Introduction to Arrays

> The first step to achieving your goal, is to take a moment to respect your goal. Know what it means to you to achieve it.
>
> DWAYNE JOHNSON

Good Evening

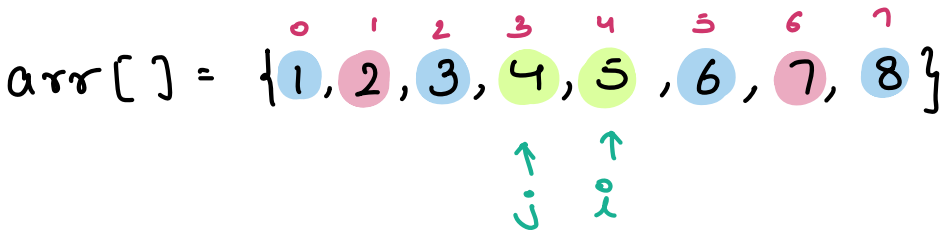
Today's content

01. Reverse the array & subarray
02. Range sum queries
03. Equilibrium Index
04. Maximum subarr sum of size K
05. Sum of all subarray sum (Google)

01. Given an array, **Reverse** the entire array

Note :- Change the given array

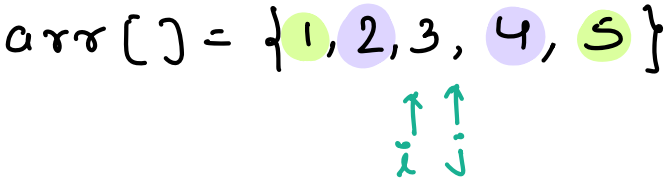
S.C = O(1)



i < j

Breaking point

i > j



Breaking point

i == j

arr[] = {5, 4, 3, 2, 1}

```
void reverse(int[] arr)
{
    i = 0 , j = N-1

    while(i<j)
    {
        swap(arr, i, j);
        i++;
        j--;
    }
}
```

No. of iterations = $N/2$

TC : O(N)

SC : O(1)

02. Given the array of N elements. Reverse the subarr from si to ei.

↳ Continous part of arr
↳ Complete arr
↳ Single ele

arr[ ] = { 1, 2, 3, 4, 5, 6, 7, 8 }
0 1 2 3 4 5 6 7

si = 2
ei = 7

arr[ ] = { 1, 2, 8, 7, 6, 5, 4, 3 }

```
void reverse (int [] ar, int si, int ei){
    i = si , j = ei

    while (i<j)
        swap(arr, i, j);
        i++;
        j--;
    }
}
```

TC: O(n)
SC: O(1)

01. Given arr[N] elements & Q queries

For each query : Given L & R calculate & print sum of elements in range [L R], L & R both included

Note :- L & R are array indices such that $0 \le L \le R < N$

arr[10] =

-3	6	2	4	5	2	8	-9	3	1
0	1	2	3	4	5	6	7	8	9

pf[ ] =

-3	3	5	9	14	16	24	15	18	19
0	1	2	3	4	5	6	7	8	9

Queries = 5

L	R	
4	8	= 9
3	7	= 10
1	3	= 12
0	4	= 14
(7)	7	= -9 → pf[7] − pf[6] = 15 − 24 = −9

Constraints = $1 \le N \le 10^3$

$1 \le Q \le N$

BF = For every query, iterate from L to R & print the sum.

TC: O(Q*N)

SC: O(1)

Q: Given Indian Cricked Team score, for first 10 overs of Batting.

After every over, total score is given as:

overs →	1	2	3	4	5	6	7	8	9	10
sc[10] →	2	8	14	29	31	49	65	79	88	97

01. Runs scored in 10^{th} over =

Runs scored till 9^{th} over + Runs scored in 10^{th} over = Runs scored in 10 overs

$$sc[9] + x = sc[10]$$

$$x = sc[10] - sc[9]$$
$$= 97 - 88 = 9$$

02 Runs scored from 6^{th} to 10^{th} over

$$sc[10] - sc[5] = 97 - 31 = 66$$

03. Runs scored from 3rd over to 8th over =

$sc[8] - sc[2] = 79 - 8 = 71$

Total no. of runs i to j = $sc[j] - sc[i-1]$

04. Runs scored till 5th over = $sc[5]$

Total no. of runs scored from 0 to j = $sc[j]$

Generic formula

$i = 0$ ans = psum[j]

else ans = psum[j] - psum[i-1];

arr =

4	6	3	-2	8
0	1	2	3	4

psum =

4	10	13	11	19

↳ sum[0-2]

psum[0] = ar[0]

psum[1] = ar[0] + ar[1]

ar[0] = psum[0]

psum[2] = ar[0] + ar[1] + ar[2]

psum[1]

psum[3] = ar[0] + ar[1] + ar[2] + ar[3]

psum[2] + ar[3]

```
if (i==0)  psum[0] = ar[0];
else {
    psum[i] = psum[i-1] + ar[i];
}
```

* Construct pf array

```
Pf[0] = ar[0]
for (i=1 ; i<n; i++){
    psum[i] = psum[i-1]+ar[i];
}
```

O(n)

TC: O(N+Q)
SC: O(N)

* Answer all queries

```
for (i=0 ; i<Q ; i++){
    s = L[i] , e = R[i]
    if (s==0)  print ( psum[e]);
    else    print ( psum[e] - psum[s-1]);
}
```

O(Q)

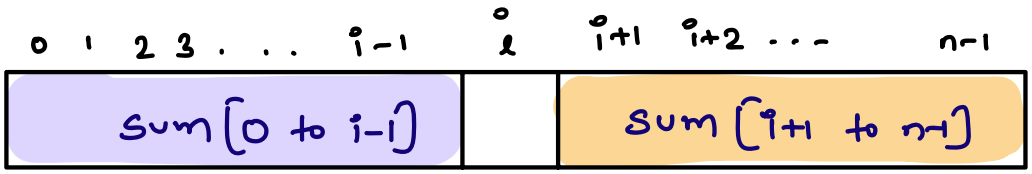
10 pm → 10:10 pm

Q. **Equilibrium Index**

Given N array elements, count no. of equilibrium idx

An idx i is known as eqm idx

sum of all elements before i idx = sum of all elements after i idx



Note:- if (i == 0) leftsum = 0

if (i == N-1) rightsum = 0

	0	1	2	3	
ar[4] = {	-3	2	4	-1 }	count = 1
lsum	0	-3	-1	3	
rsum	5	3	-1	0	

	0	1	2	3	4	5	6	
ar[7] = {	-7	1	5	2	-4	3	0 }	count = 2
lsum =	0	-7	-6	-1	1	-3	0	
rsum =	7	6	1	-1	3	0	0	

Brute force → For every index, iterate & calculate lsum & rsum. if (lsum == rsum) c=c+1;

return count:

$TC = O(n^2)$

$SC = O(1)$

Optimised step

Code → {TODO}

01. Create a pfsum array

02. For every index,

- → calculate lsum using pfsum array
- → calculate rsum using pfsum array
- → if (lsum == rsum)
  - → c = c + 1

}

return c;

01. Given N elements, print **max subarray sum** of **length 'k'**.

arr[10] =

-3	4	-2	5	3	-2	8	2	-1	4
0	1	2	3	4	5	6	7	8	9

k=5

st	end	sum
0	4	7
1	5	8
2	6	12
3	7	16
4	8	10
5	9	11

Idea 1 → For every subarr of size k, iterate & calculate the sum and update the max accordingly

TC : #subarr of size k * K

= O((n-k+1) * k)

k=1 → TC: O(n)

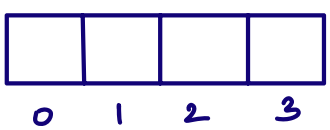
k=n → TC: O(n)

$k = n/2 \rightarrow O(n - n/2 + 1) * \frac{n}{2}$

$= O(n/2 + 1) * \frac{n}{2}$

$\simeq O(n^2)$

TC: $O(n^2)$
SC: O(1)



N-K+1

k=1 Ans = 4

k=2 Ans = 3

k=3 Ans = 2

k=4 Ans = 1

Use psum approach ⟶ TC: O(n) SC: O(n)

Idea 3 → Sliding Window TC: O(n) SC: O(1)

ar[10] =

-3	4	-2	5	3	-2	8	2	-1	4
0	1	2	3	4	5	6	7	8	9

K=5

st	end	sum
0	4	7
1	5	sum = sum − ar[0] + ar[5] = 8
2	6	sum = 8 − ar[1] + ar[6] = 12
3	7	sum = 12 − ar[2] + ar[7] = 16
4	8	sum →
5	9	sum →

```
int maxsubarrsum (int [] ar, int k)
{
    ans = -∞ . sum = 0

    for(i=0 ; i<k ; i++){
        sum = sum + ar[i]
    }

    ans = max(ans, sum);

    s = 1 , e = k

    while (e < n)
    {
        sum = sum - ar[s-1] + ar[e]

        ans = max(ans, sum);

        s = s+1;
        e = e+1;
    }
}
```

k iterations — 1st window

(n-k) iteration

$TC = O(k + n - k) \simeq O(n)$

$SC = O(1)$

Q Given an ar[n], return sum of all subarr sums.

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

Ans = 94

subarr		sum
[0 0]	→ 6	6
[0 1]	→ 6+8	14
[0 2]	→ 6+8+(-1)	13
[0 3]	→ 6+8+(-1)+7	20
[1 1]	→ 8	8
[1 2]	→ 8+(-1)	7
[1 3]	→ 8+(-1)+7	14
[2 2]	→ (-1)	-1
[2 3]	→ -1+7	6
[3 3]	→ 7	7

6 * 4 = 24
8 * 6 = 48
-1 * 6 = -6
7 * 4 = 28
94

Idea I → For every subarr, iterate & get the sum → add it to the total su

```
for (i=0 ⟶ n-1)
    for (j=i ⟶ n-1)
        sum=0;
        for (k=i ⟶ j){
            sum+=arr[k];
        }
        totalsum+=sum;
```

TC: O(n³)
SC: O(1)

* Psum approach → TC: O(n²)
SC: O(n)

* Contribution technique → Find the count of the no. of times a particular ele is contributing * particular ele

arr[] = {3, -2, 4, -1, 2, 6}
0 1 2 3 4 5

In how many subarrays, index 3 will be present

st	end
0	3
1	4
2	5
3	

subarrs: (0,3) (0,4) (0,5)
(1,3) (1,4) (1,5)
(2,3) (2,4) (2,5)
(3,3) (3,4) (3,5)

Ans = 4 * 3 = 12

arr[] = { 1 2 3 4 }
0 1 2 3

In how many subarray, index 1 is present

st	end
0	1
1	2
	3

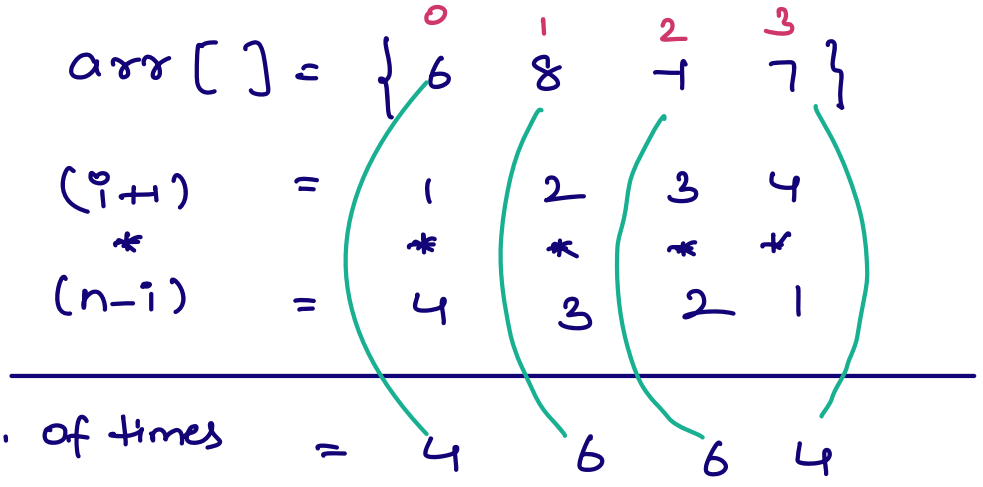
Total no. of subarray = 2 * 3 = 6

* Given N elements, how many subarrays have this i^{th} idx present

ar[] = { a_0 a_1 a_2 ... a_{i-1} a_i a_{i+1} .... a_{n-1} }

(i+1) * (n-i)

No. of times a partiular ele is going to contribute $= (i+1) * (n-i)$



i^th ele is going to contribute

Overall contribution $= 24 + 48 + (-6) + 28 = 94$

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

TC = O(n)

SC = O(1)