Lazy Propagation in Segment Trees

- Gaurish Baliga

Goal

- Range Updates
- Introduction to Lazy Propagation
- Implementation of Lazy Propagation
- Problem Solving
- Priyansh's General Purpose Segment Tree Template

1 l r → sum a[l ... r]

2 l r x → add x to all elements $a_l \ldots a_r$

$n, q \leq 3 \times 10^5$

Segment tree

~~TLE~~

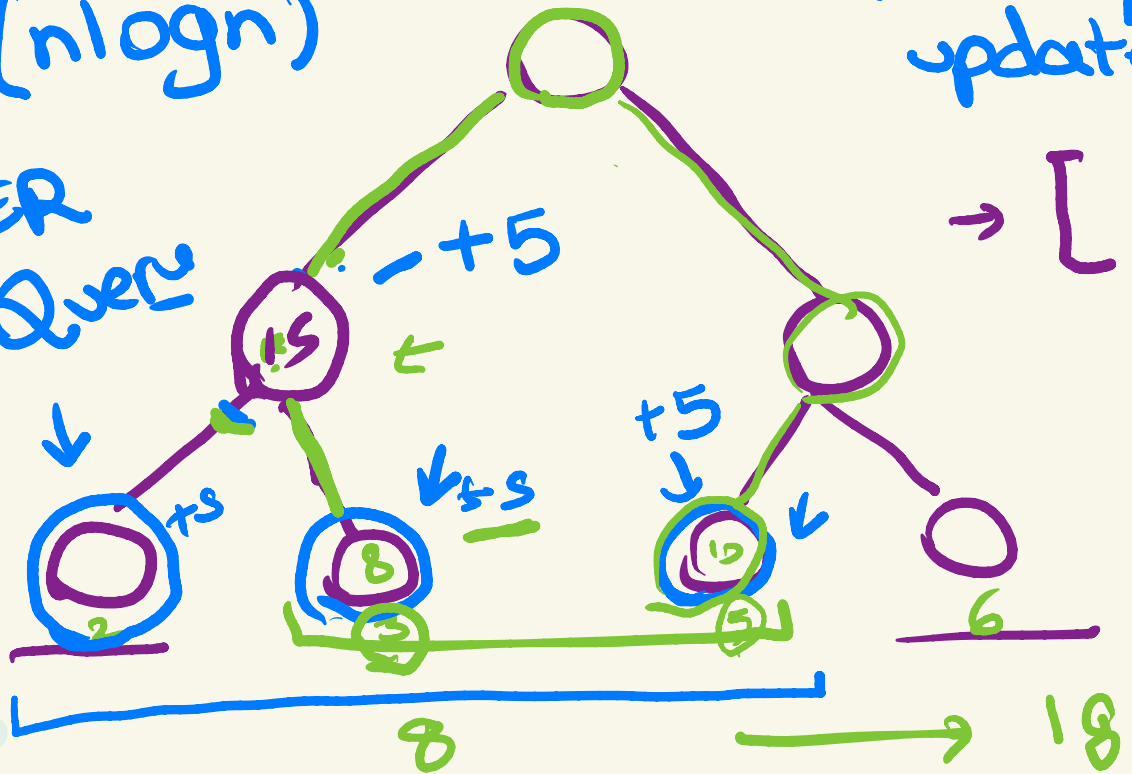
O(nlogn)

PER Query

→ every element, update it

→ [l, r]

↳ O(log₂n)



Use of Lazy Propagation?

- With a normal segment tree, we can perform range queries and point updates in $O(\log_2 N)$. With lazy propagation on segment tree, we can do both range updates and range queries in $O(\log_2 N)$

- Given an array of ***N*** elements and ***Q*** queries. Each i^{th} query is of 2 types
  - L_i R_i - Get the sum from indices L_i to R_i
  - L_i R_i X_i - Add X_i to all indices from L_i to R_i

How does it work?

- Range queries remain the same.

- For range updates, we lazily update the segment tree.
  - Eg: Add 5 to all indices from 3 to 8
  - Now, notice that we don't need to individually update each node, we can update on the ranges that contain them that they have been updated and propagate downwards later.

Illustration of Segment Tree

Now Consider adding 5 to all elements in range [3, 7]

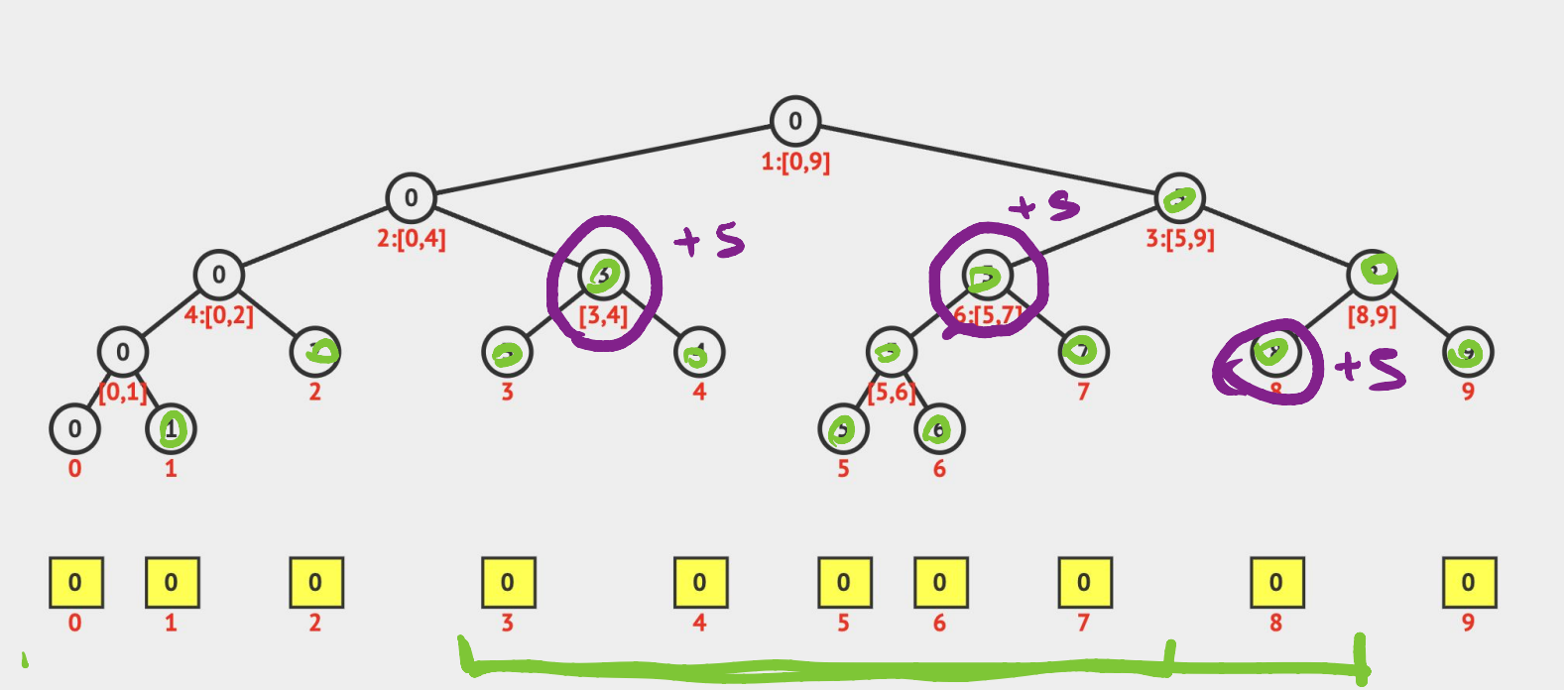
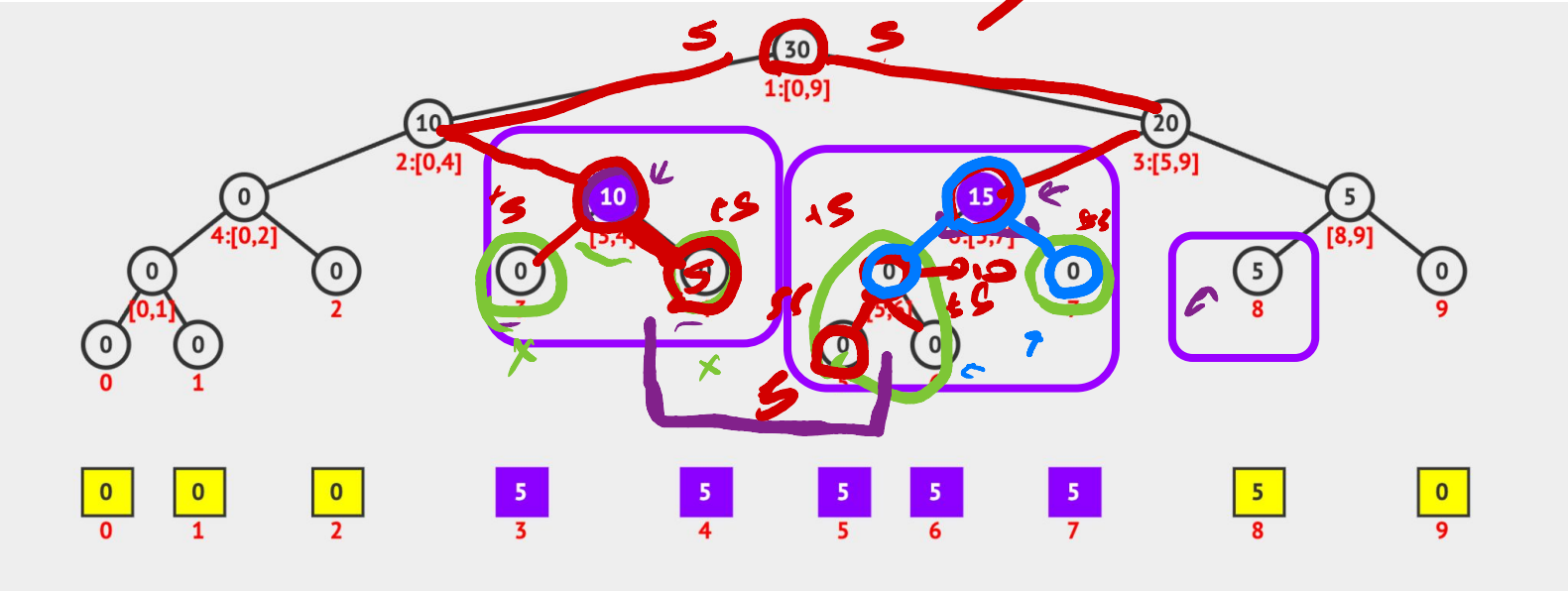


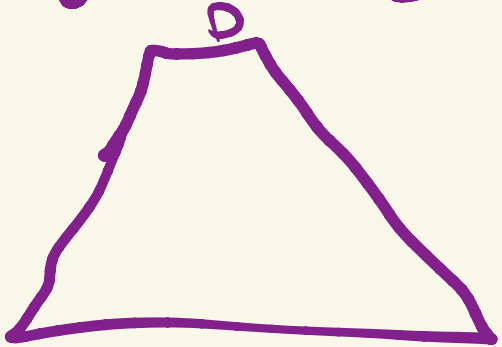
Illustration of Lazy Propagation



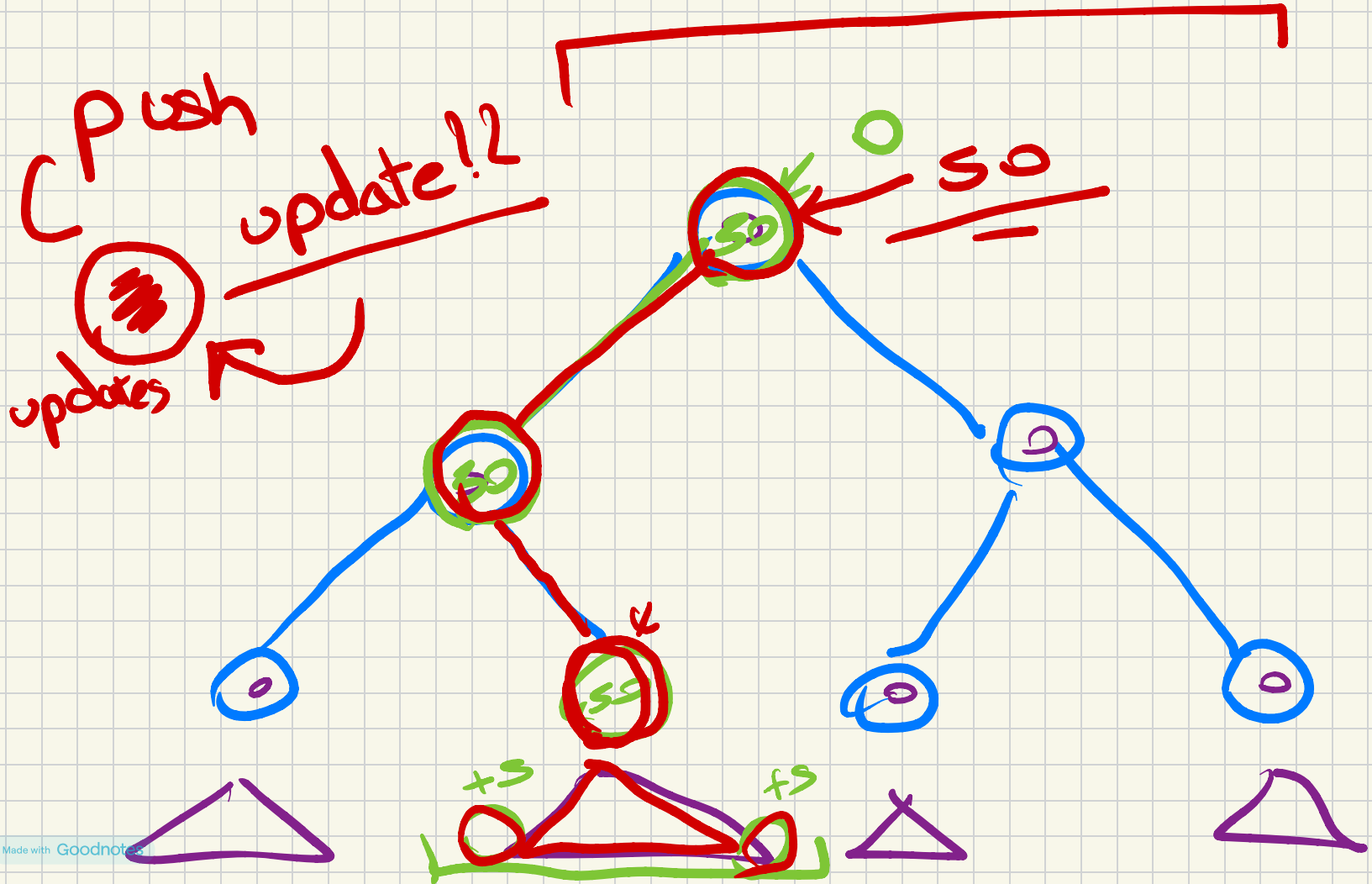
→ vector <int> lazy (4*n)

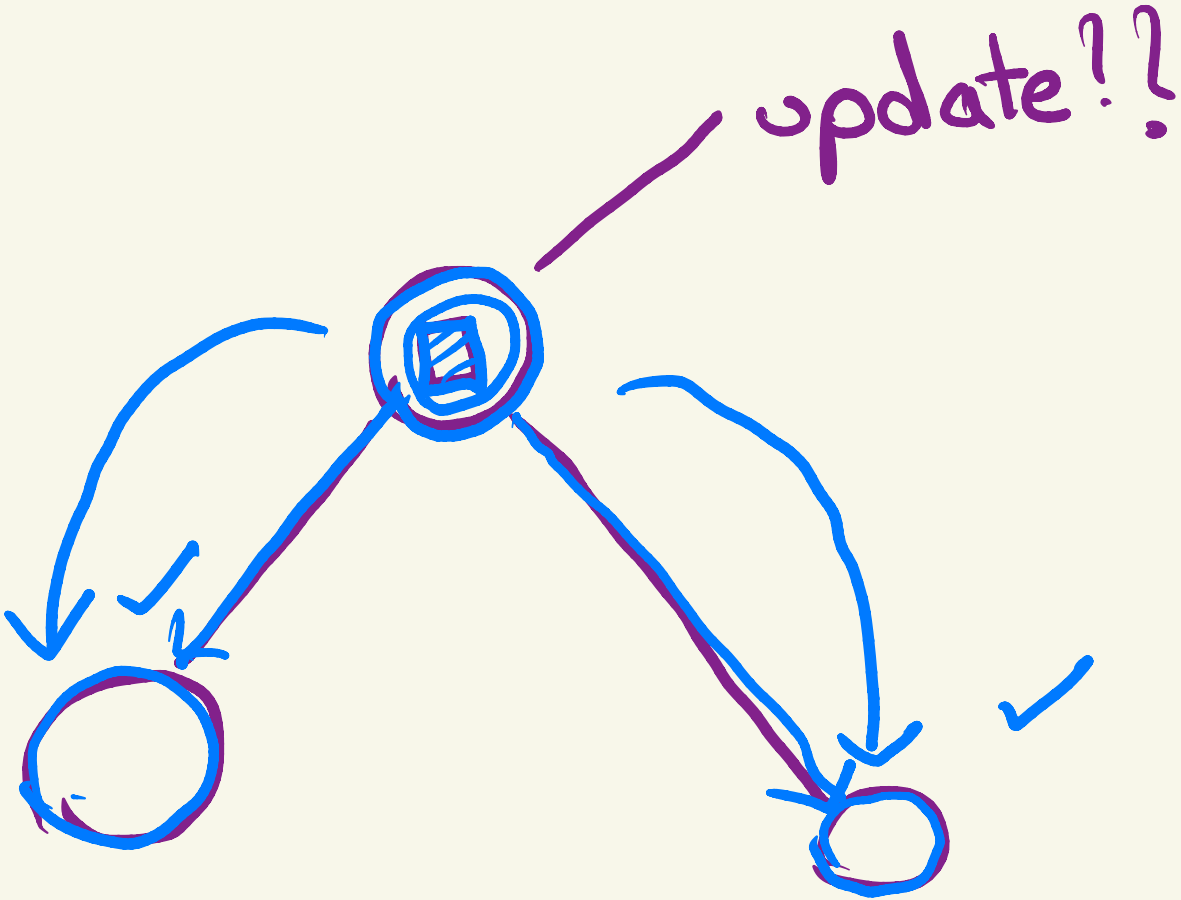
↳ updates

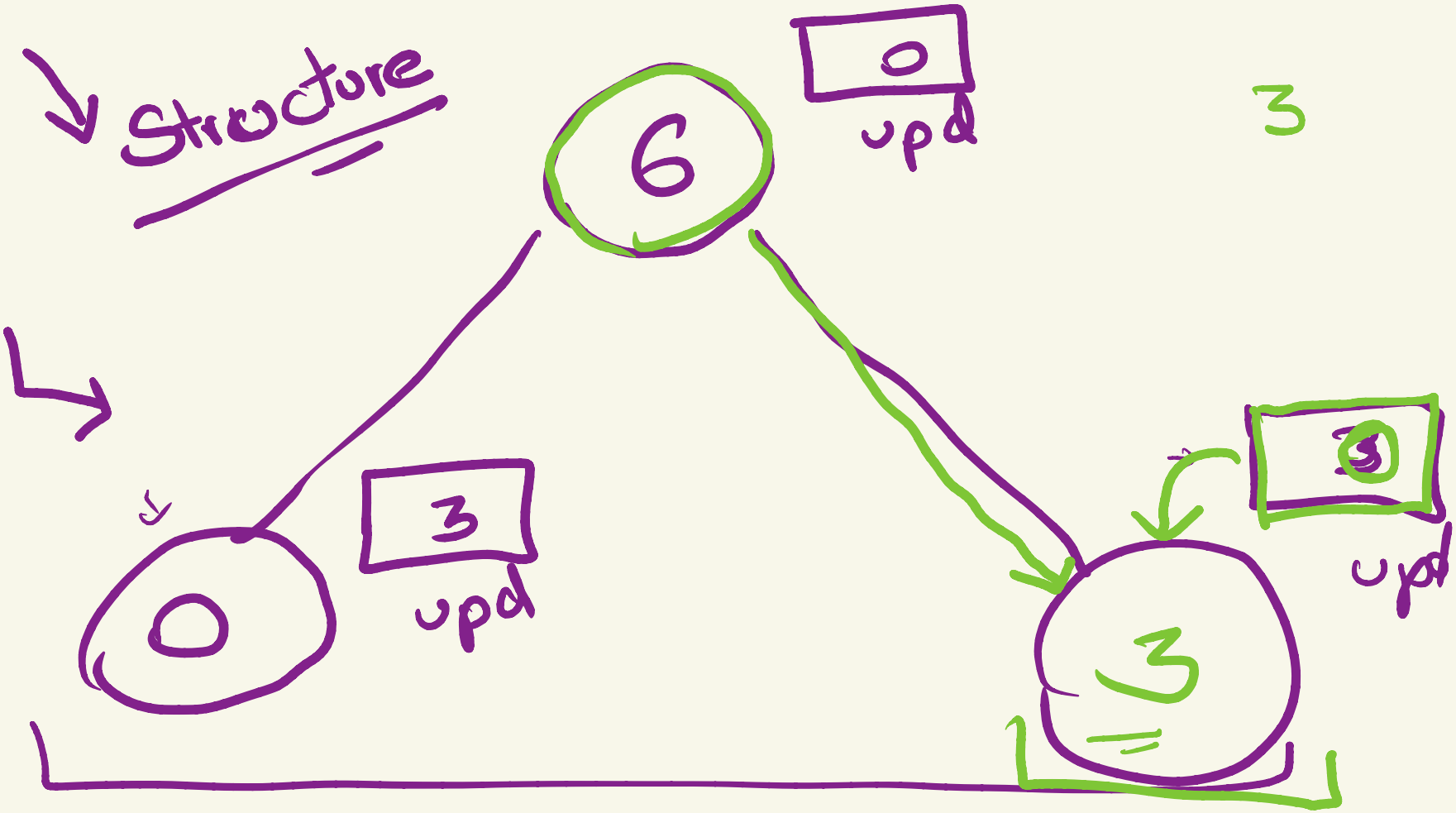
↳ lazy [node] = 0



↳ Lazy [node] ≠ 0


push
update!2
updates
0
50
50
50
55
+5
+5





Things to keep in mind!

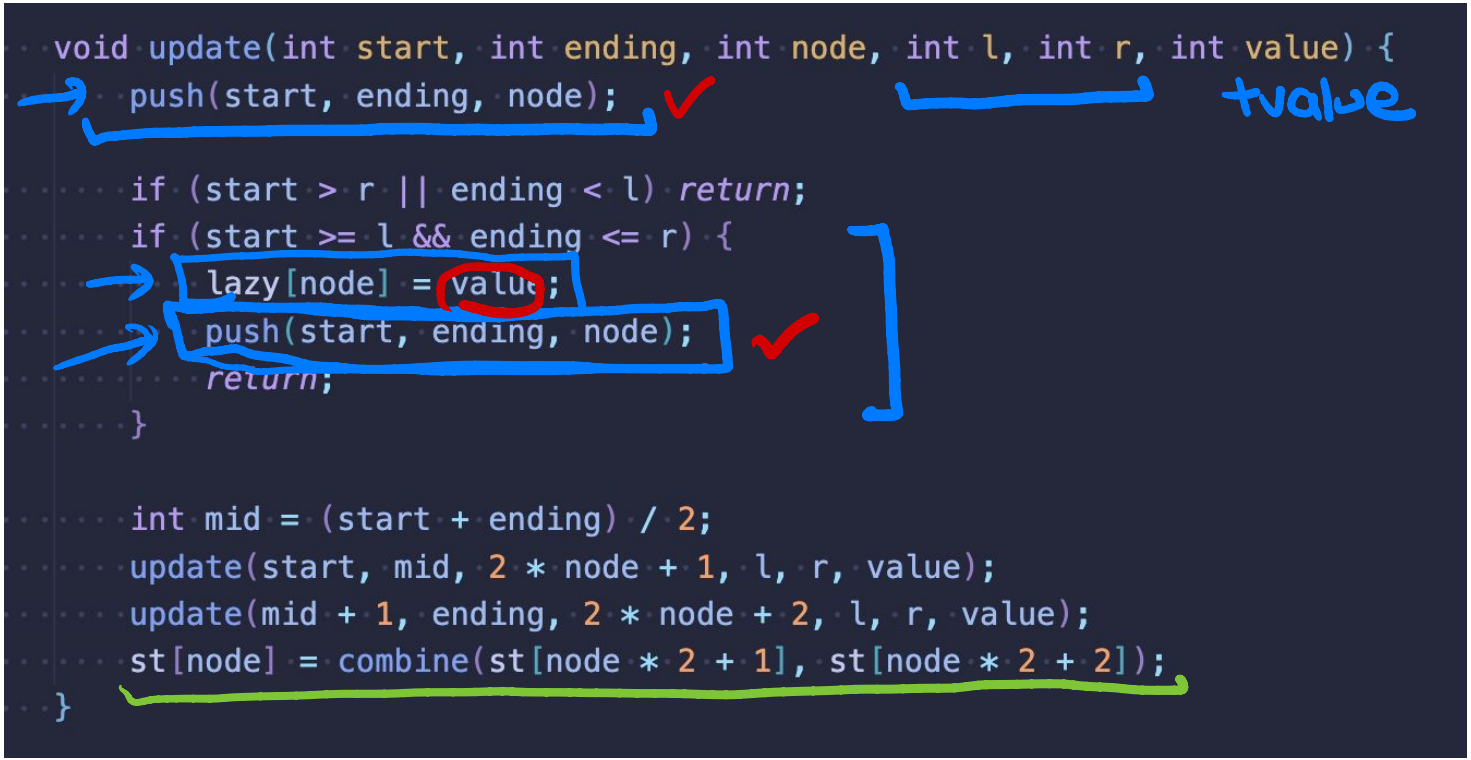
- Whenever you visit a node, it must be in the most updated state
- Whenever you want go to children, first update them and transfer the lazy updates down to them.
- Whenever you're performing a range update, return from the nodes which have either a complete overlap or no overlap.
- **Not all range updates and queries go together**
  - **Eg: Add X to all elements from L to R, get GCD of values from [L, R]**

Time Complexity

- Range queries - same as normal segment tree - $O(\log_2 N)$
- Range updates - now similar to range queries - $O(\log_2 N)$
- Building segment tree - 2 * normal segment tree - O(N)
- Overall slower than normal segment tree and takes more space.
  - Only slightly worse and doesn't matter in contests.

Link to Implementation: https://pastebin.com/rcBEDAPv

Implementation (Update Function)



```
void update(int start, int ending, int node, int l, int r, int value) {
    push(start, ending, node);

    if (start > r || ending < l) return;
    if (start >= l && ending <= r) {
        lazy[node] = value;
        push(start, ending, node);
        return;
    }

    int mid = (start + ending) / 2;
    update(start, mid, 2 * node + 1, l, r, value);
    update(mid + 1, ending, 2 * node + 2, l, r, value);
    st[node] = combine(st[node * 2 + 1], st[node * 2 + 2]);
}
```

Implementation (Push Function)

```
void push(int start, int ending, int node) {
    if (lazy[node]) {
        st[node] += (ending - start + 1) * lazy[node];
        if (start != ending){
            lazy[2 * node + 1] += lazy[node];
            lazy[2 * node + 2] += lazy[node];
        }
        lazy[node] = 0;
    }
}
```

Implementation (Query Function)

```
int query(int start, int ending, int l, int r, int node) {
    push(start, ending, node);

    if (start > r || ending < l) return 0;
    if (start >= l && ending <= r) return st[node];

    int mid = (start + ending) / 2;
    int q1 = query(start, mid, l, r, 2 * node + 1);
    int q2 = query(mid + 1, ending, l, r, 2 * node + 2);

    return combine(q1, q2);
}
```

```
segment tree tree;
tree.init(n);
tree.build(vec);

tree.query(l, r);
tree.update(l, r, x);
```

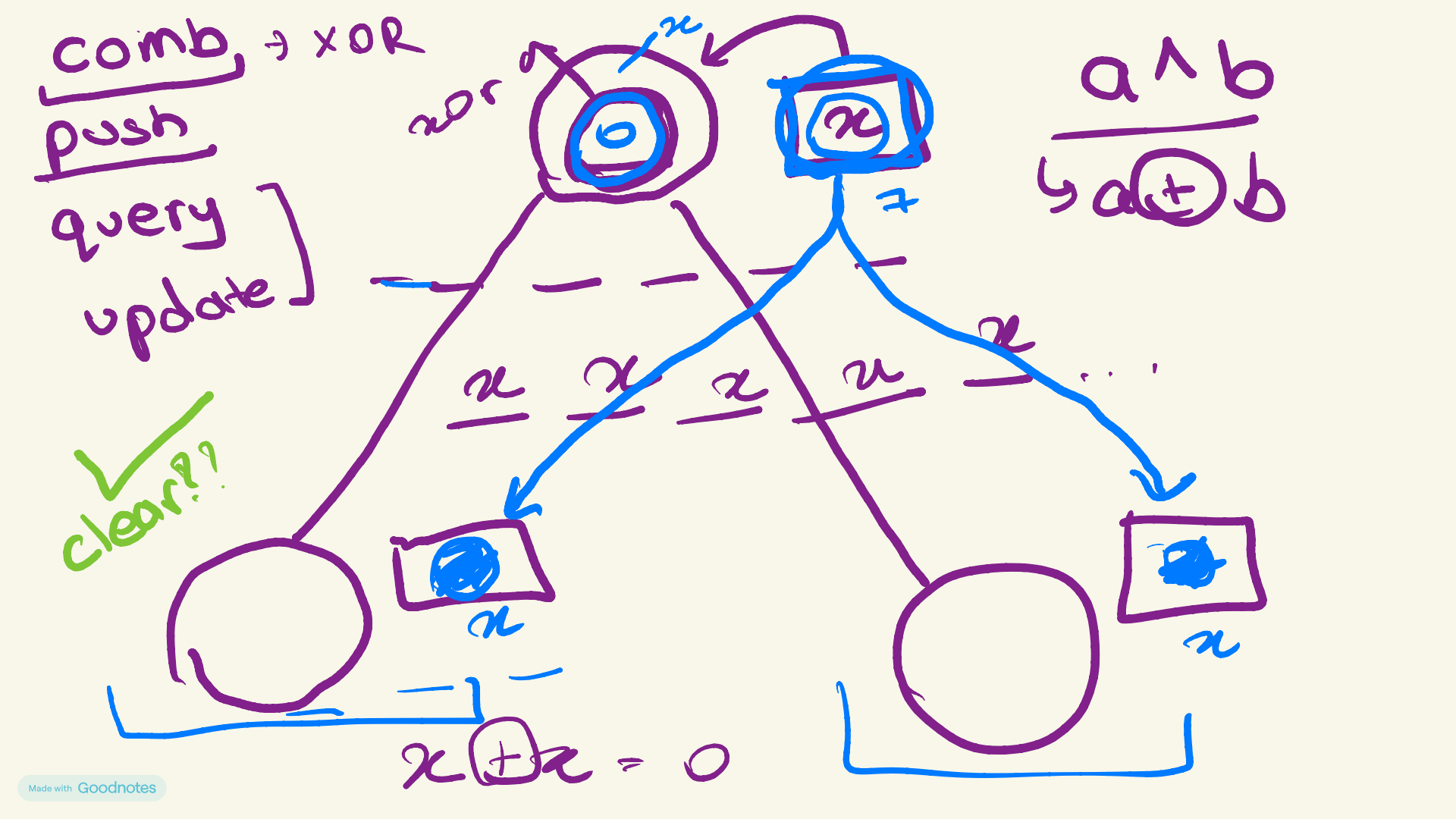
Problems

temp

- 1 L R X -> Add X to all elements from L to R
- 2 L R -> Return the sum of all elements from L to R

- 1 L R X -> Add X to all elements from L to R
- 2 L R -> Return the min of all elements from L to R

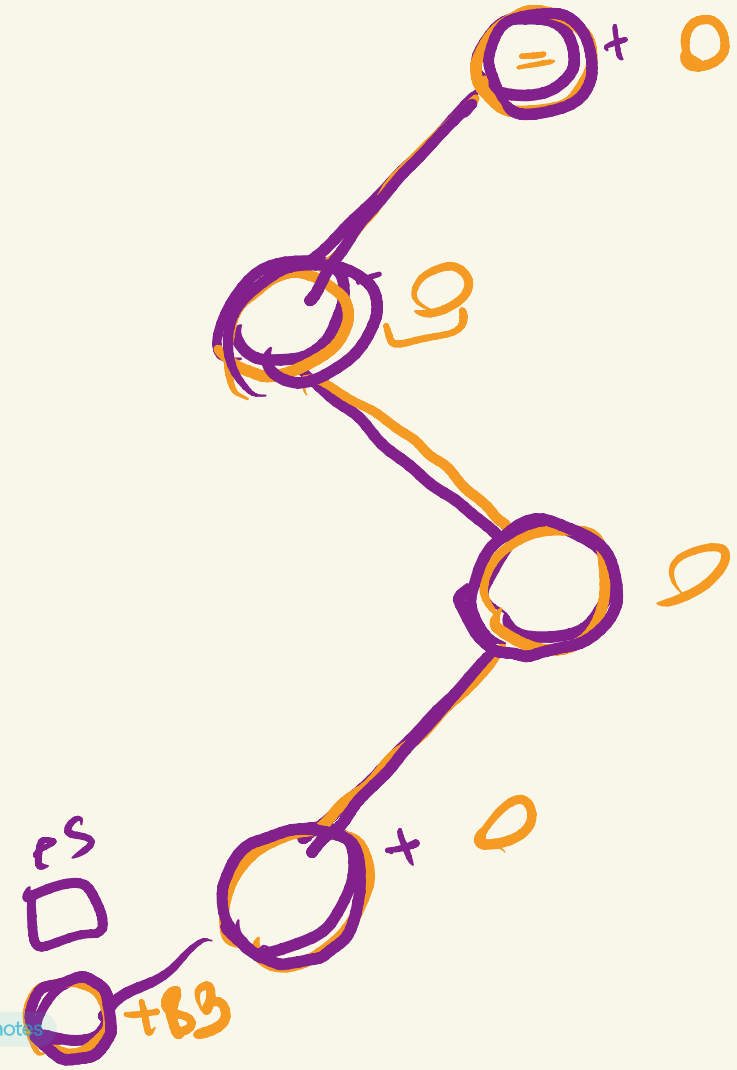
- 1 L R X -> Make all elements from L to R = X
- 2 L R -> Return the xor of all elements from L to R


comb → XOR
push
query
update
clear??
a ^ b
y a ± b
x ± x = 0

Lazy[node] = □ No update

-1 0

≈ 1 10^9



Keep this in mind

push

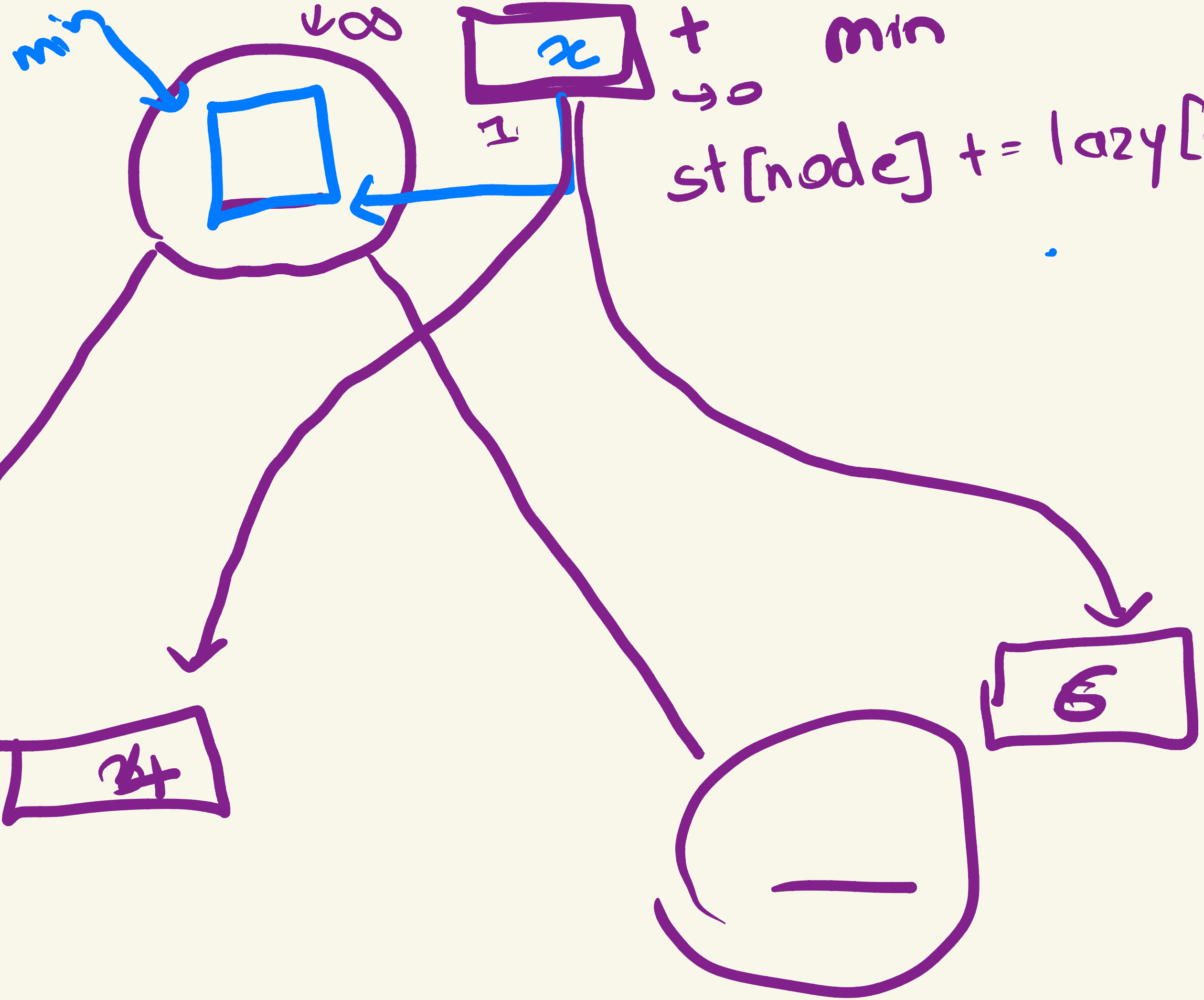
update ??

query, YES

✓Combine

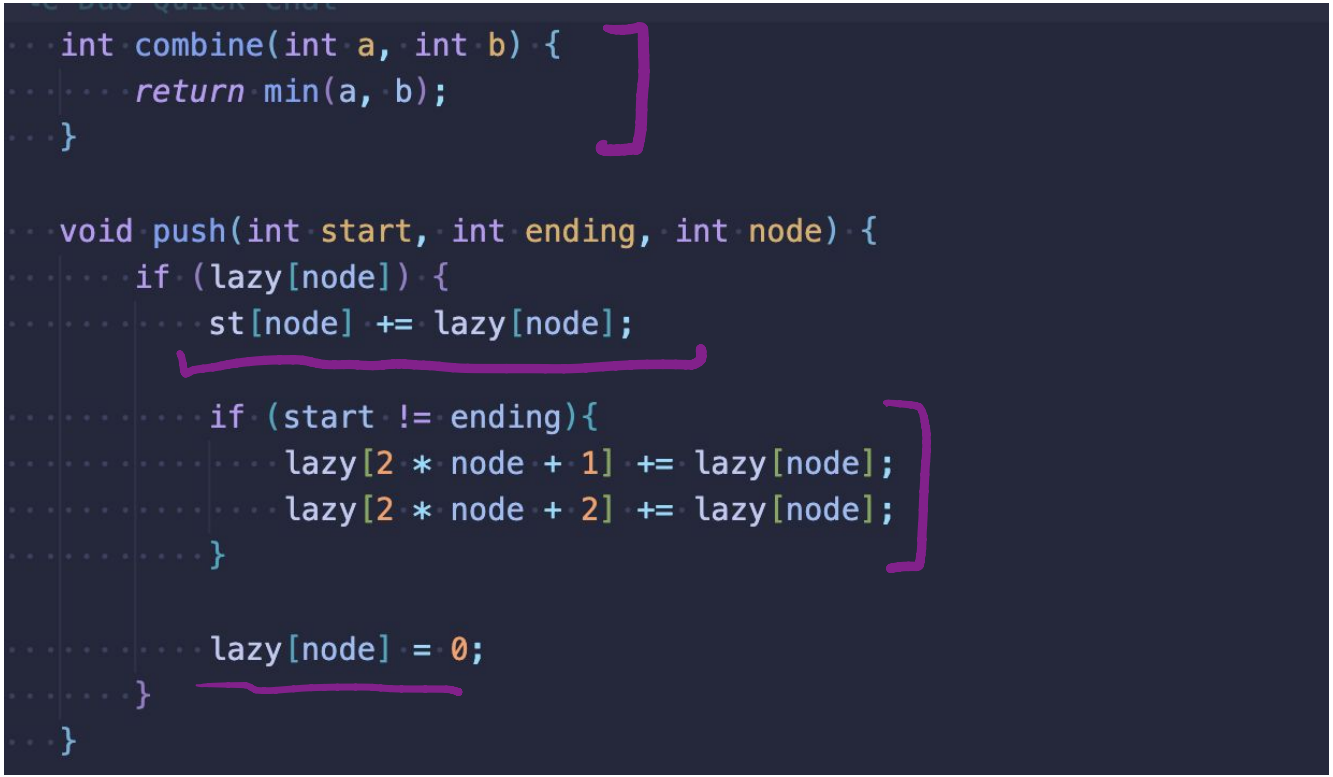
↳ +

↳ min



st[node] += lazy[node]

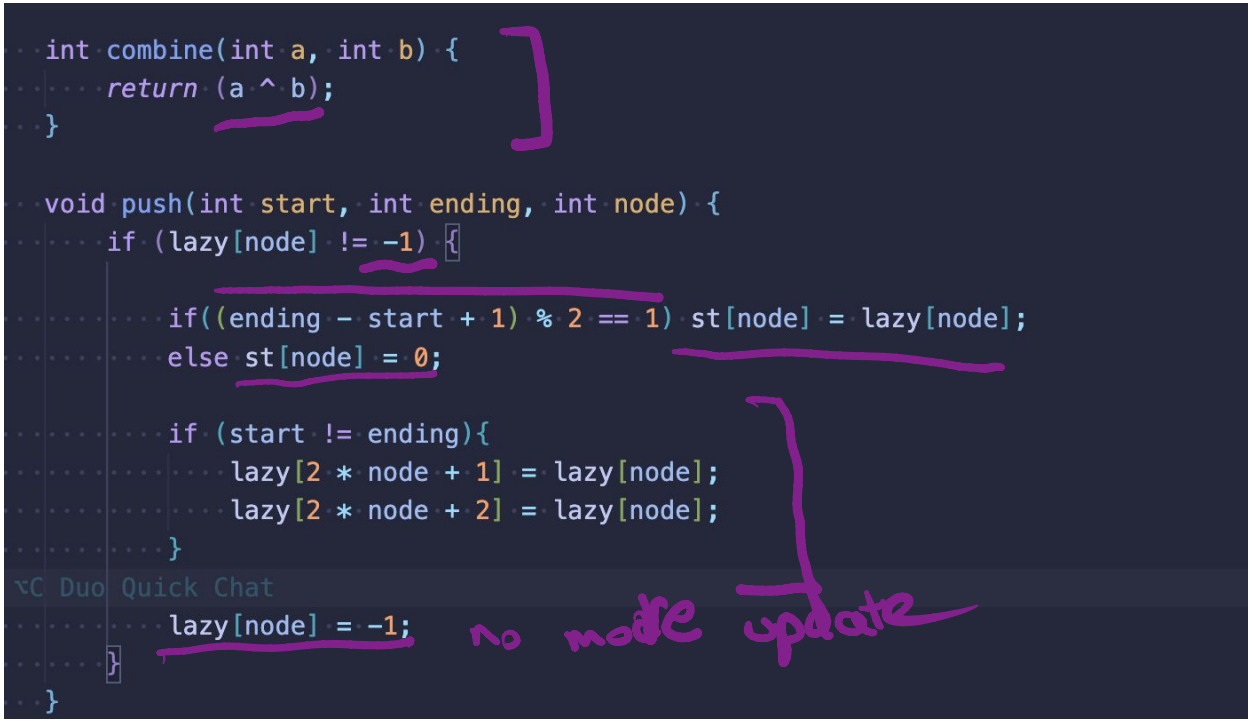
Changes to Make (2nd Problem)



```
int combine(int a, int b) {
    return min(a, b);
}

void push(int start, int ending, int node) {
    if (lazy[node]) {
        st[node] += lazy[node];

        if (start != ending){
            lazy[2 * node + 1] += lazy[node];
            lazy[2 * node + 2] += lazy[node];
        }

        lazy[node] = 0;
    }
}
```

query
y ∞
Out of
Range

Changes to Make (3rd Problem)



Generic Lazy Segment Tree by Priyansh

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