


```

24
25 // MultiSet
26 class SplayTree {
27     constructor() {
28         this.root = null;
29         this.cmp = (x, y) => x >= y ? 0 : 1;
30     }
31     zig(x) { // right rotation
32         let y = x.parent;
33         if (x.right != null) x.right.parent = y;
34         y.left = x.right;
35         x.right = y;
36         if (y.isLeft()) {
37             y.parent.left = x;
38         } else if (y.isRight()) {
39             y.parent.right = x;
40         }
41         x.parent = y.parent;
42         y.parent = x;
43         y.update();
44         x.update();
45     }
46     zag(x) { // left rotation
47         let y = x.parent;
48         if (x.left != null) x.left.parent = y;
49         y.right = x.left;
50         x.left = y;
51         if (y.isLeft()) {
52             y.parent.left = x;
53         } else if (y.isRight()) {
54             y.parent.right = x;
55         }
56         x.parent = y.parent;
57         y.parent = x;
58         y.update();
59         x.update();
60     }
61     zigzig(x) { // RR
62         this.zig(x.parent);
63         this.zig(x);
64     }
65     zigzag(x) { // RL
66         this.zig(x);
67         this.zag(x);
68     }
69     zagzag(x) { // LL
70         this.zag(x.parent);
71         this.zag(x);
72     }
73     zagzig(x) { // LR
74         this.zag(x);
75         this.zig(x);
76     }
77     splay(node, guard = null) { // splay a "node" just under a "guard", which is default to splay to the "root".
78         while (!node.isRoot(guard)) {
79             if (node.parent.isRoot(guard)) {
80                 if (node.isLeft()) {
81                     this.zig(node);
82                 } else {
83                     this.zag(node);
84                 }
85             } else {
86                 if (node.parent.isLeft()) {
87                     if (node.isLeft()) {
88                         this.zigzig(node);
89                     } else {
90                         this.zagzig(node);
91                     }
92                 } else {
93                     if (node.isRight()) {
94                         this.zagzag(node);
95                     } else {
96                         this.zigzag(node);
97                     }
98                 }
99             }
100         }

```

```

101     if (guard == null) this.root = node; // reset "root" to "node".
102 }
103 LastNode(x) {
104     this.splay(x);
105     let node = x.left;
106     if (node == null) return null;
107     while (node.right != null) node = node.right;
108     this.splay(node);
109     return node;
110 }
111 NextNode(x) {
112     this.splay(x);
113     let node = x.right;
114     if (node == null) return null;
115     while (node.left != null) node = node.left;
116     this.splay(node);
117     return node;
118 }
119 find(value) {
120     return this.findFirstOf(value);
121 }
122 findFirstOf(value) {
123     let node = this.root, res = null, last_visited = null;
124     while (node != null) {
125         last_visited = node;
126         if (this.cmp(value, node.val)) {
127             node = node.left;
128         } else if (this.cmp(node.val, value)) {
129             node = node.right;
130         } else {
131             res = node;
132             node = node.left;
133         }
134     }
135     if (last_visited != null) this.splay(last_visited);
136     return res;
137 }
138 findLastOf(value) {
139     let node = this.root, res = null, last_visited = null;
140     while (node != null) {
141         last_visited = node;
142         if (this.cmp(value, node.val)) {
143             node = node.left;
144         } else if (this.cmp(node.val, value)) {
145             node = node.right;
146         } else {
147             res = node;
148             node = node.right;
149         }
150     }
151     if (last_visited != null) this.splay(last_visited);
152     return res;
153 }
154 findRankOf(node) {
155     this.splay(node);
156     return node.left == null ? 0 : node.left.sz;
157 }
158 findSuccessorOf(value) {
159     let node = this.root, res = null, last_visited = null;
160     while (node != null) {
161         last_visited = node;
162         if (this.cmp(value, node.val)) {
163             res = node;
164             node = node.left;
165         } else {
166             node = node.right;
167         }
168     }
169     if (last_visited != null) this.splay(last_visited);
170     return res;
171 }
172 findPrecursorOf(value) {
173     let node = this.root, res = null, last_visited = null;
174     while (node != null) {
175         last_visited = node;
176         if (this.cmp(node.val, value)) {
177             res = node;

```

```

178         node = node.right;
179     } else {
180         node = node.left;
181     }
182 }
183 if (last_visited != null) this.splay(last_visited);
184 return res;
185 }
186 findKthNode(rank) {
187     if (rank < 0 || rank >= this.size()) return null;
188     let node = this.root;
189     while (node != null) {
190         let leftsize = node.left == null ? 0 : node.left.sz;
191         if (leftsize == rank) break;
192         if (leftsize > rank) {
193             node = node.left;
194         } else {
195             rank -= leftsize + 1;
196             node = node.right;
197         }
198     }
199     this.splay(node);
200     return node;
201 }
202 make(value) {
203     return new SplayNode(value);
204 }
205 removeNode(node) {
206     node = null;
207 }
208
209 // ----- Public Usage -----
210 insert(value) { // allow duplicates, tree nodes allow same value O(logN)
211     if (this.root == null) {
212         this.root = this.make(value);
213         return this.root;
214     }
215     let node = this.root;
216     while (node != null) {
217         if (this.cmp(value, node.val)) {
218             if (node.left == null) {
219                 node.left = this.make(value);
220                 node.left.parent = node;
221                 node = node.left;
222                 break;
223             }
224             node = node.left;
225         } else {
226             if (node.right == null) {
227                 node.right = this.make(value);
228                 node.right.parent = node;
229                 node = node.right;
230                 break;
231             }
232             node = node.right;
233         }
234     }
235     this.splay(node);
236     return node;
237 }
238 remove(value) { // remove one node, not remove all O(logN)
239     let node = this.find(value);
240     if (node == null) return false;
241     this.splay(node);
242     if (node.left == null) {
243         this.root = node.right;
244         if (node.right != null) node.right.parent = null;
245         this.removeNode(node);
246         return true;
247     }
248     if (node.right == null) {
249         this.root = node.left;
250         if (node.left != null) node.left.parent = null;
251         this.removeNode(node);
252         return true;
253     }
254     let last_node = this.LastNode(node);

```

```

255     let next_node = this.NextNode(node);
256     this.splay(last_node);
257     this.splay(next_node, last_node);
258     this.removeNode(next_node.left);
259     next_node.left = null;
260     next_node.update();
261     last_node.update();
262     return true;
263 }
264 has(value) { // O(logN)
265     return this.count(value) > 0;
266 }
267 count(value) { // O(logN)
268     let x = this.findFirstOf(value);
269     if (x == null) return 0;
270     let rank_x = this.findRankOf(x);
271     let y = this.findLastOf(value);
272     let rank_y = this.findRankOf(y);
273     return rank_y - rank_x + 1;
274 }
275 rankOf(value) { // The number of elements strictly less than value O(logN)
276     let x = this.findPrecursorOf(value);
277     return x == null ? 0 : this.findRankOf(x) + 1;
278 }
279 findKth(rank) { // (0-indexed) O(logN)
280     let x = this.findKthNode(rank);
281     return x == null ? null : (x.val);
282 }
283 higher(value) { // > upper_bound O(logN)
284     let node = this.findSuccessorOf(value);
285     return node == null ? null : (node.val);
286 }
287 lower(value) { // < O(logN)
288     let node = this.findPrecursorOf(value);
289     return node == null ? null : (node.val);
290 }
291 first() {
292     return this.findKth(0);
293 }
294 last() {
295     return this.findKth(this.size() - 1);
296 }
297 poll() {
298     let res = this.first();
299     this.remove(res);
300     return res;
301 }
302 pollLast() {
303     let res = this.last();
304     this.remove(res);
305     return res;
306 }
307 size() {
308     return this.root == null ? 0 : this.root.sz;
309 }
310 isEmpty() {
311     return this.root == null;
312 }
313 show() { // Get sorted values in the splay tree O(n).
314     let res = [];
315     const dfs = (x) => {
316         if (x == null) return;
317         dfs(x.left);
318         res.push(x.val);
319         dfs(x.right);
320     };
321     dfs(this.root);
322     return res;
323 }
324 }
325
326 const continuousSubarrays = (a) => {
327     let tree = new SplayTree(), n = a.length, l = 0, res = 0;
328     for (let i = 0; i < n; i++) {
329         tree.insert(a[i]);
330         while (tree.last() - tree.first() > 2) tree.remove(a[l++]);
331         res += i - l + 1;

```

```
332     }
333     return res;
334 };
```

☐ Custom Testcase

Use Example Testcases

Run

Submit

Submission Result: **Accepted** (/submissions/detail/984264071/) ⓘ More Details ▶ (/submissions/detail/984264071/)

Share your acceptance!