```cpp
class SegTree {
private:
  struct val_t {
    bool enable;
    ll upd, add, min, max, sum;
  };
  int n, N; // n is the original size, while N is the extended size
  int base;
  vector<val_t> nodes;
  vi idl, idr, cover_size;
  void merge(int id) {
    nodes[id].min = min(nodes[idl[id]].min + nodes[idl[id]].add,
      nodes[idr[id]].min + nodes[idr[id]].add);
    nodes[id].max = max(nodes[idl[id]].max + nodes[idl[id]].add,
      nodes[idr[id]].max + nodes[idr[id]].add);
    nodes[id].sum = nodes[idl[id]].sum + nodes[idl[id]].add * cover_size[idl[id]]
      + nodes[idr[id]].sum + nodes[idr[id]].add * cover_size[idr[id]];
  }
  void lazy(int id) {
    if (id >= base) return;
    if (nodes[id].enable) {
      ll upd = nodes[id].upd + nodes[id].add;
      nodes[idl[id]] = { true, upd, 0, upd, upd, upd * cover_size[idl[id]] };
      nodes[idr[id]] = { true, upd, 0, upd, upd, upd * cover_size[idr[id]] };
      nodes[id] = { false, 0, 0, upd, upd, upd * cover_size[id] };
    }
    else {
      nodes[idl[id]].add += nodes[id].add;
      nodes[idr[id]].add += nodes[id].add;
      nodes[id].add = 0;
      merge(id);
    }
  }
  enum change_t {
    UPD, ADD
  };
  void change_rec(int s, int t, int l, int r, int id, ll x, change_t op) {
    if (s == l && t == r) {
      if (op == UPD) nodes[id] = { true, x, 0, x, x, x * cover_size[id] };
      else if (op == ADD) nodes[id].add += x;
    }
    else {
      lazy(id);
      int m = (l + r) / 2;
      if (s < m && m < t) {
        change_rec(s, m, l, m, idl[id], x, op);
        change_rec(m, t, m, r, idr[id], x, op);
      }
      else if (s < m) {
        change_rec(s, t, l, m, idl[id], x, op);
      }
      else if (m < t) {
        change_rec(s, t, m, r, idr[id], x, op);
      }
      merge(id);
    }
  }
  enum solve_t {
    MIN, MAX, SUM
  };
  ll solve_rec(int s, int t, int l, int r, int id, solve_t op) {
    ll v = 0;
    if (s == l && t == r) {
      if (op == MIN) v = nodes[id].min;
      else if (op == MAX) v = nodes[id].max;
      else if (op == SUM) v = nodes[id].sum;
    }
    else {
      lazy(id);
      int m = (l + r) / 2;
      if (s < m && m < t) {
```

```
72          ll v0 = solve_rec(s, m, l, m, idl[id], op);
73          ll v1 = solve_rec(m, t, m, r, idr[id], op);
74          if (op == MIN) v = min(v0, v1);
75          else if (op == MAX) v = max(v0, v1);
76          else if (op == SUM) v = v0 + v1;
77        }
78        else if (s < m) {
79          v = solve_rec(s, t, l, m, idl[id], op);
80        }
81        else if (m < t) {
82          v = solve_rec(s, t, m, r, idr[id], op);
83        }
84      }
85      if (op == MIN) v += nodes[id].add;
86      else if (op == MAX) v += nodes[id].add;
87      else if (op == SUM) v += nodes[id].add * (t - s);
88      return v;
89    }
90  public:
91    SegTree(int n, ll init) {
92      this->n = n;
93      N = (int)pow(2, ceil(log2(n)));
94      base = N - 1;
95      nodes = vector<val_t>(base + N, { false, 0, 0, LLONG_MAX, LLONG_MIN, 0 });
96      idl.resize(base + N, -1);
97      idr.resize(base + N, -1);
98      Loop(i, base) {
99        idl[i] = i * 2 + 1;
100       idr[i] = i * 2 + 2;
101     }
102     cover_size.resize(base + N);
103     Loop(i, n) {
104       cover_size[base + i] = 1;
105     }
106     Loopr(i, base) {
107       cover_size[i] = cover_size[idl[i]] + cover_size[idr[i]];
108     }
109     upd(0, n, init);
110   }
111   void upd(int s, int t, ll x) {
112     change_rec(s, t, 0, N, 0, x, UPD);
113   }
114   void add(int s, int t, ll x) {
115     change_rec(s, t, 0, N, 0, x, ADD);
116   }
117   ll minof(int s, int t) {
118     return solve_rec(s, t, 0, N, 0, MIN);
119   }
120   ll maxof(int s, int t) {
121     return solve_rec(s, t, 0, N, 0, MAX);
122   }
123   ll sumof(int s, int t) {
124     return solve_rec(s, t, 0, N, 0, SUM);
125   }
126 };
```