```java
class Solution {
    public int[][] merge(int[][] intervals) {
        Arrays.sort(intervals, Comparator.comparingInt(a -> a[0]));
        int st = intervals[0][0], ed = intervals[0][1];
        List<int[]> ans = new ArrayList<>();
        for (int i = 1; i < intervals.length; ++i) {
            int s = intervals[i][0], e = intervals[i][1];
            if (ed < s) {
                ans.add(new int[] {st, ed});
                st = s;
                ed = e;
            } else {
                ed = Math.max(ed, e);
            }
        }
        ans.add(new int[] {st, ed});
        return ans.toArray(new int[ans.size()][]);
    }
}
```

Java Auto

```java
class MyQueue {
    private Deque<Integer> stk1 = new ArrayDeque<>();
    private Deque<Integer> stk2 = new ArrayDeque<>();

    public MyQueue() {
    }

    public void push(int x) {
        stk1.push(x);
    }

    public int pop() {
        move();
        return stk2.pop();
    }

    public int peek() {
        move();
        return stk2.peek();
    }

    public boolean empty() {
        return stk1.isEmpty() && stk2.isEmpty();
    }

    private void move() {
        while (stk2.isEmpty()) {
            while (!stk1.isEmpty()) {
                stk2.push(stk1.pop());
            }
        }
```

```java
import java.util.Stack;

class DequeUsingStack<T> {
    private Stack<T> frontStack;
    private Stack<T> backStack;

    public DequeUsingStack() {
        frontStack = new Stack<>();
        backStack = new Stack<>();
    }


    public void addFront(T item) {
        frontStack.push(item);
    }

    public void addRear(T item) {
        backStack.push(item);
    }

    public T removeFront() {
        if (frontStack.isEmpty()) {
            while (!backStack.isEmpty()) {
                frontStack.push(backStack.pop());
            }
        }
        return frontStack.isEmpty() ? null : frontStack.pop();
    }

    public T removeRear() {
        if (backStack.isEmpty()) {
            while (!frontStack.isEmpty()) {
                backStack.push(frontStack.pop());
            }
        }
        return backStack.isEmpty() ? null : backStack.pop();
    }

    public boolean isEmpty() {
        return frontStack.isEmpty() && backStack.isEmpty();
```

```
    }
    return backStack.isEmpty() ? null : backStack.pop();
}

public boolean isEmpty() {
    return frontStack.isEmpty() && backStack.isEmpty();
}

public static void main(String[] args) {
    DequeUsingStack<Integer> deque = new DequeUsingStack<>();

    deque.addFront(1);
    deque.addRear(2);
    deque.addFront(3);
    deque.addRear(4);

    System.out.println(deque.removeFront()); // 3
    System.out.println(deque.removeRear());  // 4
    System.out.println(deque.removeFront()); // 1
    System.out.println(deque.removeRear());  // 2
}
```

```java
import java.util.Stack;

class MinStack {
    private Stack<Integer> mainStack;
    private Stack<Integer> minStack;

    public MinStack() {
        mainStack = new Stack<>();
        minStack = new Stack<>();
    }

    public void push(int val) {
        mainStack.push(val);
        // If minStack is empty or the new value is smaller/equal to the cur
        if (minStack.isEmpty() || val <= minStack.peek()) {
            minStack.push(val);
        }
    }

    public void pop() {
        if (mainStack.isEmpty()) return;

        int removed = mainStack.pop();
        if (removed == minStack.peek()) {
            minStack.pop();
        }
    }

    public int top() {
        return mainStack.peek();
    }

    public int getMin() {
        return minStack.peek();
    }

    public static void main(String[] args) {
        MinStack minStack = new MinStack();
        minStack.push(5);
```

```
}

public int top() {
    return mainStack.peek();
}

public int getMin() {
    return minStack.peek();
}

public static void main(String[] args) {
    MinStack minStack = new MinStack();
    minStack.push(5);
    minStack.push(2);
    minStack.push(8);
    minStack.push(1);
    System.out.println("Minimum: " + minStack.getMin());
    minStack.pop();
    System.out.println("Minimum: " + minStack.getMin());
}
```

```java
import java.util.Stack;

class MaxStack {
    private Stack<Integer> mainStack;
    private Stack<Integer> maxStack;

    public MaxStack() {
        mainStack = new Stack<>();
        maxStack = new Stack<>();
    }

    public void push(int val) {
        mainStack.push(val);
        if (maxStack.isEmpty() || val >= maxStack.peek()) {
            maxStack.push(val);
        }
    }

    public void pop() {
        if (mainStack.isEmpty()) return;
        int removed = mainStack.pop();
        if (removed == maxStack.peek()) {
            maxStack.pop();
        }
    }

    public int top() {
        return mainStack.peek();
    }

    public int getMax() {
        return maxStack.peek();
    }

    public static void main(String[] args) {
        MaxStack maxStack = new MaxStack();
        maxStack.push(3);
        maxStack.push(1);
        maxStack.push(5);
        maxStack.push(2);
```