数据结构

单调栈

栈内元素始终单调（栈顶 **更** 满足要求）

> 如更大，更小

```
vector<int> stp;
stack<int> st;
for(int i=1;i<=n;++i)
{
        while(st.size() && check(a[i])) st.pop();
        stp.push_back(st.size() ? st.top() : -1);
        st.push(a[i]);
}
```

左侧第一个比自己小/大元素

```
// 左侧小
vector<int> ans;
stack<int> st;
for(int i=1;i<=n;++i)
{
        while(st.size() && st.top() >= a[i])    st.pop();
        ans.push_back(st.size() ? st.top() : -1);
        st.push(a[i]);
}
// 左侧大
for(int i=1;i<=n;++i)
{
        while(st.size() && st.top() <= a[i])    st.pop();
        ans.push_back(st.size() ? st.top() : -1);
        st.push(a[i]);
}
```

单调队列

滑动窗口

```
vector<int> ans;
deque<int> dq;

// 宽度为 k 的窗口中的最大值
for(int i=1;i<=n;++i)
{
    while(dq.size() && dq.front() <= i-k)       dq.pop_front();
    while(dq.size() && a[dq.back()] <= a[i])    dq.pop_back();
    dq.push_back(i);
    if(i >= k)  ans.push_back(a[dq.front()]);
}
// 宽度为 k 的窗口中的最小值
for(int i=1;i<=n;++i)
{
    while(dq.size() && dq.front() <= i-k)       dq.pop_front();
    while(dq.size() && a[dq.back()] <= a[i])    dq.pop_back();
    dq.push_back(i);
    if(i >= k)  ans.push_back(a[dq.front()]);
}
```

并查集

路径压缩

```
int rt[N];

int find(int x)// 找到根节点
{
        return rt[x] = (rt[x] == x ? x : find(rt[x]));
}

void merge(int x,int y)// 合并两联通块
{
        rt[find(x)] = find(y);
}
for(int i=1;i<=N;++i)   rt[i] = i;
```

树状数组

单点修改维护区间和

```
int n;
int a[N], t[N];
int lowbit(int z)
{
    return z&-z;
}
void update(int p,int z)// 单点修改
{
    for(int i=p;i<=n;i+=lowbit(i))  t[i] += z;
}
void query(int l,int r)// 区间查询
{
    int res1 = 0, res2 = 0;
    for(int i=l-1;i>=1;i-=lowbit(i))    res1 += t[i];
        for(int i=r;i>=1;i-=lowbit(i))          res2 += t[i];
    return res2-res1;
}
```

区间修改维护区间和

维护差分

其实没啥用了, 我会线段树

$\sum_{i=1}^{r} a_i = \sum_{i=1}^{r}(r+1)d_i - \sum_{i=1}^{r} i \cdot d_i$

```
int n;
vector<int> a(N), t(N),ti(N);
int lowbit(int i)
{
        return i&-i;
}
void update(int p,int z)
{
        for(int i=p;i<=n;i+=lowbit(i))  t[i] += z, ti[i] += p*z;
}
int query(int l,int r)
{
        int res1 = 0, res2;
        for(int i=l-1;i>=1;i-=lowbit(i))        res1 += (l)*t[i] - ti[i];
        for(int i=r;i>=1;i-=lowbit(i))          res2 += (r+1)*t[i] - ti[i];
        return res2 - res1;
}
```

维护逆序对

按出现顺序依次加入树状数组，每次可得当前小于（大于）等于自己的数目（也就是前面比自己小/大的数目），计算即可得逆序对。

```
int n,ans;
vector<int> t(N)
vector<int> dis,;// dis离散化

int get(int x)// 寻找离散化下标
{
        return lower_bound(dis.begin(),dis.end(),x)-dis.begin()+1;
        // 下标存入树状数组，保证下标从1开始
}

int lowbit(int z)
{
        return z&-z;
}
void update(int p)
{
        for(int i=p;i<=n;i+=lowbit(i))  t[i] ++;
}
int query(int p)
{
        int res = 0;
        for(int i=p;i>=1;i-=lowbit(i))  res += t[i];
        return res;
}

void func(void)
{
        cin >> n;
        vector<int> a(n);
        for(int i=0;i<n;++i)    cin >> a[i];
        dis = a;
        sort(dis.begin(),dis.end());
        dis.erase(unique(dis.begin(),dis.end()),dis.end());
        for(int i=0;i<n;++i)
        {
                ans += i-query(get(a[i]));
                update(get(a[i]));
        }
        cout << ans << '\n';
}
```

线段树

加法线段树

```
int n;
vector<int> a(N), t(N<<2), lz(N<<2);

void update(int z,int be,int ed,int p)
{
        t[p] += (ed-be+1) * z;
        lz[p] += z;
}

void push_up(int p)
{
        t[p] = t[p<<1] + t[p<<1|1];
}

void push_down(int be,int ed,int p)
{
        int mid = (be + ed) >> 1;
        update(lz[p],be,mid,p<<1),update(lz[p],mid+1,ed,p<<1|1);
        lz[p] = 0;
}

void build_tree(int be=1,int ed=n,int p=1)
{
        if(be == ed)
        {
                t[p] = a[be];
                return ;
        }
        int mid = (be + ed) >> 1;
        build_tree(be,mid,p<<1), build_tree(mid+1,ed,p<<1|1);
        push_up(p);
}

void put(int l,int r,int z,int be=1,int ed=n,int p=1)
{
        if(l <= be && ed <= r)
        {
                update(z,be,ed,p);
                return ;
```

```
        }
        push_down(be,ed,p);
        int mid = (be + ed) >> 1;
        if(l <= mid)    put(l,r,z,be,mid,p<<1);
        if(mid+1 <= r)  put(l,r,z,mid+1,ed,p<<1|1);
        push_up(p);
}

int query(int l,int r,int be=1,int ed=n,int p=1)
{
        if(l <= be && ed <= r)  return t[p];
        push_down(be,ed,p);
        int mid = (be + ed) >> 1, res = 0;
        if(l <= mid)    res += query(l,r,be,mid,p<<1);
        if(mid+1 <= r)  res += query(l,r,mid+1,ed,p<<1|1);
        return res;
}
```

权值线段树

```
int n,q;
vector<int> t(N<<2);

void push_up(int p)
{
        t[p] = t[p<<1] + t[p<<1|1];
}

void add(int x,int be=1,int ed=n,int p=1)
{
        if(be == ed)
        {
                t[p] ++;
                return ;
        }
        int mid = (be+ed) >> 1;
        if(x <= mid)    add(x,be,mid,p<<1);
        else    add(x,mid+1,ed,p<<1|1);
        push_up(p);
}

int query_cnt(int l,int r,int be=1,int ed=n,int p=1)
{
        if(l <= be && ed <= r)  return t[p];
        int mid = (be+ed) >> 1,cnt = 0;
        if(l <= mid)    cnt += query_cnt(l,r,be,mid,p<<1);
        if(mid+1 <= r)  cnt += query_cnt(l,r,mid+1,ed,p<<1|1);
        return cnt;
}

int query_k(int k,int be=1,int ed=n,int p=1)
{
        if(be == ed)    return be;
        int mid = (be+ed) >> 1, lsum = t[p<<1];
        if(lsum >= k)   return query_k(k,be,mid,p<<1);
        else return query_k(k-lsum,mid+1,ed,p<<1|1);
}
```

主席树

```
struct node
{
        int cnt, ls, rs;
};

int n,q,idx,mx;
vector<int> a(N), dis, root(N);
vector<node> t(N<<5);
int get(int x)
{
        return (lower_bound(dis.begin(),dis.end(),x) - dis.begin()+1);
}

void insert(int &p,int pre,int val,int be=1,int ed=mx)
{
         p = ++ idx;
         t[p] = t[pre];
         t[p].cnt ++;
         if(be == ed)   return ;
         int mid = (be+ed) >> 1;
         if(val <= mid) insert(t[p].ls,t[pre].ls,val,be,mid);
         else   insert(t[p].rs,t[pre].rs,val,mid+1,ed);
}

int query(int lo,int ro,int k,int be=1,int ed=mx)
{
        if(be == ed)    return be;
        int mid = (be + ed) >> 1, lcnt = t[t[ro].ls].cnt - t[t[lo].ls].cnt;
        if(k <= lcnt)   return query(t[lo].ls,t[ro].ls,k,be,mid);
        else    return query(t[lo].rs,t[ro].rs,k-lcnt,mid+1,ed);

}

void func(void)
{
        cin >> n >> q;
        for(int i=1;i<=n;++i)   cin >> a[i];
        for(int i=1;i<=n;++i)   dis.push_back(a[i]);
        sort(dis.begin(),dis.end());
        dis.erase(unique(dis.begin(),dis.end()),dis.end());
        mx = dis.size();
```

```
        for(int i=1;i<=n;++i)   insert(root[i],root[i-1],get(a[i]));
        while(q--)
        {
                int l,r,k;
                cin >> l >> r >> k;
                cout << dis[query(root[l-1],root[r],k)-1] << '\n';
        }
}
```