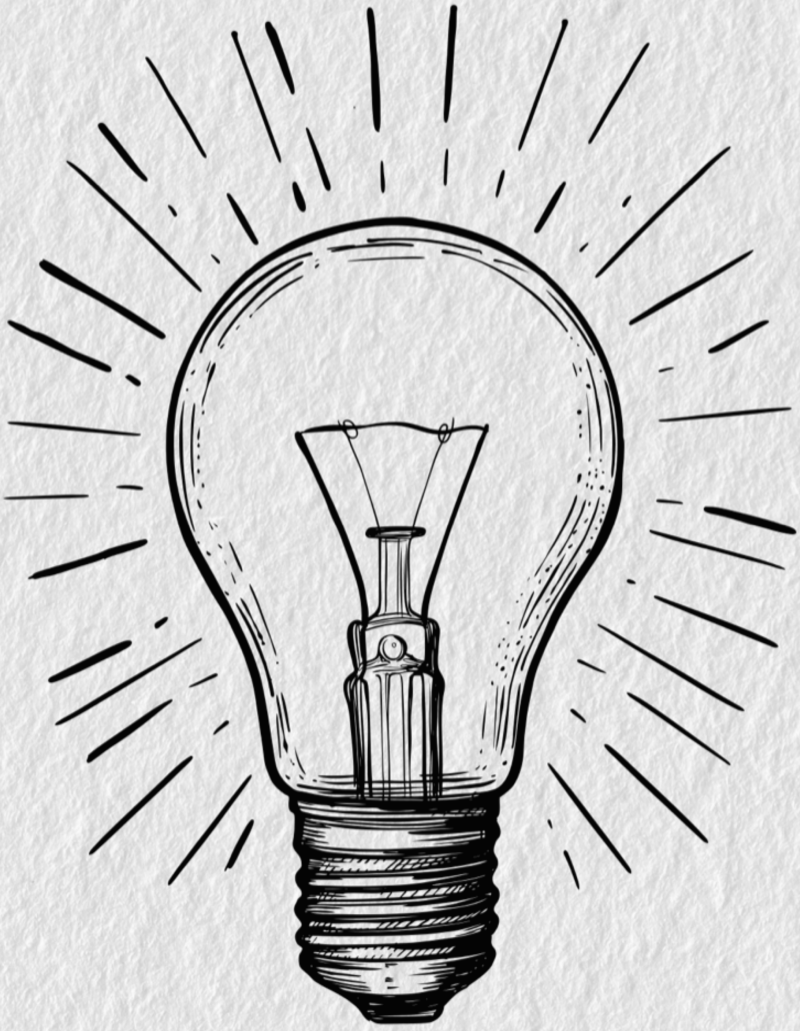
NOTEBOOK



NOTE IN

Creative notes

By reading we enrich the mind;
by writing we polish it.

Segment Trees

* Why use Segment Trees?

→ Segment trees:

worst case O(N)

a = [1, 6, 8, 2, 7, 16]
0 1 2 3 4 5

O(N) not good enough, find O(log(N))

Solution.

Ans → Segment tree

Segment Tree → perform query on a range in O(log N) time

(Sum, max, average, min, prod)

*To find sum, max, average, min, or product between a range of indices in an array we will use segment trees & intuition

update → O(log N)

Segment tree is a binary tree (every node has 2 children) which has interval

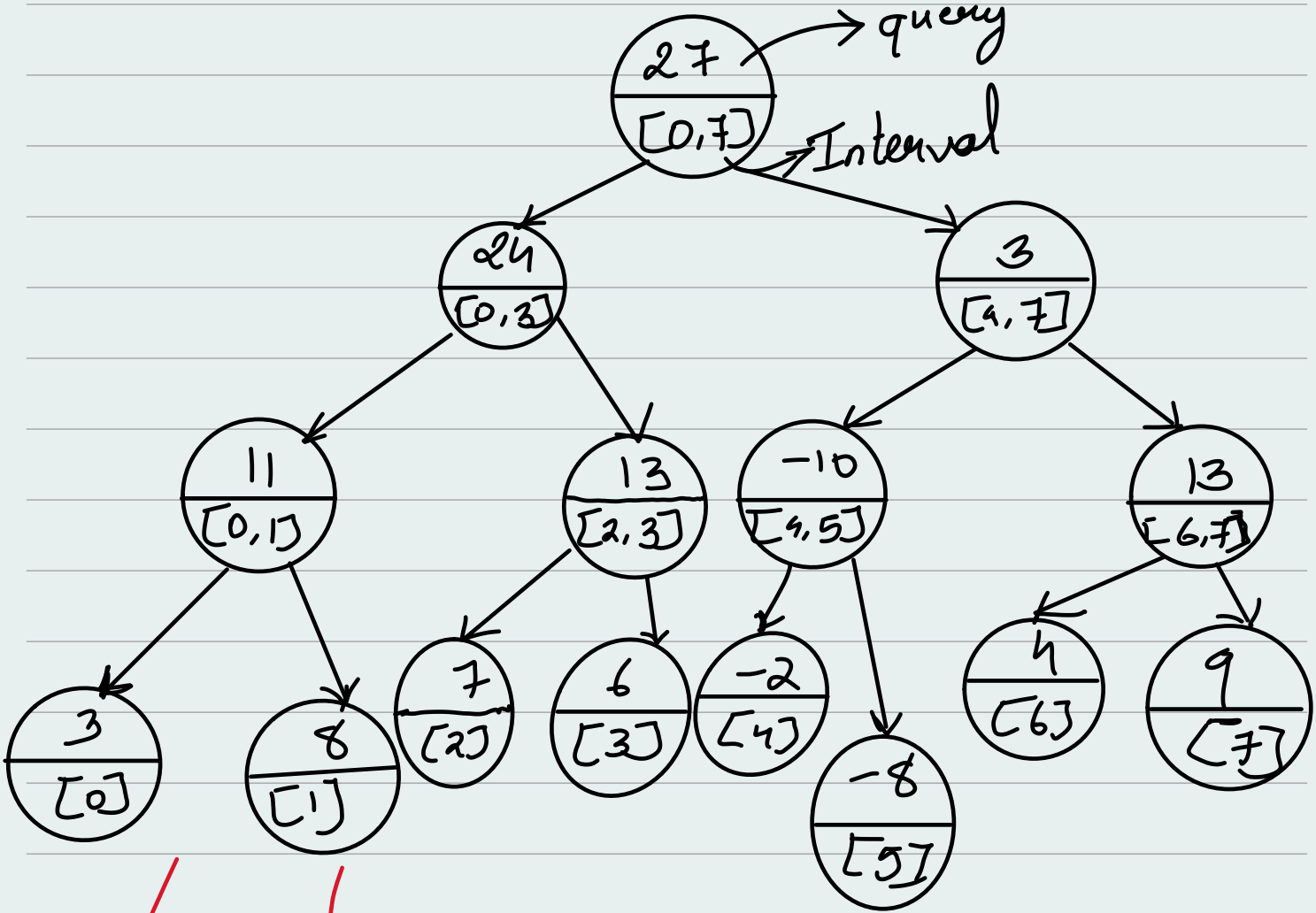
Information and operation

Ex: Sum between any 2 Integer

$a = [\underbrace{3, 8, 7, 6}, \underbrace{-2, -8, 4, 9}]$

indices: 0 1 2 3 4 5 6 7

$N = 8$



leaf nodes → cause only 1 element in an array.

∴ ST is a full Binary Tree

leaf Nodes = N
Internal nodes = N-1
Total nodes = 2N-1

→ Disadvantage of ST is that it uses extra space

Ans: Sum between [2, 6] = [2,3] + [4,5] + [6,6] etc.
Example only

Cases: case ① Node interval is inside query interval
Ex: [4, 5] → return the value

case ② Node interval is completely outside query interval i.e Node (start index > query end index) [2, 6] → [7, 7]

or (Node end index < query start index)

→ return the default value of the query In this case 0. Cause the range 7, 7 is completly outside the

range from [2, 6]

case ③ Overlapping (complete overlapping).

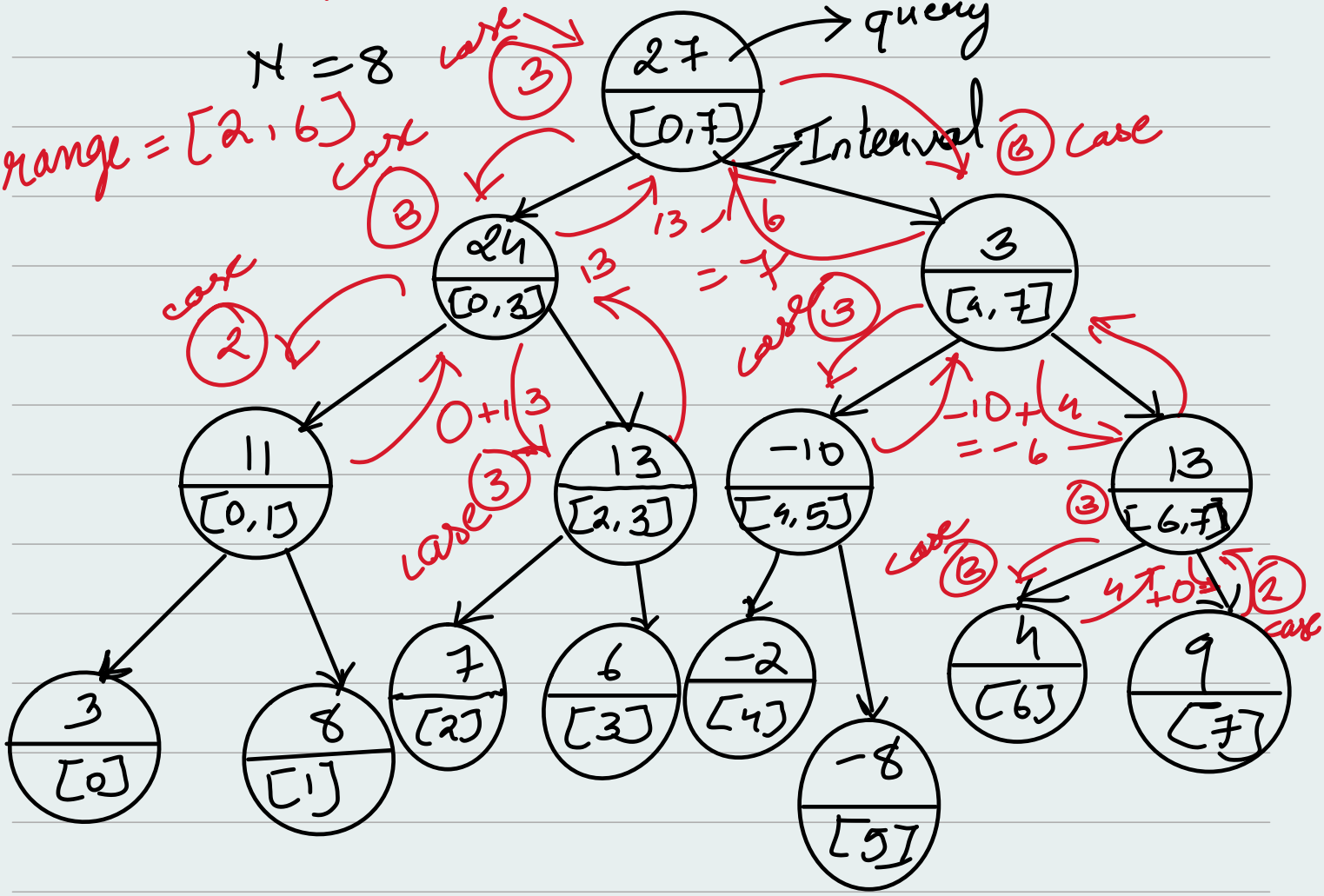
Now by using 3 cases we will get the ans in recursion diagram.

a = [3, 8, 7, 6, -2, -8, 4, 9]

0 1 2 3 4 5 6 7

N = 8

range = [2, 6]



range ⇒ [0, 7] in that [2, 6] range complete overlapping ie case no ③

$[2,3] + [4,5] + [6,6] = 7$

Worst case would be $\log N$.

B) How to update in $O(\log N)$ time.

Change Index 3 with 14 $\Rightarrow (3, 14)$

① check whether index lies in interval $[0, 7]$.

② If yes, whichever side it lies just update the value

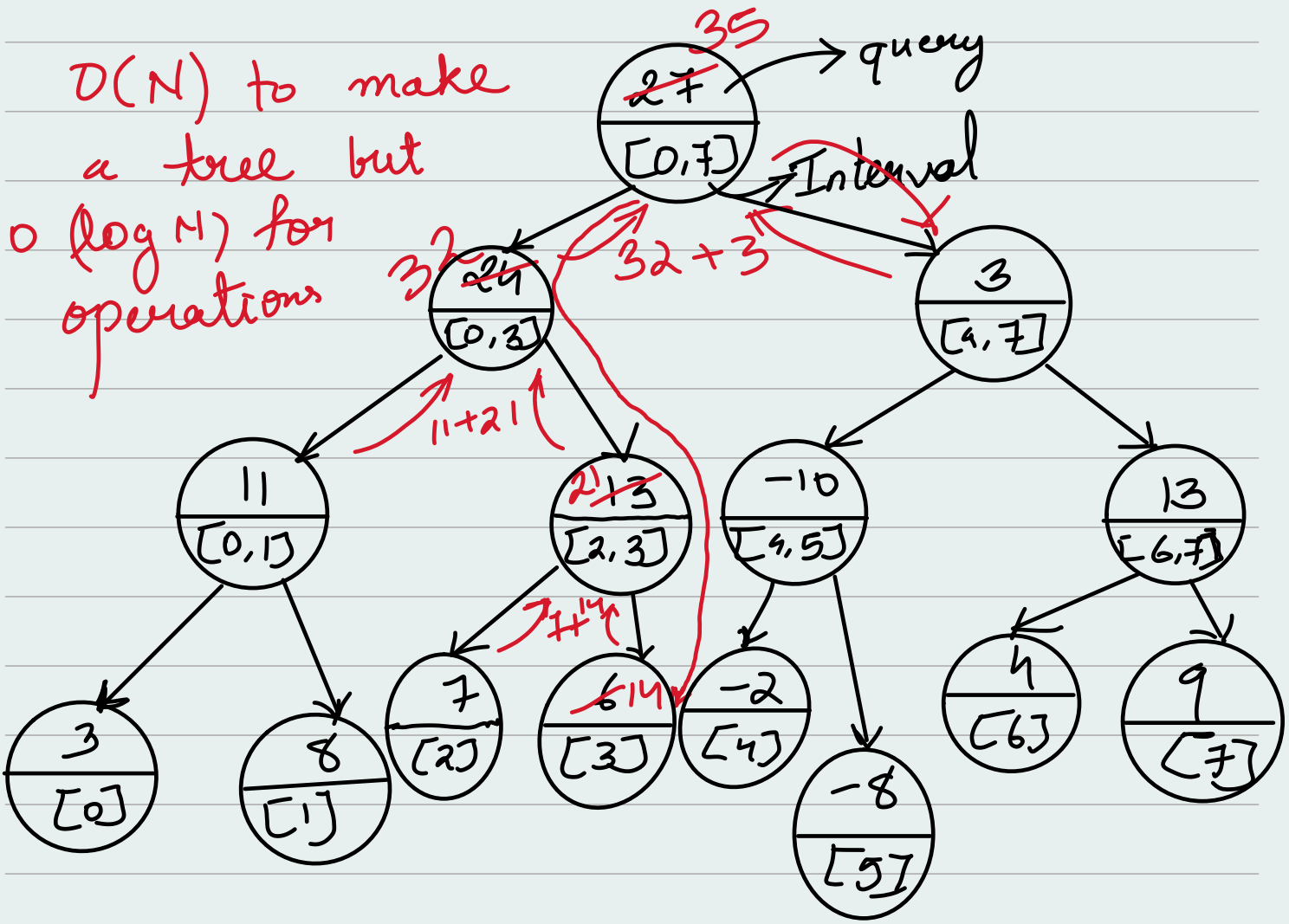
a) It does not lies in the R.H.S cause $[4, 7]$ is the range

b) It does lie in the range from $[0,3]$. From there we will again divide the array $[0,1]$ $[2,3]$

- $[0,1]$ → It doesn't lie here, do not update the value just return the old value
- $[2,3]$ → It does lie here again divide the array $[2]$ & $[3]$
  - $[2]$ → leaf node
  - $[3]$ → update it

Update the tree

O(N) to make a tree but O(log N) for operations



Code :

```
public class segmentTree{
    public static void main(String[] args){
        int[] arr = {3, 8, 6, 7, -2, -8, 4, 9};
        tree.display();
        System.out.println(tree.query(
                                qi:1 , qe:6));
}
```

```
private static class Node{ // inner class
    int data,
    int startInterval;
    int endInterval;
    Node left;
    Node right;

    public Node (int startInterval, int
                    endInterval ){
        this.startInterval = startInterval;
        this.endInterval = endInterval,
}

Node root;

public segmentTree (int[] arr){
    // create a tree using this array
    this.root = constructTree( arr, 0,
                    arr.length-1 ); //
        call the helper function

    private Node constructTree (int[] arr,
                int start, int end){
        if (start == end ){
            Node leaf = new Node(start, end);
```

```
            leaf.data = arr[start];
            return leaf;
        }
        // create a new node with index you are at
        Node node = new Node(start, end);
        int mid = (start + end) / 2;

        node.left = this.constructTree(arr,
                    start, mid),
        node.right = this.constructTree(arr,
                    mid+1, end),
        node.data = node.left.data + node
                    right.data;
        return node;

    }
    public void display{
        display(this.root),
    private void display(Node node){

    }
// Now implementing the three cases
```

```
public int query (int qsi, int qei) {
    return this.query (this.root, qsi, qei);
}
private int query(Node node, int qsi, int qei){
    // the node is complety lying inside the
       query
    if (node.startInterval >= qsi &&
        node.endInterval <= qei ){
        return node.data,
    } else if ( node.startInterval > qei
        node.endInterval < qsi ){
        // Completely outside.
        return 0;
    } else{
        return this.query (node.left, qsi,
        qei) + this.query ( node.right, qsi,
        qei);
    }
}

public void update (int index, int value){
    this.root.data = update (this.root,
                    index, value );
}
```

```
private int update (Node node, int index,
                    int value ){
    if (index >= node. startInterval &&
        index <= node. endInterval ){
    // if the index is in the range
        if (index == node. startInterval
            && index == node endInterval){
        // if the index is at the leaf node:
            node data = value;
            return node. data;
        } else{
            int leftAns = update (
                node left, index, value);
            int rightAns = update (node.
                right, index, value);
            node. data = leftAns +
                        rightAns ;
            return node. data;
        }
    }
    return node. data.
}
```