H 小刻觉得应该来个平衡树

出题人:buns out

考察点:STL容器,pb_ds容器,线段树,二分

> **宣传一下我的模板库:**
>
> Release ACM模板 · zxyDEDIRE/ACM-Code-Library (github.com)
>
> zxyDEDIRE/ACM-Code-Library at QAQ (github.com)
>
> 有LaTeX源码也有PDF版本

解法非常非常多，STL的各种容器都可以，pb_ds里边的容器也都可以，或者上平衡树，或者线段树+二分。

大家的解法也都分成丰富，很值得夸奖。虽说有人搜到了pb_ds库里边的平衡树，不过记得及时掌握，pb_ds库里边也有很多有趣的小东西。也有人直接手抄了Sply平衡树，码量确实很大，sply等等平衡树并不建议现在学，STL以及pb_ds大部分容器都掌握才是最有价值的。

不过可恶的是，居然有人用vector卡过去了，没有想到vector insert这么快，没有专门造数据卡。

解法一:
直接调用pb_ds的平衡树，或者直接上平衡树(平衡树代码就不贴出来了，太长了)。

pb_ds只要会用就可以了。

```
#include <ext/pb_ds/assoc_container.hpp>
#include<bits/stdc++.h>
using namespace __gnu_cxx;
using namespace __gnu_pbds;
using namespace std;
#define endl "\n"
using pii=pair<int,int>;
const int INF=1e9+7;

typedef tree<pii, null_type, less<pii>, rb_tree_tag,
tree_order_statistics_node_update> Tree;
Tree t;


template<typename T>
int getRank(T x)
{
    return t.order_of_key(x)+1;
}

template<typename T>
T getVal(int k)
{
    auto it=t.find_by_order(k-1);
```

```
    if(it!=t.end())
        return *it;
    else
        return {INF,0};
}
int get(int x){
    auto v=getVal<pii>(x);
    return v.first;
}

void solve()
{
    int n,x;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>x;
        t.insert({x,i});
        int mid_l = ceil(1.0*i/3);
        int mid_r = ceil(2.0*i/3);
        cout<<get(mid_l)<<" "<<get(mid_r)<<endl;
    }
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);cout.tie(nullptr);
        solve();
    return 0;
}
```

解法二：
各种容器，我用的是multiset。

第一个容器维护中位数之前的数据，第二个容器维护中位数之后的数据。如果说第一个容器中最大的数大于 第二个容器中最小的数，那么将第一个容器中最大的数移动到第二个容器。

```
#include<bits/stdc++.h>
using namespace std;
#define endl "\n"
#define pp(x) array<int,x>
using ull=unsigned long long;
using ll=long long;
using pii=pair<int,int>;
using pdd=pair<double,double>;
const int dx[]={0,0,1,-1,1,-1,1,-1};
const int dy[]={1,-1,0,0,1,-1,-1,1};
const int mod=998244353;
const int inf=0x3f3f3f3f;
const int INF=1e9+7;
const int maxn=1e6+100;
struct node{
    multiset<int,greater<int>>sa;
    multiset<int>sb;
    int lena=0,lenb=0;
    void insert(int x){
```

```
        sa.insert(x);
        lena++;
        bal();
    }
    void bal(){
        while(sa.size()&&sb.size()&&(*sa.begin())>(*sb.begin())){
            sb.insert(*sa.begin());
            sa.erase(sa.begin());
            lena--;
            lenb++;
        }
    }
    void out(int x){
        while(lena<x){
            sa.insert(*sb.begin());
            sb.erase(sb.begin());
            lena++;
            lenb--;
        }
        while(lena>x)
        {
            sb.insert(*sa.begin());
            sa.erase(sa.begin());
            lena--;
            lenb++;
        }
        cout<<*sa.begin()<<" ";
    }
}sa,sb;
int n;
void solve()
{
    cin>>n;
    int len=0;
    while(n--)
    {
        int x;
        cin>>x;
        len++;
        int mid_1=ceil(1.0*len/3);
        int mid_2=ceil(2.0*len/3);
        sa.insert(x);
        sb.insert(x);
        sa.out(mid_1);
        sb.out(mid_2);
        cout<<endl;
    }
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);cout.tie(nullptr);
        solve();
    return 0;
}
/*
```

```
*/
```

解法三：
离散化+线段树+二分，也有人用这种方法过了，NICE。

这种做法具体就是离散化后，建一棵线段树，线段树维护从小到大各个值的出现次数。然后就可以二分，找到具体是哪一个值。

```
#include<bits/stdc++.h>
using namespace std;
#define endl "\n"
const int maxn=1e6+100;
struct Seg{
    int l,r,val;
    Seg operator+(const Seg&a)const{
        Seg now;
        now.l=l;now.r=a.r;
        now.val=val+a.val;
        return now;
    }
}t[maxn<<2];
int p[maxn];
int n,len;
void build(int rt,int l,int r){
    t[rt]={l,r,0};
    if(l==r)return;
    int mid=(l+r)>>1;
    build(rt<<1,l,mid);
    build(rt<<1|1,mid+1,r);
}
void add(int rt,int l){
    if(t[rt].l==t[rt].r){
        t[rt].val++;
        return ;
    }
    int mid=(t[rt].l+t[rt].r)>>1;
    if(l<=mid)add(rt<<1,l);
    else add(rt<<1|1,l);
    t[rt]=t[rt<<1]+t[rt<<1|1];
}
int query(int rt,int l,int r){
    if(l<=t[rt].l&&t[rt].r<=r)
        return t[rt].val;
    int mid=(t[rt].l+t[rt].r)>>1;
    int ans=0;
    if(l<=mid)ans+=query(rt<<1,l,r);
    if(r>mid)ans+=query(rt<<1|1,l,r);
    return ans;
}
int q(int x)
{
    int l=1,r=len,ans=r;
    auto check=[&](int mid)->bool{
        return query(1,1,mid)>=x;
    };
```

```
    while(l<r-1){
        int mid=(l+r)>>1;
        if(check(mid))ans=mid,r=mid;
        else l=mid;
    }
    while(ans>1&&check(ans-1))ans--;
    return ans;
}
void solve()
{
    cin>>n;
    vector<int>v(n);
    for(int i=0;i<n;i++)
        cin>>p[i],v[i]=p[i];
    sort(v.begin(),v.end());
    v.erase(unique(v.begin(),v.end()),v.end());
    len=v.size();
    build(1,1,len);
    for(int i=0;i<n;i++)
    {
        int x=p[i];
        x=lower_bound(v.begin(),v.end(),x)-v.begin()+1;
        add(1,x);
        int mid_1=ceil(1.0*(i+1)/3);
        int mid_2=ceil(2.0*(i+1)/3);
        int pos1=q(mid_1)-1;
        int pos2=q(mid_2)-1;
        cout<<v[pos1]<<" "<<v[pos2]<<"\n";
    }
}
signed main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);cout.tie(nullptr);
        solve();
    return 0;
}
```