

```

1 class SegTree {
2 private:
3     struct val_t {
4         bool enable;
5         ll upd, add, min, max, sum;
6     };
7     int n, N; // n is the original size, while N is the extended size
8     int base;
9     vector<val_t> nodes;
10    int left_of(int id) {
11        if (id >= base) return -1;
12        else return id * 2 + 1;
13    }
14    int right_of(int id) {
15        if (id >= base) return -1;
16        else return id * 2 + 2;
17    }
18    int parent_of(int id) {
19        if (id == 0) return -1;
20        else return (id - 1) >> 1;
21    }
22    void merge(int id, int id_l, int id_r) {
23        nodes[id].min = min(nodes[id_l].min + nodes[id_l].add, nodes[id_r].min + nodes[id_r].add);
24        nodes[id].max = max(nodes[id_l].max + nodes[id_l].add, nodes[id_r].max + nodes[id_r].add);
25        nodes[id].sum = nodes[id_l].sum + nodes[id_l].add * cover_size(id_l)
26            + nodes[id_r].sum + nodes[id_r].add * cover_size(id_r);
27    }
28    void lazy(int id) {
29        if (id >= base) return;
30        int id_l = left_of(id);
31        int id_r = right_of(id);
32        if (nodes[id].enable) {
33            ll upd = nodes[id].upd + nodes[id].add;
34            nodes[id_l] = { true, upd, 0, upd, upd, upd * cover_size(id_l) };
35            nodes[id_r] = { true, upd, 0, upd, upd, upd * cover_size(id_r) };
36            nodes[id] = { false, 0, 0, upd, upd, upd * cover_size(id) };
37        }
38        else {
39            nodes[id_l].add += nodes[id].add;
40            nodes[id_r].add += nodes[id].add;
41            nodes[id].add = 0;
42            merge(id, id_l, id_r);
43        }
44    }
45    enum change_t {
46        UPD, ADD
47    };
48    void change_rec(int s, int t, int l, int r, int id, ll x, change_t op) {
49        if (s == l && t == r) {
50            if (op == UPD) nodes[id] = { true, x, 0, x, x, x * cover_size(id) };
51            else if (op == ADD) nodes[id].add += x;
52        }
53        else {
54            lazy(id);
55            int m = (l + r) / 2;
56            int id_l = left_of(id);
57            int id_r = right_of(id);
58            if (s < m && m < t) {
59                change_rec(s, m, l, m, id_l, x, op);
60                change_rec(m, t, m, r, id_r, x, op);
61            }
62            else if (s < m) {
63                change_rec(s, t, l, m, id_l, x, op);
64            }
65            else if (m < t) {
66                change_rec(s, t, m, r, id_r, x, op);
67            }
68            merge(id, id_l, id_r);
69        }
70    }
71    enum solve_t {

```

```

72     MIN, MAX, SUM
73 };
74 ll solve_rec(int s, int t, int l, int r, int id, solve_t op) {
75     ll v = 0;
76     if (s == l && t == r) {
77         if (op == MIN) v = nodes[id].min;
78         else if (op == MAX) v = nodes[id].max;
79         else if (op == SUM) v = nodes[id].sum;
80     }
81     else {
82         lazy(id);
83         int m = (l + r) / 2;
84         int id_l = left_of(id);
85         int id_r = right_of(id);
86         if (s < m && m < t) {
87             ll v0 = solve_rec(s, m, l, m, id_l, op);
88             ll v1 = solve_rec(m, t, m, r, id_r, op);
89             if (op == MIN) v = min(v0, v1);
90             else if (op == MAX) v = max(v0, v1);
91             else if (op == SUM) v = v0 + v1;
92         }
93         else if (s < m) {
94             v = solve_rec(s, t, l, m, id_l, op);
95         }
96         else if (m < t) {
97             v = solve_rec(s, t, m, r, id_r, op);
98         }
99     }
100     if (op == MIN) v += nodes[id].add;
101     else if (op == MAX) v += nodes[id].add;
102     else if (op == SUM) v += nodes[id].add * (t - s);
103     return v;
104 }
105 public:
106 SegTree(int n, ll init) {
107     this->n = n;
108     N = (int)pow(2, ceil(log2(n)));
109     base = N - 1;
110     nodes = vector<val_t>(base + N, { false, 0, 0, LLONG_MAX, LLONG_MIN, 0 });
111     upd(0, n, init);
112 }
113 int cover_size(int id) {
114     int cnt = 1;
115     while (left_of(id) != -1) {
116         id = left_of(id);
117         cnt *= 2;
118     }
119     int l = id - base;
120     int r = min(l + cnt, n);
121     return max(0, r - l);
122 }
123 void upd(int s, int t, ll x) {
124     change_rec(s, t, 0, N, 0, x, UPD);
125 }
126 void add(int s, int t, ll x) {
127     change_rec(s, t, 0, N, 0, x, ADD);
128 }
129 ll minof(int s, int t) {
130     return solve_rec(s, t, 0, N, 0, MIN);
131 }
132 ll maxof(int s, int t) {
133     return solve_rec(s, t, 0, N, 0, MAX);
134 }
135 ll sumof(int s, int t) {
136     return solve_rec(s, t, 0, N, 0, SUM);
137 }
138 };

```