题目描述

同一个数轴X上有两个点的集合A={A1, A2, …, Am}和B={B1, B2, …, Bn}，Ai和Bj均为正整数，A、B已经按照从小到大排好序，A、B均不为空，给定一个距离R(正整数)，列出同时满足如下条件的所有（Ai, Bj）数对：

1. Ai <= Bj
2. Ai, Bj之间的距离小于等于R
3. 在满足1,2的情况下,每个Ai只需输出距离最近的Bj
4. 输出结果按Ai从小到大的顺序排序

输入描述

第一行三个正整数m,n,R

第二行m个正整数,表示集合A

第三行n个正整数,表示集合B

输入限制：

1<=R<=100000, 1<=n,m<=100000, 1<=Ai,Bj<=1000000000

输出描述

每组数对输出一行Ai和Bj,以空格隔开

用例

输入	4 5 5 1 5 5 10 1 3 8 8 20
输出	1 1 5 8 5 8
说明	无

题目解析

本题数量级非常大，因此使用双重for是肯定会超时的。

我的解题思路是利用二分查找。

关于标准二分查找的实现可以参考Java的Arrays.binarySearch，其他语言实现可以看：

中关于binarySearch的具体实现，特别是其中关于有序插入位置的实现原理。

本题，我们只需要遍历每一个a[i]，然后通过二分查找去找他们在b中的位置 j：

- 如果 j >= 0，则说明b数组中有一个b[j] == a[i]，此时a[i] b[j]就是符合要求的组合
- 如果 j < 0，则说明b数组中没有b[j] == a[i]，此时 j 其实就是 - insertIdx - 1，其中insertIdx就是a[i]在b中的有序插入位置，因此b[insertIdx] > a[i]，如果 b[insertIdx] - a[i] <= r，那么a[i] b[insertIdx]就是符合要求的组合

上面算法的事件复杂度只有O(nlogn)，不会超时。

Java算法源码

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

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

    Integer[] tmp =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    int m = tmp[0];
    int n = tmp[1];
    int r = tmp[2];

    Integer[] a =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    Integer[] b =
        Arrays.stream(sc.nextLine().split(" ")).map(Integer::parseInt).toArray(Integer[]::new);

    getResult(r, a, b);
  }

  public static void getResult(int r, Integer[] a, Integer[] b) {
    for (int ai : a) {
      int j = Arrays.binarySearch(b, ai);

      // 如果在b数组中可以找到ai，则此时j就是ai在b数组中的位置，此时ai和bj是满足要求，且最接近的
      if (j >= 0) {
        System.out.println(ai + " " + b[j]);
      } else {
        // 如果在b数组中找不到ai，则此时j就是ai在b数组中有序插入位置-insertIdx-1，
        // 因此insertIdx = -j-1，此时b[insertIdx]满足大于ai，我们只需要检查b[insertIdx] - ai<=r即可
        int insertIdx = -j - 1;
        if (b[insertIdx] - ai <= r) System.out.println(ai + " " + b[insertIdx]);
      }
    }
  }
}
```

JS算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const { userInfo } = require("os");
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 3) {
    let [m, n, r] = lines[0].split(" ").map((ele) => parseInt(ele));
    let a = lines[1].split(" ").map(Number);
    let b = lines[2].split(" ").map(Number);
    getResult(a, b, r);
    lines.length = 0;
  }
});

function getResult(a, b, r) {
  for (let ai of a) {
    const j = binarySearch(b, ai);
    if (j >= 0) {
      console.log(`${ai} ${b[j]}`);
    } else {
      const insertIdx = -j - 1;
      if (b[insertIdx] - ai <= r) console.log(`${ai} ${b[insertIdx]}`);
    }
  }
}

```

```
function binarySearch(arr, target) {
  let low = 0;
  let high = arr.length - 1;

  while (low <= high) {
    const mid = (low + high) >> 1;
    const midVal = arr[mid];

    if (midVal > target) {
      high = mid - 1;
    } else if (midVal < target) {
      low = mid + 1;
    } else {
      return mid;
    }
  }

  return -low - 1;
}
```

Python算法源码

```python
# 输入获取
m, n, r = map(int, input().split())
a = list(map(int, input().split()))
b = list(map(int, input().split()))


def binarySearch(arr, target):
    low = 0
    high = len(arr) - 1

    while low <= high:
        mid = (low + high) >> 1
        midVal = arr[mid]

        if midVal > target:
            high = mid - 1
        elif midVal < target:
            low = mid + 1
        else:
            return mid

    return -low - 1


# 算法入口
def getResult():
    for ai in a:
        j = binarySearch(b, ai)
        if j >= 0:
            print(f"{ai} {b[j]}")
        else:
            insertIdx = -j - 1
            if b[insertIdx] - ai <= r:
                print(f"{ai} {b[insertIdx]}")


# 算法调用
getResult()
```