$



Approach1: Using prefix sum

arr[]: [-3 4 -2 5 3 -2 8 2 -1 4] , N=10 , K=5

psum → [-3 1 -1 4 7 5 13 15 14 18]

all windows →

start	end	sum in windows
0	4	psum[4] → 7
1	5	psum[5] − psum[0] = 5 − (−3) = 8
2	6	psum(6) − psum(1) = 13 − 1 = 12
3	7	psum(7) − psum(2) = 15 − (−1) = 16
4	8	psum(8) − psum(3) = 14 − 4 = 10
5	9	psum(9) − psum(4) = 18 − 7 = 11

function maxSumSubarray with length K (int[] arr, int K) {

```
int[] psum = calculatePrefixSum(arr);
int i = 0;
int j = k+1;
int sum = psum[j];
i++;
j++;
int ans = sum;
while(j<n){
    sum = psum[j] - psum[i-1];
    ans = max(ans, sum);
    i++;
    j++;
}
}
```

→ n Itr.

→ n space for prefix sum

→ n-k+1

total Itr = n + (n-k+1)

= 2n-k+1

⇒ T.C: O(n)

S.C: O(n)

approach 2: sliding window approach:

arr[]: [~~-3~~ ~~4~~ 5 3 -2 8 2 -1 4] , N=10 , K=5

0 1 2 3 4 5 6 7 8 9

first window sum = -3+4-2+5+3 = 7 ans = ~~7~~ 8 ~~12~~ 16

2nd " " = ~~-3~~ +4-2+5+3-2 = 8

3rd " " = ~~4~~ -2+5+3-2+8 = 12

4th " " = ~~-3~~ +5+3-2+8+2 = 16

5th " " = ~~5~~ +3-2+8+2-1 = 10

6th " " = ~~3~~ -2+8+2-1+4 = 11

Steps:

① prepare sum for first window (0 to k-1)

② ans = sum;

③ i++, j++; ⟶ second window.

④ while(j<n){
  acquire j^{th} index & release $(i-1)^{th}$ index
  maximise ans with new sum.
  move window for next one
}

⑤ return ans;

```
function slidingWindowMax(int[] arr, int k){
    int sum=0;
    for(int i=0; i<k; i++){ ⟶ k Iteration
        sum += arr[i];
    }
    int ans = sum;
    int i=1;
    int j=k;
    while(j<n){ ⟶ (n-k) times.
        //acquire jth index & release (i-1)th index
        sum+= arr[j];
        sum -= arr[i-1];
        ans = max(ans, sum);
        i++;
        j++;
    }
    return ans;
}
```

total iter = k + (n - k)
= n iter

⇒ T.C : O(n)
S.C : O(1)

Sum of all Subarrays Sum

Given an array of integers, find the total sum of all possible subarrays.
Note: This question has been previously asked in Google and Facebook.

arr[]: [1 , 2 , 3]

all Subarrays:

[1] ⟶ 1 ⟶ 1
[1 2] ⟶ 1+2 ⟶ 3
[1 2 3] ⟶ 1+2+3 ⟶ 6
[2] ⟶ 2 ⟶ 2
[2 3] ⟶ 2+3 ⟶ 5
[3] ⟶ 3 ⟶ 3

Total sum = 1+3+6+2+5+3
= 20

Bruteforce Approach:

→ make overall sum variable
→ iterate on all possible subbarry calculate sum
→ Calculate sum & add that sum in overall sum.
→ Return overall sum.

sum [prefix sum] → O(n) space
Carry forward → O(1) space

T.C: $O(n^2)$
S.C: ⟶ prefix sum ⇒ O(n)
Carry forward ⇒ O(1)

TODO: → write code for Both the approaches.

arr[] : [1, 2, 3]

[1] ⟶ 1

[1 2] ⟶ 1 + 2

[1 2 3] ⟶ 1 + 2 + 3

[2] ⟶ 2

[2 3] ⟶ 2 + 3

[3] ⟶ 3

Sum = $x*1 + y*2 + z*3$

$x = 3,\ y = 4,\ z = 3$

$\Rightarrow$ Sum = $3*1 + 4*2 + 3*3$

$= 3 + 8 + 9$

$= 20$

How many times arr[i] becomes added in overall sum?

Quizlo:

In how many subarrays, the element at index 1 will be present?

A: [3, -2, 4, -1, 2, 6]

0 1 2 3 4 5

Start ⟶ 0, 1

end ⟶ 1, 2, 3, 4, 5

(2) x (5)

⇒ 10 possibility

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

[1,1], [1,2], [1,3], [1,4], [1,5]

[Start, end] of subarrays

Count = 10

Quiz 11:

In how many subarrays, the element at index 2 will be present?

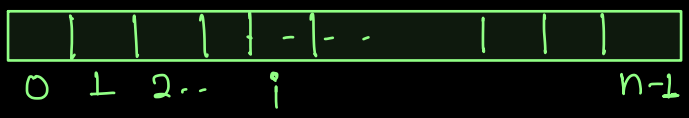
A: [3, -2, 4, -1, 2, 6]

0 1 2 3 4 5

Start	ending point
0	2
1	3
2	4
	5

(3) * (4) = 12 Subarrays is possible

Optimise way: Contribution of arr[i] in overall sum.



How many time arr[i] will be there in overall sum ?

Start → [0 to i] ⟶ i - 0 + 1 = (i+1)

[a to b] = b - a + 1

Endig → [i to n-1] ⇒ (n-1-i+1) = (n-i)

count of arr[i] in overall sum	= (i+1) * (n-i)

count = (i+1) * (n-i)

arr: [1, 2, 3]
0 1 2

overallsum = ~~0~~ 3 ~~4~~ 20

n=3

i	count	contribution
0	1*3=3	3*arr[0] → (3)
1	2*2=4	4*arr[1] = → (8)
2	3*1=3	3*arr[2] = (9)
(3)		

loop stop → (20)

function overallSum of subarraySum(int[] arr){

```
int overallSum = 0;
for(int i=0; i< n; i++){
    int count = (i+1)*(n-i);
    int contri = count * arr[i];
    overallSum += contri;
}
return overallSum;
}
```

T.C: O(n)
S.C: O(1)