Rio and Inversion

Limits: 1s, 512 MB

One day Rio was returning home from his university and unfortunately lost his phone on the way. He asked his brother to buy him a luxurious smartphone. But his brother is not an easy person to please. He will give Rio an interesting array and some queries. If Rio can answer all the queries perfectly, only then his brother will buy him a smartphone.

The array consists of **N** elements. And the given queries will be of two types.
0 u v : If we swap the two elements in index **u** & **v**, what will be the number of inversions in the new array?
1 u v : If we remove the sub-array between the elements at index **u** and **v**(the elements at indexes **u** and **v** will also be removed), what will be the number of inversions in the newly formed array?

Here the number of inversion of an array **A** is the total number of pair of indexes **(i , j)** such that **i < j** and **A[i] > A[j]**. And of course **(0 <= i < j < N)**.

Rio was stuck in this problem for few days and could not figure out how to solve this. But he really needs a phone now. As one of his best friends, can you help him?

Input

First line contains a single element **N**. In the next line there will be **N (1 <= N <= 10^4)** integers. For all the values of **A_i the limit will be (0 <= A_i < 100)**. In the next line, there will be an integer **Q (1 <= Q <= 10^4)**. Then from next line there will be **Q** queries of the given types **0** or **1**.

All the queries will be zero based indexed and **(0 <= u,v < N)**.

Changes made in a query are temporary. They will not affect the queries that come after it. After each query, the array will revert back to its initial form.

Output

For each of the query, print the number of inversions in a single line.

Samples

Input	Output
12 3 8 2 5 12 1 7 14 9 5 3 8 10 1 8 11 1 2 5 1 6 9 1 3 8 1 6 6 0 8 11 0 2 5 0 6 9 0 3 8 0 6 6	10 13 12 5 23 26 26 26 30 27

Note that there is no update operation for the array elements. Array will always be fixed to its original configuration after a query has been performed.

```cpp
// https://toph.co/p/rio-and-inversion
#include<bits/stdc++.h>
using namespace std;
#define N tree[node] #define L tree[lson] #define R tree[rson]
int a[10002],dp[10002][10002];
vector<int>tree[4*10002];
void build(int node, int b, int e){
    if(b==e){ tree[node].push_back(a[b]); return; }
    int lson=(node<<1), rson=lson+1, m=(b+e)/2;
    build(lson,b,m);
    build(rson,m+1,e);
    merge(L.begin(),L.end(),R.begin(),R.end(),back_inserter(N));
}
int query(int node,int b,int e,int x,int y,int v,int c){
    if(x>y)return 0;
    if(b==x && e==y){
        if(c==0){
            int cnt = lower_bound(N.begin(),N.end(),v)-N.begin();
            return cnt;
        }else{
            int cnt = upper_bound(N.begin(),N.end(),v)-N.begin();
            return (e-b+1)-cnt;
        }
    }
    int lson=(node<<1), rson=lson+1, m=(b+e)/2;
    if(y<=m) return query(lson,b,m,x,y,v,c);
    else if(x>m) return query(rson,m+1,e,x,y,v,c);
    else return query(lson,b,m,x,m,v,c) + query(rson,m+1,e,m+1,y,v,c);
}
int main(){
    int n; scanf("%d",&n);
    for(int i=1; i<=n; i++)scanf("%d",&a[i]);

    build(1,1,n);

    int ans = 0;
    for(int i=1; i<=n; i++){
        ans += query(1,1,n,i+1,n,a[i],0);
    }

    dp[0][0] = dp[1][0] = dp[n+1][n] = ans;
    for(int j=1; j<=n; j++){
        dp[1][j] = dp[1][j-1] - query(1,1,n,j,n,a[j],0);
    }
    for(int i=n; i>=1; i--){
        dp[i][n] = dp[i+1][n] - query(1,1,n,1,i,a[i],1);
    }
    for(int i=2; i<n; i++){
        for(int j=n-1; j>=i; j--){
            dp[i][j] = dp[i-1][j] + dp[i][j+1] - dp[i-1][j+1];
            if(a[i-1]>a[j+1]) dp[i][j]++;
        }
    }
    int q; scanf("%d",&q);
    while(q--){
        int type; scanf("%d",&type);
        if(type==0){
            int x,y; scanf("%d%d",&x,&y); x++, y++; if(x>y)swap(x,y);
            int res = ans;
            if(a[x]!=a[y]){
                res -= query(1,1,n,x+1,n,a[x],0);
                res -= query(1,1,n,1,x-1,a[x],1);
                res -= query(1,1,n,y+1,n,a[y],0);
                res -= query(1,1,n,1,y-1,a[y],1);
                if(a[x]>a[y])res++;

                res += query(1,1,n,x+1,n,a[y],0);
                res += query(1,1,n,1,x-1,a[y],1);
                res += query(1,1,n,y+1,n,a[x],0);
                res += query(1,1,n,1,y-1,a[x],1);
                if(a[x]<a[y])res++;
            }
            printf("%d\n",res);
        }else{
            int x,y; scanf("%d%d",&x,&y); x++, y++; if(x>y)swap(x,y);
            int res = dp[x][y];
            printf("%d\n",res);
        }
    }
    return 0;
}
```