Java-排序数组-给定一组闭区间，其中部分区间存在交集

题目描述：

给定一组闭区间，其中部分区间存在交集。任意两个给定区间的交集，称为公共区间（如：[1,2],[2,3]的公共区间为[2,2]，[3,5],[3,6]的公共区间为[3,5]）。公共区间之间若存在交集，则需要合并（如：[1,3],[3,5]区间存在交集[3,3]，须合并为[1,5]）。按升序排列输出合并后的区间列表。

输入描述：

一组区间列表，

区间数为 N：

0 <= N <= 1000；

区间元素为 X：

-10000 <= X <= 10000。

输出描述：

升序排列的合并后区间列表

补充说明：

1、区间元素均为数字，不考虑字母、符号等异常输入。

2、单个区间认定为无公共区间。

示例 1

输入：

0 3

1 3

3 5

3 6

输出：

1 5

说明：

[0,3]和[1,3]的公共区间为[1,3]，[0,3]和[3,5]的公共区间为[3,3]，[0,3]和[3,6]的公共区间为[3,3]，[1,3]和[3,5]的公共区间为[3,3]，[1,3]和[3,6]的公共区间为[3,3]，[3,5]和[3,6]的公共区间为[3,5]，公共区间列表为[[1,3],[3,3],[3,5]]；[1,3],[3,3],[3,5]存在交集，须合并为[1,5]。

示例 2

输入：

0 3

1 4

4 7

5 8

输出：

1 3

4 4

5 7

说明：

示例 3

输入：

1 2

3 4

输出：
None
说明：
[1,2]和[3,4]无交集

```
import java.util.Scanner;
import java.util.*;
// 注意类名必须为 Main, 不要有任何 package xxx 信息
public class Main {
    public static class Node implements Comparable<Node>{
        public int x;
        public int y;
        public Node(int x,int y){
            this.x=x;
            this.y=y;
        }
        @Override
        public int compareTo(Node o) {
            return x-o.x;
        }
    }
    public static void main(String[] args) {
        Scanner in =new Scanner(System.in);
        ArrayList<Node> list= new ArrayList<>();
        while(in.hasNextInt()){
            int a=in.nextInt();
            int b=in.nextInt();
            list.add(new Node(a,b));
        }
        ArrayList<Node> same= findsame(list);
        ArrayList<Node> union=find(same);
        if(union.size()==0){
            System.out.println("None");
        }else {
            for(Node node:union){
                System.out.println(node.x+" "+ node.y);
            }
        }
    }
    public static ArrayList<Node> findsame(ArrayList<Node> list){
        Collections.sort(list);
        ArrayList<Node> ans=new ArrayList<>();
        int size=list.size();
        for(int i=0;i<size;i++){
            for(int j=i+1;j<size;j++){
```

```
                    if(list.get(j).x<=list.get(i).y){
                        int x=list.get(j).x;
                        int y=Math.min(list.get(i).y,list.get(j).y);
                        ans.add(new Node(x,y));
                    }
                }
            }
            return ans;
        }
        public static ArrayList<Node> find(ArrayList<Node> same){
            ArrayList<Node> union=new ArrayList<>();
            Collections.sort(same);
            for(int i=0;i<same.size();i++){
                int x=same.get(i).x;
                int y=same.get(i).y;
                int j=i+1;
                while(j<same.size()&&same.get(j).x<=y){
                    y=Math.max(y,same.get(j).y);
                    j++;
                }
                union.add(new Node(x,y));
                i=j-1;
            }
            return union;
        }
}
```