Todays Content:

1. 0/1 prefixSum
2. Count Triplets
3. Buy & Sell stocks

Questions :

psum[i] = Total sum of elements from [0..i]

Sum of elements l..r using psum[] =

```
if(l==0){ //[0..r]
    print(pf[r]);
}else{
    print(pf[r]- pf[l-1]);
}
```

When can we store pfsum[] → arr[]

1. If both have same data type: pf[] & arr[].

Q1: Given ar[N] elements & Qmat[Q][2]

In Qmat matrix, we have Q: rows & 2: columns

Each row in Qmat represents a query.

0^{th} col in row represents : start point of query → s = Qmat[i][0]

1^{st} col in row represents : end point of query → e = Qmat[i][1]

For every query calculate no: of even elements of index s..e in ar[] & print

Constraints:

$1 \le N \le 10^5$

$1 \le ar[i] \le 10^9$

$1 \le Q \le 10^5$

$0 \le s \le e < N$.

} Count of even elements

Min: 0 Max: $N \le 10^5$ → int range

Can store pf[] → ar[] itself

Ex:

	0	1	2	3	4	5	6	7	8	9
ar[10] = {	2	4	3	7	9	8	6	3	4	9 }

Qmat[4][2]

	0	1	Output
0	4	8	3
1	3	7	2
2	1	3	1
3	0	4	2

Ideal:

For every query:

Iterate from s..e calculate no: of even elements

Estimated TC : O(Q*N) SC: O(1)

void QueriesEven(int ar[], int N, int mat[][], int Q){ :TODO

Optimization:

ar[i] is even : 1 ar[i] is odd : 0 : count of even numbers

	0	1	2	3	4	5	6	7	8	9
ar[10] :	2	4	3	7	9	8	6	3	4	9
update ar[] :	1	1	0	0	0	1	1	0	1	0
sum = 0	1	2	2	2	2	3	4	4	5	5
pf[10] :	1	2	2	2	2	3	4	4	5	5

&mat[4][2]

	0	1
0	4	8
1	3	7
2	1	3
3	0	4

Output

pf[8] - pf[3] = 5 - 2 = 3

pf[7] - pf[2] = 4 - 2 = 2

pf[3] - pf[0] = 2 - 1 = 1

pf[4] - pf[-1]

→ Count of even numbers [0..4] = pf[4] = 2

Note: Above pf based technique: 0/1 pf sum

TC: O(N+N+Q) SC: O(N) Can store pf() → ar() itself: TODO

```
void QueriesEven(int ar[], int N, int mat[][], int Q){
    Step1:
    for(int i=0; i<N; i++){
        if(ar[i]%2==0){ ar[i]=1;}
        else{ ar[i]=0;}
    }

    Step2:
    int pf[N], sum=0;
    for(int i=0; i<N; i++){
        sum=sum+ar[i];
        pf[i]=sum;
    }

    Step3;
    for(int i=0; i<Q; i++){
        int s=Q mat[i][0], e=Q mat[i][1];
        if(s==0){ // [0..e]
            print(pf[e])
        }else{
            print(pf[e]-pf[s-1]);
        }
    }
}
```

Q2

Count of Triplets

Given ar[N], calculate no: of triplets i < j < k & ar[i] < ar[j] < ar[k]

Constraints:

$1 \le N \le 10^3 \rightarrow O(N^3) = (10^3)^3 = 10^9 > 10^8$ TLE

$\rightarrow O(N^2) = (10^3)^2 = 10^6 < 10^8$ ✓

0 1 2 3 4

Eg1: ar[5] = {2 6 9 4 10}

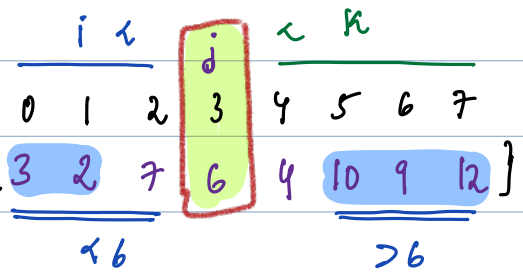
i < j < k	ar[i] < ar[j] < ar[k]
0 1 2	2 < 6 < 9
0 1 4	2 < 6 < 10
1 2 4	6 < 9 < 10
0 3 4	2 < 4 < 10
0 2 4	2 < 9 < 10

Ideal: Generate all triplets:

check if i<j<k && ar[i] < ar[j] < ar[k] = cnt++;

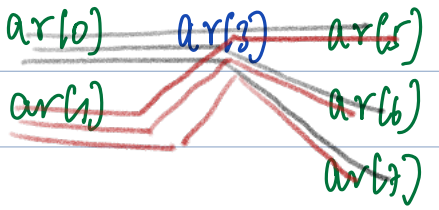
TC: O(N3) ; TODO SC: O(1)

Hint: For triplet based questions try to fix center element & search for left & right.



$i < j < k$

$ar[i] < ar[j] < ar[k]$



Ideal: For every ar[j]

Calculate no: elements < ar[j] in left = c_L

Calculate no: elements > ar[j] in right = c_R

Triplets += $c_L * c_R$

Estimated TC:

Trace:

	0	1	2	3	4	5	6	7 ↑
Ex: ar[8] = {	3	2	7	6	4	10	9	12 }
Count less :	0	0	2	2	2	5	5	7
Count more :	6	6	3	3	3	1	1	0
Triplets =	0	0	6	6	6	5	5	0

ans = 28

Note: For above question, we cannot apply carry forward, because for every element we will calculate no: of elements less than itself, data keeps changing.

Buy & Sell Stocks:

Given an ar[N], where ar[i] is price of given stock in i^{th} day

Return max profit which can be achieved by exactly 1 transaction

Note1: If we buy a stock in i^{th} day: We can sell on any day { $i+1, i+2, i+3, \ldots N-1$ }

Note2: If cannot achieve any profit: return 0;

Constraints:

$1 <= N <= 10^5$

$1 <= ar[i] <= 10^9$

Ex1:

ar[] = { 7 1 5 3 6 4 } ans =

Ex2:

	0	1	2	3	4	5	6
ar[] = {	4	6	10	4	2	9	1 }

Idea: In Stock we

i	0	1	2	3	4	5	6	Selling Price	Profit day i^{th}
{	4	6	10	4	2	9	1 }		

App:

TC: SC:

Ideal:

	0	1	2	3	4	5	6
i	{ 4	6	10	4	2	9	1 }

: max

: profit

```
int maxProfit(int arr[], int n){
```

3