```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;
  int left_of(int id) {
    if (id >= base) return -1;
    else return id * 2 + 1;
  }
  int right_of(int id) {
    if (id >= base) return -1;
    else return id * 2 + 2;
  }
  int parent_of(int id) {
    if (id == 0) return -1;
    else return (id - 1) >> 1;
  }
  void merge(int id, int id_l, int id_r) {
    nodes[id].min = min(nodes[id_l].min + nodes[id_l].add, nodes[id_r].min + nodes[id_r].add);
    nodes[id].max = max(nodes[id_l].max + nodes[id_l].add, nodes[id_r].max + nodes[id_r].add);
    nodes[id].sum = nodes[id_l].sum + nodes[id_l].add * cover_size(id_l)
      + nodes[id_r].sum + nodes[id_r].add * cover_size(id_r);
  }
  void lazy(int id) {
    if (id >= base) return;
    int id_l = left_of(id);
    int id_r = right_of(id);
    if (nodes[id].enable) {
      ll upd = nodes[id].upd + nodes[id].add;
      nodes[id_l] = { true, upd, 0, upd, upd, upd * cover_size(id_l) };
      nodes[id_r] = { true, upd, 0, upd, upd, upd * cover_size(id_r) };
      nodes[id] = { false, 0, 0, upd, upd, upd * cover_size(id) };
    }
    else {
      nodes[id_l].add += nodes[id].add;
      nodes[id_r].add += nodes[id].add;
      nodes[id].add = 0;
      merge(id, id_l, id_r);
    }
  }
  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;
      int id_l = left_of(id);
      int id_r = right_of(id);
      if (s < m && m < t) {
        change_rec(s, m, l, m, id_l, x, op);
        change_rec(m, t, m, r, id_r, x, op);
      }
      else if (s < m) {
        change_rec(s, t, l, m, id_l, x, op);
      }
      else if (m < t) {
        change_rec(s, t, m, r, id_r, x, op);
      }
      merge(id, id_l, id_r);
    }
  }
  enum solve_t {
```

```cpp
    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;
      int id_l = left_of(id);
      int id_r = right_of(id);
      if (s < m && m < t) {
        ll v0 = solve_rec(s, m, l, m, id_l, op);
        ll v1 = solve_rec(m, t, m, r, id_r, op);
        if (op == MIN) v = min(v0, v1);
        else if (op == MAX) v = max(v0, v1);
        else if (op == SUM) v = v0 + v1;
      }
      else if (s < m) {
        v = solve_rec(s, t, l, m, id_l, op);
      }
      else if (m < t) {
        v = solve_rec(s, t, m, r, id_r, op);
      }
    }
    if (op == MIN) v += nodes[id].add;
    else if (op == MAX) v += nodes[id].add;
    else if (op == SUM) v += nodes[id].add * (t - s);
    return v;
  }
public:
  SegTree(int n, ll init) {
    this->n = n;
    N = (int)pow(2, ceil(log2(n)));
    base = N - 1;
    nodes = vector<val_t>(base + N, { false, 0, 0, LLONG_MAX, LLONG_MIN, 0 });
    upd(0, n, init);
  }
  int cover_size(int id) {
    int cnt = 1;
    while (left_of(id) != -1) {
      id = left_of(id);
      cnt *= 2;
    }
    int l = id - base;
    int r = min(l + cnt, n);
    return max(0, r - l);
  }
  void upd(int s, int t, ll x) {
    change_rec(s, t, 0, N, 0, x, UPD);
  }
  void add(int s, int t, ll x) {
    change_rec(s, t, 0, N, 0, x, ADD);
  }
  ll minof(int s, int t) {
    return solve_rec(s, t, 0, N, 0, MIN);
  }
  ll maxof(int s, int t) {
    return solve_rec(s, t, 0, N, 0, MAX);
  }
  ll sumof(int s, int t) {
    return solve_rec(s, t, 0, N, 0, SUM);
  }
};
```