```
/*
 *  This is a data implement of persist segment tree, which is a data
 *  structure support interval update and undo and redo.
 */

#ifndef N
#define N 200100
#endif
#define INF MOD
template <class t> struct per_segment_node{
	int lef, rig;
	t minv, add, sum;
};

template <class t> struct per_segment_tree{
	int l, cnt;
	per_segment_node <t> tree[N * 40];

	inline per_segment_tree(){ l = 0; cnt = 1; }

	/*
	 *  Allocate a new node.
	 */
	inline int newnode(int no) {
		tree[cnt++] = tree[no];
		return cnt - 1;
	}
	inline int newnode(int lef, int rig, int minv = -1, int sum = 0, int add = 0) {
		tree[cnt].lef = lef;
		tree[cnt].rig = rig;
		tree[cnt].minv = minv;
		tree[cnt].add = add;
		tree[cnt].sum = sum;
		cnt++;
		return cnt - 1;
	}

	/*
	 *  An initialize function.
	 */
	int build(int l, int r, t orig = 0, t *a = NULL){
		if(l > r) r = l;
		if(l == r){
			if(a != NULL)
				return newnode(-1, -1, a[l], a[l]);
			else
				return newnode(-1, -1, orig, orig);
		}
		int mid = (l + r) / 2;
		int rt = newnode(build(l, mid, orig, a), build(mid + 1, r, orig, a), 0, 0);
		int lef = tree[rt].lef, rig = tree[rt].rig;
		tree[rt].minv = min(tree[lef].minv, tree[rig].minv);
		tree[rt].sum = tree[lef].sum + tree[rig].sum;
		return rt;
	}

	/*
```

```
 * To do lazy operation, return new node index
 */
inline int relax(int no, int l, int r) {
	int le = newnode(tree[no].lef), ri = newnode(tree[no].rig);
	int mid = (l + r) >> 1;
	tree[le].add += tree[no].add;
	tree[le].sum += tree[no].add * (mid - l + 1);
	tree[le].minv += tree[no].add;
	tree[ri].add += tree[no].add;
	tree[ri].sum += tree[no].add * (r - mid);
	tree[ri].minv += tree[no].add;
	return newnode(le, ri, tree[no].minv, tree[no].sum);
}

/*
 * Update the value between l to r, return the new root index
 */
int down(int l, int r, int rl, int rr, int no, t ranadd){
	if(l <= rl && r >= rr){
		int rno = newnode(no);
		tree[rno].add += ranadd;
		tree[rno].sum += ranadd * (rr - rl + 1);
		tree[rno].minv += ranadd;
		return rno;
	}
	if(tree[no].add && rl != rr)
		no = relax(no, rl, rr);
	else
		no = newnode(no);
	int mid = (rl + rr) >> 1;
	if(r >= rl && l <= mid)
		tree[no].lef = down(l, r, rl, mid, tree[no].lef, ranadd);
	if(r >= mid + 1 && l <= rr)
		tree[no].rig = down(l, r, mid + 1, rr, tree[no].rig, ranadd);

	tree[no].sum = tree[tree[no].lef].sum + tree[tree[no].rig].sum;
	tree[no].minv = min(tree[tree[no].rig].minv, tree[tree[no].lef].minv);
	return no;
}

/*
 *  Return the sum of value between l to r
 */
pair <t, int> getsum(int l, int r, int rl, int rr, int no){
	if(l > r) return mpr(0, -1);
	if(l <= rl && r >= rr)
		return mpr(tree[no].sum, no);
	if(tree[no].add && rl != rr)
		no = relax(no, rl, rr);
	else
		no = newnode(no);
	t ans = 0;
	int mid = (rl + rr) >> 1;
	if(r >= rl && l <= mid) {
		pair <t, int> tans = getsum(l, r, rl, mid, tree[no].lef);
		ans += tans.first;
		tree[no].lef = tans.second;
```

```
        }
        if(r >= mid + 1 && l <= rr) {
            pair <t, int> tans = getsum(l, r, mid + 1, rr, tree[no].rig);
            ans += tans.first;
            tree[no].rig = tans.second;
        }
        return mpr(ans, no);
    }

    /*
     *  Return the minimum value between l to r
     */
    pair <t, int> getmin(int l, int r, int rl, int rr, int no){
        if(l > r) return mpr(0, -1);
        if(l <= rl && r >= rr)
            return mpr(tree[no].minv, no);
        if(tree[no].add && rl != rr)
            no = relax(no, rl, rr);
        else
            no = newnode(no);
        t ans = INF;
        int mid = (rl + rr) >> 1;
        if(r >= rl && l <= mid) {
            pair <t, int> tans = getmin(l, r, rl, mid, tree[no].lef);
            ans = min(ans, tans.first);
            tree[no].lef = tans.second;
        }
        if(r >= mid + 1 && l <= rr) {
            pair <t, int> tans = getmin(l, r, mid + 1, rr, tree[no].rig);
            ans = min(ans, tans.first);
            tree[no].rig = tans.second;
        }
        return mpr(ans, no);
    }
};
```