Segment Tree Min-Max

\\ Base code //

```
struct segmentTree{

    // // Min
    // const int NEUTRAL_ELEMENT = INT_MAX;
    // int baseOperation(int a, int b) {
    //     return min(a, b);
    // }
    // // Max
    // int NEUTRAL_ELEMENT = 0;
    // int baseOperation(int a, int b) {
    //     return max(a, b);
    // }

    int size;
    vector<int> values;
    void init(int n){
        size = 1;
        while(size < n){
            size *= 2;
        }

        values.assign(2 * size, NEUTRAL_ELEMENT);
    }
};
```

\\ Build Code //

```
void build(vector<int> &nums, int x, int lx, int rx){
    if(rx - lx == 1){
        if(lx < (int)nums.size()) {
            values[x] = nums[lx];
        }
        return;
    }

    int mid = (rx + lx) / 2;
    build(nums, 2 * x + 1, lx, mid);
    build(nums, 2 * x + 2, mid, rx);

    values[x] = baseOperation(values[2 * x + 1], values[2 * x + 2]);
}
void build(vector<int> &nums){
    build(nums, 0, 0, size);
}
```

\\ Set Code //

```
void set(int i, int v, int x, int lx, int rx){
    if(rx - lx == 1){
        values[x] = v;
        return;
    }
    int mid = (rx + lx) / 2;
    if(i < mid){
        set(i, v, 2 * x + 1, lx, mid);
    }
    else{
        set(i, v, 2 * x + 2, mid, rx);
    }
    values[x] = baseOperation(values[2 * x + 1], values[2 * x + 2]);
}
void set(int i, int v){
    set(i, v, 0, 0, size);
}
```

\\ Get Code //

```
long long get(int l, int r, int x, int lx, int rx){
    if(l ⩾ rx || lx ⩾ r) return NEUTRAL_ELEMENT;
    if(lx ⩾ l && rx ⩽ r) return values[x];

    int mid = (rx + lx) / 2;
    long long a = get(l, r, 2 * x + 1, lx, mid);
    long long b = get(l, r, 2 * x + 2, mid, rx);

    return baseOperation(a, b);
}
long long get(int l, int r){
    return get(l, r, 0, 0, size);
}
```

\\ Set Range Code //

```
void setRange(int l, int r, int v, int x, int lx, int rx){
    if(l ⩾ rx || lx ⩾ r) return;
    if(lx ⩾ l && rx ⩽ r) return void(values[x] = baseOperation(values[x], v));

    int mid = (rx + lx) / 2;
    setRange(l, r, v, 2 * x + 1, lx, mid);
    setRange(l, r, v, 2 * x + 2, mid, rx);
}
void setRange(int l,int r, int v){
    setRange(l, r, v, 0, 0, size);
}
```

\\ Calc Code [get value from setRange()] //

```
void calc(int i, int x, int lx, int rx, int &ans){

    if(rx - lx == 1){
        ans = baseOperation(values[x], ans);
        return;
    }

    int mid = (rx + lx) / 2;
    if(i < mid){
        calc(i, 2 * x + 1, lx, mid, ans);
    }
    else{
        calc(i, 2 * x + 2, mid, rx, ans);
    }
    ans = baseOperation(values[x], ans);
}
int calc(int idx){
    int ans = NEUTRAL_ELEMENT;
    calc(idx, 0, 0, size, ans);
    return ans;
}
```

Get First Index of an element in range less than a value (v)

```
// Use it to get the first index that is smaller than v in the range [l, r) ___ (V not included)
// Make sure that segmentTree built with min operation
void getFirstIndexSmallerThan_V_Between_L_R(int v, int l, int r, int x, int lx, int rx, int &ans)
{
    if(~ans || values[x] > v) return;
    if(l ⩾ rx || lx ⩾ r) return;
    if(rx - lx == 1){
        if(values[x] < v) ans = lx; // Make "<" ⟶ "⩽" if you want to include the value v

        return;
    }

    int mid = (rx + lx) / 2;

    // Make "<" ⟶ "⩽" if you want to include the value v
    if(values[2 * x + 1] < v){
        getFirstIndexSmallerThan_V_Between_L_R(v, l, r, 2 * x + 1, lx, mid, ans);
    }

    // Make "<" ⟶ "⩽" if you want to include the value v
    if(values[2 * x + 2] < v && !~ans){
        getFirstIndexSmallerThan_V_Between_L_R(v, l, r, 2 * x + 2, mid, rx, ans);
    }

}
long long getFirstIndexSmallerThan_V_Between_L_R(int v, int l, int r){
    int ans = -1;
    getFirstIndexSmallerThan_V_Between_L_R(v, l, r, 0, 0, size, ans);
    return ans;
}
```

Get Last Index of an element in range less than a value (v)

```
// Use it to get the last index that is smaller than v in the range [l, r) ___ (V not included)
// Make sure that segmentTree built with min operation
void getLastIndexSmallerThan_V_Between_L_R(int v, int l, int r, int x, int lx, int rx, int &ans){
    if(~ans || values[x] > v) return;
    if(l ⩾ rx || lx ⩾ r) return;
    if(rx - lx == 1){
        if(values[x] < v) ans = lx; // Make "<" ⟶ "⩽" if you want to include the value v

        return;
    }

    int mid = (rx + lx) / 2;

    // Make "<" ⟶ "⩽" if you want to include the value v
    if(values[2 * x + 2] < v){
        getLastIndexSmallerThan_V_Between_L_R(v, l, r, 2 * x + 2, mid, rx, ans);
    }

    // Make "<" ⟶ "⩽" if you want to include the value v
    if(values[2 * x + 1] < v  && !~ans){
        getLastIndexSmallerThan_V_Between_L_R(v, l, r, 2 * x + 1, lx, mid, ans);
    }

}
long long getLastIndexSmallerThan_V_Between_L_R(int v, int l, int r){
    int ans = -1;
    getLastIndexSmallerThan_V_Between_L_R(v, l, r, 0, 0, size, ans);
    return ans;
}
```

President Segment Tree

```cpp
struct PresidentSegmentTree{
public:
    PresidentSegmentTree(int n){
        sz = n;
        roots.push_back(new node(0));
    }
    void set(int idx , int val , int version = 1){
        version--;
        roots[version] = set(idx , val , roots[version] , 1 , sz);
    }
    void newVersion(int version){
        version--;
        roots.push_back(roots[version]);
    }
    ll query(int l , int r , int version){
        version--;
        return query(l , r , roots[version] , 1 , sz);
    }
private:
    struct node{
        ll val;
        node *l , *r;
        node(ll v){
            val = v;
            l = r = this;
        }
        node(node *_l , node *_r){
            val = 0;
            l = _l;
            r = _r;
            if(l) val+=l->val;
            if(r) val+=r->val;
        }
    };
    vector<node*> roots;
    int sz;
    node* set(int idx , int val , node *cur , int l , int r){
        if(l == r) return new node(val);
        int md = l + (r-l) / 2;
        if(idx <= md) return new node(set(idx , val , cur->l , l , md) , cur->r);
        return new node(cur->l , set(idx , val , cur->r , md+1 , r));
    }
    ll query(int l , int r , node *cur , int lx , int rx){
        if(lx >= l && rx <= r) return cur->val;
        if(lx > r || rx < l) return 0;
        int md = lx + (rx-lx) / 2;
        return query(l , r , cur->l , lx , md) + query(l , r , cur->r , md+1 , rx);
    }
};
```

Ordered Set & Map

```cpp
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template<typename K, typename V, typename Comp = less<K>>
using ordered_map = tree<K, V, Comp, rb_tree_tag, tree_order_statistics_node_update>;
template<typename K, typename Comp = less<K>>
using ordered_set = ordered_map<K, null_type, Comp>;

template<typename K, typename V, typename Comp = less_equal<K>>
using ordered_multimap = tree<K, V, Comp, rb_tree_tag, tree_order_statistics_node_update>;
template<typename K, typename Comp = less_equal<K>>
using ordered_multiset = ordered_multimap<K, null_type, Comp>;

```

Kadane

```cpp
int kadane(const vector<int>& arr) {
    int max_current = arr[0];
    int max_global = arr[0];

    for (size_t i = 1; i < arr.size(); ++i) {
        max_current = max(arr[i], max_current + arr[i]);
        max_global = max(max_global, max_current);
    }

    return max_global;
}
```

Eclidian GCD

```cpp
void move(int &a, int &b, int q){
    int n = a- q*b;
    a = b;
    b = n;
}

int eGCD(int a, int b, int &x0, int &y0){
    int r0 = a, r1 = b;
    int x1, y1;
    x1 = y0 = 0;
    x0 = y1 = 1;

    while(r1 != 0){
        int q = r0 / r1;

        move(r0, r1, q);
        move(x0, x1, q);
        move(y0, y1, q);
    }
    return r0;
}
```

Factorize

```cpp
map<int, int> factors;
void factorize(long long n)
{
    factors.clear();

    for (int i = 2; i * i <= n; ++i) {
        int power = 0;
        while (n % i == 0) {
            ++power;
            n /= i;
        }
        if (power > 0) factors[i] = power;
    }
    if (n != 1) factors[n] = 1;
}

```

Is Prime 1e18

```cpp
inline long long modMul(const long long &a,const long long &b, const long long moood){
    return __int128(a) * (b % moood + moood) % moood;
}

long long power(long long a, long long b, long long moood) {
    long long res = 1;
    a %= moood;
    while (b) {
        if (b & 1) res = modMul(res, a, moood);
        a = modMul(a, a, moood);
        b >>= 1;
    }
    return res;
}

// Miller-Rabin test for a single base
bool millerTest(ll d, ll n, ll a) {
    ll x = power(a, d, n);
    if (x == 1 || x == n - 1)
        return true;

    while (d != n - 1) {
        x = modMul(x, x, n);
        d *= 2;

        if (x == 1) return false;
        if (x == n - 1) return true;
    }

    return false;
}

// Miller-Rabin Primality Test (deterministic for < 2^64)
bool isPrime(ll n) {
    if (n <= 1) return false;
    if (n <= 3) return true;
    if (n % 2 == 0) return false;

    ll d = n - 1;
    while (d % 2 == 0)
        d /= 2;

    // Deterministic bases for 64-bit integers
    vector<ll> bases = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

    for (ll a : bases) {
        if (a >= n) break;
        if (!millerTest(d, n, a))
            return false;
    }

    return true;
}
```

Factorize 1e18

\\ Continue //

```cpp
inline long long modMul(const long long &a,const long long &b, const long long moood){
    return __int128(a) * (b % moood + moood) % moood;
}

long long power(long long a, long long b, long long moood) {
    long long res = 1;
    a %= moood;
    while (b) {
        if (b & 1) res = modMul(res, a, moood);
        a = modMul(a, a, moood);
        b >>= 1;
    }
    return res;
}

// Miller-Rabin test for a single base
bool millerTest(ll d, ll n, ll a) {
    ll x = power(a, d, n);
    if (x == 1 || x == n - 1)
        return true;

    while (d != n - 1) {
        x = modMul(x, x, n);
        d *= 2;

        if (x == 1) return false;
        if (x == n - 1) return true;
    }

    return false;
}

// Miller-Rabin Primality Test (deterministic for < 2^64)
bool isPrime(ll n) {
    if (n <= 1) return false;
    if (n <= 3) return true;
    if (n % 2 == 0) return false;

    ll d = n - 1;
    while (d % 2 == 0)
        d /= 2;

    // Deterministic bases for 64-bit integers
    vector<ll> bases = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37};

    for (ll a : bases) {
        if (a >= n) break;
        if (!millerTest(d, n, a))
            return false;
    }

    return true;
}
```

```cpp
// Pollard's Rho algorithm
long long pollards_rho(long long n) {
    if (n % 2 == 0) return 2;
    long long x = 2, y = 2, d = 1, c = rand() % (n - 1) + 1;
    while (d == 1) {
        x = (modMul(x, x, n) + c) % n;
        y = (modMul(y, y, n) + c) % n;
        y = (modMul(y, y, n) + c) % n;
        d = gcd((x > y) ? x - y : y - x, n);
        if (d == n) return pollards_rho(n);
    }
    return d;
}

// Recursive factorization
void factor(long long n, map<long long, int>& factors) {
    if (n == 1) return;
    if (isPrime(n)) {
        ++factors[n];
        return;
    }
    long long d = pollards_rho(n);
    factor(d, factors);
    factor(n / d, factors);
}

map<long long, int> factorize(long long n){
    map<long long, int> factors;

    if(n == 1) {
        factors[0] = -1;
    }
    else{
        factor(n, factors);
    }

    return factors;
}
```

LCA

```cpp
struct LCA {
    int n;
    const int LOG = 20;
    vector<vector<pair<int, int>>> adj;
    vector<vector<int>> up, mn, mx;
    vector<int> depth, total;

    LCA(int _n) : n(_n) {
        adj.resize(n);
        total.resize(n);
        up.assign(n, vector<int>(LOG));
        mn.assign(n, vector<int>(LOG, INT_MAX));
        mx.assign(n, vector<int>(LOG, 0));
        depth.assign(n, 0);
    }

    void addEdge(int u, int v, int w = 0) {
        adj[u].emplace_back(v, w);
        adj[v].emplace_back(u, w);
    }

    void dfs(int node, int parent) {
        total[node] += total[parent];

        for (auto [child, weight] : adj[node]) {
            if (child == parent) continue;

            depth[child] = depth[node] + 1;
            up[child][0] = node;
            mn[child][0] = weight;
            mx[child][0] = weight;

            total[child] += weight;
            for (int j = 1; j < LOG; j++) {
                up[child][j] = up[up[child][j - 1]][j - 1];
                mn[child][j] = min(mn[child][j - 1], mn[up[child][j - 1]][j - 1]);
                mx[child][j] = max(mx[child][j - 1], mx[up[child][j - 1]][j - 1]);
            }

            dfs(child, node);
        }
    }

    void build(int root = 0) {
        up[root][0] = root;
        dfs(root, root);
    }
```

\\ Continue //

```cpp

    int getLCA(int a, int b) {
        if (depth[a] < depth[b]) swap(a, b);
        int diff = depth[a] - depth[b];
        for (int i = 0; i < LOG; i++) {
            if (diff & (1 << i))
                a = up[a][i];
        }

        if (a == b) return a;

        for (int i = LOG - 1; i >= 0; i--) {
            if (up[a][i] != up[b][i]) {
                a = up[a][i];
                b = up[b][i];
            }
        }

        return up[a][0];
    }

    int getKthAncestor(int node, int k) {
        if(k > depth[node]){
            return -1;
        }

        for(int i=0; i < LOG; i++){
            if(k & (1 << i)){
                node = up[node][i];
            }
        }
        return node;
    }

    pair<int, int> getKthMinMax(int node, int k) {
        if (k > depth[node]) return {-1, -1};
        pair<int, int> ret = {INT_MAX, 0};

        for (int i = 0; i < LOG; i++) {
            if (k & (1 << i)) {
                ret.first = min(ret.first, mn[node][i]);
                ret.second = max(ret.second, mx[node][i]);
                node = up[node][i];
            }
        }

        return ret;
    }

    // Returns {minEdge, maxEdge} on path between a and b
    pair<int, int> getMinMaxEdgeOnPath(int a, int b) {
        int lca = getLCA(a, b);
        auto aa = getKthMinMax(a, depth[a] - depth[lca]);
        auto bb = getKthMinMax(b, depth[b] - depth[lca]);

        return {min(aa.first, bb.first), max(aa.second, bb.second)};
    }
};
```

NUMS

```cpp
constexpr int MOD = 1e9 + 7;
constexpr int MAX = 3 * 1e6 + 5;

long long fact[MAX], invFact[MAX];

inline long long modAdd(long long a, long long b) {
    return (a + b + 2 * MOD) % MOD;
}

inline long long modSub(long long a, long long b) {
    return (a - b + MOD) % MOD;
}

inline long long modMul(const long long &a,const long long &b){
    return (a % MOD + MOD) * (b % MOD + MOD) % MOD;
}

long long power(long long a, long long b) {
    long long res = 1;
    a %= MOD;
    while (b) {
        if (b & 1) res = modMul(res, a);
        a = modMul(a, a);
        b >>= 1;
    }
    return res;
}

inline long long modInverse(long long a) {
    return power(a, MOD - 2);
}

inline long long modDivide(long long a, long long b) {
    return modMul(a, modInverse(b));
}


inline long long nCr(int n, int r) {
    if (r > n || r < 0) return 0;
    return modMul(fact[n], modMul(invFact[r], invFact[n - r]));
}

inline long long nPr(int n, int r) {
    if (r > n || r < 0) return 0;
    return modMul(fact[n], invFact[n - r]);
}

inline long long catalanNumber(int n){
    return nCr(2 * n, n) / (n + 1);
}

int preprocess = []() {
    fact[0] = invFact[0] = 1;
    for (int i = 1; i < MAX; i++) {
        fact[i] = modMul(fact[i - 1], i);
    }
    invFact[MAX - 1] = modInverse(fact[MAX - 1]);
    for (int i = MAX - 2; i >= 1; i--) {
        invFact[i] = modMul(invFact[i + 1], i + 1);
    }
    return 0;
}();
```

Len. of the longest inc. subseq

```cpp
#define ll long long
struct FenwickTree{
    int N;vector<int>bit;
    FenwickTree(int n,int init=0)
    {
        N=n+1;
        bit=vector<int>(n+1,init);
    }
    void update(int idx,int val)
    {
        while(idx<N)
        {
            bit[idx]=max(bit[idx],val);
            idx+=idx&-idx;
        }
    }
    int query(int idx)
    {
        int ret=0;
        while(idx>0)
        {
            ret=max(ret,bit[idx]);
            idx-=idx&-idx;
        }
        return ret;
    }
};
signed main()
{
    ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0);
    int n;cin>>n;
    int pos[n+1];
    for(int i=1;i<=n;i++)
    {
        int x;cin>>x;
        pos[x]=i;
    }
    pair<int,int>s[n];
    for(int i=0;i<n;i++)
    {
        cin>>s[i].first;
        s[i].second=i+1;
    }
    sort(s,s+n,[&](pair<int,int>&i,pair<int,int>&j){
        return pos[i.first]<pos[j.first];
    });
    FenwickTree tree(n);
    for(int i=0;i<n;i++)
    {
        int pre=tree.query(s[i].second);
        tree.update(s[i].second,pre+1);
    }
    cout<<tree.query(n);
}
```

MST

```cpp
class DSU {

public:
    vector <int> par;
    vector<vector<int>> group;
    DSU(int __N) {
        par.resize(__N + 1);
        group.resize(__N + 1);
        for(int i = 0; i <= __N; i++) {
            group[i].push_back(i);
            par[i] = i;
        }
    }

    int find(int u) {
        if(par[u] == u) return u;
        return par[u] = find(par[u]);
    }

    bool unite(int s, int t) {
        t = find(t), s = find(s);

        if(s == t) return false;
        if(group[t].size() < group[s].size())
            swap(t, s);

        for(auto i: group[s])
            group[t].push_back(i);

        group[s].clear();
        par[s] = t;

        return true;
    }

    bool connected(int x, int y) {
        return find(x) == find(y);
    }

    vector<vector<int>> boundries(){
        map<int, vector<int>> tmp;
        vector<vector<int>> ret;
        for(int i = 1; i < group.size()-1; i++){
            tmp[find(i)].push_back(i);
        }
        for(auto &[a, b]: tmp) ret.push_back(b);
        return ret;
    }
};
```

\\ Continue //

```cpp
struct Edge {
    int u, v, weight, idx;
    bool operator<(const Edge& other) const {
        return weight < other.weight;
    }
};

class MST {
private:
    int n;
    std::vector<Edge> edges;

public:
    MST(int nodes){
        n = nodes;
    }

    void addEdge(int u, int v, int weight, int idx) {
        edges.push_back({u, v, weight, idx});
    }

    vector<int> kruskal() {
        std::sort(edges.begin(), edges.end());
        DSU dsu(n);
        int totalWeight = 0;

        vector<int> ret;
        for (const auto& edge : edges) {
            if (!dsu.connected(edge.u, edge.v)) {
                dsu.unite(edge.u, edge.v);
                totalWeight += edge.weight;
                ret.push_back(edge.idx);
                // std::cout << "Edge added: (" << edge.u << ", " << edge.v << ") → " << edge.weight << "\n";
            }
        }
        sort(BegEnd(ret));
        return ret;
    }
};
```

MEX

```cpp
int MEX(set<int> &MEX_set){
    int MEX_result = 0;
    vector<int>loop(MEX_set.begin() , MEX_set.end());
    for(int i = 0 ; i <(int)loop.size() ; i ++){
        MEX_result = loop[i] + 1;
        if(i != loop[i])return i ;
    }
    return MEX_result;
}
```