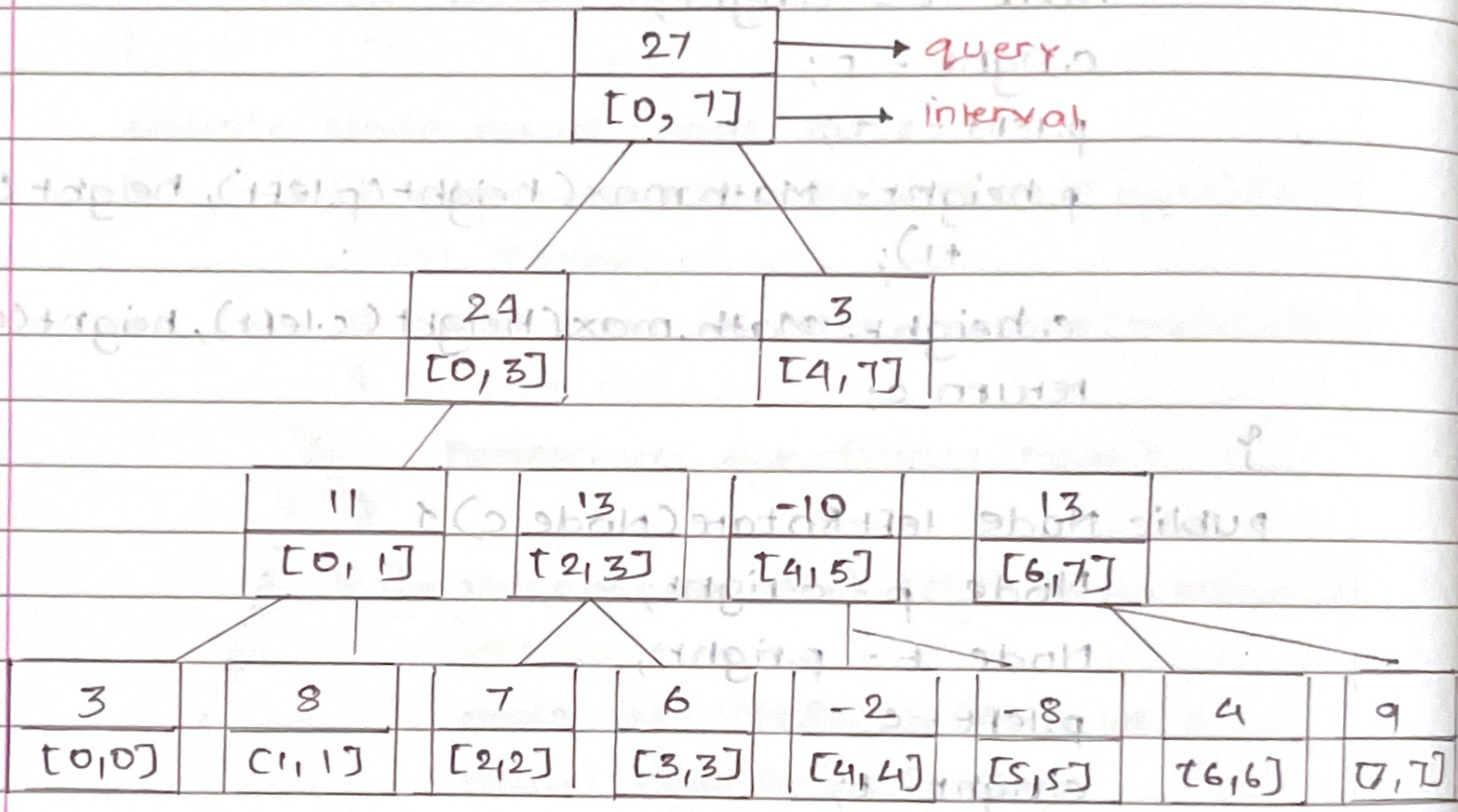
★ When Question/Queries given in the range

- Segment Tree
  Perform query in the range, sum, max, min, average, product in log(N) and updates also takes O(log N).
  Segment tree is an binary tree which has internal info and operations.

Example: [3 8 7 6 -2 -8 4 9]
0 1 2 3 4 5 6 7



- Sum between [2,6] = [2,3] + [4,5] + [6,6] etc.
- Node interval is inside query interval
  Ex: [4, 5] → Return value
- Node interval is completely outside the query interval
  Ex: (node start index) > (query end index)
  return the default value of the query
- Overlaping: [2,6] [0,3]

- How to update in O(logN) time:

1. Check whether index lies in interval.
2. If yes then check child nodes, If child range is out no change in value just return.
3. In the end you will reach leaf node, update leaf and recursion will update the tree

Implementation:

```
class SegmentTree {
        private static class Node{
                int data;
                int startInterval;
                ind endInterval;
                Node left;
                Node right;
                public Node(int startInterval, int endInterval){
                        this.startInterval = startInterval;
                        this.endInterval = endInterval;
                }
        }
        Node root;
        public SegmentTree(int[]arr){
                this.root= constructTree(arr, 0, arr.length-1);
        }
        private node constructTree(int[]arr, int start, int end){
                if(start == end{
                        Node leaf = new Node(start,end);
                        leaf.data=arr[start]
                        return leaf;
                }
                Node node = new Node(start, end)
        int mid = (start + end)/2;
        node.left = this.constructTree(arr, start, ~~end~~ mid);
        node.right = this.construct Tree(arr, mid+1, end);
        node.data = node.left.data + node.right.data;
        return node;
    }

    public int query(int qsi, int qei) {
        return this.query(this.root, qsi, qei);
    }

    private int query (Node node, int qsi, int qei) {
        if (node.startInterval >= qsi &&
                node.endInterval <= qei) {
            return node.data;
        } else if (node.startInterval > qei || ~~node.~~
                node.endInterval < qsi) {
            return 0;
        } else {
            return this.query(node.left, qsi, qei) +
                this.query(node.right, qsi, qei);
        }
    }

    private int update (Node node, int index, int value
        if( index >= node.startInterval && index <=
                node.endInterval) {
            ~~node.data = value;~~
            ~~return node.data;~~
          if(index == node.startInterval &&
                index == node.endInterval) {
                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;
    }
}
```