**Inversions**

**Problem**

Given an array of size n. We need to count the number of indices j for each index i (0 <= i < n) following the condition

*j < i*

*a[j] > a[i]*

Note: Above condition is known as the property of inversion.

**Constraints**

*1 <= n <= 105*

Example input

5

4 1 3 5 2

Output

0 1 1 0 3

**Brute force approach**

Pseudo Code

for(int i=0; i<n; i++){

for(int j=i-1; j>=0; j--){

if(a[j] > a[i]){

ans++;

}

}

}

**Optimized Approach (Present Sir approach)**

1. After input of each element x, mark that element on the number line as “Present sir” by marking 1 at the index x.
2. Create a segment tree of sum.
3. For each x, just count the number of 1’s from x+1 till n.

**Code**

#include "bits/stdc++.h"

using namespace std;

#define int long long

const int N = 1e5+2, MOD = 1e9+7;

int tree[4\*N], a[N];

int query(int node, int st, int en, int l, int r){

if(st>r || en<l)

return 0;

if(l<=st && en<=r)

return tree[node];

int mid = (st + en)/2;

int q1 = query(2\*node, st, mid, l, r);

int q2 = query(2\*node+1, mid+1, en, l, r);

return q1 + q2;;

}

void update(int node, int st, int en, int idx, int val){

if(st == en){

tree[node] = val;

return;

}

int mid = (st+en)/2;

if(idx <= mid){

update(2\*node, st, mid, idx, val);

}

else

{

update(2\*node+1, mid+1, en, idx, val);

}

tree[node] = tree[2\*node] + tree[2\*node+1];

}

signed main()

{

for(int i=0; i<4\*N; i++){

tree[i] = 0;

}

int n;

cin >> n;

int x;

for(int i=0; i<n; i++){

cin >> x;

int ans = query(1,1,n,x,n);

cout << ans <<" ";

update(1,1,n,x,1);

}

return 0;

}