E. Anton and Permutation

time limit per test 4 seconds
memory limit per test 512 megabytes
input standard input
output standard output

Anton likes permutations, especially he likes to permute their elements. Note that a permutation of n elements is a sequence of numbers $\{a_1, a_2, ..., a_n\}$, in which every number from 1 to n appears exactly once.

One day Anton got a new permutation and started to play with it. He does the following operation q times: he takes two elements of the permutation and swaps these elements. After each operation he asks his friend Vanya, how many inversions there are in the new permutation. The number of inversions in a permutation is the number of distinct pairs (i, j)such that $1 \le i < j \le n$ and $a_i > a_j$.

Vanya is tired of answering Anton's silly questions. So he asked you to write a program that would answer these questions instead of him.

Initially Anton's permutation was $\{1, 2, ..., n\}$, that is $a_i = i$ for all i such that $1 \le i \le n$.

Input

The first line of the input contains two integers n and q $(1 \le n \le 200\,000, 1 \le q \le 50\,000)$ — the length of the permutation and the number of operations that Anton does.

Each of the following q lines of the input contains two integers l_i and r_i $(1 \le l_i, r_i \le n)$ — the indices of elements that Anton swaps during the i-th operation. Note that indices of elements that Anton swaps during the i-th operation can coincide. Elements in the permutation are numbered starting with one.

Output

Output q lines. The i-th line of the output is the number of inversions in the Anton's permutation after the i-th operation.

Examples

input Copy

```
5 4
4 5
2 4
2 5
2 2
```

output Copy

```
1
4
3
3
```

input Copy

```
2 1
2 1
```

output Copy

```
1
```

input Copy

```
6 7
1 4
3 5
2 3
3 3
3 6
2 1
5 1
```

output Copy

```
5
6
7
7
10
11
8
```

Note

Consider the first sample.

After the first Anton's operation the permutation will be $\{1, 2, 3, 5, 4\}$. There is only one inversion in it: $(4, 5)$.

After the second Anton's operation the permutation will be $\{1, 5, 3, 2, 4\}$. There are four inversions: $(2, 3)$, $(2, 4)$, $(2, 5)$ and $(3, 4)$.

After the third Anton's operation the permutation will be $\{1, 4, 3, 2, 5\}$. There are three inversions: $(2, 3)$, $(2, 4)$ and $(3, 4)$.

After the fourth Anton's operation the permutation doesn't change, so there are still three inversions.

At first observe that there is 0 inversions in the initial permutation.

Let's divide our queries in $\sqrt{q}$ blocks. Now learn how to answer all the queries in one block in $O(n+q+\sqrt{nq})$.

At first, let's divide our positions in the permutation in fixed and mobile positions. Mobile positions are all the positions that are changed in the current block. Fixed positions are the rest of the positions. Observe that the number of mobile positions is not more than $2 \cdot \sqrt{q}$.

Now all the inversions are divided into three types:

1. Inversions only between fixed positions;
2. Inversions only between mobile positions;
3. Inversions between fixed and mobile positions.

To keep inversions of the first type is the easiest of all: their number doesn't change. So we can precalcualte them in the beginning of the block. How can we do it? Let's remember the answer for the query that was directly before the beginning of the block (if the block starts with the first query, this number is equal to 0). This number is equal to the total number of inversions in the beginning of the block. To get the number of fixed inversions, we can just subtract from this number the number of inversions of the second and the third types.

It's also easy to calculate the number of inversions of the second type. In the beginning their number can be counted even using a naive algorithm in $O(\sqrt{q}^2) = O(q)$. How to keep this number? We can recalculate each time not all the inversions but only these ones that contain changed elements. Totally we can count them in $O(\sqrt{q} \cdot \sqrt{q}) = O(q)$ for a block.

It's a little bit harder to keep inversions of the third type. To count them we'll use the similar approach as with inversions of the second type. We'll also recount the number of inversions only for changed elements. So, we must learn how to count the number of inversions between fixed elements and some mobile element on the position x. What fixed elements make an inversion with it? It's obvious that these are the elements which are earlier than the x-th and wherein bigger than it (denote this query $countLower(x, p_x)$) or elements which are later than the x-th and wherein smaller than it (denote this query $countUpper(x, p_x)$). Here p_x denotes the x-th element of permutation. Observe that there are $O(\sqrt{q})$ such queries.

How can we calculate the answers for these queries quickly? At first note that we can count them offline, because fixed elements doesn't change and mobile elements are not counted in these queries. Consider how we can count answers for $countLower(x, y)$(for $countUpper(x, y)$ we can use the same approach). Let's sort all the $countLower$queries by non-decreasing x. Now if we have some structure that can add a value to an element and count a sum on a segment, we can easily do it. Let we added the fixed elements that stay earlier than x (it can be easily done because all the queries are sorted by non-decreasing x). So the answer is the sum on the segment $(y+1, n)$.

What data structure can we use? We can use sqrt-decomposition, because it takes $O(1)$for adding and $O(\sqrt{n})$ for sum query. Totally all the $O(\sqrt{q})$ $countLower$ and $countUpper$ queries in a block are processed in $O(n+\sqrt{nq})$.

Time complexity is $O(\sqrt{q} \cdot (n+q+\sqrt{nq})) = O(n\sqrt{q} + q\sqrt{q} + q\sqrt{n})$.

There also exists an $O((n+q) \cdot \log^2 n)$ solution. You can find it by yourself as an exercise. Such solution should be written carefully, otherwise it doesn't fit to the time limit or memory limit.

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace std;
using namespace __gnu_pbds;
typedef long long ll;
typedef pair<int,int> ii;
typedef tree<
        int,
        null_type,
        less<int>,
        rb_tree_tag,
        tree_order_statistics_node_update
> orderset;
const int maxn = 2e5+10;
orderset t[maxn<<2];
ll arr[maxn], n, q, ans = 0;
void build(int node, int l, int r) {
        for(int i=l; i<=r; i++) t[node].insert(arr[i]);
        if(l == r) return;
        int mid = l + r >> 1;
        build(node<<1, l, mid);
        build(node<<1|1, mid+1, r);
}
int query(int node, int l, int r, int i, int j, int val) {
        if(r < i || l > j || i > j) return 0;
        if(i <= l && r <= j) return t[node].order_of_key(val);
        int mid = l + r >> 1;
        return query(node<<1, l, mid, i, j, val) +
               query(node<<1|1, mid+1, r, i, j, val);
}
void update(int node, int l, int r, int x, int y) {
        bool a = (l <= x && x <= r);
        bool b = (l <= y && y <= r);
        if(a && !b) {
                t[node].erase(arr[x]);
                t[node].insert(arr[y]);
        } else if(b && !a) {
                t[node].erase(arr[y]);
                t[node].insert(arr[x]);
        } else if(!a && !b) return;
        if(r == l) return;
        int mid = l + r >> 1;
        update(node<<1, l, mid, x, y);
        update(node<<1|1, mid+1, r, x, y);
}
void SWP(int x, int y) {
        if(x == y) return;
        if(x > y) swap(x, y);
        int len = y-x-1;
        int a = query(1, 0, n-1, x+1, y-1, arr[y]);
        int b = query(1, 0, n-1, x+1, y-1, arr[x]);
        ans += 2*(a-b);
        update(1, 0, n-1, x, y);
        swap(arr[x], arr[y]);
        if(arr[x] > arr[y]) ans++;
        else ans--;
}
int main() {
        cin>>n>>q;
        for(int i=0; i<n; i++) arr[i] = i+1;
        build(1, 0, n-1);
        while(q--) {
                int x, y;
                cin>>x>>y; x--, y--;
                SWP(x,y);
                cout<<ans<<endl;
        }
    return 0;
}
```

```cpp
#include <bits/stdc++.h>
#define LL long long
using namespace std;

const int N=300000+5;
const int K = 500;
const int M = N/K + 3;
int a[N];
int s[N];
long long inv=0;
#define id(x) (x/K)*K

void build(){
    for (int i=0; i<N; i++){
        a[i]=i;
        s[i]=i;
    }
}
void swp(int l, int r){
    swap(a[l], a[r]);
    for (int i=id(l); i<id(l)+K; i++)   s[i]=a[i];
    for (int i=id(r); i<id(r)+K; i++)   s[i]=a[i];
    sort(s + id(l), s+id(l) + K);
    sort(s + id(r), s+id(r) + K);
}
long long query(int l, int r){
    if (l==r)   return inv;
    int x = a[l], y = a[r];
    int mn = min(a[l], a[r]);
    int mx = max(a[l], a[r]);

    int ans=0;

    if (id(l) == id(r)){
        for (int i=l; i<=r; i++)
            if (mn<a[i] && a[i]<mx) ans++;

        if (x<y)    inv+= 2*ans+1;
        else        inv-= 2*ans+1;
        return inv;
    }

    for (int i=id(l)+K; i<id(r); i+=K)
        ans += upper_bound(s+i, s+i+K, mx) - upper_bound(s+i, s+i+K, mn);

    for (int i=l; i<id(l)+K; i++)
        if (mn<a[i] && a[i]<mx) ans++;

    for (int i=id(r); i<=r; i++)
        if (mn<a[i] && a[i]<mx) ans++;

    if (x<y)    inv+= 2*ans+1;
    else        inv-= 2*ans+1;
    return inv;
}
int main (){
    int n, q; cin>>n>>q;

    build();

    for (int i=0; i<q; i++){
        int l, r; cin>>l>>r; l--; r--;
        if (l>r)    swap(l, r);
        cout<<query(l, r)<<endl;
        swp(l, r);
    }
    return 0;
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;
#define LL long long
#define pii pair<int,int>
const int Size = 200010;
const int SQRT = 450; /// Maximum no of block
int N, Q, A[Size], bit[SQRT][Size], BUCKET_SIZE;
LL inversion = 0;

void update (int bit[], int u, int v, int MAX) {
    while (u < MAX){
        bit[u] += v;
        u += (u & -u);
    }
}

int query (int bit[], int u) {
    int sum = 0;
    while (u > 0){
        sum += bit[u];
        u -= (u & -u);
    }
    return sum;
}

int findBlock(int u){
    int b = (u - 1)/BUCKET_SIZE + 1;
    return b;
}

/// Returns number of elements smaller than A[u] and A[v] from u+1 to v-1;
pii queryBit(int u, int v){
    int b1  = findBlock(u);
    int st1 = (b1-1)*BUCKET_SIZE + 1;
    int ed1 = b1*BUCKET_SIZE;
    ed1 = min(ed1, N);

    int b2 = findBlock(v);
    int st2 = (b2-1)*BUCKET_SIZE + 1;
    int ed2 = b2*BUCKET_SIZE;
    ed2 = min(ed2, N);

    int res1 = 0,res2 = 0;
    if(b1 == b2){
        for(int p=u+1; p<v; p++){
            if(A[p]<A[u]) res1++;
            if(A[p]<A[v]) res2++;
        }
        return make_pair(res1,res2);
    }

    for(int p=u+1; p<=ed1; p++){
        if(A[p]<A[u]) res1++;
        if(A[p]<A[v]) res2++;
    }

    for(int p=st2; p<v; p++){
        if(A[p]<A[u]) res1++;
        if(A[p]<A[v]) res2++;
    }

    for(int b=b1+1; b<b2; b++){
        res1 += query(bit[b], A[u]-1);
        res2 += query(bit[b], A[v]-1);
    }
    return make_pair(res1,res2);
}
```

```
68
69  void updateBit(int u, int v){
70      int b1 = findBlock(u);
71      int b2 = findBlock(v);
72      update(bit[b1], A[u], -1, Size);
73      update(bit[b2], A[v], -1, Size);
74      update(bit[b1], A[v],  1, Size);
75      update(bit[b2], A[u],  1, Size);
76  }
77
78  void solve(int u, int v){
79      pii rtt = queryBit(u, v);
80      int tot = v - u + 1 - 2;
81      int smlU = rtt.first;
82      int smlV = rtt.second;
83      int bigU = tot - smlU;
84      int bigV = tot - smlV;
85
86      inversion -= smlU;
87      inversion += bigU;
88
89      inversion -= bigV;
90      inversion += smlV;
91
92      if(A[u]>A[v]) inversion--;
93      else inversion++;
94
95      updateBit(u, v);
96
97      swap(A[u], A[v]);
98      printf("%lld\n",inversion);
99  }
100
101 int main(){
102         scanf("%d %d",&N,&Q);
103         BUCKET_SIZE = sqrt(N);
104         memset(bit,0,sizeof(bit));
105         for(int i=1; i<=N; i++){
106         A[i] = i;
107         int b = findBlock(i);
108         update(bit[b], A[i], 1, Size);
109         }
110
111         for(int i=0; i<Q; i++){
112         int u,v; scanf("%d %d",&u,&v);
113         if(u>v) swap(u,v);
114         if(u == v){
115             printf("%lld\n",inversion);
116             continue;
117         }
118         solve(u, v);
119         }
120         return 0;
121 }
```

```cpp
#include<bits/stdc++.h> using namespace std;
#define mx 200000ll #define ll long long
int n, sq, ar[mx+2], bu[mx+2];
ll ans=0;
void init() {
    sq=sqrt(n+.0)+1;
    for(int i=0; i<n; i++) bu[i] = ar[i];
    for(int i=0; i*sq<n; i++) {
        int l=i*sq; int r=min((i+1)*sq-1,n);
        sort(bu+l,bu+r+1);
    }
}
/// How many numbers are there from l to ri greater than v
int qug(int l,int r,int v) {
    int nl=l/sq, nr=r/sq; int res=0;
    if(nl==nr) {
        for(int i=l; i<=r; i++)if(ar[i]>v)res++;
        return res;
    }
    for(int i=l,j=(nl+1)*sq-1; i<=j; i++)if(ar[i]>v)res++;
    for(int i=nl+1; i<nr; i++) {
        int lo=i*sq; int hi=min((i+1)*sq-1,n); int sesh=hi;
        int ans=0;
        while(lo<=hi) {
            int mid=(lo+hi)/2;
            if(bu[mid]>v) {
                ans=(sesh-mid+1);
                hi=mid-1;
            } else lo=mid+1;
        }
        res+=ans;
    }
    for(int i=nr*sq; i<=r; i++)if(ar[i]>v)res++;
    return res;
}
/// How many numbers are there from l to ri less than v
int qul(int l,int r,int v) {
    int nl=l/sq, nr=r/sq; int res=0;
    if(nl==nr) {
        for(int i=l; i<=r; i++)if(ar[i]<v)res++;
        return res;
    }
    for(int i=l,j=(nl+1)*sq-1; i<=j; i++)if(ar[i]<v)res++;
    for(int i=nl+1; i<nr; i++) {
        int lo=i*sq; int hi=min((i+1)*sq-1,n); int suru=lo;
        int ans=0;
        while(lo<=hi) {
            int mid=(lo+hi)/2;
            if(bu[mid]<v) {
                ans=(mid-suru+1);
                lo=mid+1;
            } else hi=mid-1;
        }
        res+=ans;
    }
    for(int i=nr*sq; i<=r; i++)if(ar[i]<v)res++;
    return res;
}
void update(int in,int val){
    int s=in/sq; int l=s*sq; int r=min(n,(s+1)*sq-1);
    ar[in]=val;
    for(int i=l; i<=r; i++) bu[i]=ar[i];
    sort(bu+l,bu+r+1);
}
void  query(int x,int y) {
    if(x==y)return;
    ll sum = qul(x, y-1, ar[y]);
    sum -= qug(x, y-1, ar[y]);
    sum += qug(x+1, y, ar[x]);
    sum -= qul(x+1, y, ar[x]);

    if(ar[x]>ar[y]) sum++;
    else sum--;

    ans+=sum;
    swap(ar[x],ar[y]);
    update(x,ar[x]);
    update(y,ar[y]);
}
int main() {
    int q; scanf("%d%d",&n,&q);
    for(int i=0; i<n; i++)ar[i]=i;
    init();
    while(q--) {
        int x,y; scanf("%d%d",&x,&y); x--,y--;
        if(x>y)swap(x,y);
        query(x,y);
        printf("%lld\n",ans);
    }
    return 0;
}
```