Inception- 6.0
Editorial

Check Bit Flip

Since there is only a single bit difference in the two numbers the answer is the XOR of the two bytes.

Example: $5 \oplus 7 = 2$ -> Answer

Its Wedding Season

It is always optimal to insert the larger flowers first. Observer that the largest flower adds up only once.

Every flower from second largest flower onwards adds up twice because we can keep the coming two flowers adjacent to the previous

largest flowers, one in clockwise and other in anticlockwise direction.

This way the first n/2-1 largest flowers adds up twice.

The resulting sum is our answer.

Win the archer

This problem basically ends to a simple observation where you need to just see if size of board is even blue will always win and for all adjacent squares of (1,1) , (1,2) is the one with required minimum value,

If size is odd answer will always be red. You can check for chessboards of size up to 10 and then come up with an answer.

Gray Similar Code

There are 2 observations that need to be made to find the pattern:

1. The codes formed will always be cyclic, i.e., if the first element is moved to the last position the code remains valid.
2. The patterns formed by bits in the pattern remains same for all valid codes.

Example:

For n=3:
Gray code is:
0 1 3 2 6 7 5 4
Representing numbers in binary:

0	**1**	**3**	**2**	**6**	**7**	**5**	**4**
0	0	0	0	1	1	1	1
0	0	1	1	1	1	0	0
0	1	1	0	0	1	1	0

The patterns formed by bits of position 0 : 01100110 is same relative to pattern formed by position 1 & 2 but if all bits at position 0 are swapped with bits at position 1 or 2 new valid patterns are formed.

So the total number of patterns are equal to the number of permutations of value of bits given to each pattern which is equal to factorial of n given and the result is multiplied by 2^n as the pattern can start from any number in the code as the code is cyclic.

That can be defined as a function $f(n) = n! * 2^n$

To solve the question in given constraints you have to use dynamic programming, pre-calculate all values of f(n) using $f(n) = f(n-1) * 2 * n$ and then answer testcases in O(1) time for each test case.

Magical Numbers

This problem can be solved using dynamic programming , here the states for dynamic programming can be (i,last,streak) where dp[i][last][streak] represents number of magical digits with i digits and whose last digit is ‘last’ and count of continuous streak of the last digit is ‘streak’.

Triplet Sum

First Solution:

Here the first solution strikes in our mind is try to check all combination but the time complexity for the same will be O(N ^ 3) which will not pass in given constraints.

Second Solution:

Assume I < j < k

We fix the middle element of triplet(let’s assume A[j]).

how many triplet you can create for fixed A[j]?

it will be total = (number of elements less than A[j] in left subarray) * (number of elements greater than A[j] in right subarray)

So A[j] will contribute to answer total * A[j].

You can also calculate contribution using same method for A[i], A[k] but here you have to fix one of them.

Now we fix A[i] so how many triplet we can create using fixed A[i], A[j].

It will be rightcnt = number of elements grater than A[j] in right subarray

So contribution for A[i] will be A[i] * rightcnt.

Same can be done for A[k] and we can calculate contribution of A[k].

Here if you carefully see number of triplet you can create for all different A[i] and A[k] is same.

So we can calculate the sum of all the elements which are less than A[j] in left subarray(leftsum) and

Sum of all elements which are greater than A[j] in right subarray(rightsum).

So at last

Total contribution for fixed A[j] will be

Contri = (leftsum * rightcnt) + (rightsum * leftcnt) + (leftcnt * rightcnt * A[j])

```
Ans = 0
For(j = 0; j < n; j++){
        leftcnt = 0, leftsum = 0;
        rightcnt = 0, rightsum = 0;
        for(k = j + 1; k < n; k++){
                if(A[k] > A[j]){
                        rightsum += A[k];
                        rightcnt++;
                }
        }
        For(I = j – 1; I >= 0; i--){
                If(A[j] > A[i]){
                        leftsum += A[i];
                        leftcnt++;
                }
        }
Ans += (leftsum * rightcnt) + (rightsum * leftcnt) + (leftcnt * rightcnt * A[j]);
        Ans %= mod;
}
```

The time complexity for this code will be O(N ^ 2).

Which is not sufficient to pass.

Third Solution:

Here we will do the same thing as done in 2nd solution.

But we will try to get leftsum, leftcnt and rightsum, rightcnt in constant time.

But how?

We will use elements of array as indexes and then create prefix sum of pair which will calculate the sum and cnt in O(1).

But here the order matters so we have to do update simultaneously which will cost us O(N).

So we have to find out a data structure which can do both of work in constant time.

Segment tree, fenewick tree have insetion/updation in o(logn) time.

So we will use two segment tree.

Reference : https://cp-algorithms.com/data_structures/segment_tree.html

Please go through basics otherwise you wont be able to understand solution.

One for leftsum and leftcnt and another one for rightsum, rightcnt.

One more thing you have to notice is A[i] < 10^9 which means we cant create a array of 1e9 size.

So here we will use data compression.

As you know the maximum number of elements are 2 * 10^5.

So we will sort the **all distinct number** and assign a new value to them starting from 0.

For ex:

[1,4, 5, 6, 8, 5]

The new array will be [0, 1, 2, 3, 4, 2]

And now we can use segment tree easily. And calculate the answer.

The time complexity for this algorithm will be O(NlogN).