

```

1.  class Treap {
2.      static minstd_rand generator;
3.
4.      struct Node {
5.          int key, priority;
6.          Node *l = nullptr, *r = nullptr;
7.          Node (int key): key(key), priority(generator()) {}
8.      } *root = nullptr;
9.
10.     static Node *merge(Node *a, Node *b){
11.         if (!a || !b){
12.             return a ? a : b;
13.         }
14.         if (a->priority > b->priority){
15.             a->r = merge(a->r, b);
16.             return a;
17.         }
18.         else {
19.             b->l = merge(a, b->l);
20.             return b;
21.         }
22.     }
23.
24.     static void split(Node *n, int key, Node *&a, Node *&b){
25.         if (!n){
26.             a = b = nullptr;
27.             return ;
28.         }
29.         if (n->key < key){
30.             //      a = n
31.             // n->l      a' b'
32.             split(n->r, key, n->r, b);
33.             a = n;
34.         }
35.         else {
36.             split(n->l, key, a, n->l);
37.             b = n;
38.         }
39.     }
40.
41. public:
42.
43.     bool find(int key){
44.         Node *less, *equal, *greater;
45.         split(root, key, less, greater);
46.         split(greater, key + 1, equal, greater);
47.         bool result = equal;
48.         root = merge(merge(less, equal), greater);
49.         return equal;
50.     }
51.
52.     void insert(int key){
53.         Node *greater, *less;
54.         split(root, key, less, greater);
55.         less = merge(less, new Node(key));
56.         root = merge(less, greater);
57.     }
58.
59.     int min(Node *n) const {
60.         if (!n)
61.             return -1;
62.         while (n->l){
63.             n = n->l;
64.         }
65.         return n->key;
66.     }
67.
68.     int next(int key){
69.         Node *greater, *less;
70.         split(root, key, less, greater);
71.         int ans = min(greater);
72.         root = merge(less, greater);
73.         return ans;
74.     }
75.
76. };
77.

```