```cpp
#include <chrono>
using namespace std;
auto t1 = chrono::high_resolution_clock::now();
auto t2 = std::chrono::high_resolution_clock::now();

auto duration = std::chrono::duration_cast<std::chrono::microseconds>( t2 - t1 ).count();

cout << duration;


void countSmallerRight(int A[], int len)
{
    set<int> s;
    int countSmaller[len];
    for (int i = len - 1; i >= 0; i--) {
        s.insert(A[i]);
        auto it = s.lower_bound(A[i]);
        countSmaller[i] = distance(s.begin(), it);
    }

    for (int i = 0; i < len; i++)
        cout << countSmaller[i] << " ";
}


//DSU
ll r[limit];
ll parent[limit];

ll find(ll x){
    if(parent[x]==x) return x;
    return parent[x] = find(parent[x]);
}

void un(ll x, ll y){
    ll xRoot = find(x);
    ll yRoot = find(y);
    if(r[xRoot]<r[yRoot]){
        ll temp = yRoot;
        yRoot = xRoot;
        xRoot = temp;
    }
    parent[yRoot] = xRoot;
    if(r[xRoot] == r[yRoot]) r[xRoot]++;

}

```

```cpp
//Hash
struct VectorHasher {
    int operator()(const vector<int> &V) const {
        int hash = V.size();
        for(auto &i : V) {
            hash ^= i + 0x9e3779b9 + (hash << 6) + (hash >> 2);
        }
        return hash;
    }
};

//Sqrt Decomposition
#define limit 1000100
ll arr[limit];
ll bloki[1100];
ll dolzinaBloka = sqrt(limit);

void update(int i, ll val){
    arr[i] = min(val, arr[i]);
    int blokId = i / dolzinaBloka;
    bloki[blokId] = min(bloki[blokId], val);
}

int query(int L, int R){
    ll res = inf*1LL*inf;

    int p = L;
    //leva stran
    while(p < R and p % dolzinaBloka){
        res = min(res, arr[p]);
        p++;
    }
    //bloki
    while(p + dolzinaBloka <= R){
        res = min(res, bloki[p/dolzinaBloka]);
        p++;
    }

    //desno
    while(p <= R){
        res = min(res, arr[p]);
        p++;
    }

    //printf("query %lld %lld -> %lld\n", L, R, res);
    return res;
}

//Matrika
struct Matrix{
```

```
    long long a[2][2];
    Matrix operator *(Matrix other){
        Matrix res = {{{0,0},{0,0}}};
        loop(i, 2){
            loop(k, 2){
                loop(j, 2){
                    res.a[i][k] += a[i][j] * other.a[j][k];
                    res.a[i][k] %= mod;
                }
            }
        }
        return res;
    }
};


//Teorija Stevil

const int MOD = 998244353;
const int N = 200043;

int add(int x, int y)
{
    return (x + y) % MOD;
}

int sub(int x, int y)
{
    return add(x, MOD - y);
}

int mul(int x, int y)
{
    return (x * 1ll * y) % MOD;
}

int binpow(int x, int y)
{
    int z = 1;
    while(y > 0)
    {
        if(y % 2 == 1)
            z = mul(z, x);
        x = mul(x, x);
        y /= 2;
    }
    return z;
}

int inv(int x)
```

```
{
    return binpow(x, MOD - 2);
}

int fact[N];

int C(int n, int k)
{
    return mul(fact[n], inv(mul(fact[k], fact[n - k])));
}


//Stringi
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}

//Template
using namespace std;
typedef long long ll;
#define pll pair<ll, ll>
#define loop(i, n) for(ll i = 0; i < n; i++)
#define FOR(i,n,m) for(ll i = n; i <= m; i++)
#define isIn(vec, item) find(vec.begin(), vec.end(), item) != vec.end()
#define fs first
#define sc second
#define pb push_back
#define mp make_pair
#define all(v) v.begin(),v.end()
#define inf 1000000005
#define mod 1000000007
#define print(v) for(auto e : v) cout << e << " "; cout << endl;

//Treap (Ne uporabljaj, ce se imas rad)




/*
VNAPREJ DOLIČI VELIKOST
```

```
VEKTORJA, KAMOR SHRANJUJEŠ
TREAPE
*/
mt19937 mt_rand(time(0));

struct Treap {
    int data, priority;
    vector<Treap*> kids;
    int subtreeSize, sum, toProp;

    Treap(int data);
};

int size(Treap *me) {
    return me == NULL ? 0 : me->subtreeSize;
}

void recalc(Treap *me) {
    if (me==NULL) return;
    me->subtreeSize = 1;
    me->sum = me->data + me->toProp*size(me);
    for (Treap* t:me->kids) if (t != NULL) me->subtreeSize += t->subtreeSize;
    for (Treap* t:me->kids) if (t != NULL) me->sum += t->sum+t->toProp*size(t);
}

void prop(Treap *me) {
    if (me==NULL) return;
    if (me->toProp == 0) return;
    for (Treap *t:me->kids) if (t != NULL) t->toProp += me->toProp;
    me->data+=me->toProp;
    me->toProp=0;
    recalc(me);
}

Treap* merge(Treap *l, Treap *r) {
    if (l==NULL) return r;
    if (r==NULL) return l;
    prop(l); prop(r);
    if (l->priority < r->priority) {
        l->kids[1]=merge(l->kids[1], r);
        recalc(l);
        return l;
    }
    else {
        r->kids[0]=merge(l, r->kids[0]);
        recalc(r);
        return r;
    }
}

```

```
vector<Treap*> split(Treap *me, int nInLeft) {
    if (me == NULL) return {NULL, NULL};
    prop(me);
    if (size(me->kids[0])>=nInLeft) {
        vector<Treap*> leftRes=split(me->kids[0], nInLeft);
        me->kids[0]=leftRes[1];
        recalc(me);
        return {leftRes[0], me};
    }
    else {
        nInLeft = nInLeft - size(me->kids[0]) - 1;
        vector<Treap*> rightRes = split(me->kids[1], nInLeft);
        me->kids[1] = rightRes[0];
        recalc(me);
        return {me, rightRes[1]};
    }
    return {NULL, NULL};
}

Treap::Treap(int _data) {
    kids={NULL, NULL};
    data = _data;
    toProp = 0;
    recalc(this);

    priority = mt_rand();
}

Treap* rangeAdd(Treap* t, int l, int r, int toAdd) {
    vector<Treap*> a=split(t, l), b=split(a[1], r-l+1);
    b[0]->toProp+=toAdd;
    return merge(a[0], merge(b[0], b[1]));
}

void inOrderTraversal(Treap *t){
    if(t == NULL) return;
    //Left, Node, Right
    inOrderTraversal(t->kids[0]);
    cout << t->data << endl;
    inOrderTraversal(t->kids[1]);
}

//HEAVY-LIGHT DECOMPOSITION

vector<int> parent, heavy, head, pos, depth;
vector<int> children[limit];
int curPos=1;

int dfs(int node){
    int size = 1;
```

```
297     int maxS = 0;
298     int best = -1;
299     for(int c : children[node]){
300         depth[c] = depth[node] + 1;
301         int ret = dfs(c);
302         size += ret;
303         if(maxS < ret){
304             best = c;
305             maxS = ret;
306         }
307     }
308 
309     //nastavi heavy edge
310     heavy[node] = best;
311 
312     return size;
313 }
314 
315 void decompose(int node, int h){
316     if(DEBUG)printf("decom %d, heavy %d\n", node, heavy[node]);
317     head[node] = h;
318     pos[node] = curPos++;
319 
320     if(heavy[node] != -1) decompose(heavy[node], h); //isti component
321     for(int c : children[node]){
322         if(c == heavy[node]) continue;
        decompose(c, c);
    }
}
```
