```cpp
#include "../../bits/stdc++.h"
// 区間update区間min 再帰遅延セグ木 0-indexed
// verified: http://judge.u-aizu.ac.jp/onlinejudge/review.jsp?rid=3433302#1
class LazySegmentTree
{
    int n;
    using T1 = int;
    using T2 = int;
    T1 id1 = INT_MAX;
    T2 id2 = -1;
    std::vector<T1> node; // min
    std::vector<T2> lazy; // update

    // 遅延評価
    void eval(int k, int l, int r)
    {
        // 遅延配列が空なら終了
        if (lazy[k] == id2)
            return;
        // 遅延配列を適用
        node[k] = lazy[k];
        if (r - l > 1)
        {
            lazy[2 * k + 1] = lazy[k];
            lazy[2 * k + 2] = lazy[k];
        }
        // 遅延配列初期化
        lazy[k] = id2;
    }

  public:
    LazySegmentTree(int _n)
    {
        int sz = _n;
        n = 1;
        while (n < sz)
            n *= 2;
        // 配列初期化
        node.resize(2 * n - 1, id1);
        lazy.resize(2 * n - 1, id2);
    }
    // 半開区間 [a, b) に対して値 val を反映させる
    void update(int a, int b, T2 val, int l = 0, int r = -1, int k = 0)
    {
        if (r < 0)
            r = n;
        // ノード k で遅延評価
        eval(k, l, r);
        if (b <= l || r <= a)
            return;
        // 区間が被覆されている場合
        if (a <= l && r <= b)
        {
            // 遅延配列更新, 評価
            lazy[k] = val;
            eval(k, l, r);
        }
        else
        {
            // 子ノードの値を評価し, 更新
            int mid = (l + r) / 2;
            update(a, b, val, l, mid, 2 * k + 1);
            update(a, b, val, mid, r, 2 * k + 2);
            node[k] = std::min(node[2 * k + 1], node[2 * k + 2]);
        }
    }

    // 半開区間 [a, b) に対してクエリを投げる
    T1 query(int a, int b, int l = 0, int r = -1, int k = 0)
    {
        if (r < 0)
            r = n;
        eval(k, l, r);
        // 範囲外なら単位元返す
        if (b <= l || r <= a)
            return id1;
        if (a <= l && r <= b)
            return node[k];
        int mid = (l + r) / 2;
        T1 vl = query(a, b, l, mid, 2 * k + 1);
        T1 vr = query(a, b, mid, r, 2 * k + 2);
        return std::min(vl, vr);
    }
};

```