

# 树套树练习

qq\_42371755 于 2019-12-06 16:14:06 发布

## 树套树讲题

T1: 二逼平衡树 (树套树)

[题解](#)

[代码](#)

[国家集训队]排队

[题解](#)

[代码](#)

[题解](#)

[代码](#)

k大数查询

[题解 \(by zxy\)](#)

[代码](#)

## T1：二逼平衡树（树套树）

[传送门](#)

### 题解

所谓树套树呢？就是把两棵不同的树套在一起，以达到意想不到的效果。

对于这道题，可以线段树套平衡树，也就是每个点维护一个平衡树，下面我来分别讲解一下每种操作。

查询k在区间内的排名

把区间分解成线段树上的点，查询k在每个点上的排名-1然后累加，最后在加上1。

查询区间内排名为k的值

先二分一个值，然后用拿到这个值的最大排名（我没有把相同的点合并为一个点），然后找到满最大排名大于等于k 的最小的mid，就是答案。

修改某一位值上的数值

暴力在线段树上单点修改，删去原来的数字后加入新给的数字。

查询k在区间内的前驱

先分解区间，然后取每个点找到的前驱中最大的。

查询k在区间内的后继

与上面的操作方式类似。

总时间复杂度 $O(n\log^3 n)$

## 代码

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstdlib>
4  #include <ctime>
5  #define ls son[num][0]
6  #define rs son[num][1]
7  using namespace std;
8  const int maxn = 5e4+5;
9  const int INF = 2147483647;
10 int readint(){
11     int x=0,f=1;char s=getchar();
12     #define sc (s=getchar())
13     while(s<'0' || s>'9'){
14         if(s=='-')
15             f=-1;
16         sc;
17     }
```

```

18     while(s>='0'&&s<='9'){
19         x=(x<<3)+(x<<1)+(s^48);
20         sc;
21     }
22     #undef sc
23     return x*f;
24 }
25 int len,n,q;
26 int a[maxn<<5];
27 int siz[maxn<<5],key[maxn<<5],son[maxn<<5][2],cnt[maxn<<5],rd[maxn<<5];
28 struct treap{
29     int rt;
30     void push_up(int num){
31         siz[num]=siz[ls]+siz[rs]+cnt[num];
32     }
33     void rotate(int &x,int d){
34         int root=son[x][d^1];
35         son[x][d^1]=son[root][d];
36         son[root][d]=x;
37         push_up(x);
38         push_up(root);
39         x=root;
40     }
41     void insert(int &num,int x){
42         if(!num){
43             num=++len;
44             siz[num]=cnt[num]=1;
45             key[num]=x;
46             rd[num]=rand();
47             return;
48         }
49         if(key[num]==x){

```

```

50         cnt[num]++;
51         siz[num]++;
52         return;
53     }
54     int d=(x>key[num]);
55     insert(son[num][d],x);
56     if(rd[num]<rd[son[num][d]])
57         rotate(num,d^1);
58     push_up(num);
59 }
60 void deleted(int &num,int x){
61     if(!num)
62         return;
63     if(x!=key[num])
64         deleted(son[num][x>key[num]],x);
65     else{
66         if(!ls&&!rs){
67             cnt[num]--;
68             siz[num]--;
69             if(cnt[num]==0)
70                 num=0;
71         }
72         else if(ls&&!rs){
73             rotate(num,1);
74             deleted(rs,x);
75         }
76         else if(!ls&&rs){
77             rotate(num,0);
78             deleted(ls,x);
79         }
80         else{
81             int d=rd[ls]>rd[rs];

```

```
82         rotate(num,d);
83         deleted(son[num][d],x);
84     }
85 }
86 push_up(num);
87 }
88 int get_rank(int num,int x){
89     if(!num)
90         return 0;
91     if(key[num]==x)
92         return siz[ls];
93     if(key[num]<x)
94         return siz[ls]+cnt[num]+get_rank(rs,x);
95     return get_rank(ls,x);
96 }
97 int find(int num,int x){
98     if(!num)
99         return 0;
100     if(siz[ls]>=x)
101         return find(ls,x);
102     else if(siz[ls]+cnt[num]<x)
103         return find(rs,x-cnt[num]-siz[ls]);
104     else
105         return key[num];
106 }
107 int getpre(int num,int x){
108     if(!num)
109         return -INF;
110     if(key[num]>=x)
111         return getpre(ls,x);
112     else
113         return max(key[num],getpre(rs,x));
```

```

114     }
115     int getback(int num,int x){
116         if(!num)
117             return INF;
118         if(key[num]<=x)
119             return getback(rs,x);
120         else
121             return min(key[num],getback(ls,x));
122     }
123     int com(int &num,int x){
124         int pre=getpre(num,x),nxt=getback(num,x);
125         int f1=x-pre,f2=nxt-x;
126         if(f1>f2){
127             deleted(num,nxt);
128             return f2;
129         }
130         else{
131             deleted(num,pre);
132             return f1;
133         }
134     }
135 }tree[maxn<<3];
136 namespace segmentTree{
137     void build(int num,int l,int r){
138         for(int i=l;i<=r;i++)
139             tree[num].insert(tree[num].rt,a[i]);
140         if(l==r)
141             return;
142         int mid=(l+r)>>1;
143         build(num<<1,l,mid);
144         build(num<<1|1,mid+1,r);
145     }

```

```

146 void update(int num,int l,int r,int pos,int val){
147     if(pos<l||r<pos)
148         return;
149     tree[num].deleted(tree[num].rt,a[pos]);
150     tree[num].insert(tree[num].rt,val);
151     if(l==r)
152         return;
153     int mid=(l+r)>>1;
154     update(num<<1,l,mid,pos,val);
155     update(num<<1|1,mid+1,r,pos,val);
156 }
157 // zhen de rank = queryrank() + 1
158 int queryrank(int num,int l,int r,int goal_l,int goal_r,int val){
159     if(goal_l>r||goal_r<l)
160         return 0;
161     if(l>=goal_l&&r<=goal_r)
162         return tree[num].get_rank(tree[num].rt,val);
163     int mid=(l+r)>>1;
164     return queryrank(num<<1,l,mid,goal_l,goal_r,val)+queryrank(num<<1|1,mid+1,r,goal_l,goal_r,val);
165 }
166 int queryfind(int goal_l,int goal_r,int rank){
167     int l=0,r=1e8,ans;
168     while(l<=r){
169         int mid=(l+r)>>1;
170         if(queryrank(1,1,n,goal_l,goal_r,mid)+1<=rank)
171             ans=mid,l=mid+1;
172         else
173             r=mid-1;
174     }
175     return ans;
176 }
177 int querypre(int num,int l,int r,int goal_l,int goal_r,int val){

```

```

178         if(goal_r<l||goal_l>r)
179             return -INF;
180         if(goal_l<=l&&r<=goal_r)
181             return tree[num].getpre(tree[num].rt,val);
182         int mid=(l+r)>>1;
183         return max(querypre(num<<1,l,mid,goal_l,goal_r,val),querypre(num<<1|1,mid+1,r,goal_l,goal_r,val));
184     }
185     int queryback(int num,int l,int r,int goal_l,int goal_r,int val){
186         if(goal_r<l||goal_l>r)
187             return INF;
188         if(goal_l<=l&&r<=goal_r)
189             return tree[num].getback(tree[num].rt,val);
190         int mid=(l+r)>>1;
191         return min(queryback(num<<1,l,mid,goal_l,goal_r,val),queryback(num<<1|1,mid+1,r,goal_l,goal_r,val));
192     }
193 }
194 int main (){
195     srand(20050301);
196     n=readint(),q=readint();
197     for(int i=1;i<=n;i++)
198         a[i]=readint();
199     segmentTree::build(1,1,n);
200     while(q--){
201         int op=readint();
202         if(op==3){
203             int pos=readint(),val=readint();
204             segmentTree::update(1,1,n,pos,val);
205             a[pos]=val;
206         }
207         else{
208             int l=readint(),r=readint(),k=readint();
209             if(op==1)

```



```

210         printf("%d\n", segmentTree::queryrank(1, 1, n, l, r, k) + 1);
211     if (op == 2)
212         printf("%d\n", segmentTree::queryfind(1, r, k));
213     if (op == 4)
214         printf("%d\n", segmentTree::querypre(1, 1, n, l, r, k));
215     if (op == 5)
216         printf("%d\n", segmentTree::queryback(1, 1, n, l, r, k));
217     }
218 }
219 return 0;
220 }

```

## [国家集训队]排队

传送门

### 题解

还需要考虑  $a_l, a_r$  之间的大小关系对全局逆序对的贡献

若  $a_l < a_r$

那么交换后会产生 1 的贡献

若  $a_l > a_r$

那么交换后会产生 -1 的贡献

所以我们需要这样一种数据结构，可以支持：

单点插入

单点删除

区间询问严格的比  $val$  小的元素有多少个

区间询问严格的比  $val$  大的元素有多少个

我们可以使用：分块或者树套树

我这里使用的是树套树(即线段树套 Treap)

## 代码

```
1  #include <cstdio>
2  #include <iostream>
3  #include <cstdlib>
4  #include <ctime>
5  #define ls son[num][0]
6  #define rs son[num][1]
7  using namespace std;
8  const int maxn = 5e4+5;
9  const int INF = 2147483647;
10 int readint(){
11     int x=0,f=1;char s=getchar();
12     #define sc (s=getchar())
13     while(s<'0' || s>'9'){
14         if(s=='-')
15             f=-1;
16         sc;
17     }
18     while(s>='0' && s<='9'){
19         x=(x<<3)+(x<<1)+(s^48);
20         sc;
21     }
22     #undef sc
23     return x*f;
24 }
25 int len,n,q;
26 int a[maxn<<5];
27 int siz[maxn<<5],key[maxn<<5],son[maxn<<5][2],cnt[maxn<<5],rd[maxn<<5];
28 struct treap{
29     int rt;
30     void push_up(int num){
```

```
31     siz[num]=siz[ls]+siz[rs]+cnt[num];
32 }
33 void rotate(int &x,int d){
34     int root=son[x][d^1];
35     son[x][d^1]=son[root][d];
36     son[root][d]=x;
37     push_up(x);
38     push_up(root);
39     x=root;
40 }
41 void insert(int &num,int x){
42     if(!num){
43         num=++len;
44         siz[num]=cnt[num]=1;
45         key[num]=x;
46         rd[num]=rand();
47         return;
48     }
49     if(key[num]==x){
50         cnt[num]++;
51         siz[num]++;
52         return;
53     }
54     int d=(x>key[num]);
55     insert(son[num][d],x);
56     if(rd[num]<rd[son[num][d]])
57         rotate(num,d^1);
58     push_up(num);
59 }
60 void deleted(int &num,int x){
61     if(!num)
62         return;
```

```

63     if(x!=key[num])
64         deleted(son[num][x>key[num]],x);
65     else{
66         if(!ls&&!rs){
67             cnt[num]--;
68             siz[num]--;
69             if(cnt[num]==0)
70                 num=0;
71         }
72         else if(ls&&!rs){
73             rotate(num,1);
74             deleted(rs,x);
75         }
76         else if(!ls&&rs){
77             rotate(num,0);
78             deleted(ls,x);
79         }
80         else{
81             int d=rd[ls]>rd[rs];
82             rotate(num,d);
83             deleted(son[num][d],x);
84         }
85     }
86     push_up(num);
87 }
88 int get_rank(int num,int x){
89     if(!num)
90         return 0;
91     if(key[num]==x)
92         return siz[ls];
93     if(key[num]<x)
94         return siz[ls]+cnt[num]+get_rank(rs,x);

```

```
95         return get_rank(ls,x);
96     }
97     int get_Rank(int num,int x){
98         if(!num)
99             return 0;
100         if(key[num]==x)
101             return siz[rs];
102         if(key[num]>x)
103             return siz[rs]+cnt[num]+get_Rank(ls,x);
104         return get_Rank(rs,x);
105     }
106     int find(int num,int x){
107         if(!num)
108             return 0;
109         if(siz[ls]>=x)
110             return find(ls,x);
111         else if(siz[ls]+cnt[num]<x)
112             return find(rs,x-cnt[num]-siz[ls]);
113         else
114             return key[num];
115     }
116     int getpre(int num,int x){
117         if(!num)
118             return -INF;
119         if(key[num]>=x)
120             return getpre(ls,x);
121         else
122             return max(key[num],getpre(rs,x));
123     }
124     int getback(int num,int x){
125         if(!num)
126             return INF;
```

```

127         if(key[num]<=x)
128             return getback(rs,x);
129         else
130             return min(key[num],getback(ls,x));
131     }
132     int com(int &num,int x){
133         int pre=getpre(num,x),nxt=getback(num,x);
134         int f1=x-pre,f2=nxt-x;
135         if(f1>f2){
136             deleted(num,nxt);
137             return f2;
138         }
139         else{
140             deleted(num,pre);
141             return f1;
142         }
143     }
144 }tree[maxn<<3];
145 namespace segmentTree{
146     void build(int num,int l,int r){
147         for(int i=l;i<=r;i++)
148             tree[num].insert(tree[num].rt,a[i]);
149         if(l==r)
150             return;
151         int mid=(l+r)>>1;
152         build(num<<1,l,mid);
153         build(num<<1|1,mid+1,r);
154     }
155     void update(int num,int l,int r,int pos,int val){
156         if(pos<l||r<pos)
157             return;
158         tree[num].deleted(tree[num].rt,a[pos]);

```

```

159     tree[num].insert(tree[num].rt,val);
160     if(l==r)
161         return;
162     int mid=(l+r)>>1;
163     update(num<<1,l,mid,pos,val);
164     update(num<<1|1,mid+1,r,pos,val);
165 }
166 // zhen de rank = queryrank() + 1
167 int queryrank(int num,int l,int r,int goal_l,int goal_r,int val){
168     if(goal_l>r||goal_r<l)
169         return 0;
170     if(l>=goal_l&&r<=goal_r)
171         return tree[num].get_rank(tree[num].rt,val);
172     int mid=(l+r)>>1;
173     return queryrank(num<<1,l,mid,goal_l,goal_r,val)+queryrank(num<<1|1,mid+1,r,goal_l,goal_r,val);
174 }
175 int queryRank(int num,int l,int r,int goal_l,int goal_r,int val){
176     if(goal_l>r||goal_r<l)
177         return 0;
178     if(l>=goal_l&&r<=goal_r)
179         return tree[num].get_Rank(tree[num].rt,val);
180     int mid=(l+r)>>1;
181     return queryRank(num<<1,l,mid,goal_l,goal_r,val)+queryRank(num<<1|1,mid+1,r,goal_l,goal_r,val);
182 }
183 int queryfind(int goal_l,int goal_r,int rank){
184     int l=0,r=1e8,ans;
185     while(l<=r){
186         int mid=(l+r)>>1;
187         if(queryrank(1,1,n,goal_l,goal_r,mid)+1<=rank)
188             ans=mid,l=mid+1;
189         else
190             r=mid-1;

```

```

191     }
192     return ans;
193 }
194 }
195 int main (){
196     //freopen("own.out", "w", stdout);
197     //freopen("testdata.in", "r", stdin);
198     //freopen("own.out", "w", stdout);
199     srand(20050301);
200     int n=readint();
201     for(int i=1;i<=n;i++)
202         a[i]=readint();
203     segmentTree::build(1,1,n);
204     int ans=0;
205     for(int i=1;i<=n;i++)
206         ans+=segmentTree::queryRank(1,1,n,1,i-1,a[i]);
207     cout<<ans<<endl;
208     int q=readint();
209     while(q--){
210         int x=readint(),y=readint();
211         int t1=a[x],t2=a[y];
212         ans-=segmentTree::queryRank(1,1,n,1,x-1,a[x]);
213         ans-=segmentTree::queryRank(1,1,n,x+1,n,a[x]);
214         ans+=segmentTree::queryRank(1,1,n,1,x-1,a[y]);
215         ans+=segmentTree::queryRank(1,1,n,x+1,n,a[y]);
216         segmentTree::update(1,1,n,x,t2);
217         ans-=segmentTree::queryRank(1,1,n,1,y-1,a[y]);
218         ans-=segmentTree::queryRank(1,1,n,y+1,n,a[y]);
219         ans+=segmentTree::queryRank(1,1,n,1,y-1,a[x]);
220         ans+=segmentTree::queryRank(1,1,n,y+1,n,a[x]);
221         segmentTree::update(1,1,n,y,t1);
222         swap(a[x],a[y]);

```



```
223         printf("%d\n",ans);
224     }
225     return 0;
226 }
```

## 动态逆序对

传送门

## 题解

按道理来说应该是树套树都能过的，况且可能我的写法太丑了，luogu只有70分

## 代码

我的：

```
1  #pragma GCC optimize(fast)
2  #include <cstdio>
3  #include <iostream>
4  #include <cstdlib>
5  #include <ctime>
6  #define ls son[num][0]
7  #define rs son[num][1]
8  using namespace std;
9  const int maxn = 1e5+5;
10 const int INF = 2147483647;
11 typedef long long LL;
12 int readint(){
13     int x=0,f=1;char s=getchar();
14     #define sc (s=getchar())
15     while(s<'0' || s>'9'){
```

```

16         if(s=='-')
17             f=-1;
18         sc;
19     }
20     while(s>='0'&&s<='9'){
21         x=(x<<3)+(x<<1)+(s^48);
22         sc;
23     }
24     #undef sc
25     return x*f;
26 }
27 int len,n,q;
28 int a[maxn<<5];
29 int key[maxn<<5],rd[maxn<<5];
30 int son[maxn<<5][2];
31 int siz[maxn<<5],cnt[maxn<<5];
32 struct treap{
33     int rt;
34     void push_up(int num){
35         siz[num]=siz[ls]+siz[rs]+cnt[num];
36     }
37     void rotate(int &x,int d){
38         int root=son[x][d^1];
39         son[x][d^1]=son[root][d];
40         son[root][d]=x;
41         push_up(x);
42         push_up(root);
43         x=root;
44     }
45     void insert(int &num,int x){
46         if(!num){
47             num=++len;

```

```

48         siz[num]=cnt[num]=1;
49         key[num]=x;
50         rd[num]=rand();
51         return;
52     }
53     if(key[num]==x){
54         cnt[num]++;
55         siz[num]++;
56         return;
57     }
58     int d=(x>key[num]);
59     insert(son[num][d],x);
60     if(rd[num]<rd[son[num][d]])
61         rotate(num,d^1);
62     push_up(num);
63 }
64 void deleted(int &num,int x){
65     if(!num)
66         return;
67     if(x!=key[num])
68         deleted(son[num][x>key[num]],x);
69     else{
70         if(!ls&&!rs){
71             cnt[num]--;
72             siz[num]--;
73             if(cnt[num]==0)
74                 num=0;
75         }
76         else if(ls&&!rs){
77             rotate(num,1);
78             deleted(rs,x);
79         }

```

```

80         else if(!ls&&rs){
81             rotate(num,0);
82             deleted(ls,x);
83         }
84         else{
85             int d=rd[ls]>rd[rs];
86             rotate(num,d);
87             deleted(son[num][d],x);
88         }
89     }
90     push_up(num);
91 }
92 LL get_rank(int num,int x){
93     if(!num)
94         return 0;
95     if(key[num]==x)
96         return siz[ls];
97     if(key[num]<x)
98         return siz[ls]+cnt[num]+get_rank(rs,x);
99     return get_rank(ls,x);
100 }
101 LL get_Rank(int num,int x){
102     if(!num)
103         return 0;
104     if(key[num]==x)
105         return siz[rs];
106     if(key[num]>x)
107         return siz[rs]+cnt[num]+get_Rank(ls,x);
108     return get_Rank(rs,x);
109 }
110 int find(int num,int x){
111     if(!num)

```

```

112         return 0;
113     if(siz[ls]>=x)
114         return find(ls,x);
115     else if(siz[ls]+cnt[num]<x)
116         return find(rs,x-cnt[num]-siz[ls]);
117     else
118         return key[num];
119 }
120 int getpre(int num,int x){
121     if(!num)
122         return -INF;
123     if(key[num]>=x)
124         return getpre(ls,x);
125     else
126         return max(key[num],getpre(rs,x));
127 }
128 int getback(int num,int x){
129     if(!num)
130         return INF;
131     if(key[num]<=x)
132         return getback(rs,x);
133     else
134         return min(key[num],getback(ls,x));
135 }
136 int com(int &num,int x){
137     int pre=getpre(num,x),nxt=getback(num,x);
138     int f1=x-pre,f2=nxt-x;
139     if(f1>f2){
140         deleted(num,nxt);
141         return f2;
142     }
143     else{

```

```

144         deleted(num,pre);
145         return f1;
146     }
147 }
148 }tree[maxn<<3];
149 namespace segmentTree{
150     void build(int num,int l,int r){
151         for(int i=l;i<=r;i++)
152             tree[num].insert(tree[num].rt,a[i]);
153         if(l==r)
154             return;
155         int mid=(l+r)>>1;
156         build(num<<1,l,mid);
157         build(num<<1|1,mid+1,r);
158     }
159     void update(int num,int l,int r,int pos){
160         if(pos<l||r<pos)
161             return;
162         tree[num].deleted(tree[num].rt,a[pos]);
163         // tree[num].insert(tree[num].rt,val);
164         if(l==r)
165             return;
166         int mid=(l+r)>>1;
167         update(num<<1,l,mid,pos);
168         update(num<<1|1,mid+1,r,pos);
169     }
170     // zhen de rank = queryrank() + 1
171     LL queryrank(int num,int l,int r,int goal_l,int goal_r,int val){
172         if(goal_l>r||goal_r<l)
173             return 0;
174         if(l>=goal_l&&r<=goal_r)
175             return tree[num].get_rank(tree[num].rt,val);

```

```

176         int mid=(l+r)>>1;
177         return queryrank(num<<1,l,mid,goal_l,goal_r,val)+queryrank(num<<1|1,mid+1,r,goal_l,goal_r,val);
178     }
179 LL queryRank(int num,int l,int r,int goal_l,int goal_r,int val){
180     if(goal_l>r||goal_r<l)
181         return 0;
182     if(l>=goal_l&&r<=goal_r)
183         return tree[num].get_Rank(tree[num].rt,val);
184     int mid=(l+r)>>1;
185     return queryRank(num<<1,l,mid,goal_l,goal_r,val)+queryRank(num<<1|1,mid+1,r,goal_l,goal_r,val);
186 }
187 int queryfind(int goal_l,int goal_r,int rank){
188     int l=0,r=1e8,ans;
189     while(l<=r){
190         int mid=(l+r)>>1;
191         if(queryrank(1,1,n,goal_l,goal_r,mid)+1<=rank)
192             ans=mid,l=mid+1;
193         else
194             r=mid-1;
195     }
196     return ans;
197 }
198 }
199 int pos[maxn];
200 int main (){
201     //freopen("own.out","w",stdout);
202     //freopen("testdata.in","r",stdin);
203     //freopen("own.out","w",stdout);
204     srand(20050301);
205     int n=readint(),q=readint();
206     for(int i=1;i<=n;i++)
207         a[i]=readint(),pos[a[i]]=i;

```

```

208     segmentTree::build(1,1,n);
209     LL ans=0;
210     for(int i=1;i<=n;i++)
211         ans+=segmentTree::queryRank(1,1,n,1,i-1,a[i]);
212     // cout<<ans<<endl;
213     while(q--){
214         //cout<<"Fuck"<<endl;
215         printf("%lld\n",ans);
216         int x=readint();
217         ans-=segmentTree::queryRank(1,1,n,1,pos[x]-1,x);
218         ans-=segmentTree::queryrank(1,1,n,pos[x]+1,n,x);
219         segmentTree::update(1,1,n,pos[x]);
220     }
221     return 0;
222 }

```

线段树套树状数组（这个代码能过）

```

1  #include<iostream>
2  #include<algorithm>
3  #include<climits>
4  #include<cstdio>
5  using namespace std;
6  struct node
7  {
8      int lson;
9      int rson;
10     int siz;
11 }tre[1000000];
12 void read(int &x)
13 {
14     x=0;

```



```
15     int f=1;
16     char c=getchar();
17     while('0'>c||c>'9')
18     {
19         if(c=='-')
20             f=-1;
21         c=getchar();
22     }
23     while('0'<=c&& c<='9')
24     {
25         x=(x<<3)+(x<<1)+c-'0';
26         c=getchar();
27     }
28     x*=f;
29 }
30 void write(long long x)
31 {
32     if(x<0)
33     {
34         putchar('-');
35         write(-x);
36         return;
37     }
38     if(x>9)
39         write(x/10);
40     putchar(x%10+'0');
41 }
42 int n,m;
43 int tot;
44 int cnt;
45 long long ans;
46 int q;
```

```

47 int p[200005];
48 int a[100005];
49 int b[200005];
50 int rt[200005];
51 int lowbit(int x)
52 {
53     return x&(-x);
54 }
55 void insert(int &x,int pos,int val,int l=1,int r=cnt)
56 {
57     if(!x)
58         x=++tot;
59     tre[x].siz+=val;
60     if(l==r)
61         return;
62     int mid=(l+r)>>1;
63     if(pos<=mid)
64         insert(tre[x].lson,pos,val,l,mid);
65     else
66         insert(tre[x].rson,pos,val,mid+1,r);
67 }
68 void modify(int x,int v)
69 {
70     int k=a[x];
71     for(int i=x;i<=n;i+=lowbit(i))
72         insert(rt[i],k,v);
73 }
74 int solve_rank(int x,int val,int l=1,int r=cnt)
75 {
76     if(!x)
77         return 0;
78     if(l==r)

```

```

79     {
80         return 0;
81     }
82     int mid=(l+r)>>1;
83     if(val<=mid)
84         return solve_rank(tre[x].lson,val,l,mid);
85     else
86         return tre[tre[x].lson].siz+solve_rank(tre[x].rson,val,mid+1,r);
87 }
88 int Solve_rank(int l,int r,int val)
89 {
90     if(l>r)
91         return 0;
92     int s=0, x=val;
93     for(int i=l-1;i>0;i-=lowbit(i))
94         s-=solve_rank(rt[i],x);
95     for(int i=r;i>0;i-=lowbit(i))
96         s+=solve_rank(rt[i],x);
97     return s;
98 }
99 int solve_size(int l,int r)
100 {
101     if(l>r)
102         return 0;
103     int s=0;
104     for(int i=l-1;i>0;i-=lowbit(i))
105         s-=tre[rt[i]].siz;
106     for(int i=r;i>0;i-=lowbit(i))
107         s+=tre[rt[i]].siz;
108     return s;
109 }
110 void efsort(int l,int r)

```

```

111 {
112     if(l == r) return ;
113     int m = (l+r)>>1;
114     efsort(l,m);
115     efsort(m+1,r);
116     int k=l,t1=l,t2=m+1;
117     while(t1<=m&& t2<=r)
118     {
119         if(b[t1]<=b[t2])
120         {
121             p[k++]=b[t1++];
122         }
123         else
124         {
125             ans+=m-t1+1;
126             p[k++]=b[t2++];
127         }
128     }
129     while(t1<=m)
130         p[k++]=b[t1++];
131     while(t2<=r)
132         p[k++]=b[t2++];
133     for(int i=l;i<=r;i++)
134         b[i]=p[i];
135 }
136 int main()
137 {
138     read(n);
139     read(m);
140     for(int i=1;i<=n;i++)
141     {
142         read(a[i]);

```

```

143     a[i]*=2;
144     b[i]=a[i];
145 }
146 cnt=2*n;
147 efsort(1,n);
148 for(int i=1;i<=n;i++)
149     b[a[i]]=i;
150 for(int i=1;i<=n;i++)
151     modify(i,1);
152 for(int i=1;i<=m;i++)
153 {
154     //cout<<"ans:"<<ans<<'\\n';
155     write(ans);
156     putchar('\\n');
157     read(q);
158     q = b[q<<1];
159     int t1=solve_size(1,q-1);
160     ans=ans-(t1-Solve_rank(1,q-1,a[q]+1));
161     //cout << "LEFT: ";
162     ans=ans-Solve_rank(q+1,n,a[q]-1);
163     modify(q,-1);
164 }
165 return 0;
166 }

```

## k大数查询

[传送门](#)

题解 (by zxy)

先%一波zxy

这道题的思路特别巧妙，树套树不一定要用区间线段树套权值线段树，还可以反过来套。

我们维护一个动态开点的权值线段树，每个点代表权值 $[l, r]$ 在整个区间的出现情况，套上一个动态开点的区间线段树，操作1对权值线段树单点修改，然后对每个点的 $[a, b]$ 区间修改。

操作2 22先算右子树根的 $[a, b]$ 和，如果答案在里面，找右子树，否则减掉之后找左子树，最后跑到叶子就得到了答案，时间复杂度 $O(n\log^2)$ ，空间复杂度 $O(n\log^2)$

这道题是需要卡常的，sum需要开unsigned int，输入的c cc要开long long long long。

## 代码

```
1  #pragma GCC optimize(2)
2  #include <cstdio>
3  #define uint unsigned int
4  #define LL long long
5  const int MAXN = 400005;
6  int read()
7  {
8      int x=0,flag=1;char c;
9      while((c=getchar())<'0' || c>'9') if(c=='-') flag=-1;
10     while(c>='0' && c<='9') x=(x<<3)+(x<<1)+(c^48),c=getchar();
11     return x*flag;
12 }
13 int n,m,a,b,rt,cnt1,cnt2;LL c;
14 struct node1
15 {
16     int ls,rs,lazy;
17     uint sum;
18 }tr[MAXN*40];
19 struct node2
20 {
21     int ls,rs,rt;
```

```

22 }Tr[MAXN*20];
23 void modify(int x,int l,int r,int v)
24 {
25     tr[x].sum+=v*(r-l+1);
26     tr[x].lazy+=v;
27 }
28 void up(int x)
29 {
30     tr[x].sum=tr[tr[x].ls].sum+tr[tr[x].rs].sum;
31 }
32 void down(int x,int l,int r)
33 {
34     if(!tr[x].lazy) return ;
35     int mid=(l+r)>>1;
36     if(!tr[x].ls) tr[x].ls=++cnt1;
37     if(!tr[x].rs) tr[x].rs=++cnt1;
38     modify(tr[x].ls,l,mid,tr[x].lazy);
39     modify(tr[x].rs,mid+1,r,tr[x].lazy);
40     tr[x].lazy=0;
41 }
42 void add(int &x,int l,int r,int L,int R)
43 {
44     if(l>R || L>r) return ;
45     if(!x) x=++cnt1;
46     if(L<=l && r<=R)
47     {
48         modify(x,l,r,1);
49         return ;
50     }
51     down(x,l,r);
52     int mid=(l+r)>>1;
53     add(tr[x].ls,l,mid,L,R);

```

```

54     add(tr[x].rs,mid+1,r,L,R);
55     up(x);
56 }
57 uint query(int x,int l,int r,int L,int R)
58 {
59     if(!x || l>R || L>r) return 0;
60     if(L<=l && r<=R)
61         return tr[x].sum;
62     down(x,l,r);
63     int mid=(l+r)>>1;
64     return query(tr[x].ls,l,mid,L,R)+query(tr[x].rs,mid+1,r,L,R);
65 }
66 void Modify(int &x,int l,int r,int id)
67 {
68     if(!x) x=++cnt2;
69     add(Tr[x].rt,1,n,a,b);
70     if(l==r) return ;
71     int mid=(l+r)>>1;
72     if(mid>=id)
73         Modify(Tr[x].ls,l,mid,id);
74     else
75         Modify(Tr[x].rs,mid+1,r,id);
76 }
77 int Query(int &x,int l,int r,int s)
78 {
79     if(!x) x=++cnt2;
80     if(l==r) return l;
81     int mid=(l+r)>>1;uint t=query(Tr[Tr[x].rs].rt,1,n,a,b);
82     //printf("%d %d %d %d %d\n",l,r,t,a,b);
83     if(t>=s)
84         return Query(Tr[x].rs,mid+1,r,s);
85     return Query(Tr[x].ls,l,mid,s-t);

```



```
86 | }
87 | int main()
88 | {
89 |     n=read();m=read();
90 |     while(m--)
91 |     {
92 |         int op=read();a=read();b=read();scanf("%lld",&c);
93 |         if(op==1)
94 |         {
95 |             Modify(rt,-n,n,c);
96 |         }
97 |         if(op==2)
98 |         {
99 |             printf("%d\n",Query(rt,-n,n,c));
100 |         }
101 |     }
102 | }
```