3.HEOI2016 排序

- 3.HEOI2016 排序
  - 题目简化
  - 题解
    - 思路
    - 代码

题目简化

给定一个长度为 n 的数组，m 次操作，每次操作对数组进行局部排序，**求第 q 位上的数**。

题解

思路

题目**只要求第 q 位上的数**，不管其他数。因此考虑二分第 q 位上的数 mid。为了简化其他不必要的排序，当 $val[i] \geq mid$ 此位置就为 1 反之则为 0。这样一来，这道题就变成了一个 01 序列排序，所以就可以用支持区间和查询与区间覆盖的线段树实现 $O(log\ n)$ 排序。

用线段树来维护区间和，代表着区间中 1 的个数，则降序排序将区间中的 1 放在最前，生序则反之。

每次二分，若 mid 的值为 1，则将 l 赋值为 $mid+1$，反之将 r 赋值为 mid，以求出最大的 mid，最终答案即为 $l-1$ （因为 l 是最大的满足 $check$ 的 $mid+1$，l 是不满足 $check$ 的）

代码

```
#include <bits/stdc++.h>
const int N = 300005;
using namespace std;
struct node
{
    int opt, l, r;
} q[N];
int n, m, st[N], val[N], lazy_tag[N << 2], tree[N << 2], k;
void push_up(int u) // 上传sum
{
    tree[u] = tree[u * 2] + tree[u * 2 + 1];
}
void push_down(int u, int l, int r) // 下传lazy_tag
{
    if (lazy_tag[u] < 0) // 无lazy_tag
        return;
    lazy_tag[u * 2] = lazy_tag[u * 2 + 1] = lazy_tag[u];
    tree[u * 2] = lazy_tag[u] * l;
    tree[u * 2 + 1] = lazy_tag[u] * r;
    lazy_tag[u] = -1; // 下移后移除
}
void init(int l = 1, int r = n, int u = 1)
{
    lazy_tag[u] = -1;
    if (l == r)
    {
        tree[u] = st[l];
        return;
    }
    int mid = (l + r) >> 1;
    init(l, mid, u * 2);
    init(mid + 1, r, u * 2 + 1);
    push_up(u);
}
int query(int L, int R, int l = 1, int r = n, int u = 1) // 查询L~R区间和
{
    if (l > R || r < L)
        return 0;
    if (l >= L && r <= R)
        return tree[u];
    int mid = (l + r) >> 1;
    push_down(u, mid - l + 1, r - mid);
    int ans = 0;
    ans += query(L, R, l, mid, u * 2);
    ans += query(L, R, mid + 1, r, u * 2 + 1);
    return ans;
}
void change(int L, int R, int val, int l = 1, int r = n, int u = 1) // L~R赋值为val
{
    if (l > R || r < L)
        return;
    if (l >= L && r <= R)
    {
        tree[u] = val * (r - l + 1);
        lazy_tag[u] = val;
        return;
    }
    int mid = (l + r) >> 1;
    push_down(u, mid - l + 1, r - mid);
    change(L, R, val, l, mid, u * 2);
    change(L, R, val, mid + 1, r, u * 2 + 1);
    push_up(u);
}
int check(int mid)
```

```
{
    for (int i = 1; i <= n; ++i)
        if (val[i] >= mid) // 按照01分类
            st[i] = 1;
        else
            st[i] = 0;
    init(); // 初始化
    for (int i = 1; i <= m; ++i)
    {
        int l = q[i].l, r = q[i].r;
        if (q[i].opt == 0)
        {
            // 升序
            int num1 = query(l, r);
            change(r - num1 + 1, r, 1);
            change(l, r - num1, 0);
        }
        else
        {
            //  降序
            int num1 = query(l, r);
            change(l, l + num1 - 1, 1);
            change(l + num1, r, 0);
        }
    }
    int tmp = query(k, k);
    return tmp;
}
int main()
{
#ifndef ONLINE_JUDGE
    freopen("3.in", "r", stdin);
#endif
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &val[i]);
    for (int i = 1; i <= m; ++i)
        scanf("%d%d%d", &q[i].opt, &q[i].l, &q[i].r);
    scanf("%d", &k);
    int L = 1, R = n;
    while (L < R)
    {
        int mid = (L + R) >> 1;
        if (check(mid))
            L = mid + 1;
        else
            R = mid;
    }
    printf("%d\n", L - 1);
    return 0;
}
```