C. Covered Points Count

time limit per test 3 seconds
memory limit per test 256 megabytes
input standard input
output standard output

You are given n segments on a coordinate line; each endpoint of every segment has integer coordinates. Some segments can degenerate to points. Segments can intersect with each other, be nested in each other or even coincide.

Your task is the following: for every $k \in [1..n]$, calculate the number of points with integer coordinates such that the number of segments that cover these points equals k. A segment with endpoints l_i and r_i covers point x if and only if $l_i \le x \le r_i$.

Input

The first line of the input contains one integer n ($1 \le n \le 2 \cdot 10^5$) — the number of segments.

The next n lines contain segments. The i-th line contains a pair of integers l_i, r_i ($0 \le l_i \le r_i \le 10^{18}$) — the endpoints of the i-th segment.

Output

Print n space separated integers $cnt_1, cnt_2, \ldots, cnt_n$, where cnt_i is equal to the number of points such that the number of segments that cover these points equals to i.

Examples

input Copy

```
3
0 3
1 3
3 8
```

output Copy

```
6 2 1
```

input Copy

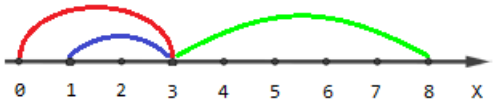
```
3
1 3
2 4
5 7
```

output Copy

```
5 2 0
```

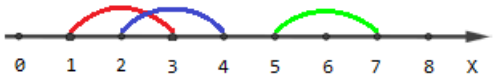
Note

The picture describing the first example:



Points with coordinates $[0, 4, 5, 6, 7, 8]$ are covered by one segment, points $[1, 2]$ are covered by two segments and point $[3]$ is covered by three segments.

The picture describing the second example:



Points $[1, 4, 5, 6, 7]$ are covered by one segment, points $[2, 3]$ are covered by two segments and there are no points covered by three segments.

```cpp
/// http://codeforces.com/contest/1000/problem/C
#include<bits/stdc++.h>
using namespace std;
#define ll long long
#define mx 200005
struct dt{ ll l,r; }st[mx];
ll n,ans[2*mx],a[2*mx],sum[2*mx];
map<ll,int>mp;
map<int,ll>vp;
set<ll>ss;
bool cmp(dt x,dt y){
    if(x.l!=y.l)return x.r < y.r;
    return x.l < y.l;
}
int main(){
    ios::sync_with_stdio(false);cin.tie(0);
    cin>>n;
    for(int i=1; i<=n; i++){
        ll l,r; cin>>l>>r;
        if(l>r)swap(l,r);
        st[i].l = l; st[i].r = r;
        ss.insert(l); ss.insert(r+1);
    }
    sort(st+1,st+n+1,cmp);
    int k = 0;
    for(auto it=ss.begin(); it!=ss.end(); it++){
        mp[(ll)*it] = ++k;
        vp[k] = (ll)*it;
    }
    for(int i=1; i<=n; i++){
        ll l = st[i].l;
        ll r = st[i].r;
        a[mp[l]] += 1;
        a[mp[r+1]] -= 1;
    }
    for(int i=1; i<=k; i++){
        sum[i] += sum[i-1]+a[i];
    }
    for(int i=1; i<k; i++){
        ans[sum[i]] += vp[i+1] - vp[i];
    }
    for(int i=1; i<=n; i++){
        if(i<n)cout << ans[i] << " ";
        else cout << ans[i] << endl;
    }
    return 0;
}
```

1000C - Covered Points Count

This problem with small coordinates can be solved using partial sums and some easy counting. Let's carry an array cnt, where cnt_i will be equal to the number of segments that cover the point with coordinate i. How to calculate cnt in $O(n + maxX)$?

For each segment (l_i, r_i) let's add $+1$ to cnt_{l_i} and -1 to cnt_{r_i+1}. Now build on this array prefix sums and notice that cnt_i equals the number of segments that cover the point with coordinate i. Then ans_i will be equal to $\sum_{j=0}^{maxX} cnt_j = i$. All the answers can be calculated in $O(maxX)$ in total. So the total complexity of this solution is $O(n + maxX)$.

But in our problem it is too slow to build an entire array cnt. So what should we do? It is obvious that if any coordinate j is not equals some l_i or some $r_i + 1$ then $cnt_i = cnt_{i-1}$. So we do not need carry all the positions explicitly. Let's carry all l_i and $r_i + 1$ in some logarithmic data structure or let's use the coordinate compression method.

The coordinate compression method allows us to transform the set of big sparse objects to the set of small compressed objects maintaining the relative order. In our problems let's make the following things: push all l_i and $r_i + 1$ in vector $cval$, sort this vector, keep only unique values and then use the position of elements in vector $cval$ instead of original value (any position can be found in $O(\log n)$ by binary search or standard methods as `lower_bound` in C++).

So the first part of the solution works in $O(n \log n)$. Answer can be calculated using almost the same approach as in solution to this problem with small coordinates. But now we know that between two adjacent elements $cval_i$ and $cval_{i+1}$ there is exactly $cval_{i+1} - cval_i$ points with answer equals to cnt_i. So if we will iterate over all pairs of the adjacent elements $cval_i$ and $cval_{i+1}$ and add $cval_{i+1} - cval_i$ to the ans_{cnt_i}, we will calculate all the answers in $O(n)$.

So the total complexity of the solution is $O(n \log n)$.