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[] =   l.... r

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

When can we store pfsum[] → ar[]

1. When datatype pfsum[] ↔ ar[] same : Major Rule.
2. If we no longer use arr[] : We can always make a copy

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

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

Each row in Qmat represents a query.

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

1st 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:

$1 <= N <= 10^5$

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

$1 <= Q <= 10^5$

$0 <= s <= e < N.$

} Count of Even elements

Min: 0   Max = $N <= 10^5$ → int ✓

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

Idea1:

For every query:

Iterate from s..e get count of even numbers & print it.

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

Code: TODO

Optimization: ar[i] is even = +1   ar[i] is odd = 0

	0	1	2	3	4	5	6	7	8	9
ar[10] :	2	4	3	7	9	8	6	3	4	9
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

Qmat[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] : Error :

No: of even ele : [0..4] = pf[4] = 2 → Write explanation?

Note: Above technique is considerd 0/1 prefix Sum

```
                          → Can optimize:
TC: O(N+N+Q)      SC: O(N)  Can store pf[] → ar[] SC: O(1)  : 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 = Qmat[i][0], e= Qmat[i][1];
       if( s==0) { //{0..e}
       }   print(pf[e]);
       else{
           print( pf[e]- pf[s-1]);
       }
   }
}
```

Q2

Count of Triplets

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

Constraints:

$1 <= N <= 10^3$ → $O(N^3) = (10^3)^3 = 10^9 > 10^8$ TLE

→ $O(N^2) = (10^3)^2 = 10^6 < 10^8$ ✓

0 1 2 3 4

Ex1: ar[5] = {2 6 9 4 10}

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

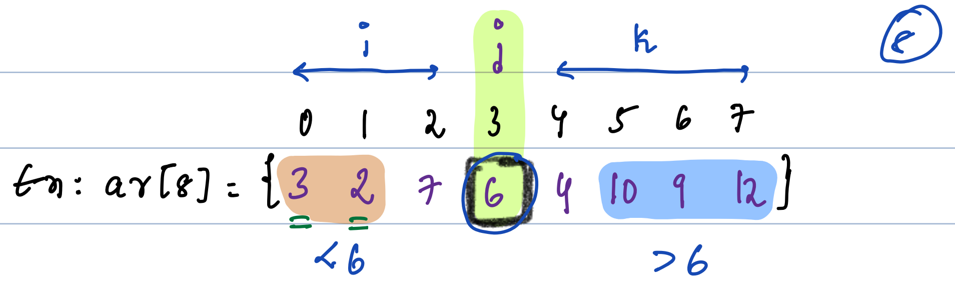
Idea1: Generate all triplets

For every triplet (i, j, k) : check $i < j < k$ && $ar[i] < ar[j] < ar[k]$

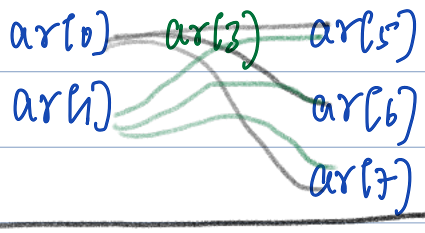
Estimated TC: $O(N^3)$. Code TODO

Idea2: We need to get triplet

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



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



If ar[3] is center ele : We have 6 triplets

Idea: For every ar[j]:

Step1: Iterate on left calculate no: of elements < ar[j] : cl

Step2: Iterate on right calculate no: of elements > ar[j] : cg

Triplets += cl * cg

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

Trace:

	0	1	2	3	4	5	6	7	
Ex: ar[8] =	{3	2	7	6	4	10	9	12}	
Count less :		0	2	2	2	5	5		
Count more :		6	3	3	3	1	1		
Triplets =		0	6	6	6	5	5		= 28

```
int triplets(int ar[], int N){ TODO

}
```

Note: In above approach we cannot apply carry forward.

1. For every ar[i]: Calculate no: of elements < ar[i] on left, data keeping changing hence we cannot apply cf

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 on i^{th} day: we can sell on any day {i+1, i+2, i+3, ... 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 9

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

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){
|
|
|
}
```