```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using vi = vector<int>;
using vll = vector<ll>;
using vvi = vector<vector<int>>;
using vvl = vector<vector<ll>>;

class RMQ {
  vector<long long> data;

public:
  static const long long INF = 100000000000;
  int n;

  RMQ(int _) {
    n = _;
    data.resize(n * 4);
    for (int i = 0; i < n * 4; i++)
      data[i] = INF;
  }

  void update(int index, long long val) {
    int i = index + n - 1;
    data[i] = val;
    while (i > 0) {
      i = (i - 1) / 2;
      data[i] = min(data[i * 2 + 1], data[i * 2 + 2]);
    }
  }

  // [a, b)
  long long query(int a, int b, int k, int l, int r) {
    if (a < 0)
      return 0;
    if (r <= a || b <= l)
      return INF;
    if (a <= l && r <= b)
      return data[k];
    else
      return min(query(a, b, k * 2 + 1, l, (l + r) / 2),
                 query(a, b, k * 2 + 2, (r + l) / 2, r));
  }

  long long query(int a, int b) { return query(a, b, 0, 0, n); }
};

template<typename T>
class tRMQ {
```

```
51    vector<T> data;
52    T unit;
53
54  public:
55    int n;
56    function<T(const T &, const T &)> f;
57
58    tRMQ(int _, T u, function<T(T, T)> bi) {
59      unit = u;
60      f = bi;
61      n = 1;
62      while (n < _) {
63        n <<= 1;
64      }
65      data.resize(n * 4);
66      for (int i = 0; i < n * 4; i++)
67        data[i] = unit;
68    }
69
70    tRMQ(vector<T> &v, T u, function<T(T, T)> bi) {
71      unit = u;
72      f = bi;
73      n = 1;
74      while (n < v.size())
75        n <<= 1;
76      data.resize(n * 4, u);
77      for (int i = 0; i < v.size(); i++) {
78        data[n + i - 1] = v[i];
79      }
80      for (int i = n - 2; i >= 0; i--) {
81        data[i] = f(data[i * 2 + 1], data[i * 2 + 2]);
82      }
83    }
84
85    void update(int index, T val) {
86      int i = index + n - 1;
87      data[i] = val;
88      while (i > 0) {
89        i = (i - 1) / 2;
90        data[i] = f(data[i * 2 + 1], data[i * 2 + 2]);
91      }
92    }
93
94    // [a, b)
95    T query(int a, int b, int k, int l, int r) {
96      if (a < 0 || r <= a || b <= l)
97        return unit;
98      if (a <= l && r <= b)
99        return data[k];
100     else
```

```
101       return f(query(a, b, k * 2 + 1, l, (l + r) / 2),
102                query(a, b, k * 2 + 2, (r + l) / 2, r));
103     }
104
105     T query(int a, int b) { return query(a, b, 0, 0, n); }
106   };
107
108   tRMQ<ll> minrmq(int n) {
109     return tRMQ<ll>(n, 10000000000000000LL, [](ll r, ll l) { return min(l, r); });
110   }
111
112   tRMQ<ll> maxrmq(int n) {
113     return tRMQ<ll>(n, -10000000000000000LL,
114                     [](ll r, ll l) { return max(l, r); });
115   }
116
117   tRMQ<ll> sumrmq(int n) {
118     return tRMQ<ll>(n, 0, [](ll l, ll r) { return l + r; });
119   }
120
121   template<typename T>
122   struct RSQRAQ2 {
123     int n;
124     T unit;
125     function<T(T, T)> update_f, sum_f, query_f;
126     vector<T> dat, lazy;
127
128     RSQRAQ2() {}
129
130     RSQRAQ2(int n_, T unit, function<T(T, T)> update_f, function<T(T, int)> sum_f,
      function<T(T, T)> query_f)
131         : unit(unit), update_f(update_f), sum_f(sum_f), query_f(query_f) {
132       n = 1;
133       while (n < n_)
134         n *= 2;
135       dat.assign(n * 2, unit);
136       lazy.assign(n * 2, unit);
137     }
138
139     RSQRAQ2(vector<T> v, T unit, function<T(T, T)> update_f, function<T(T, int)>
      sum_f, function<T(T, T)> query_f)
140         : unit(unit), update_f(update_f), sum_f(sum_f), query_f(query_f) {
141       n = 1;
142       while (n < v.size()) n <<= 1;
143       dat.assign(n * 2, unit);
144       lazy.assign(n * 2, unit);
145       for (int i = 0; i < v.size(); i++) {
146         dat[n + i - 1] = v[i];
147       }
148       for (int i = n - 2; i >= 0; i--) {
```

```
149       dat[i] = query_f(dat[i * 2 + 1], dat[i * 2 + 2]);
150     }
151   }
152
153   void eval(int len, int k) {
154     if (lazy[k] == unit)
155       return;
156     if (k * 2 + 1 < n * 2 - 1) {
157       lazy[2 * k + 1] = update_f(lazy[2 * k + 1], lazy[k]);
158       lazy[2 * k + 2] = update_f(lazy[2 * k + 2], lazy[k]);
159     }
160     dat[k] = update_f(dat[k], sum_f(lazy[k], len));
161     lazy[k] = unit;
162   }
163
164   // [a, b)
165   T update(int a, int b, T x, int k, int l, int r) {
166     eval(r - l, k);
167     if (b <= l || r <= a)
168       return dat[k];
169     if (a <= l && r <= b) {
170       lazy[k] = update_f(lazy[k], x);
171       return query_f(dat[k], sum_f(lazy[k], r - l));
172     }
173     return dat[k] = query_f(update(a, b, x, 2 * k + 1, l, (l + r) / 2),
174                             update(a, b, x, 2 * k + 2, (l + r) / 2, r));
175   }
176
177   T update(int a, int b, T x) { return update(a, b, x, 0, 0, n); }
178
179   // [a, b)
180   T query(int a, int b, int k, int l, int r) {
181     eval(r - l, k);
182     if (b <= l || r <= a)
183       return unit;
184     if (a <= l && r <= b)
185       return dat[k];
186     return query_f(query(a, b, 2 * k + 1, l, (l + r) / 2), query(a, b, 2 * k + 2, (l + r) / 2,
      r));
187   }
188
189   T query(int a, int b) { return query(a, b, 0, 0, n); }
190 };
191 // ref:https://www65.atwiki.jp/kyopro-lib/pages/15.html
192
193 int main() {
194   vector<ll> v({1, 2, 3, 4, 5, 6, 7, 8});
195   RSQRAQ2<ll> q(v, 0, [](ll l, ll r) {
196                 return l + r;
197               }, [](ll l, int len) {
```

```
198                 return l;
199               }, [](ll l, ll r) {
200                 return max(l, r);
201               }
202     );
203     for (int i = 0; i < v.size(); i++) cerr << q.query(i, i + 1) << endl;
204     q.update(0, 8, 3);
205     cerr << q.query(0, 8) << endl;
206     for (int i = 0; i < v.size(); i++) cerr << q.query(i, i + 1) << endl;
207
208     return 0;
209   }
```