```
class Treap {
    static minstd_rand generator;

    struct Node {
        int priority, size, value, min_value, add = 0, rev = 0;
        Node *l = nullptr, *r = nullptr;
        Node (int value): value(value), priority(generator()), size(1), min_value(value) {}
    } *root = nullptr;

    static int get_size(Node *n){
        return n ? n -> size : 0;
    }

    static int get_min_value(Node *n){
        return n ? n -> min_value + n -> add : INF;
    }

    static void push(Node *n){
        if (n){
            if (n -> add){
                n -> value += n -> add;
                n -> min_value += n -> add;
                if (n -> l){
                    n -> l -> add += n -> add;
                }
                if (n -> r){
                    n -> r -> add += n -> add;
                }
                n -> add = 0;
            }
            if (n -> rev){
                swap(n -> l, n -> r);
                n -> l -> rev ^= 1;
                n -> r -> rev ^= 1;
                n -> rev = 0;
            }
        }
    }

    static void update(Node *&n){
        if (n){
            n -> size = get_size(n -> l) + 1 + get_size(n -> r);
            n -> min_value = min(get_min_value(n -> l), min(n -> value, get_min_value(n -> r)));
        }
    }

    static Node *merge(Node *a, Node *b){
        push(a);
        push(b);
        if (!a || !b){
            return a ? a : b;
        }
        if (a->priority > b->priority){
            a->r = merge(a->r, b);
            update(a);
            return a;
        }
        else {
            b->l = merge(a, b->l);
            update(b);
            return b;
        }
    }

    static void split(Node *n, int k, Node *&a, Node *&b){
        push(n);
        if (!n){
            a = b = nullptr;
            return ;
        }
        if (get_size(n -> l) < k){
            split(n->r, k - get_size(n -> l) - 1, n->r, b);
            a = n;
        }
        else {
            split(n->l, k, a, n->l);
            b = n;
        }
        update(a);
        update(b);
    }

    void show_tree(Node *n){
        if (!n)
            return ;
        cout << n -> value << endl;
        show_tree(n -> l);
        show_tree(n -> r);
```

```
    }

public:

    int get(int index){
        Node *greater, *equal, *less;
        split(root, index, less, greater);
        split(greater, 1, equal, greater);
        int result = equal -> value;
        root = merge(merge(less, equal), greater);
        return result;
    }

    void push_back(int value){
        root = merge(root, new Node(value));
    }

    void push_front(int value){
        root = merge(new Node(value), root);
    }

    void insert(int index, int value){
        Node *greater, *less;
        split(root, index, less, greater);
        root = merge(merge(less, new Node(value)), greater);
    }

    void erase(int index){
        Node *greater, *equal, *less;
        split(root, index, less, greater);
        split(greater, 1, equal, greater);
        root = merge(less, greater);
    }

    void erase(int l, int r){
        Node *greater, *equal, *less;
        split(root, l, less, greater);
        split(greater, r - l + 1, equal, greater);
        root = merge(less, greater);
    }

    void movetofront(int l, int r){
        Node *greater, *equal, *less;
        split(root, l, less, greater);
        split(greater, r - l + 1, equal, greater);
        root = merge(merge(equal, less), greater);
    }

    void revolve(int l, int r, int x){
        Node *greater, *equal, *less;
        split(root, l, less, greater);
        split(greater, r - l + 1, equal, greater);
        int len = get_size(equal);
        x %= len;
        // переставляем x последних элементов в начало
        Node *left, *right;
        split(equal, len - x, left, right);
        equal = merge(right, left);
        root = merge(merge(less, equal), greater);
    }

    int get_min(int l, int r){
        Node *greater, *equal, *less;
        split(root, l, less, greater);
        split(greater, r - l + 1, equal, greater);
        int result = get_min_value(equal);
        root = merge(merge(less, equal), greater);
        return result;
    }

    void range_add(int l, int r, int value){
        Node *greater, *equal, *less;
        split(root, l, less, greater);
        split(greater, r - l + 1, equal, greater);
        equal -> add += value;
        root = merge(merge(less, equal), greater);
    }

    void reverse(int l, int r){
        Node *greater, *equal, *less;
        split(root, l, less, greater);
        split(greater, r - l + 1, equal, greater);
        equal -> rev ^= 1;
        root = merge(merge(less, equal), greater);
    }

    void show_tree(){
        show_tree(root);
```

```
    }

    int size(){
        return get_size(root);
    }
};

minstd_rand Treap::generator
```