Persistent Segment Trees

Q) Given an array answer queries of the form l r k -> find the kth largest element in a[l….r].
Segment trees with each node storing an ordered set -> (logn)^3 per query

Node 1 -> 1,2...n -> array(1,....n) in sorted form
Node i -> l,....r -> array(l,..r) in sorted form

T(n) = 2*T(n/2) + O(n)
We can build this seg tree in O(nlogn)

Kth element is x
We'll find rank of x in this range l,r
Split into m ranges
$Node_i$, $node_{i+1}$, $node_{i+2}$, …, $node_m$

$rank_i(x)$ + …. $rank_m(x)$ >= k

O(logn * logn * logn)

Cnt[1,.....n] -> cnt[i] = count of value i in the array

V_0 -> cnt[i]= 0 for all i
V_i -> cnt[j] = count of value j in a[1..i]

Cnt[3] in a[l...r]
Cnt[3] in V_r - cnt[3] in V_{l-1}

```
//Range update, range max queries.
struct Vertex
{
        Vertex *l, *r;
        int sum, val;
        bool is;
```

```
        Vertex(int va) : l(nullptr), r(nullptr), sum(va), val(0), is(false) {}
        Vertex(Vertex *L, Vertex *R) : l(L), r(R), sum(0), val(0), is(0)
        {
        if (l) sum = max(sum,l->sum);
        if (r) sum = max(sum,r->sum);
        }
};

Vertex* newlazykid(Vertex *v, int delta, int L, int R)
{
        Vertex *ret = new Vertex(0);
        ret->l = v->l;
        ret->r = v->r;
        ret->val = v->val;
        ret->is = 1;
        ret->val += delta;
        ret->sum = v->sum + delta;

        return ret;
}

void propagate(Vertex *v, int L, int R)
{
        if(v->is && v->val)
        {
   if(L!=R)
   {
        v->l = newlazykid(v->l, v->val, L, (L+R)/2);
        v->r = newlazykid(v->r, v->val, (L+R)/2+1, R);
   }
        }
        v->is=0;
        v->val=0;
}

Vertex* build(int tl, int tr)
```

```
{
        if (tl == tr)
        return new Vertex(0);
        int tm = (tl + tr) / 2;
        return new Vertex(build(tl, tm), build(tm+1, tr));
}

int get_max(Vertex* v, int tl, int tr, int l, int r)
{
        if (l > r)
        return 0;
        if (l <= tl && tr <= r)
        return v->sum;
        propagate(v, tl, tr);
        int tm = (tl + tr) / 2;
        return max(get_max(v->l, tl, tm, l, min(r,tm))
        ,get_max(v->r, tm+1, tr, max(l,tm+1), r));
}

Vertex* update(Vertex* v, int tl, int tr, int L, int R, int delta)
{
        if(tr < L || R < tl) return v;
        if (L<=tl && tr<=R)
        return newlazykid(v,delta, tl, tr);
        propagate(v, tl, tr);
        int tm = (tl + tr) / 2;
        return new Vertex(update(v->l, tl, tm, L, min(R,tm), delta),
        update(v->r, tm+1, tr, max(L,tm+1), R, delta));
}
```