

```

1.  class Treap {
2.      static mt19937_64 generator;
3.
4.      struct Node {
5.          int key, priority;
6.          int value, max_val, add = 0;
7.          Node *l = nullptr, *r = nullptr;
8.          Node (int key, int value): key(key), priority(generator()), value(value), max_val(value) {}
9.      } *root = nullptr;
10.
11.      static int getMaxValue(Node *n){
12.          return n ? n -> max_val + n -> add : -1e9;
13.      }
14.
15.      static void push(Node *&n){
16.          if (n && n -> add){
17.              n -> value += n -> add;
18.              n -> max_val += n -> add;
19.              if (n -> l){
20.                  n -> l -> add += n -> add;
21.              }
22.              if (n -> r){
23.                  n -> r -> add += n -> add;
24.              }
25.              n -> add = 0;
26.          }
27.      }
28.
29.      static void update(Node *&n){
30.          if (n){
31.              n -> max_val = max(max(getMaxValue(n -> l), n -> value), getMaxValue(n -> r));
32.          }
33.      }
34.
35.      static Node *merge(Node *a, Node *b){
36.          push(a);
37.          push(b);
38.          if (!a || !b){
39.              return a ? a : b;
40.          }
41.          if (a->priority > b->priority){
42.              a->r = merge(a->r, b);
43.              update(a);
44.              return a;
45.          }
46.          else {
47.              b->l = merge(a, b->l);
48.              update(b);
49.              return b;
50.          }
51.      }
52.
53.      static void split(Node *n, int key, Node *&a, Node *&b){
54.          push(n);
55.          if (!n){
56.              a = b = nullptr;
57.              return ;
58.          }
59.          if (n -> key < key){
60.              //      a = n
61.              // n->l      a' b'
62.              split(n->r, key, n->r, b);
63.              a = n;
64.          }
65.          else {
66.              split(n->l, key, a, n->l);
67.              b = n;
68.          }
69.          update(a);
70.          update(b);
71.      }
72.
73.  public:
74.
75.      bool find(int key){
76.          Node *greater, *equal, *less;
77.          split(root, key, less, greater);
78.          split(greater, key + 1, equal, greater);
79.          bool result = equal;
80.          root = merge(merge(less, equal), greater);
81.          return result;
82.      }
83.
84.      void insert(int key, int value){
85.          Node *greater, *less;

```

```

86.     split(root, key, less, greater);
87.     less = merge(less, new Node(key, value));
88.     root = merge(less, greater);
89. }
90.
91. void erase(int key){
92.     Node *greater, *equal, *less;
93.     split(root, key, less, greater);
94.     split(greater, key + 1, equal, greater);
95.     root = merge(less, greater);
96. }
97.
98. long long getMax(int l, int r){
99.     Node *left, *middle, *right;
100.     split(root, l, left, middle);
101.     split(middle, r + 1, middle, right);
102.     long long ans = getMaxValue(middle);
103.     root = merge(merge(left, middle), right);
104.     return ans;
105. }
106.
107. void rangeAdd(int l, int r, int value){
108.     Node *left, *middle, *right;
109.     split(root, l, left, middle);
110.     split(middle, r + 1, middle, right);
111.     if (middle){
112.         middle -> add += value;
113.     }
114.     root = merge(merge(left, middle), right);
115. }
116.
117. };
118.
119. mt19937_64 Treap::generator;

```