LESSON_14 参考试题

一、 选择题

1. 若对n个元素进行**插入排序**的过程中，共需进行(　)趟。

   A. n

   B. n+1

   C. n-1

   D. 2n

   【答案】C

2. （样题）排序算法的稳定性是指(　)。

   A. 相同元素在排序后的相对顺序保持不变

   B. 排序算法的性能稳定

   C. 排序算法对任意输入都有较好的效果

   D. 排序算法容易实现

   【答案】A

3. （2023年6月）排序算法是稳定的（Stable Sorting），就是指排序算法可以保证，在待排序数据中有两个相等记录的关键字 R 和 S（R 出现在 S 之前），在排序后的列表中 R 也一定在 S 前。下面关于排序稳定性的描述，正确的是(　)。

   A. 冒泡排序是不稳定的

   B. 插入排序是不稳定的

   C. 选择排序是不稳定的

   D. 以上都不正确

   【答案】C

二、 判断题

1. 插入排序，是指将数据按照一定的顺序一个一个的插入到有序的表中，最终得到的序列就是已经排序好的数据。

   【答案】正确

2. 使用插入排序算法对序列18, 23, 19, 9, 23, 15 进行升序排序, 第 3 趟排序后的结果为9, 18, 19, 23, 23,15。

   【答案】正确

3. （样题）冒泡排序是一种稳定的排序算法。

   【答案】正确

三、 编程题

1. 插入排序

【问题描述】

插入排序是一种非常常见且简单的排序算法。小 Z 是一名大一的新生，今天 H 老师刚刚在上课的时候讲了**插入排序**算法。

假设比较两个元素的时间为 O(1)，则插入排序可以 $O(n^2)$的时间复杂度完成长度为 n 的数组的排序。不妨假设这 n 个数字分别存储在 $a_1, a_2, \ldots\ldots, a_n$之中，则如下伪代码给出了插入排序算法的一种最简单的实现方式。

这下面是 C/C++ 的示范代码：

```
for (int i = 1; i <= n; i++)
   for (int j = i; j >= 2; j--)
     if (a[j] < a[j-1]){
          int t = a[j-1];
          a[j-1] = a[j];
          a[j] = t;
     }
```

为了帮助小 Z 更好的理解插入排序，H 老师留下了这么一道家庭作业：

H 老师给了一个长度为 n 的数组 a，数组下标从 1 开始，并且数组中的所有元素均为非负整数。小 Z 需要支持在数组 a 上的 Q 次操作，操作共两种，参数分别如下：

① x v：这是第一种操作，会将 a 的第 x 个元素，也就是 a_x的值，修改为 v。保证 $1 \le x \le n$，$1 \le v \le 10^9$。注意这种操作会改变数组的元素，修改得到的数组会被保留，也会影响后续的操作。

② x：这是第二种操作，假设 H 老师按照上面的伪代码对 a 数组进行排序，你需要告诉 H 老师原来 a 的第 x 个元素，也就是 a_x，在排序后的新数组所处的位置。保证 $1 \le x \le n$。注意这种操作不会改变数组的元素，排序后的数组不会被保留，也不会影响后续的操作。

H 老师不喜欢过多的修改，所以他保证类型 1 的操作次数不超过 5000。

小 Z 没有学过计算机竞赛，因此小 Z 并不会做这道题。他找到了你来帮助他解决这个问题。

【输入描述】

第一行，包含两个正整数 n, Q，表示数组长度和操作次数。

第二行，包含 n 个空格分隔的非负整数，其中第 i 个非负整数表示 a_i。

接下来 Q 行，每行 2~3 个正整数，表示一次操作，操作格式见【题目描述】。

【输出描述】

对于每一次类型为 2 的询问，输出一行一个正整数表示答案。

【样例输入】

```
3 4
3 2 1
2 3
1 3 2
2 2
2 3
```

【样例输出】

1

1

2

【提示】

对于所有测试数据，满足 $1 \le n \le 8000$，$1 \le Q \le 2 \times 10^5$，$1 \le x \le n$，$1 \le v, a_i \le 10^9$。

【参考代码】

```
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 8005;
int n, q;
//t[i]表示原来的第 i 个元素在排序后的新位置
int t[MAXN];
struct node {
    int val, id; //val 值和原位置
} a[MAXN];
//排序过程保证稳定性
bool cmp(node x, node y) {
    if (x.val != y.val) return x.val < y.val;
    return x.id < y.id;
}
int main() {
    scanf("%d%d", &n, &q);
    for (int i = 1; i <= n; i++) { //读入数组元素
        scanf("%d", &a[i].val);
        a[i].id = i;
    }
    sort(a + 1, a + n + 1, cmp); //升序排序
    //记录排序后每个元素的位置
    for (int i = 1; i <= n; i++){
        t[a[i].id] = i;
    }
```

```
    //q 次操作
    for (int i = 1; i <= q; i++) {
        int opt, x, v;
        scanf("%d", &opt);
        if (opt == 1) { //修改操作
            scanf("%d%d", &x, &v); //ax->v
            a[t[x]].val = v;
            //向前插入一趟插入排序
            for (int j = t[x]; j >= 2; j--)
                if (cmp(a[j], a[j - 1])) {
                    node k = a[j];
                    a[j] = a[j - 1];
                    a[j - 1] = k;
                }
            //向后一趟插入排序
            for (int j = t[x]; j < n; j++)
                if (cmp(a[j + 1], a[j])) {
                    node k = a[j];
                    a[j] = a[j + 1];
                    a[j + 1] = k;
                }
            //更新位置
            for (int i=1;i<=n;i++)
                t[a[i].id] = i;
        } else if(opt==2) { //输出原 ax 的新位置
            scanf("%d", &x);
            printf("%d\n", t[x]);
        }
    }
    return 0;
}
```