how long will you take to code a segment tree from scratch

Generic Segment Trees

20 min for complete code

10 min for modifying an existing template

- Priyansh Agarwal

What is a Generic Segment Tree

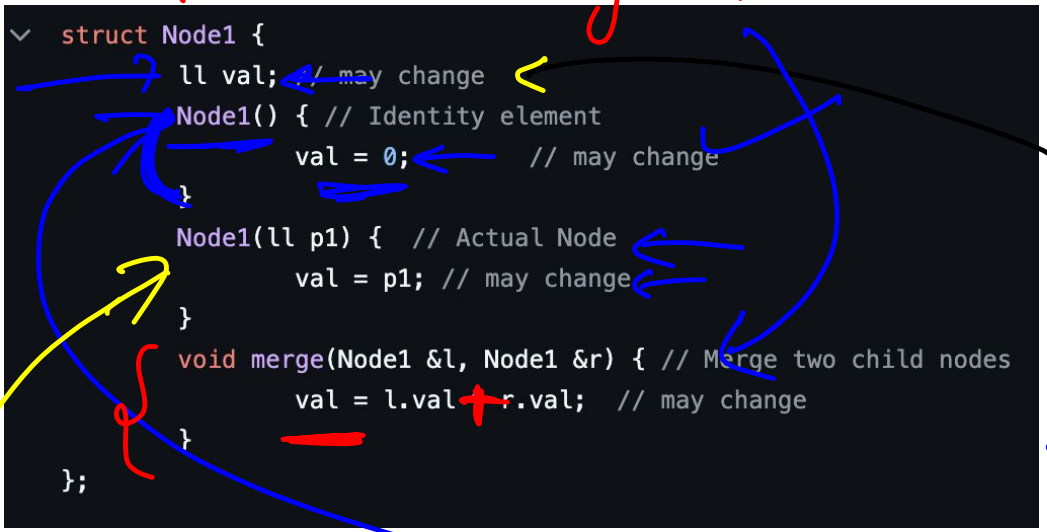
- A generic segment tree is a segment tree template that can be modified to fit any use case by making the minimum number of changes.
- It separates out the static part and the dynamic part making it easier to make edits where it is relevant. Eg: going down the tree for an update query is static in all segment trees but what happens on the leaf node in that update is dynamic.
- It also allows us to create multiple segment trees in the same code without having to rewrite the static part more than once.

* also helps avoid many typos and logical errors

Generic Segment Tree Template:

https://github.com/Priyansh19077/CP-Templates/blob/master/Range%20Queries/Segment_tree.cpp

Generic Segment Tree Node Struct



```
struct Node1 {
    ll val; // may change
    Node1() { // Identity element
        val = 0;    // may change
    }
    Node1(ll p1) {  // Actual Node
        val = p1; // may change
    }
    void merge(Node1 &l, Node1 &r) { // Merge two child nodes
        val = l.val + r.val;  // may change
    }
};
```

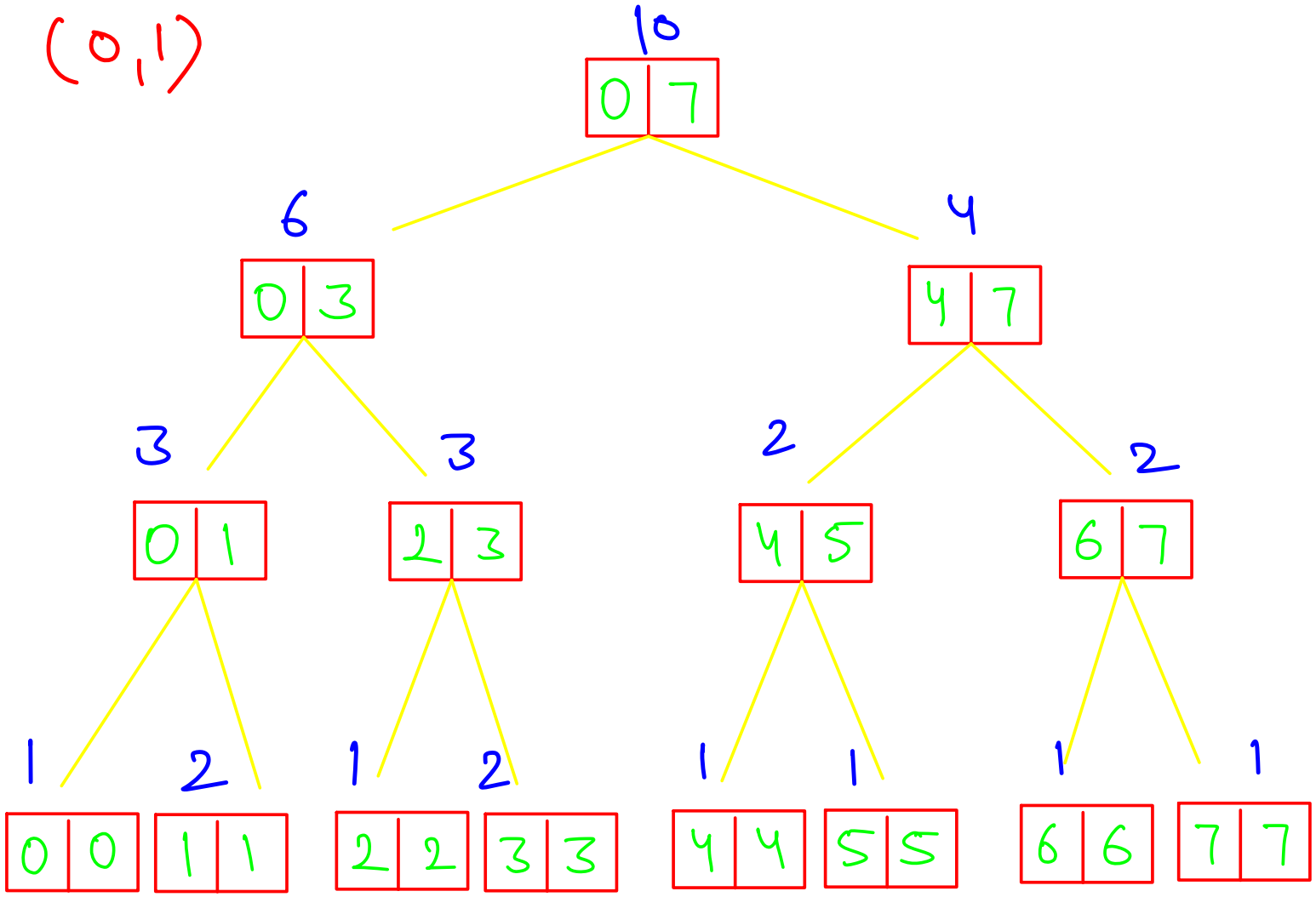
how to merge two children to get ans of parent

what we store in every segment tree node must be modifiable

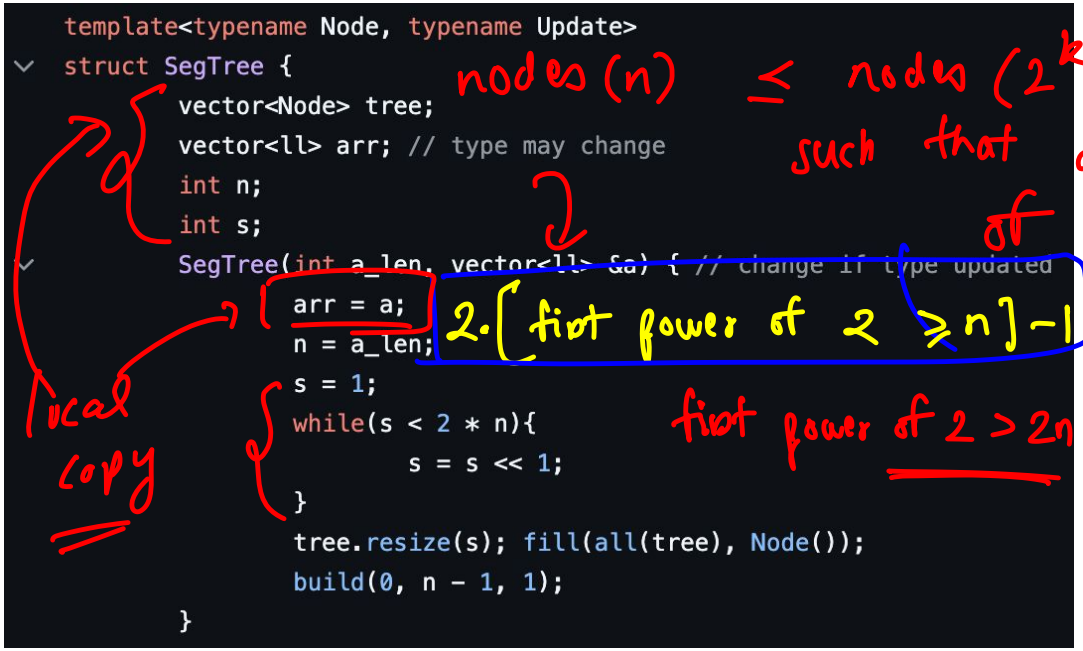
what to return when there is no overlap (default value)

how to populate all the information needed from the array index

query → (0,1)



Generic Segment Tree Constructor



```
template<typename Node, typename Update>
struct SegTree {
    vector<Node> tree;
    vector<ll> arr; // type may change
    int n;
    int s;
    SegTree(int a_len, vector<ll> &a) { // change if type updated
        arr = a;
        n = a_len;
        s = 1;
        while(s < 2 * n){
            s = s << 1;
        }
        tree.resize(s); fill(all(tree), Node());
        build(0, n - 1, 1);
    }
```

nodes(n) ≤ nodes(2^k) such that 2^k is the first power of $2 \geq n$

local copy

$2 \cdot [\text{first power of } 2 \geq n] - 1$

first power of $2 > 2n$

$\rightarrow$ nodes(n) < 4n

$n = 9 \rightarrow 36$

$2^k = 16$

$2(2^k) - 1 = 31$

$n = 15 \rightarrow 4n = 60$

$\rightarrow 31 \quad 32$

Generic Segment Tree Build Function

```
        build(0, n - 1, 1);
}
void build(int start, int end, int index)  // Never change this
{
        if (start == end)       {
                tree[index] = Node(arr[start]);
                return;
        }
        int mid = (start + end) / 2;
        build(start, mid, 2 * index);
        build(mid + 1, end, 2 * index + 1);
        tree[index].merge(tree[2 * index], tree[2 * index + 1]);
}
```

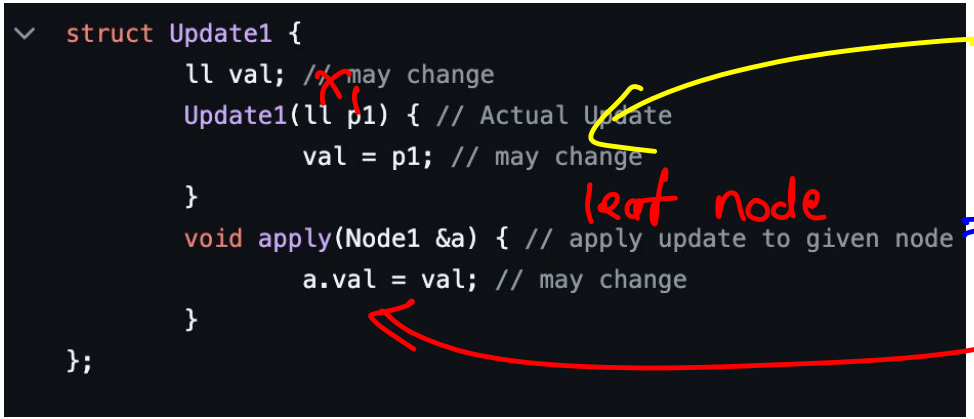
leaf node

↑ parameterized constructor

↑ merge function of node

update → a[i] → (a[i].x + y)

Generic Segment Tree Update Struct



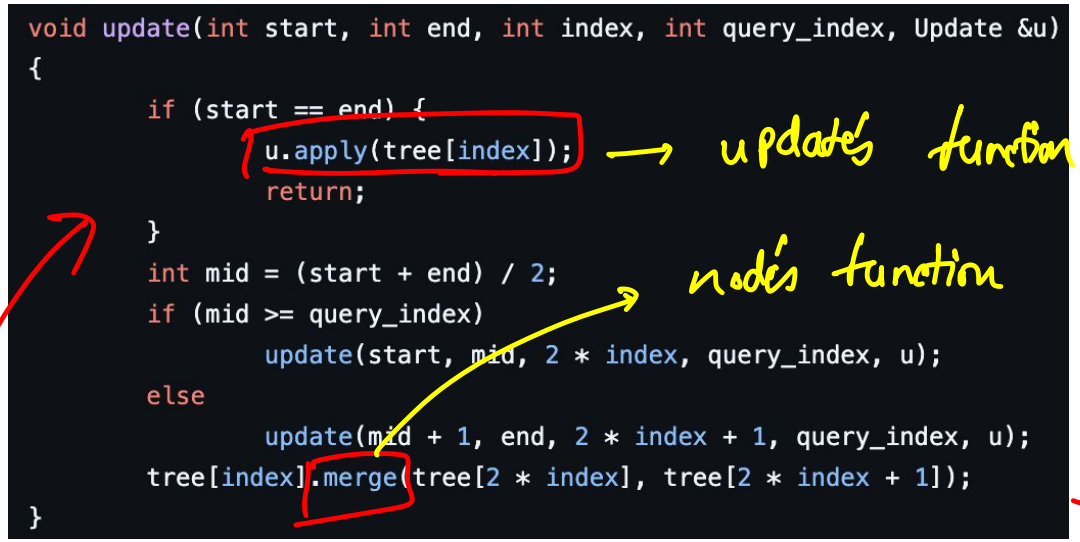
```
struct Update1 {
        ll val; // may change
        Update1(ll p1) { // Actual Update
                val = p1; // may change
        }
        void apply(Node1 &a) { // apply update to given node
                a.val = val; // may change
        }
};
```

leaf node

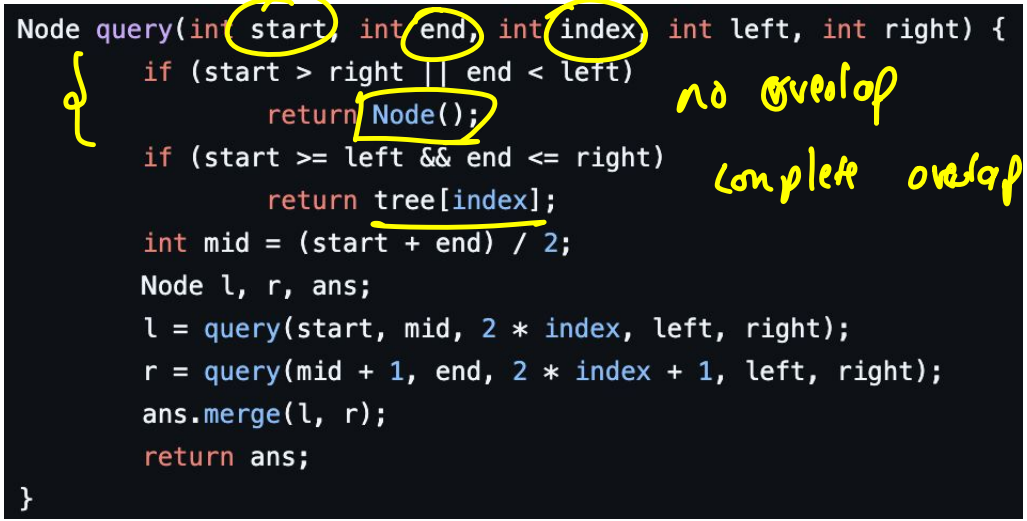
pass as many variables as we want

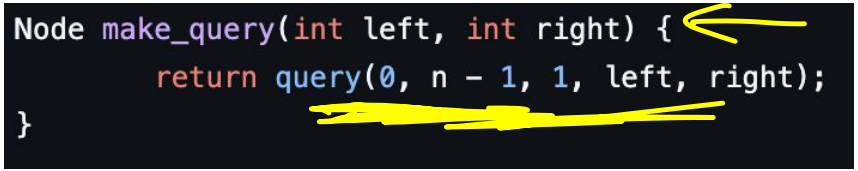
the way in which we update the leaf node

Generic Segment Tree Update Function



Generic Segment Tree Query Function



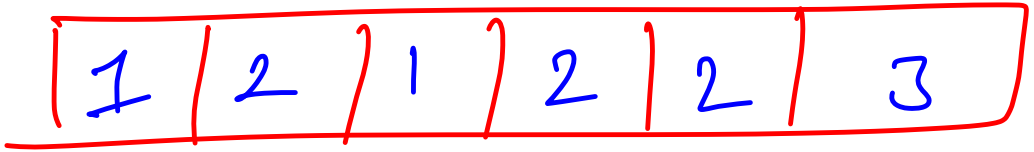


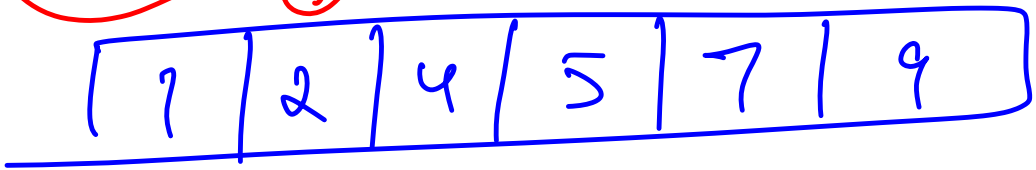
Problems

- Range Min
- Range Xor
- Range GCD

More Interesting Problems

- Range Second Min
- Range Min Index
- First element > X in Range





2nd min

lst min 2nd min

what to store in leaf → { val, inf }

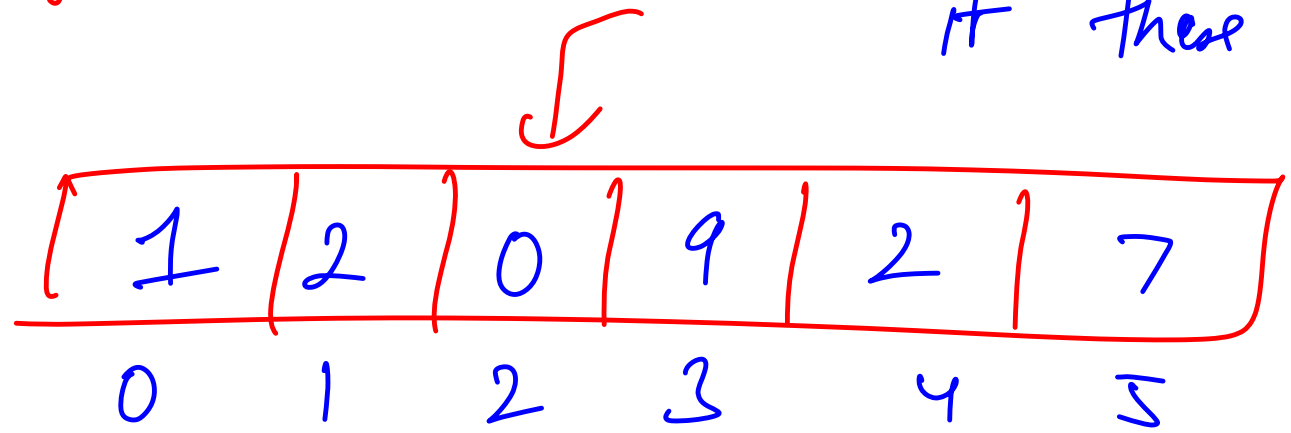
merge left and right

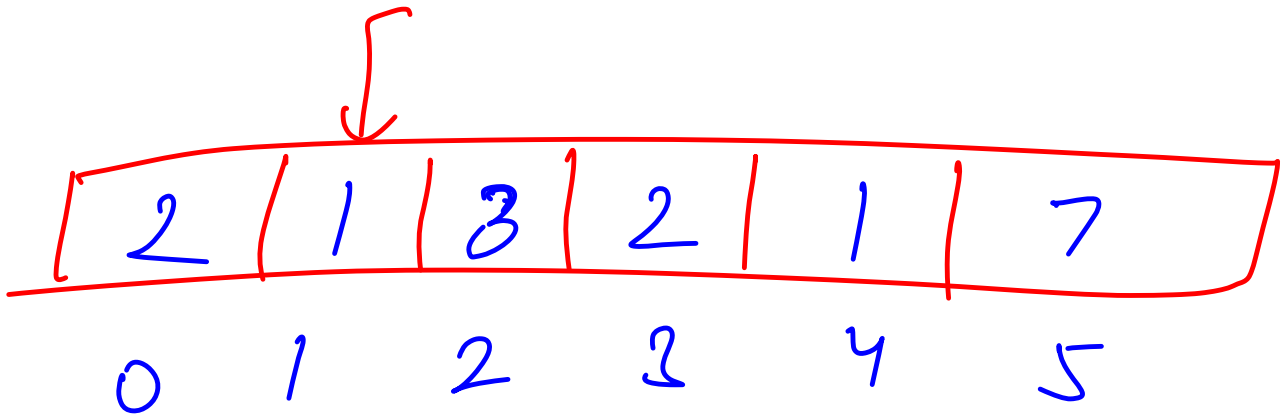
[l_1 l_2 r_1 r_2) → sort

lst min = arr[0], 2nd min = arr[1]

Default value → { leaf, inf }

Range min index (return index of 1st min if there are multiple instances of the min in the range)





store → val, index

merge → L.val, L.index R.val, R.index

if (L.val ≤ R.val) → L

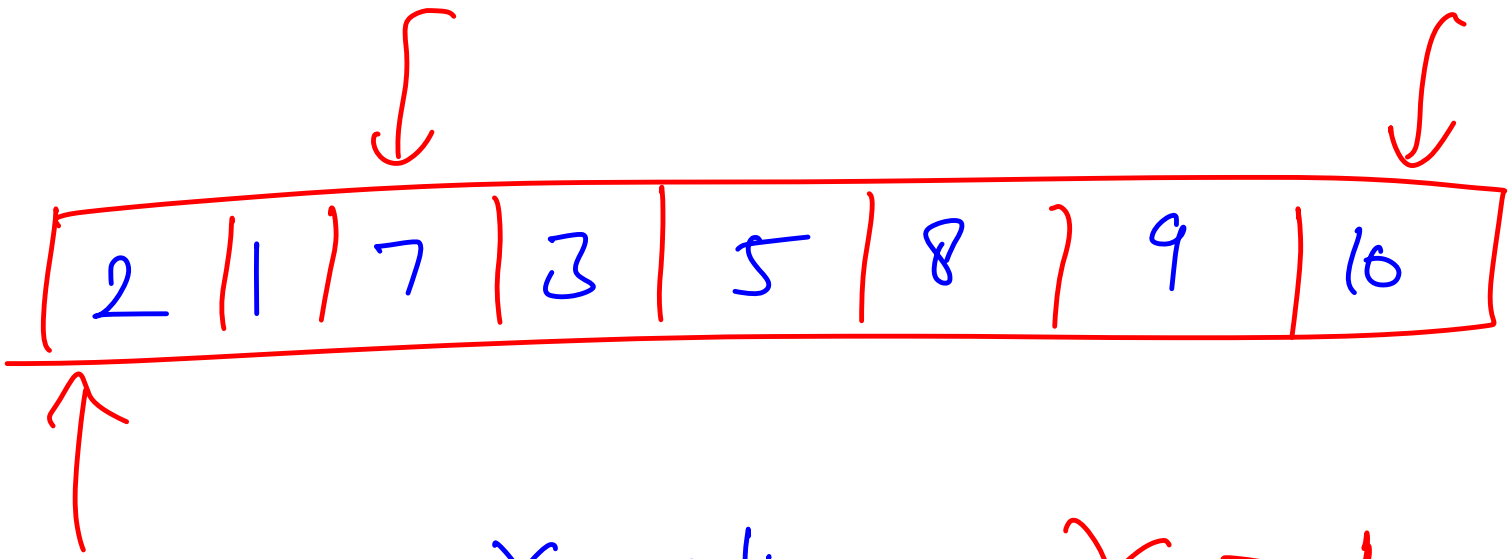
else ⟶ R

anything

Defaurt value ⟶ { inf , inf }

wont matter

1st element > x in the range



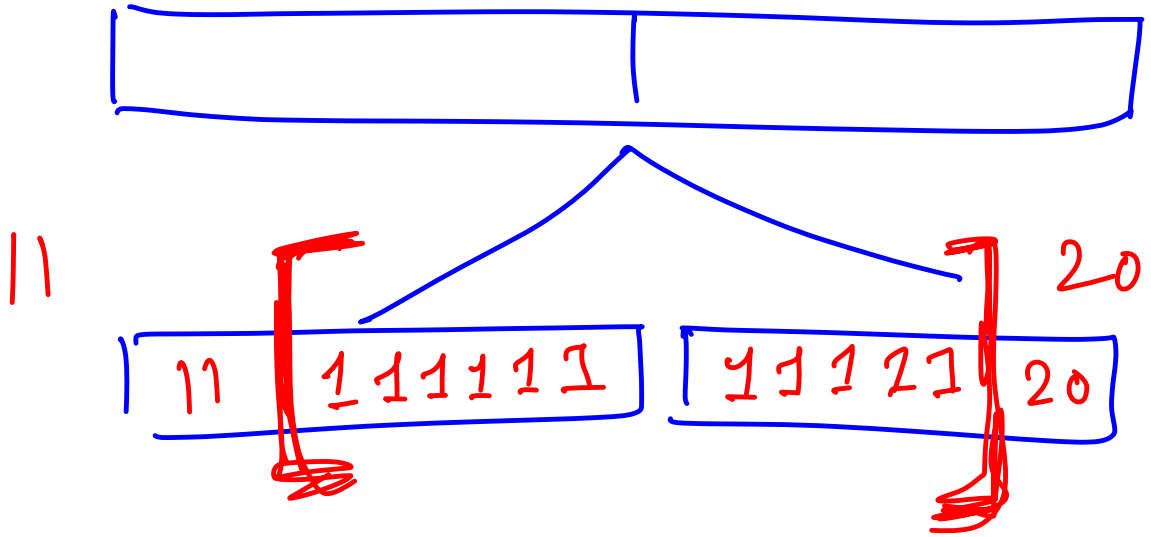
$x = 4$ $x = 1$

$=$

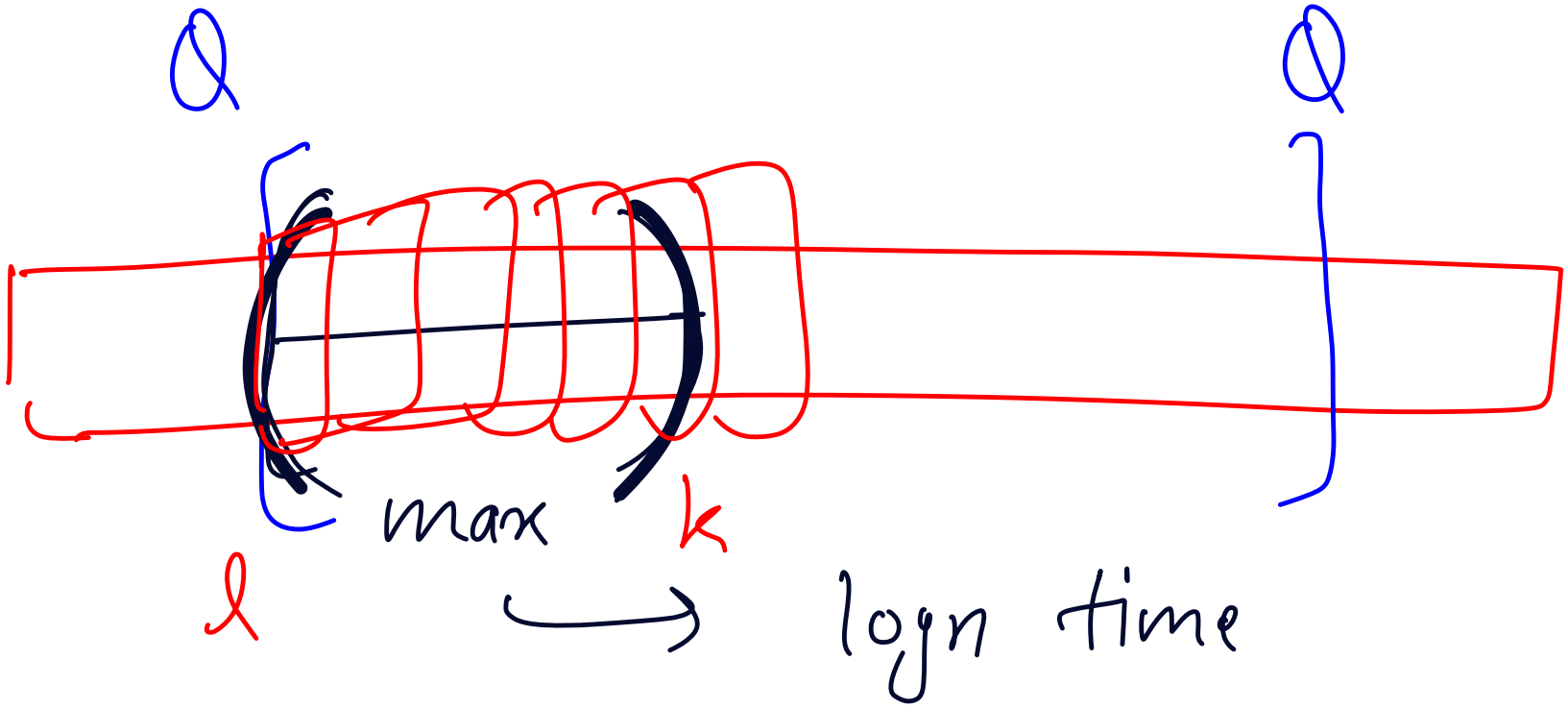
$x = 9$

store maximum at every node

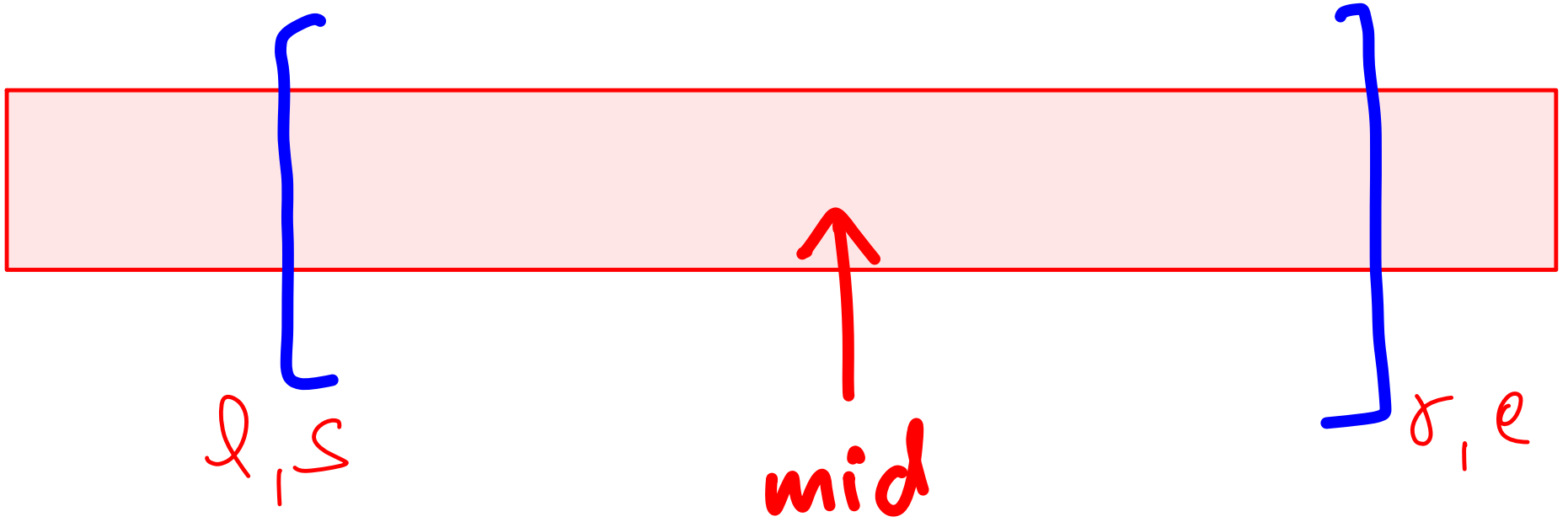
$x = 10$



Store the maximum at every node $X = 20$



$\max(l \text{ to } k) \leq \max(1 \text{ to } k+1)$



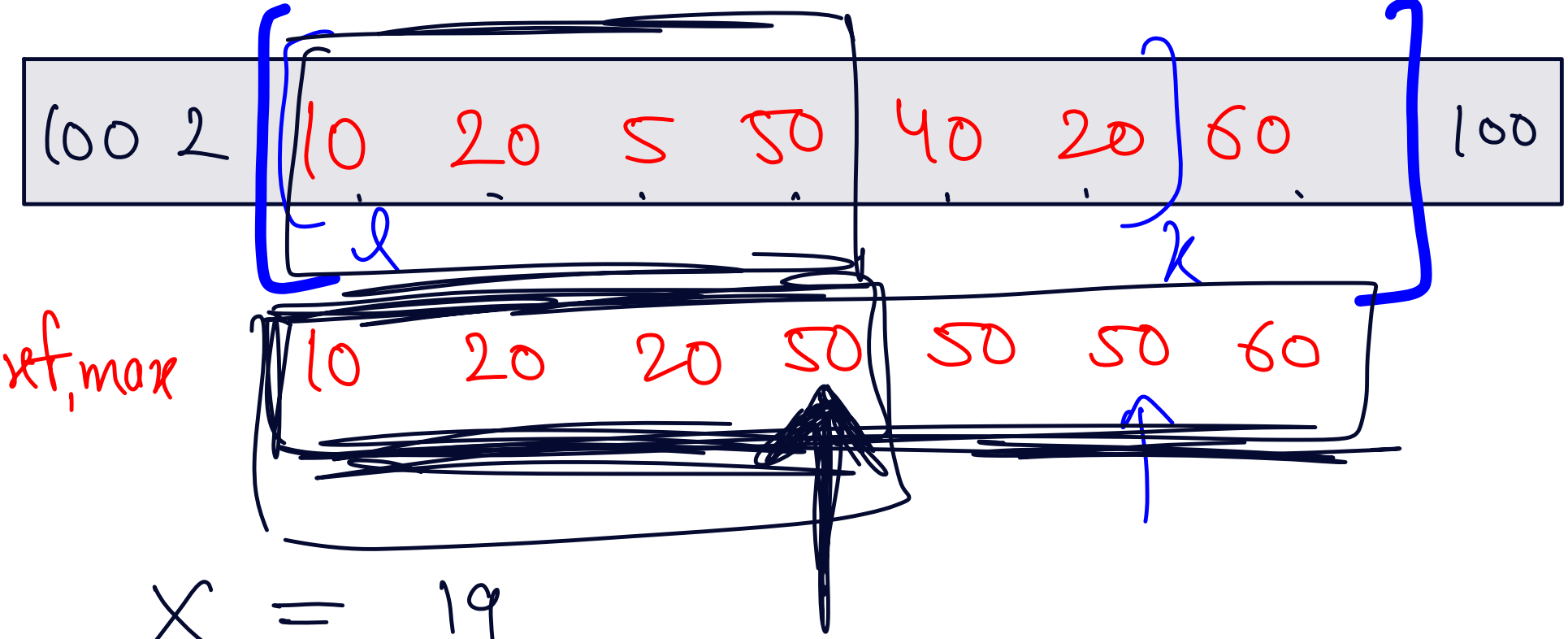
if (sq.make_query(l, mid) > x)

ans = mid ,

e = mid − 1

else

s = mid + 1



$X = 19$

$X = 55$