华为OD机试 - 模拟消息队列（Java & JS & Python）

题目描述

让我们来模拟一个 消息队列 的运作，有一个发布者和若干消费者，发布者会在给定的时刻向消息队列发送消息，

- 若此时消息队列有消费者订阅，这个消息会被发送到订阅的消费者中优先级最高（输入中消费者按优先级升序排列）的一个；
- 若此时没有订阅的消费者，该消息被消息队列丢弃。

消费者则会在给定的时刻订阅消息队列或取消订阅。

- 当消息发送和订阅发生在同一时刻时，先处理订阅操作，即同一时刻订阅的消费者成为消息发送的候选。
- 当消息发送和取消订阅发生在同一时刻时，先处理取消订阅操作，即消息不会被发送到同一时刻取消订阅的消费者。

输入描述

输入为两行。

第一行为2N个正整数，代表发布者发送的N个消息的时刻和内容（为方便解折，消息内容也用正整数表示）。第一个数字是第一个消息的发送时刻，第二个数字是第一个消息的内容，以此类推。用例保证发送时刻不会重复，但注意消息并没有按照发送时刻排列。

第二行为2M个正整数，代表M个消费者订阅和取消订阅的时刻。第一个数字是第一个消费者订阅的时刻，第二个数字是第一个消费者取消订阅的时刻，以此类推。用例保证每个消费者的取消订阅时刻大于订阅时刻，消费者按优先级 升序排列 。

两行的数字都由空格分隔。N不超过100，M不超过10，每行的长度不超过1000字符。

输出描述

输出为M行，依次为M个消费者收到的消息内容，消息内容按收到的顺序排列，且由空格分隔；

若某个消费者没有收到任何消息，则对应的行输出-1.

用例

输入	2 22 1 11 4 44 5 55 3 33 1 7 2 3
输出	11 33 44 55 22
说明	消息11在1时刻到达，此时只有第一个消费者订阅，消息发送给它； 消息22在2时刻到达，此时两个消费者都订阅了，消息发送给优先级最高的第二个消费者； 消息33在时刻3到达，此时只有第一个消费者订阅，消息发送给它； 余下的消息按规则也是发送给第一个消费者。

输入	5 64 11 64 9 97 9 11 4 9
输出	97 64
说明	消息64在5时刻到达，此时只有第二个消费者订阅，消息发送给它； 消息97在9时刻到达，此时只有第一消费者订阅(因为第二个消费者刚好在9时刻取消订阅)，消息发送给它； 11时刻也到达了一个内容为64的消息，不过因为没有消费者订阅，消息被丢弃。

```java
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
import java.util.StringJoiner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int[] pubArr = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
    int[] subArr = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();

    getResult(pubArr, subArr);
  }

  public static void getResult(int[] pubArr, int[] subArr) {
    int n = pubArr.length;
    int m = subArr.length;

    // 将第一行输入两个一组，收集到：发布者数组中
    int[][] publisher = new int[n / 2][];
    for (int i = 0, k = 0; i < n; i += 2) {
      publisher[k++] = new int[] {pubArr[i], pubArr[i + 1]}; // [发布时刻，  发布内容]
    }

    // 将第二行输入两个一组，收集到：订阅者数组中
    int[][] subscriber = new int[m / 2][];
    for (int j = 0, k = 0; j < m; j += 2) {
      subscriber[k++] = new int[] {subArr[j], subArr[j + 1]}; // [订阅时刻，  取消订阅时刻]
    }

    // 按照发布时刻升序：发布者数组
    Arrays.sort(publisher, (a, b) -> a[0] - b[0]);

    // 为每一个订阅者构造一个的订阅内容集合
```

```
    ArrayList<ArrayList<Integer>> subContent = new ArrayList<>();
    for (int j = 0; j < subscriber.length; j++) subContent.add(new ArrayList<>());

    // 遍历发布者
    for (int[] pub : publisher) {
      int pubTime = pub[0]; // 发布时刻
      int pubContent = pub[1]; // 发布内容

      // 倒序遍历订阅者，因为后面的订阅者优先级更高，因此倒序可以实现高优先级的订阅者先匹配到发布者
      for (int j = subscriber.length - 1; j >= 0; j--) {
        int subTime = subscriber[j][0]; // 订阅时刻
        int unSubTime = subscriber[j][1]; // 取消订阅时刻

        // 如果 订阅时刻 <= 发布时刻 < 取消订阅时刻
        if (pubTime >= subTime && pubTime < unSubTime) {
          // 那么该订阅者就可以收到发布的内容
          subContent.get(j).add(pubContent);
          // 题目说，发布内容只会被最高优先级的订阅者收到
          break;
        }
      }
    }

    // 打印
    for (ArrayList<Integer> contents : subContent) {
      if (contents.size() == 0) {
        System.out.println("-1");
      } else {
        StringJoiner sj = new StringJoiner(" ");
        for (Integer content : contents) sj.add(content + "");
        System.out.println(sj.toString());
      }
    }
  }
}
```