

queries:  $[l, r]$  Range Sum Query  $\Rightarrow$   $\text{prefix}[r] - \text{prefix}[l-1]$

$$[6, 7] \Rightarrow (-10 + -5 = -15) \Rightarrow p[7] - p[5] = 45 - 60 = -15$$

$$[2, 6] \Rightarrow -12 + 20 + 30 + (-15) + (-10) = 13 \Rightarrow p[6] - p[1] = 50 - 37 = 13$$

$$[0, 2] \Rightarrow 10 + 27 - 12 = 25 \Rightarrow p[2] - p[1] = 25$$

$q$  queries  $\Rightarrow \mathcal{O}(1) \times q = \underline{\mathcal{O}(q)}$

# Prefix Sum will work in  $O(n+q)$  time  
if array is "immutable"

```
class NumArray {
    int[] prefix;
    public NumArray(int[] nums) {
        this.prefix = nums;
        for(int i = 1; i < nums.length; i++)
            prefix[i] += prefix[i - 1];
    }

    public int sumRange(int left, int right) {
        if(left == 0) return prefix[right];
        return prefix[right] - prefix[left - 1];
    }
}
```

$O(n)$

$O(1)$  query  
 $q$

Space =  $O(1)$   
Time =  $O(n+q)$   
 $O^4 + 1 O^4$   
Leetcode 303

Approach 1)

Time =  $O(n^2)$   
per query

Prefix array for each

3	0	1	4	2
5	6	3	2	1
1	2	0	1	5
4	1	0	1	7
1	0	3	0	5

3	3	4	8	10
5	11	14	16	17
1	3	3	4	9
4	5	5	6	13
1	1	4	4	9

3	0	1	4	2
5	( $r_1, c_1$ )	6	3	2
1	2	0	1	( $r_2, c_2$ ) 5
4	1	0	1	7
1	0	3	0	5

Prefix matrix

3	3	4	8	10
8	14	18	24	27
9	17	21	27	36
-	-	-	-	-
-	-	-	-	-

Approach 2

$O(n^2)$  pre calculation

$\hookrightarrow \text{dp}[i][j] = \text{nums}[i][j]$

$$\begin{aligned} &+ \text{dp}[i-1][j] \\ &+ \text{dp}[i][j-1] \\ &- \text{dp}[i-1][j-1] \end{aligned}$$

$\text{ans} \xrightarrow{\text{query}} \text{query}(r_1, c_1, r_2, c_2)$

$$= \text{dp}[r_2][c_2] - \text{dp}[r_2][c_1-1] - \text{dp}[r_1-1][c_2] \\ + \text{dp}[r_1-1][c_1-1]$$

```

class NumMatrix {
    int[][] dp; // Precalculation -> O(N ^ 2)
    public NumMatrix(int[][] dp) {
        this.dp = dp;
        for(int i = 0; i < dp.length; i++){
            for(int j = 0; j < dp[0].length; j++){
                if(i > 0) dp[i][j] += dp[i - 1][j];
                if(j > 0) dp[i][j] += dp[i][j - 1];
                if(i > 0 && j > 0) dp[i][j] -= dp[i - 1][j - 1];
            }
        }
    }
    public int sumRegion(int r1, int c1, int r2, int c2) {
        int ans = dp[r2][c2];
        if(c1 > 0) ans -= dp[r2][c1 - 1];
        if(r1 > 0) ans -= dp[r1 - 1][c2];
        if(r1 > 0 && c1 > 0) ans += dp[r1 - 1][c1 - 1];
        return ans;
    }
}

```

•  $O(3n^2)$

$\rightarrow O(1)$  per query  $\times Q$

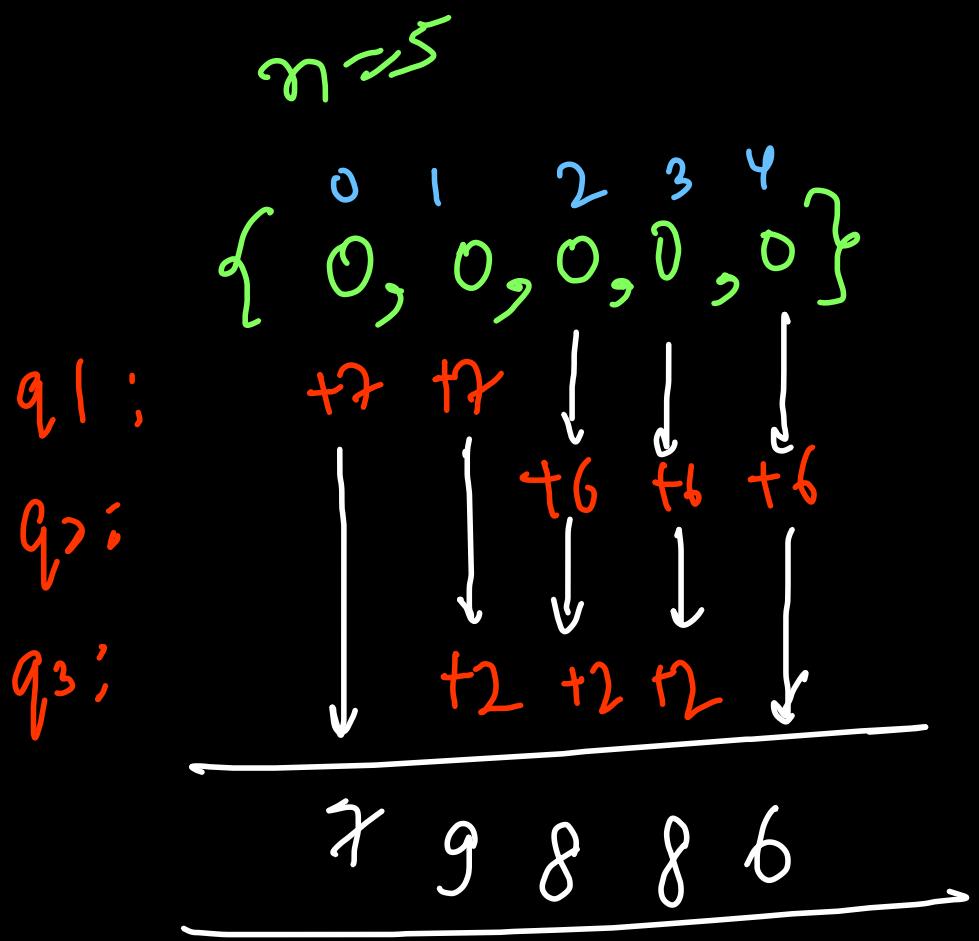
Time

$$\leq O(n^2 + q)$$

Space

$$= O(1)$$

Update if  $S \neq 0$



update queries

Input:

1  
5 3  
0 1 7  
2 4 6  
1 3 2

3  
0  
3  
4  
} get ith index

Output:

7  
8  
6

Brute force

per query  
 $O(n)$

Total time

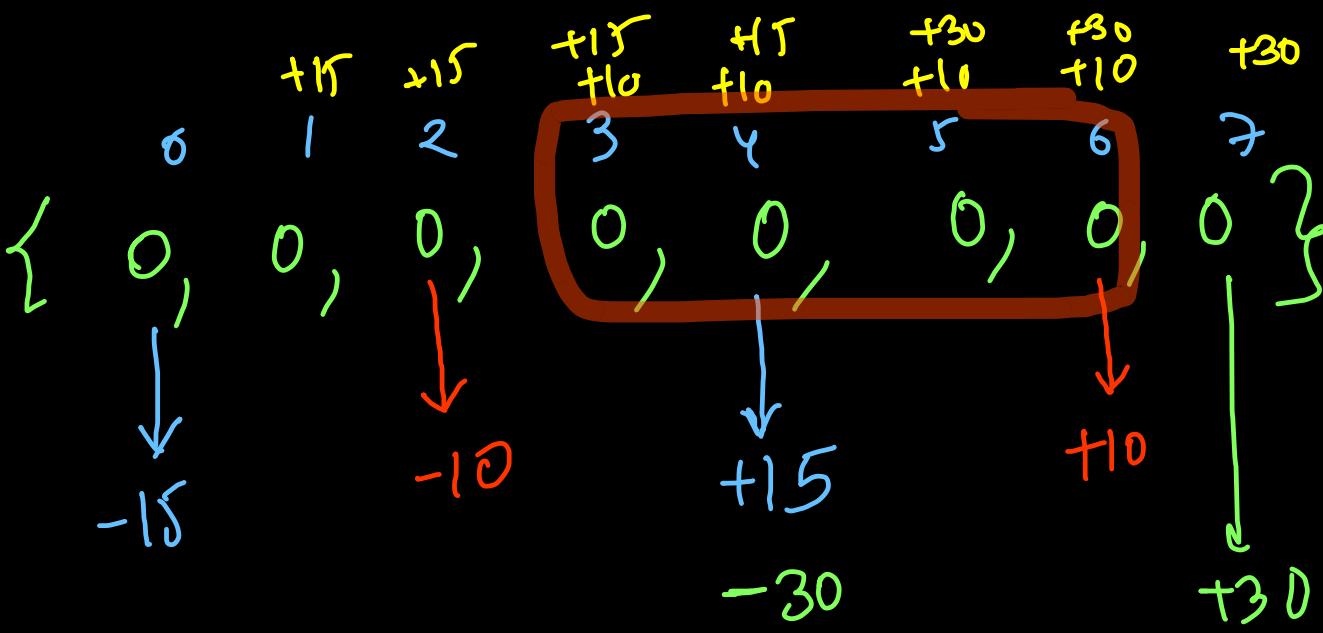
$= O(n \times q)$

Time

$10^4 \times 10^5$

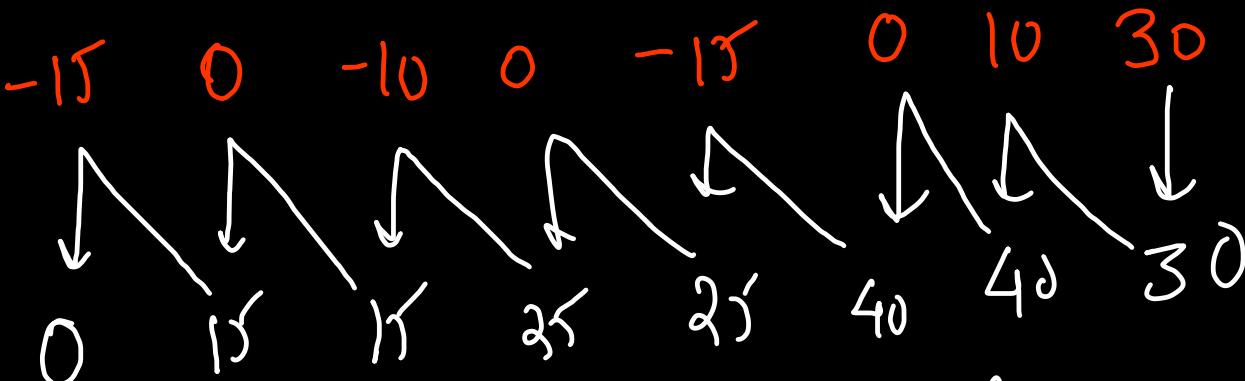
per query  
val

$q_1 : (3, 6, +10)$   
 $\ell \succ val$



$q_2 : (1, 4, +15)$   
 $\ell \succ val$

$q_3 : (5, 7, +30)$   
 $\ell \succ val$



final query  $\Rightarrow O(n)$

```

public static void solve() throws Exception {
    int t = scn.nextInt();
    while(t-- > 0){
        int n = scn.nextInt(); → length of array
        int u = scn.nextInt(); → no. of update queries
        long[] diff = new long[n];
        while(u-- > 0){
            int l = scn.nextInt();
            int r = scn.nextInt();
            int val = scn.nextInt();

            diff[r] += val;
            if(l > 0) diff[l - 1] -= val;
        }
        for(int i = n - 2; i >= 0; i--) }→ Array construct
            diff[i] += diff[i + 1];
        int g = scn.nextInt();
        while(g-- > 0){
            int idx = scn.nextInt();
            out.println(diff[idx]);
        }
    }
}

```

~~Difference array~~  
~~Update it  $O(n)$~~

$\rightarrow O(1)$  per query  $\times q = O(q)$

+

$\rightarrow O(n)$   $\times O(n) = O(n^2)$

+

$\rightarrow$  get  $i^{th}$  query  $\Rightarrow O(1) \times g$   
 $= O(g)$

{ 10, 5, 15, 7, 13 }

~~UFU~~ If input is not  
containing 0's

↓ Diff Array

{ 10<sup>5</sup>, 5<sup>15</sup>, 15<sup>7</sup>, 7<sup>13</sup>, 13 }  
prefix array

no update queries

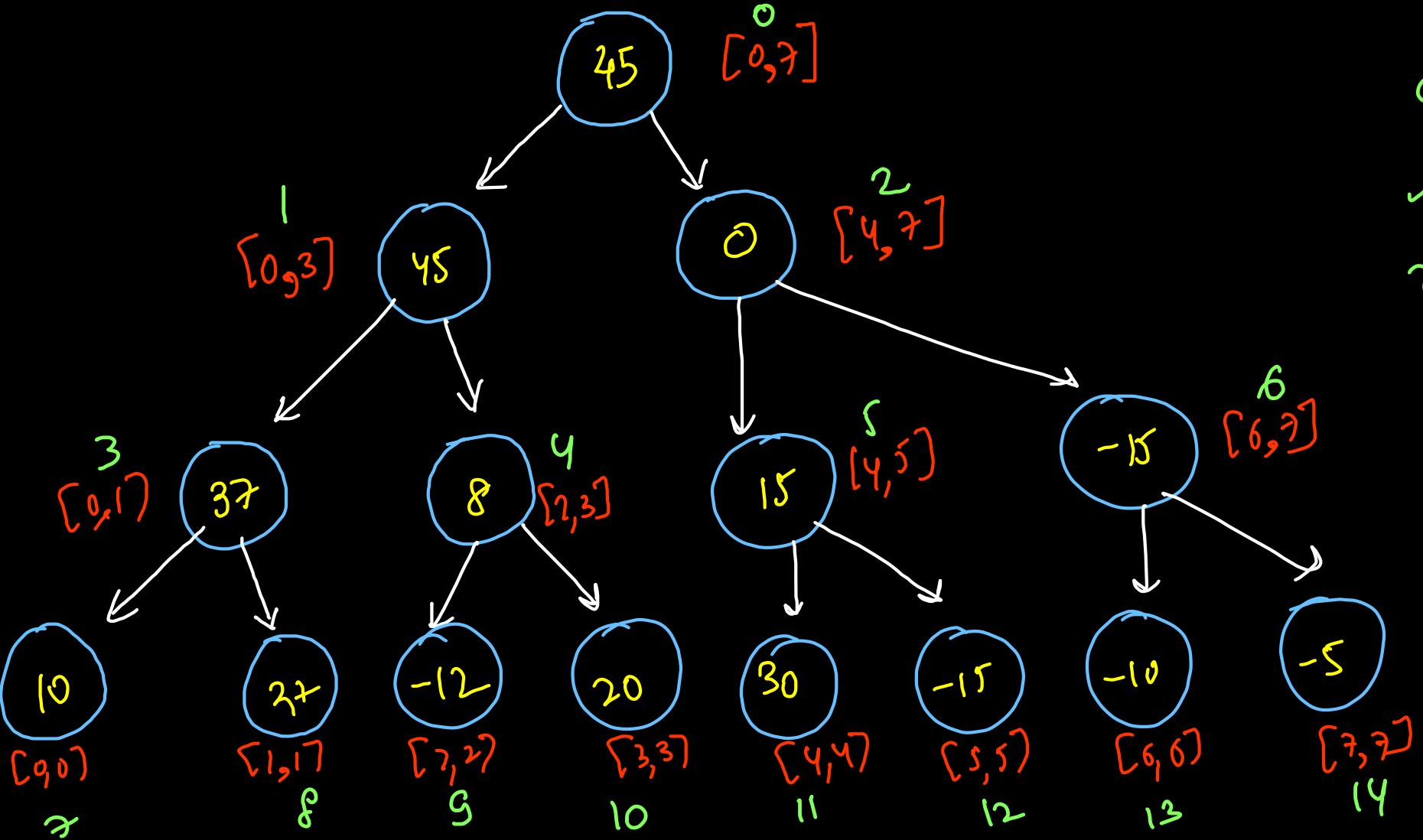
$$\text{diff}(i) = \overbrace{A[0:i]}^{\text{prefix}} - \overbrace{A[i+1:n]}^{\text{suffix}}$$

Range Sum Query Segment Tree

$\Rightarrow$  Range query  $\Rightarrow$  Point update

~~ary:~~ { 0 1 2 3 4 5 6 7  
10, 27, -12, 20, 30, -15, -10, -5 }

ary: { 0, 1, 2, 3, 4, 5, 6, 7  
 10, 27, -12, 20, 30, -15, -10, -5 }



par  $\rightarrow (\text{idx}-1)/2$

curr  $\rightarrow \text{idx}$

left  $\rightarrow 2\text{idx}+1$

right  $\rightarrow 2\text{idx}+2$

```
TreeNode[] nodes;
int[] nums; (data member)

public int construct(int idx, int left, int right){
    if(left > right){
        nodes[idx] = new TreeNode(left, right, 0);
        return 0;
    }
    if(left == right){
        nodes[idx] = new TreeNode(left, right, nums[left]);
        return nums[left];
    }

    nodes[idx] = new TreeNode(left, right, 0);
    int mid = left + (right - left) / 2;
    nodes[idx].sum += construct(2 * idx + 1, left, mid);
    nodes[idx].sum += construct(2 * idx + 2, mid + 1, right);
    return nodes[idx].sum;
}

public NumArray(int[] nums) {
    this.nums = nums;
    int n = nums.length;
    nodes = new TreeNode[n * 4 + 5];
    construct(0, 0, n - 1);
}
```

class TreeNode{  
 int left, right, sum;  
 TreeNode(int left, int right, int sum){  
 this.left = left;  
 this.right = right;  
 this.sum = sum;  
 }  
}

LC307

→ constructor parameter

Objekt  
Oberklasse

```
int[] nodes;

public void construct(int[] nums, int idx, int left, int right){
    if(left > right) return;
    if(left == right){
        nodes[idx] = nums[left];
        return;
    }
    int mid = left + (right - left) / 2;
    construct(nums, 2 * idx + 1, left, mid);
    construct(nums, 2 * idx + 2, mid + 1, right);
    nodes[idx] = nodes[2 * idx + 1] + nodes[2 * idx + 2];
}

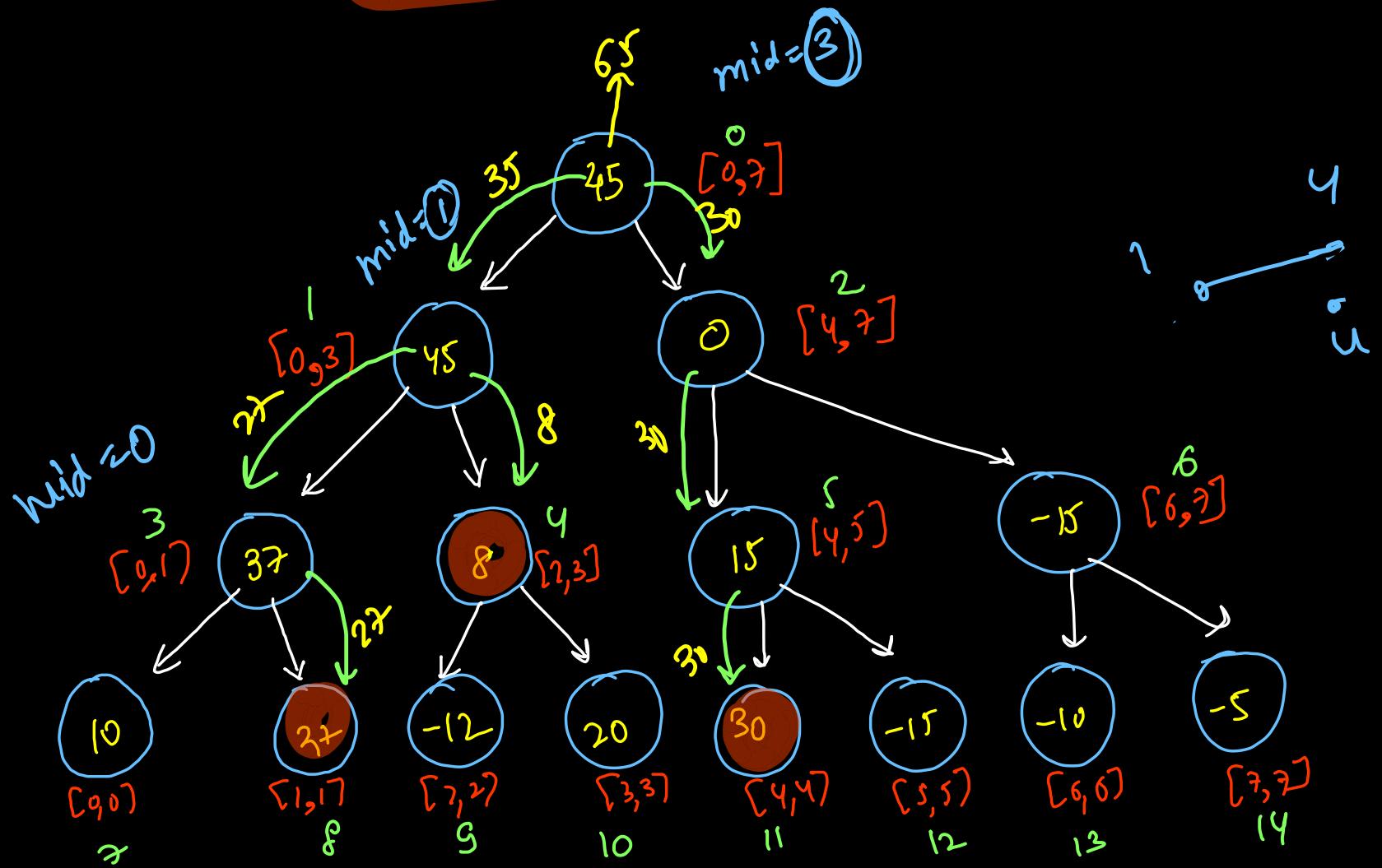
public NumArray(int[] nums) {
    int n = nums.length;
    nodes = new int[n * 4 + 5];
    construct(nums, 0, 0, n - 1);
}
```

Approach 2) Construction

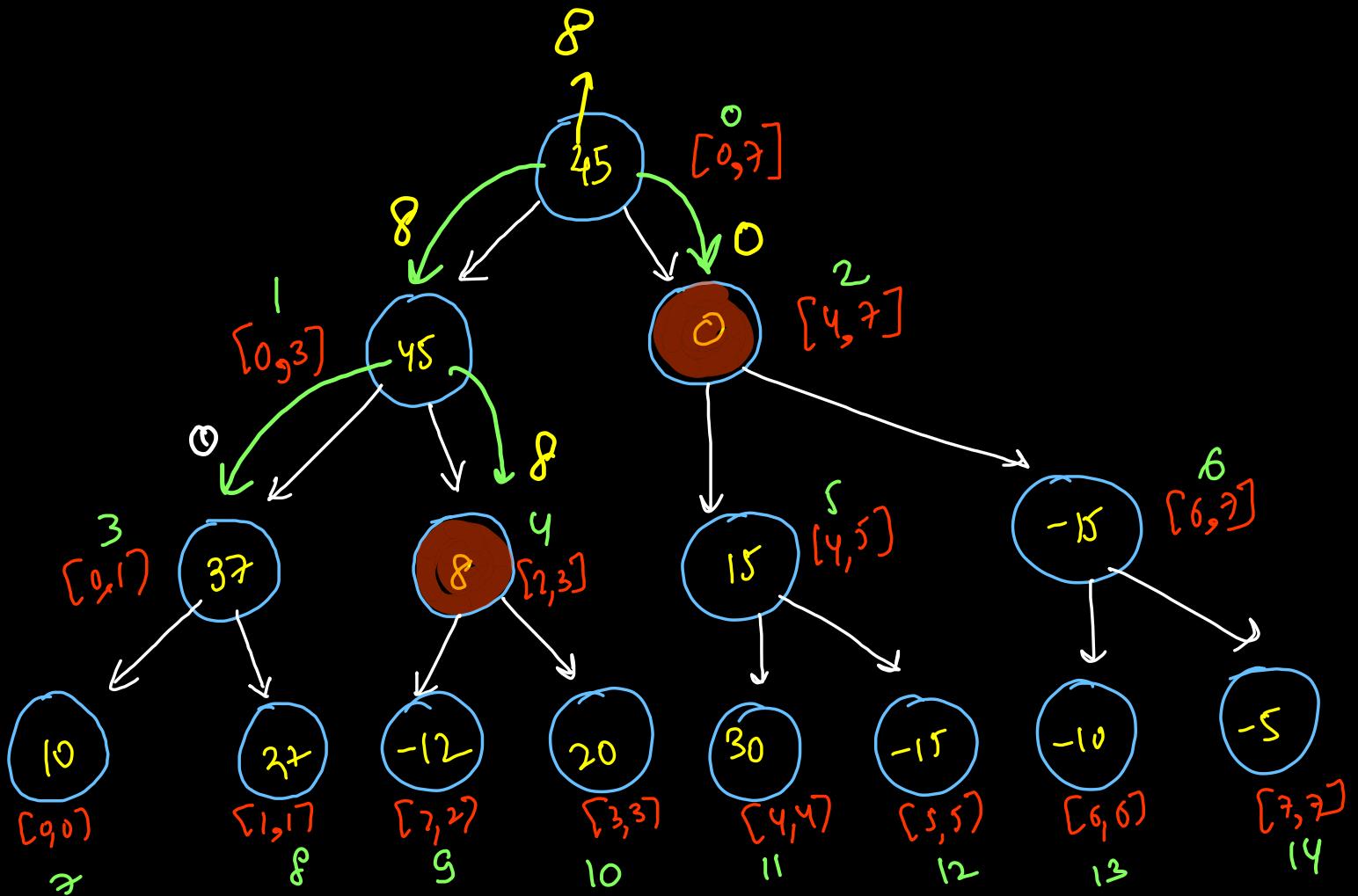
$O(n \log n)$  time

$O(n)$  space

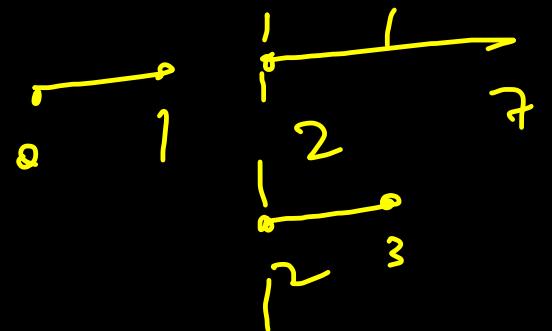
array : { 0, 1, 2, 3, 4, 5, 6, 7, 8 }  
10, 27, -12, 20, 30, -15, -10, -5 }



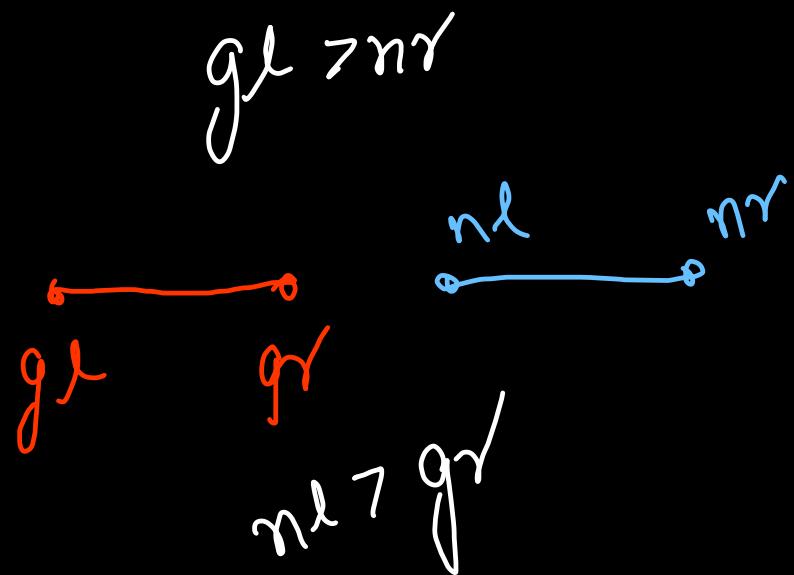
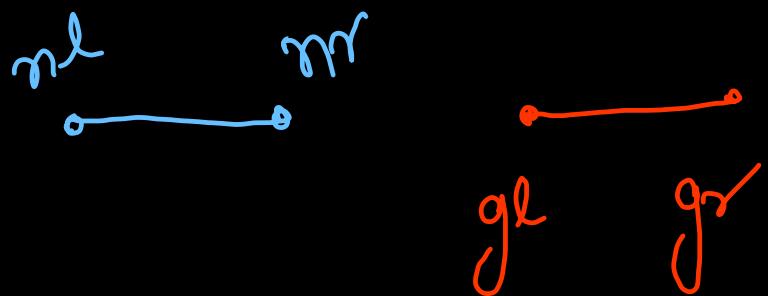
arry: [ 10, 27, -12, 20, 30, -15, -10, -5 ]



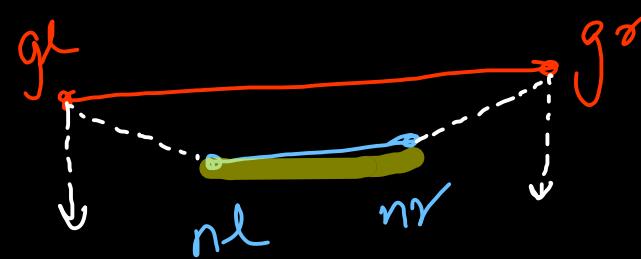
query = [2,7]



return 0;  
no overlapping



node completely ly  
inside the given range

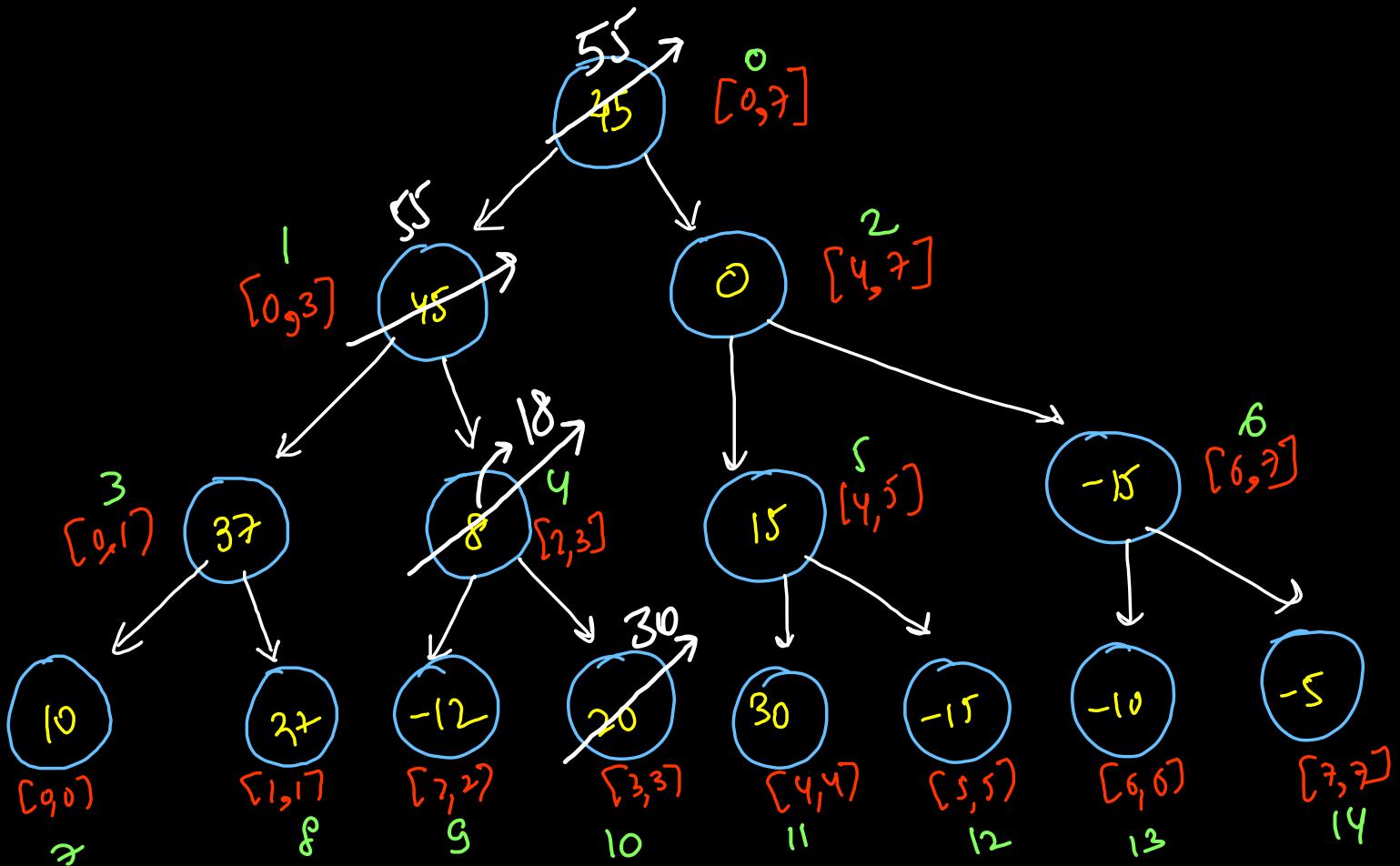


```
public int dfs(int gl, int gr, int idx, int nl, int nr){  
    if(gl > nr || nl > gr) return 0; // no overlapping  
    if(nl >= gl && nr <= gr) return nodes[idx]; // completely inside  
  
    int mid = (nl + nr) / 2;  
    return dfs(gl, gr, 2 * idx + 1, nl, mid)  
        + dfs(gl, gr, 2 * idx + 2, mid + 1, nr);  
}  
  
public int sumRange(int left, int right) {  
    return dfs(left, right, 0, 0, n - 1);  
}
```

time  
 $= O(\log n)$   
per query

~~point update~~

array:

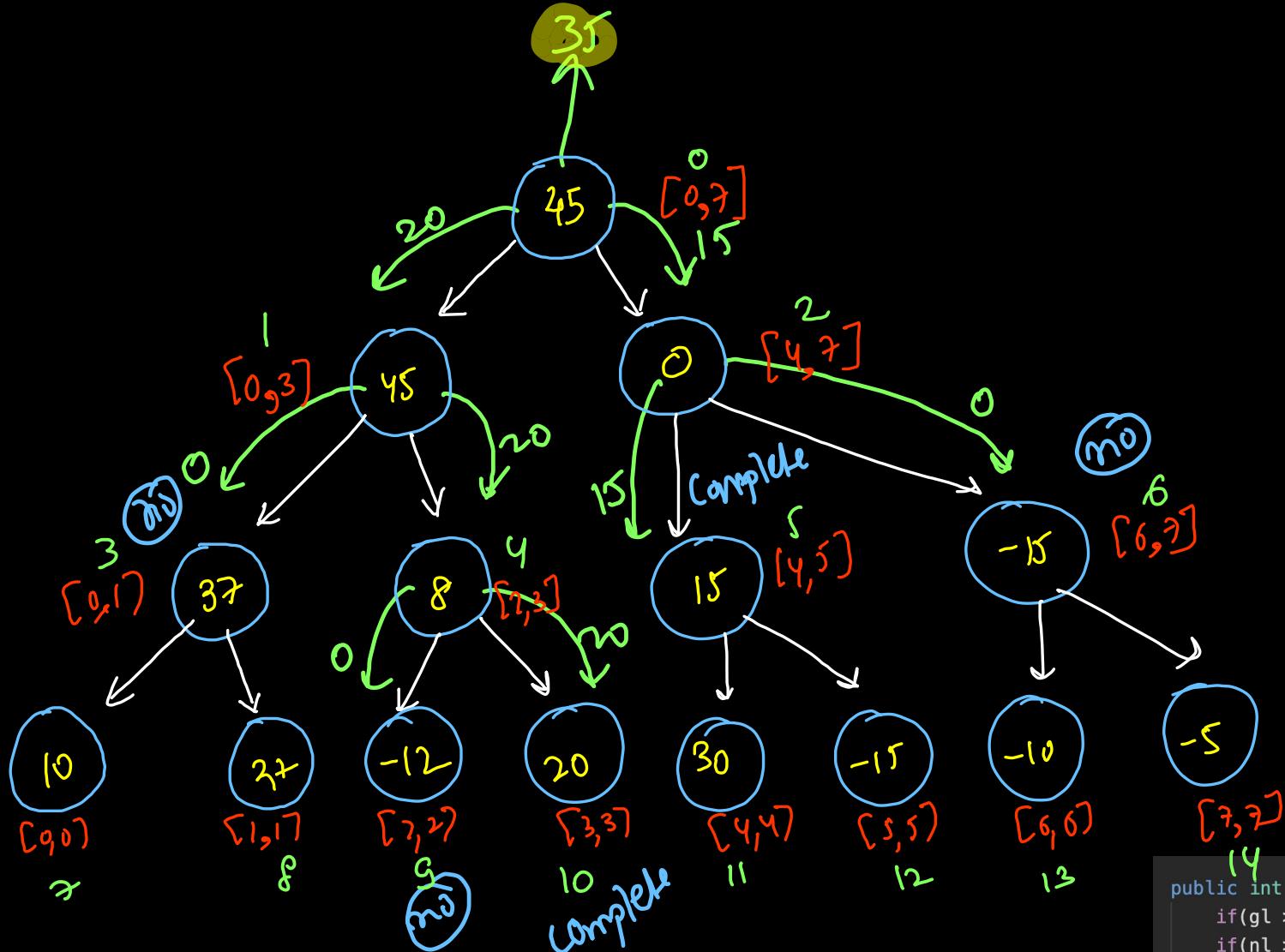
$$\{ 10, 27, -12, 20, 30, -15, -10, -5 \}$$


update  
nums[3] → 30

$$\text{diff} = 30 - 20 \\ \downarrow \quad \downarrow \\ \text{val} - \text{oldval}$$

Range sum  
query

{ 3, 5 }



```

public int dfs(int gl, int gr, int idx, int nl, int nr){
    if(gl > nr || nl > gr) return 0; // no overlapping
    if(nl >= gl && nr <= gr) return nodes[idx]; // completely inside

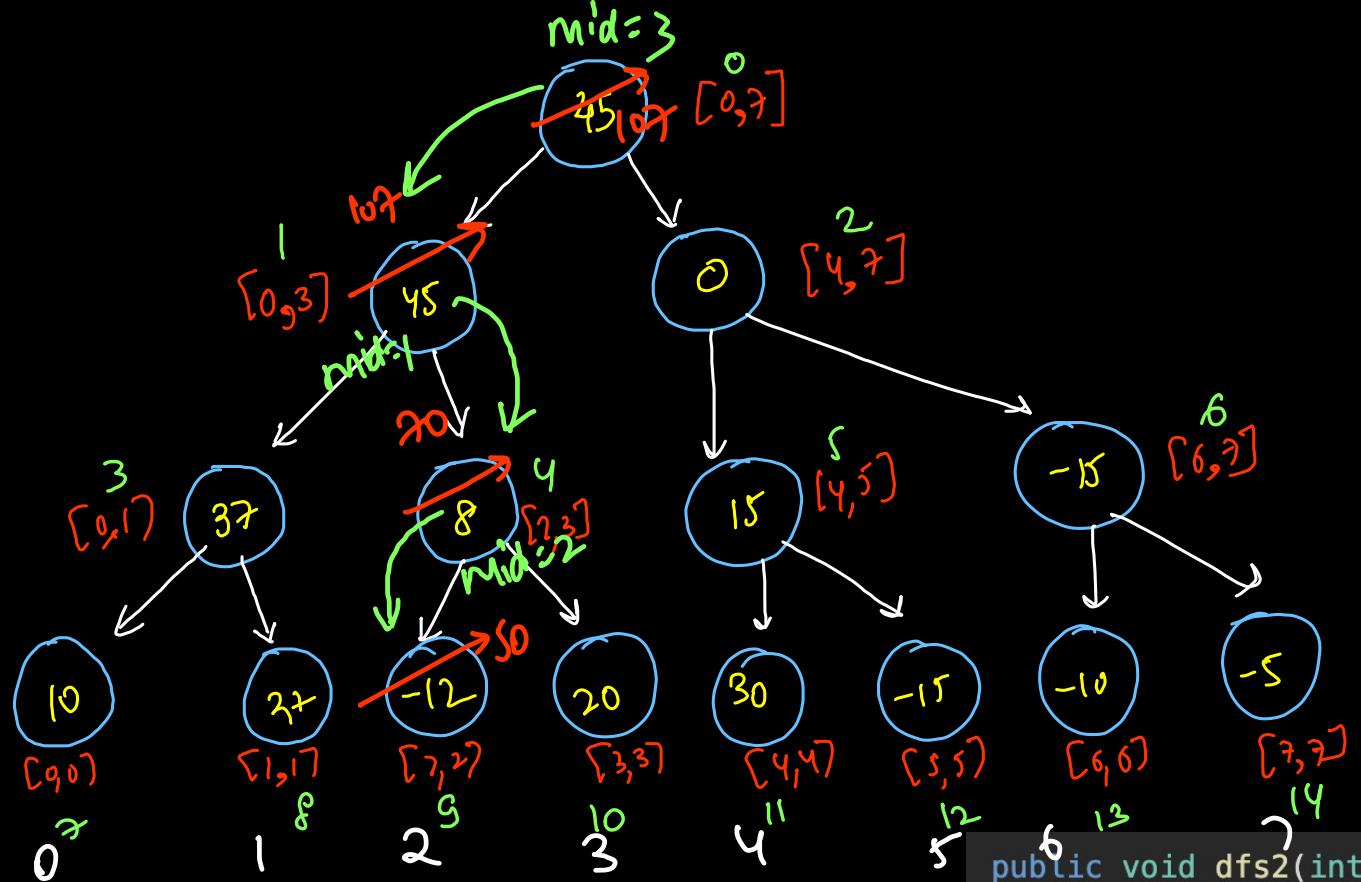
    int mid = (nl + nr) / 2;
    return dfs(gl, gr, 2 * idx + 1, nl, mid)
        + dfs(gl, gr, 2 * idx + 2, mid + 1, nr);
}

public int sumRange(int left, int right) {
    return dfs(left, right, 0, 0, n - 1);
}

```

Point update

$$\text{sums}[2] = 50$$



```
public void dfs2(int idx, int left, int right, int index, int val){  
    if(left > right) return;  
    if(left == right) {  
        nodes[idx] = val;  
        return;  
    }  
  
    int mid = left + (right - left) / 2;  
    if(index <= mid) dfs2(2 * idx + 1, left, mid, index, val);  
    else    dfs2(2 * idx + 2, mid + 1, right, index, val);  
    nodes[idx] = nodes[2 * idx + 1] + nodes[2 * idx + 2];  
}
```

2  
50  
 $O(\log n)$

## Range Minimum/Maxm Query

~~array~~:  $\{ 10, 27, -12, 20, 30, -15, -10, -5 \}$

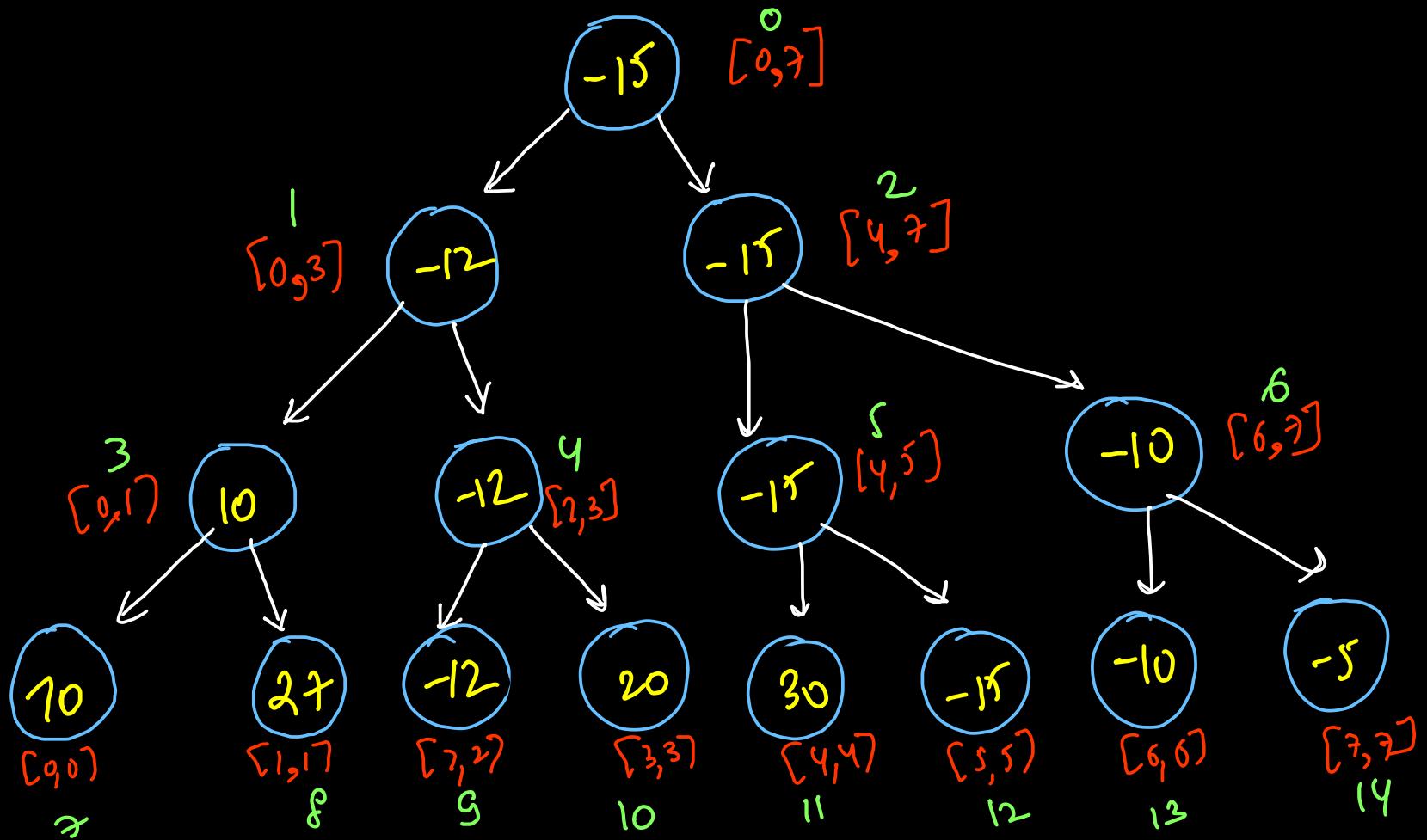
$$[1, 4] = -12$$

$$[3, 7] = -15$$

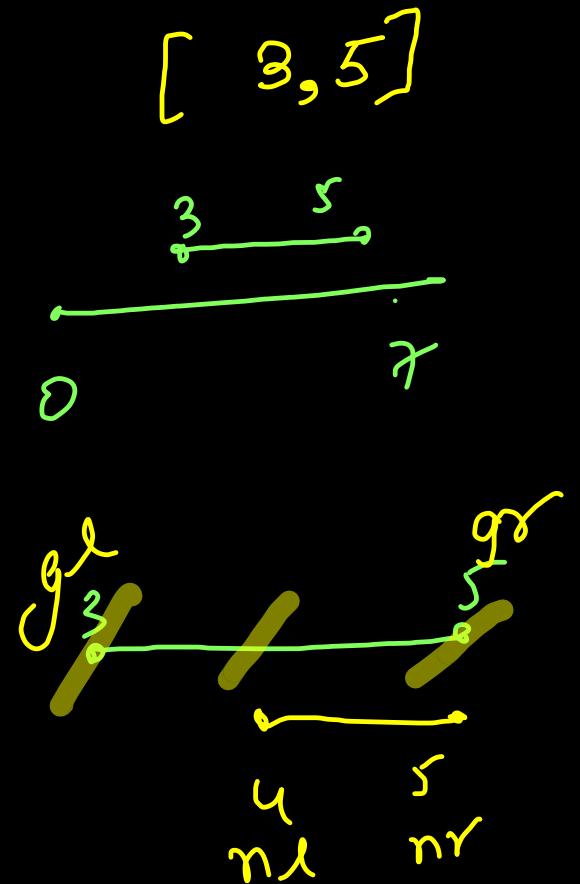
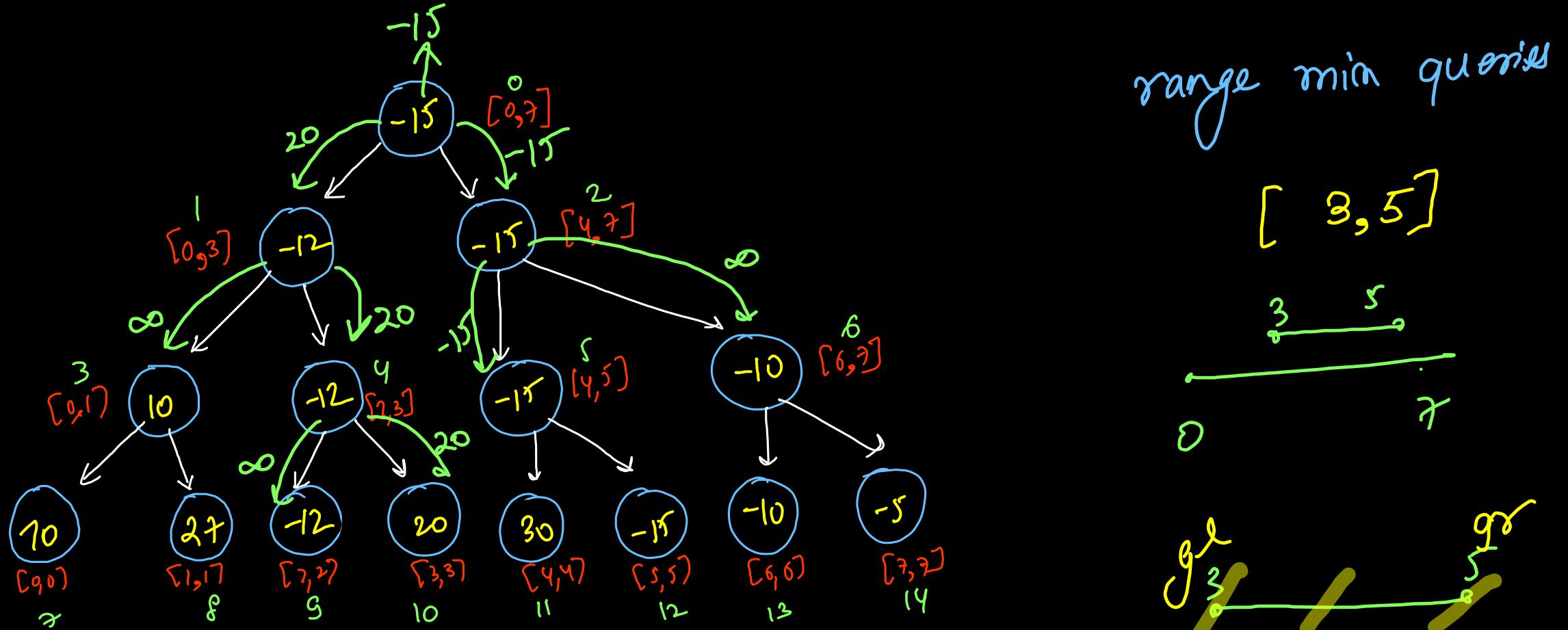
$$[4, 4] = 30$$

$$[0, 7] = -15$$

arry: { 0, 1, 2, 3, 4, 5, 6, 7  
 10, 27, -12, 20, 30, -15, -10, -5 }



{ }  $\Rightarrow m|m^m$   
 $= +\infty$   
 $l == r$   
 $\Rightarrow \text{num}[l]$



```

int[] nodes;
public static void construct(int idx, int left, int right, int[] nums){
    if(left > right) return;
    if(left == right) {
        nodes[idx] = nums[left];
        return;
    }

    int mid = left + (right - left) / 2;
    construct(2 * idx + 1, left, mid, nums);
    construct(2 * idx + 2, mid + 1, right, nums);
    nodes[idx] = Math.min(nodes[2 * idx + 1], nodes[2 * idx + 2]);
}

```

```

public static int minQuery(int idx, int nl, int nr, int gl, int gr){
    if(nr < gl || gr < nl) return Integer.MAX_VALUE; // no overlap
    if(nl >= gl && nr <= gr) return nodes[idx]; // complete overlap

    int mid = nl + (nr - nl) / 2;
    int left = minQuery(2 * idx + 1, nl, mid, gl, gr);
    int right = minQuery(2 * idx + 2, mid + 1, nr, gl, gr);
    return Math.min(left, right);
}

O(log n)

```

```

public static void update(int idx, int left, int right, int index, int value){
    if(left == right){
        nodes[idx] = value;
        return;
    }

    int mid = left + (right - left) / 2;

    if(index <= mid) update(2 * idx + 1, left, mid, index, value);
    else update(2 * idx + 2, mid + 1, right, index, value);
    nodes[idx] = Math.min(nodes[2 * idx + 1], nodes[2 * idx + 2]);
}

```

$O(q)$

```

public static void solve() throws Exception {
    int n = scn.nextInt();
    int q = scn.nextInt();

    int[] nums = new int[n];
    for(int i = 0; i < n; i++){
        nums[i] = scn.nextInt(); } } O(n)

    nodes = new int[4 * n + 5];
    Arrays.fill(nodes, Integer.MAX_VALUE);
    construct(0, 0, n - 1, nums); } } n logn D&C

    while(q-- > 0){
        int type = scn.nextInt();

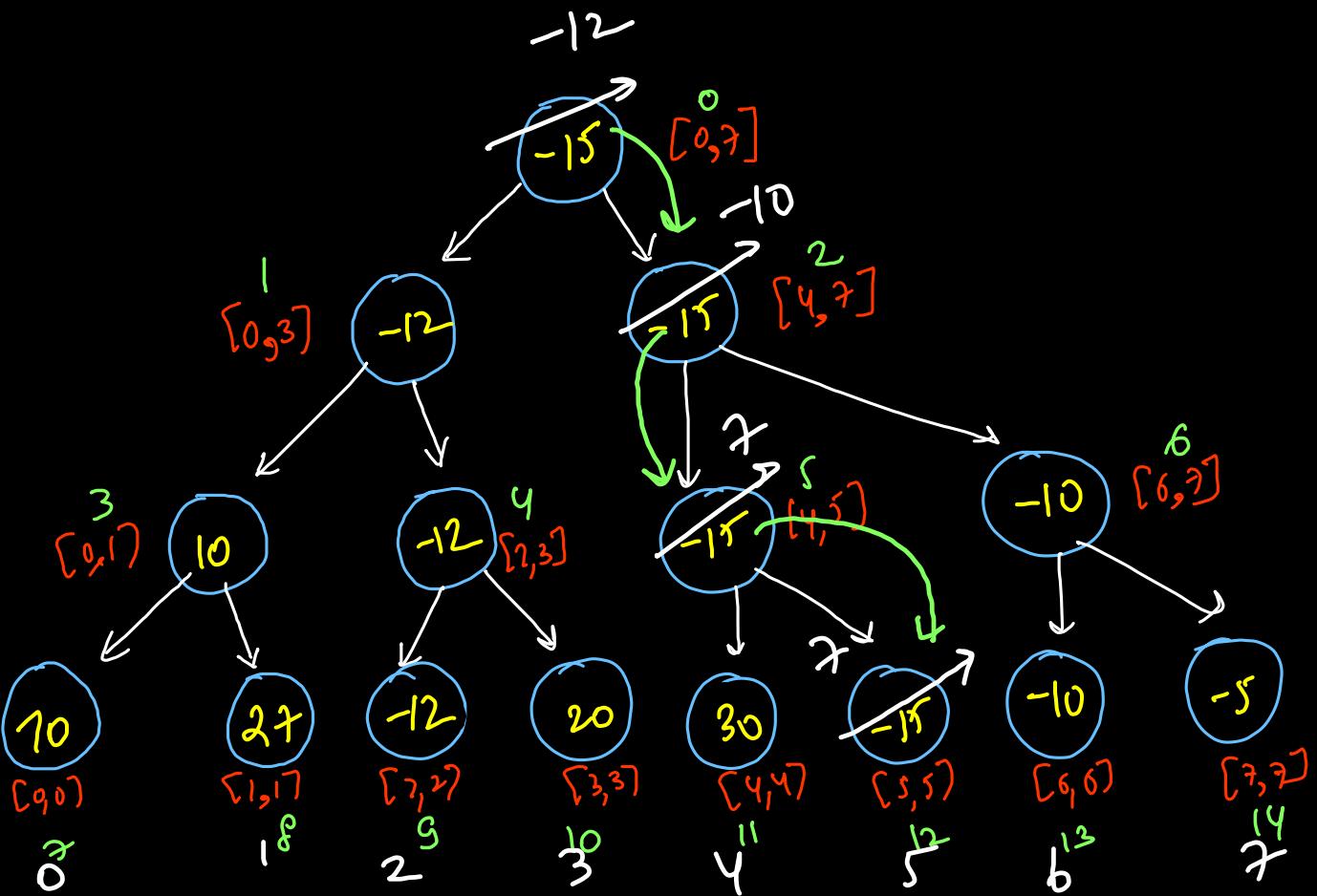
        if(type == 1){
            int index = scn.nextInt();
            int value = scn.nextInt();
            update(0, 0, n - 1, index - 1, value);
        } else {
            int left = scn.nextInt();
            int right = scn.nextInt();
            out.println(minQuery(0, 0, n - 1, left - 1, right - 1));
        }
    }
}

```

$O(\log n)$

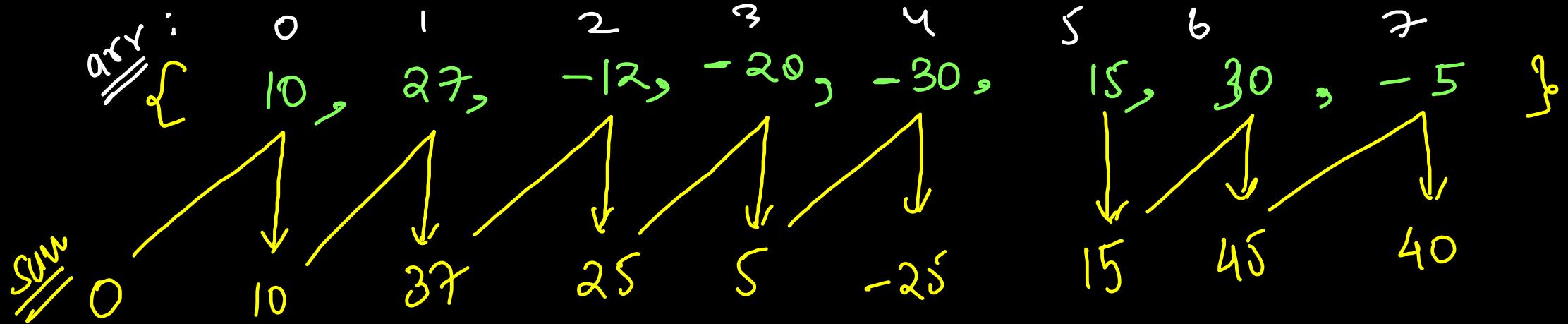
Time  
 $= n \log n + q \log n$

$n \in 10^5, q$



$\text{nums}[5] = +7$

↑  
index      ↑  
              value

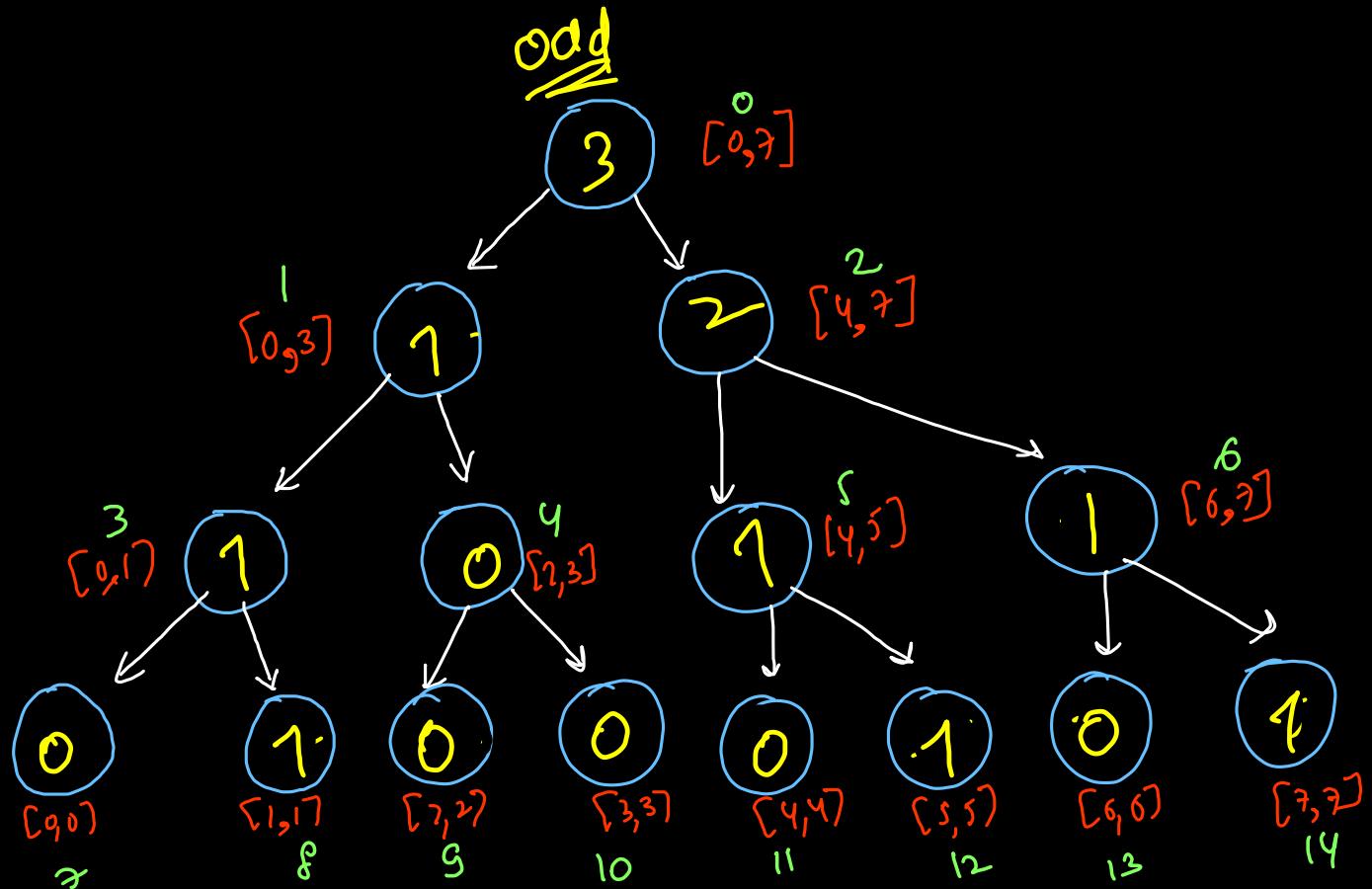


$\max^m$  sum subarray → Kadane's algorithm

$O(n)$  time  
 $O(1)$  space

$\max^m = -\infty \nearrow 10 \nearrow 37 \nearrow 45 \Rightarrow$

arry: { 0, 1, 2, 3, 4, 5, 6, 7  
 10, 27, -12, 20, 30, -15, -10, -5 }



$$\text{odd } [1, 3] = 1$$

$$\text{even } [1, 3] = 2$$

$$\text{odd } [4, 7] = 2$$

$$\text{even } [4, 7] = 2$$

```

class Solution {
    static int[] nodes;

    public static void construct(int idx, int left, int right, int[] nums) {
        if (left > right)
            return;
        if (left == right) {
            nodes[idx] = (nums[left] % 2 == 1) ? 1 : 0;
            return;
        }

        int mid = left + (right - left) / 2;
        construct(2 * idx + 1, left, mid, nums);
        construct(2 * idx + 2, mid + 1, right, nums);
        nodes[idx] = nodes[2 * idx + 1] + nodes[2 * idx + 2];
    }

    public static int countOdd(int idx, int nl, int nr, int gl, int gr) {
        if (nr < gl || gr < nl)
            return 0; // no overlap
        if (nl >= gl && nr <= gr)
            return nodes[idx]; // complete overlap

        int mid = nl + (nr - nl) / 2;
        int left = countOdd(2 * idx + 1, nl, mid, gl, gr);
        int right = countOdd(2 * idx + 2, mid + 1, nr, gl, gr);
        return left + right;
    }
}

```

```

public static void update(int idx, int left, int right, int index, int value) {
    if (left == right) {
        nodes[idx] = (value % 2 == 1) ? 1 : 0;
        return;
    }

    int mid = left + (right - left) / 2;
    if (index <= mid)
        update(2 * idx + 1, left, mid, index, value);
    else
        update(2 * idx + 2, mid + 1, right, index, value);
    nodes[idx] = nodes[2 * idx + 1] + nodes[2 * idx + 2];
}

public static List<Integer> solve(int n, int[] nums, int t, int[][] queries) {
    nodes = new int[4 * n + 5];
    construct(idx: 0, left: 0, n - 1, nums);

    List<Integer> res = new ArrayList<>();
    for (int[] q : queries) {
        if (q[0] == 0)
            update(idx: 0, left: 0, n - 1, q[1], q[2]);
        else if (q[0] == 1) {
            // count even
            int left = q[1], right = q[2];
            int odd = countOdd(idx: 0, nl: 0, n - 1, left, right);
            res.add((right - left + 1) - odd);
        } else {
            // count odd
            int left = q[1], right = q[2];
            int odd = countOdd(idx: 0, nl: 0, n - 1, left, right);
            res.add(odd);
        }
    }
    return res;
}

```

```

class Node{
    int sum, low, high;
    Node left, right;

    Node(int low, int high, int sum){
        this.low = low;
        this.high = high;
        this.sum = sum;
    }
}

```

## Point update

```

static void update(Node root, int index, int value){
    if(root.low == root.high){
        root.sum = value;
        return;
    }

    int mid = root.low + (root.high - root.low) / 2;
    if(index <= mid) update(root.left, index, value);
    else update(root.right, index, value);
    root.sum = root.left.sum + root.right.sum;
}

```

// leaf node  $O(log n)$

```

static Node build(int low, int high, int[] nums){
    if(low > high) return new Node(low, high, 0);
    if(low == high) return new Node(low, high, nums[low]);

    Node root = new Node(low, high, 0);
    int mid = low + (high - low) / 2;

    root.left = build(low, mid, nums);
    root.right = build(mid + 1, high, nums);

    root.sum = root.left.sum + root.right.sum;
    return root;
}

```

$\sum O(n \log n)$

```

static int query(Node root, int low, int high){
    if(high < root.low || root.high < low)
        return 0; // non overlap
    if(root.low >= low && root.high <= high)
        return root.sum; // completely inside
    return query(root.left, low, high)
        + query(root.right, low, high); // partial overlap
}

```

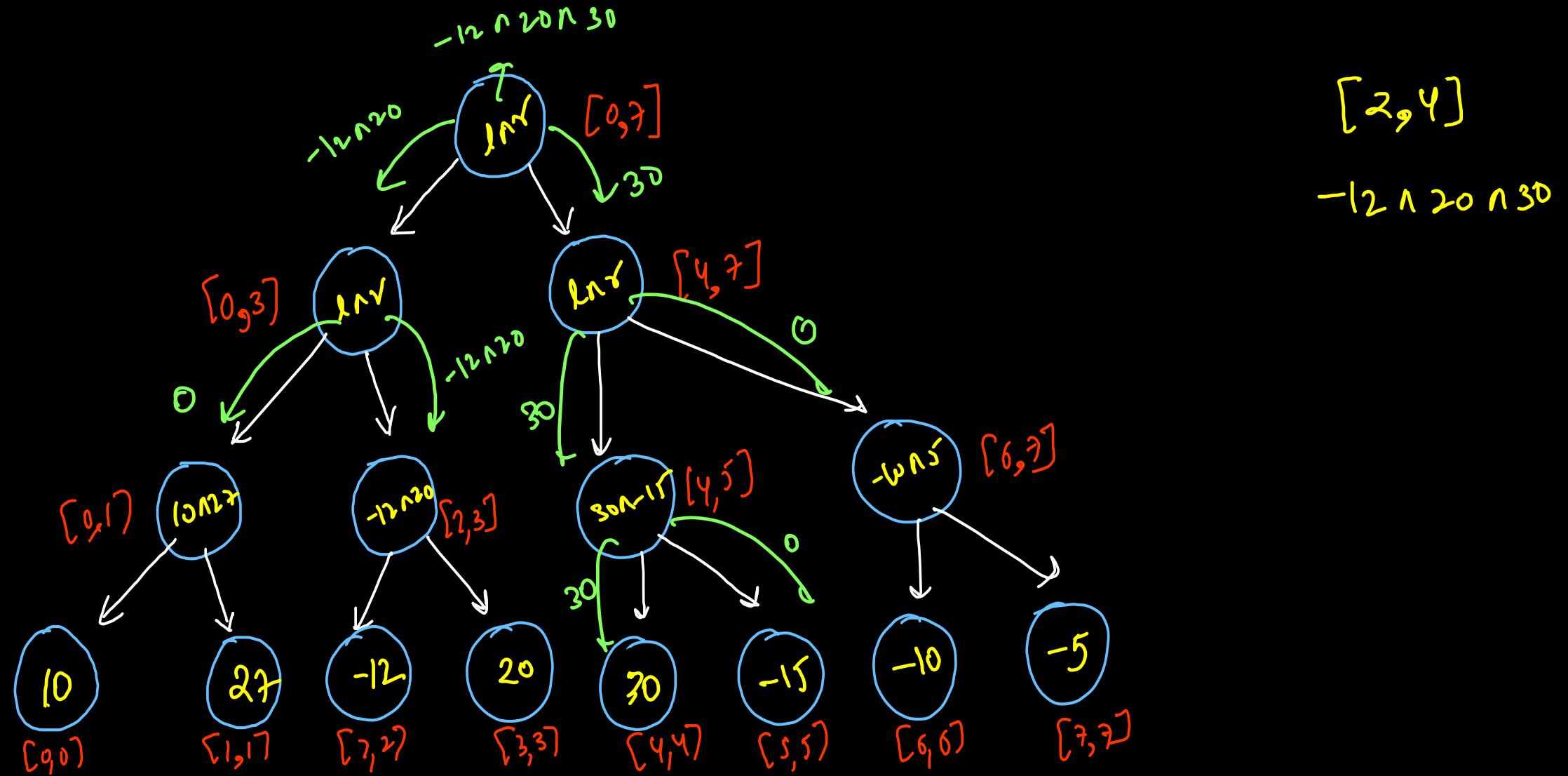
$\sum O(\log n)$

```

class NumArray {
    Node root;
    public NumArray(int[] nums) { root = Tree.build(0, nums.length - 1, nums); }
    public void update(int index, int value) { Tree.update(root, index, value); }
    public int sumRange(int low, int high) { return Tree.query(root, low, high); }
}

```

~~array~~:  $\{ 10, 27, -12, 20, 30, -15, -10, -5 \}$



```

class Node{
    int xor, low, high;
    Node left, right;

    Node(int low, int high, int xor){
        this.low = low;
        this.high = high;
        this.xor = xor;
    }
}

class Tree{
    static Node build(int low, int high, int[] nums){
        if(low > high) return new Node(low, high, 0);
        if(low == high) return new Node(low, high, nums[low]);

        Node root = new Node(low, high, 0);
        int mid = low + (high - low) / 2;

        root.left = build(low, mid, nums);
        root.right = build(mid + 1, high, nums);

        root.xor = root.left.xor ^ root.right.xor;
        return root;
    }

    static int query(Node root, int low, int high){
        if(high < root.low || root.high < low)
            return 0; // non overlap
        if(root.low >= low && root.high <= high)
            return root.xor; // completely inside
        return query(root.left, low, high)
            ^ query(root.right, low, high); // partial overlap
    }
}

```

$n \log n$   
 $\times$   
 $q \log n$

```

class Solution {
    public int[] xorQueries(int[] arr, int[][] queries) {
        Node root = Tree.build(0, arr.length - 1, arr);

        int[] res = new int[queries.length];
        for(int i = 0; i < res.length; i++){
            int left = queries[i][0];
            int right = queries[i][1];
            res[i] = Tree.query(root, left, right);
        }

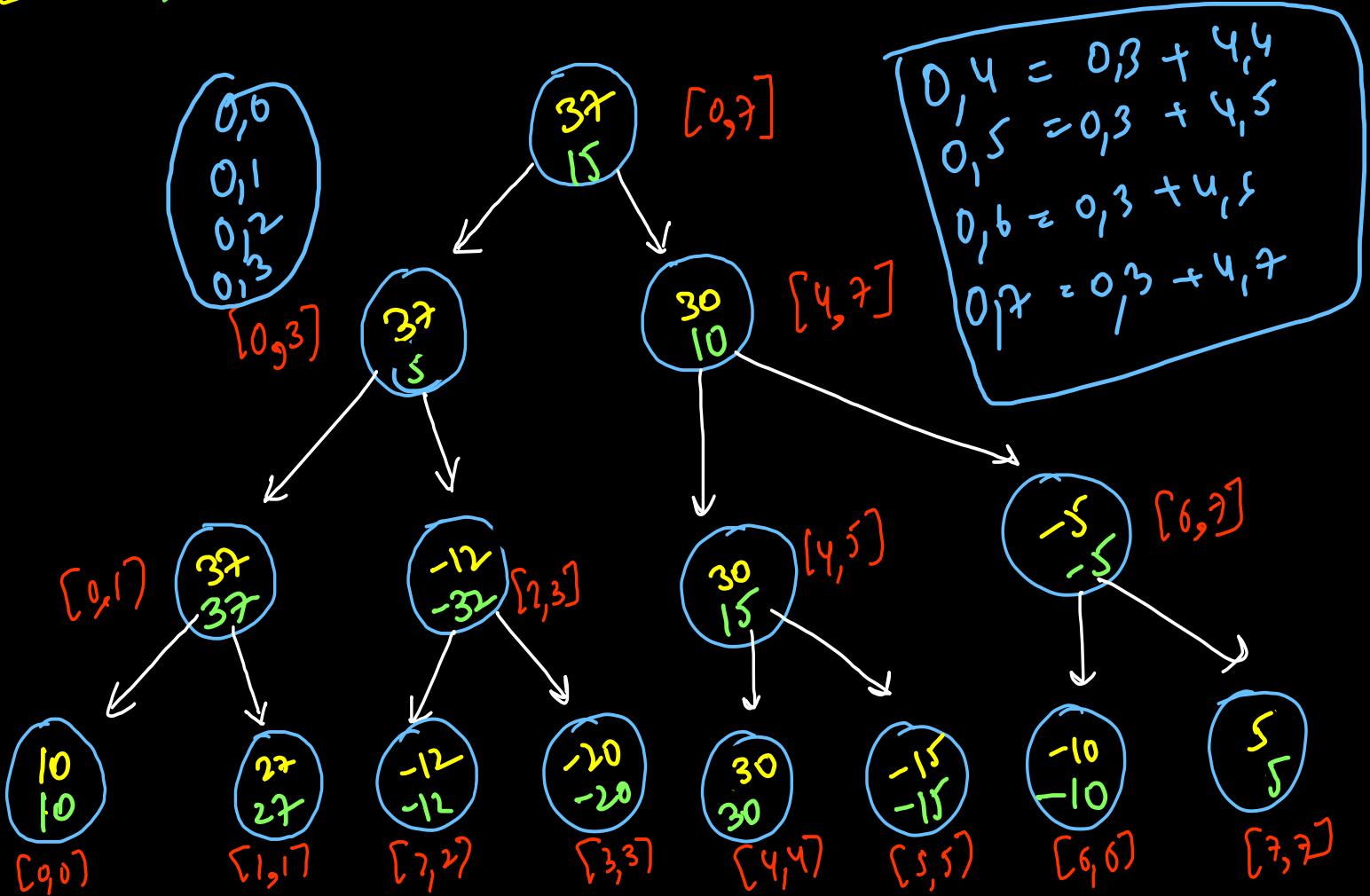
        return res;
    }
}

```

$$\text{Time} = (n+q) \log n$$

$$\text{Space} = O(4*n) \quad \underline{\text{extra Space}}$$

~~arry:~~ { 0, 1, 2, 3, 4, 5, 6, 7 }  
 arr: { 10, 27, -12, -20, 30, -15, -10, 5 }



max prefix sum

{ start, end }  
 { 2, 5 }

yellow → prefix sum  
 green → total sum

✓  $\text{root} \cdot \text{prefix} = \max(\text{root} \cdot \text{left} \cdot \text{prefix}$   
 $\text{root} \cdot \text{right} \cdot \text{prefix} + \text{root} \cdot \text{left} \cdot \text{total});$

✓  $\text{root} \cdot \text{total} = \text{root} \cdot \text{left} \cdot \text{total} + \text{root} \cdot \text{right} \cdot \text{total};$

```

static class Node {
    long prefix, total;
    int low, high;
    Node left, right;

    Node(int low, int high, long total, long prefix) {
        this.low = low;
        this.high = high;
        this.total = total;
        this.prefix = prefix;
    }
}

static class Tree {
    static Node build(int low, int high, long[] nums) {
        if (low > high)
            return new Node(low, high, 0, 0);
        if (low == high)
            return new Node(low, high, nums[low], nums[low]);

        Node root = new Node(low, high, 0, 0);
        int mid = low + (high - low) / 2;

        root.left = build(low, mid, nums);
        root.right = build(mid + 1, high, nums);

        root.total = root.left.total + root.right.total;
        root.prefix = Math.max(root.left.prefix, root.left.total + root.right.prefix);
        return root;
    }
}

```

```

    static long[] query(Node root, int low, int high) {
        if (high < root.low || root.high < low)
            return new long[]{0, 0}; // non overlap

        if (root.low >= low && root.high <= high)
            return new long[]{root.prefix, root.total}; // completely inside

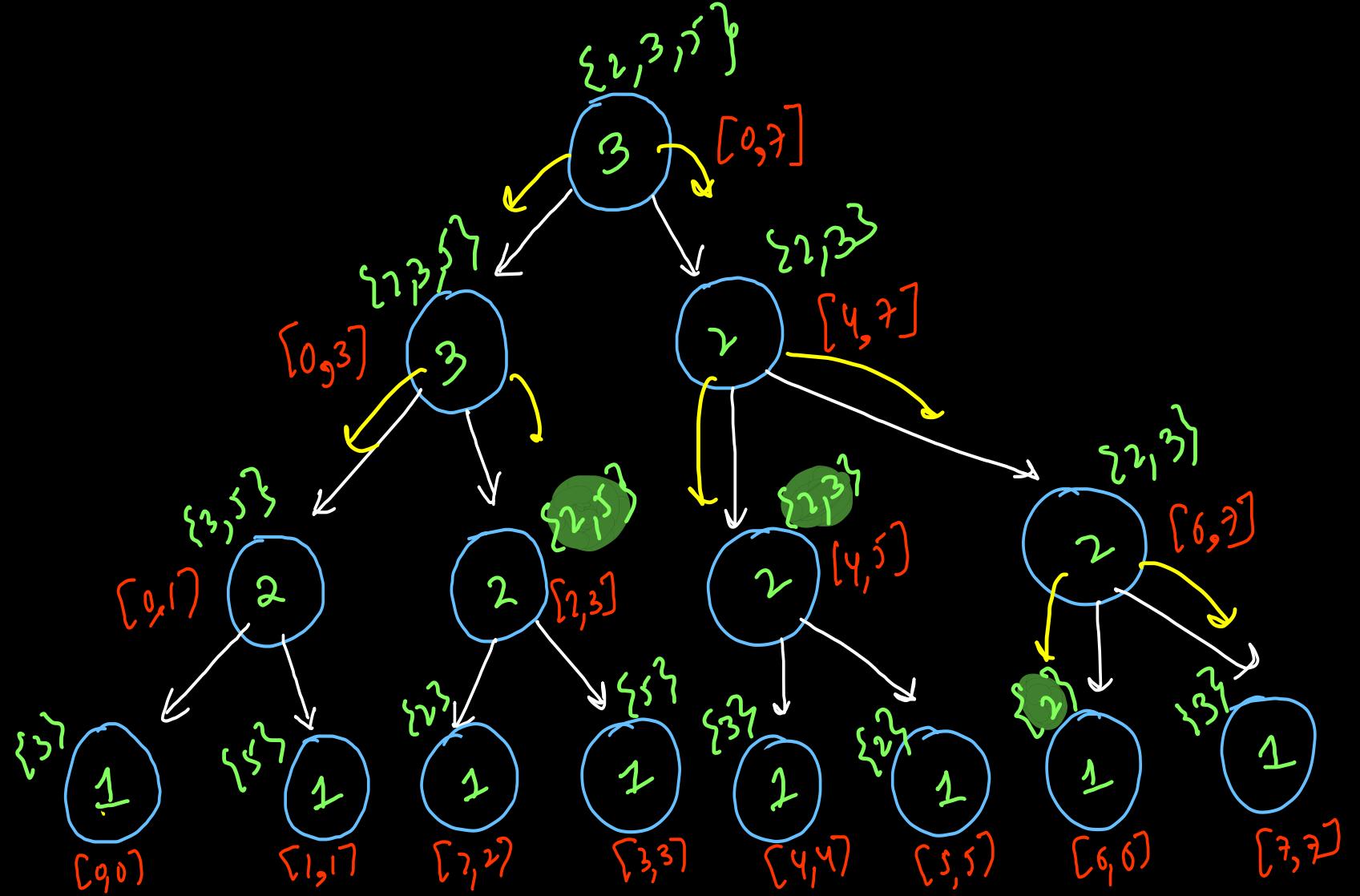
        long[] left = query(root.left, low, high);
        long[] right = query(root.right, low, high);

        long[] ans = new long[2];
        ans[0] = Math.max(left[0], left[1] + right[0]); // prefix sum
        ans[1] = left[1] + right[1];
        return ans;
    }

    long[] maxPrefixes(long a[], long L[], long R[], long N, long Q) {
        Node root = Tree.build(0, a.length - 1, a);
        long[] res = new long[(int)Q];
        for(int i = 0; i < res.length; i++)
            res[i] = Tree.query(root, (int)L[i], (int)R[i])[0];
        return res;
    }
}

```

$\{$      $0, 1, 2, 3, 4, 5, 6, 7$      $\}$   
 $3, 5, 2, 5, 3, 2, 2, 3$

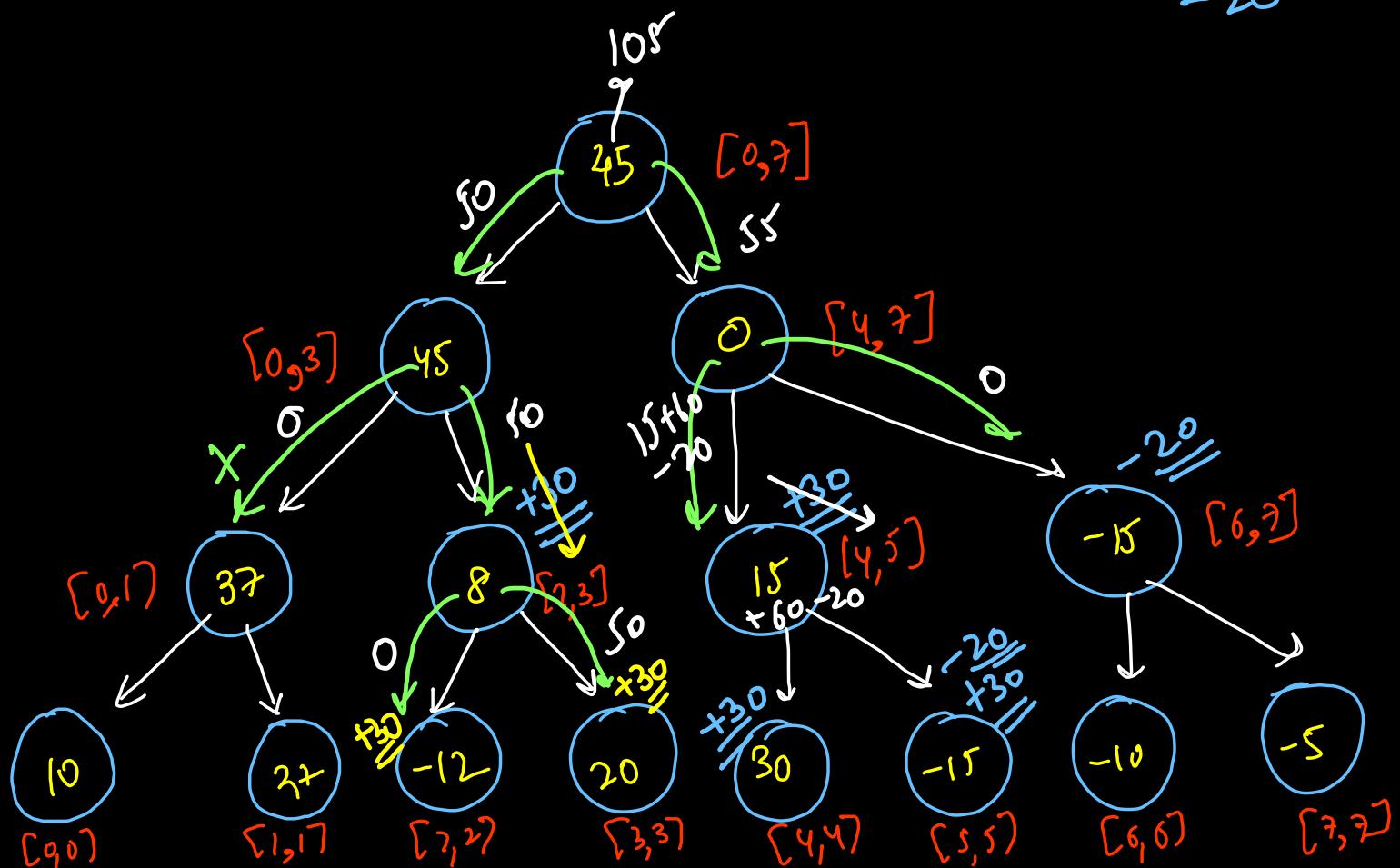


$\{ 2, 6 \}$   
 $\Rightarrow 3$

$[ 5, 7 ]$   
 $\Rightarrow 2$

- Range Sum Query } → Prefix Sum } → segment + range  
→ Point update update  
→ Range update → brute force ⇒ point update for each index  
 $O(R \log n)$  avg,  $O(n \log n)$  worst

~~ary~~:   
 { 0, 1, 2, 3, 4, 5, 6, 7 }   
 { 10, 27, -12, 20, 30, -15, -10, -5 }



build  $\rightarrow O(n \log n)$

rangesumquery  $\rightarrow O(\log n)$

ptupdate  $\rightarrow O(\log n)$

range  
 update  
 $O(\log n)$   
 { 2, 5 }  $\Rightarrow +30$   
 { 5, 7 }  $\Rightarrow -20$

rangesumquery

{ 0, 1 }  $\Rightarrow 37$   
 { 3, 5 }  $\Rightarrow 105$

Binary Indexed Tree (BIT)

or

Fenwick Tree

1 based indexing

Range Sum Query

- ① Code will be very small
- ② Space  $\rightarrow O(n)$  as array

0	1	H2	3	H2+3+4	5	5+6	7	(1-8)	9	9+10	11	9+10+11+2	13	13+14	15	(1-16)
---	---	----	---	--------	---	-----	---	-------	---	------	----	-----------	----	-------	----	--------

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
22	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0
21	0	0	1	0	0	1	1	0	1	1	1	0	0	1	1	0
20	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0

#rangesum  
query

{3, 15}

sum[15] →  
-sum[2] →

$$\text{bit}[15] + \text{bit}[15 - (\text{lsb})] + \text{bit}[14 - \text{lsb}] + \dots + \text{bit}[0]$$

$\text{bit}[14]$        $\text{bit}[12]$        $\downarrow$   
 $\text{bit}[2]$        $\text{bit}[0]$

lsb = n & -n

# point  
update

6, +30 ↗

bit[6]

+30

bit[6+lsb]  
= bit[8]

+30

bit[8+lsb]  
= bit[16]  
= 30