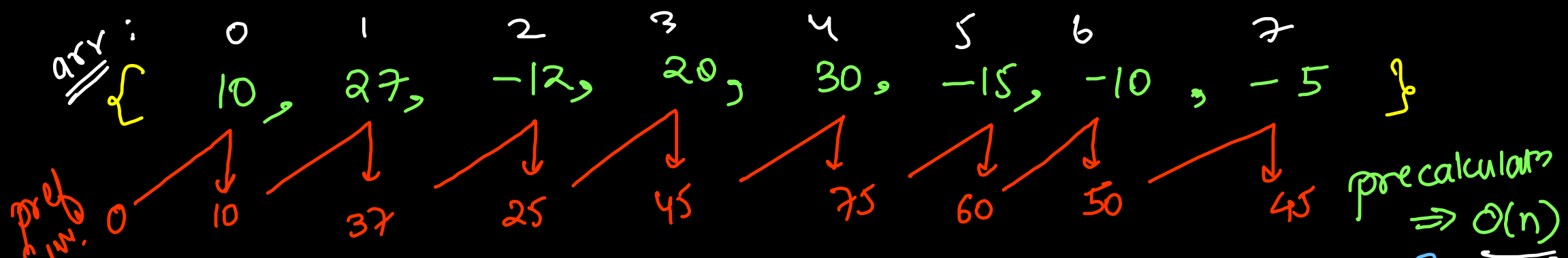


queries: $[l, r]$ Range Sum Query ⇒ prefix[r] − prefix[l−1]

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

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

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

q queries ⇒ $O(1) \times q = O(q)$

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

```java
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) per query q

Space = O(1)

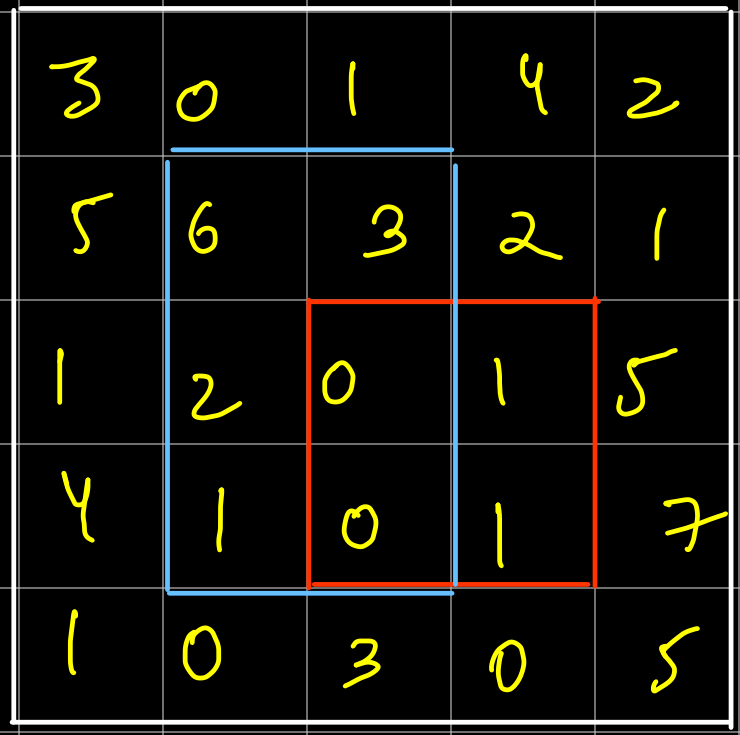
Time = O(n+q)

$10^4 + 10^4$

Leetcode 303

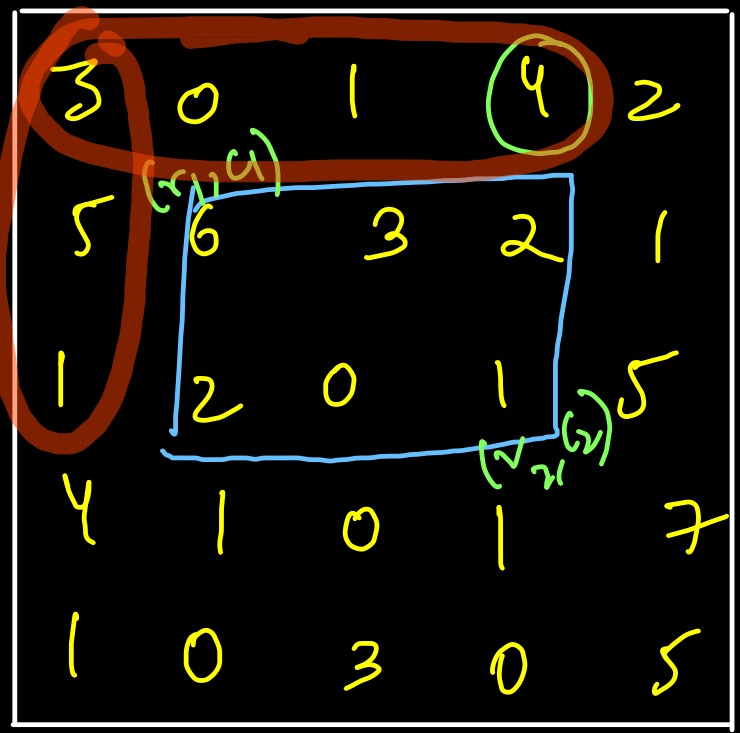
Approach 1) Time = O(row) per query

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

Prefix matrix

Approach 2

$dp[i][j] =$ nums[i][j] + dp[i-1][j] + dp[i][j-1] − dp[i-1][j-1]

→ O(n²) pre calculation

Ans query (r_1, c_1, r_2, c_2) → O(1) time

$= dp[r_2][c_2] - dp[r_2][c_1-1] - dp[r_1-1][c_2] + dp[r_1-1][c_1-1]$

```java
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;
    }
}
```

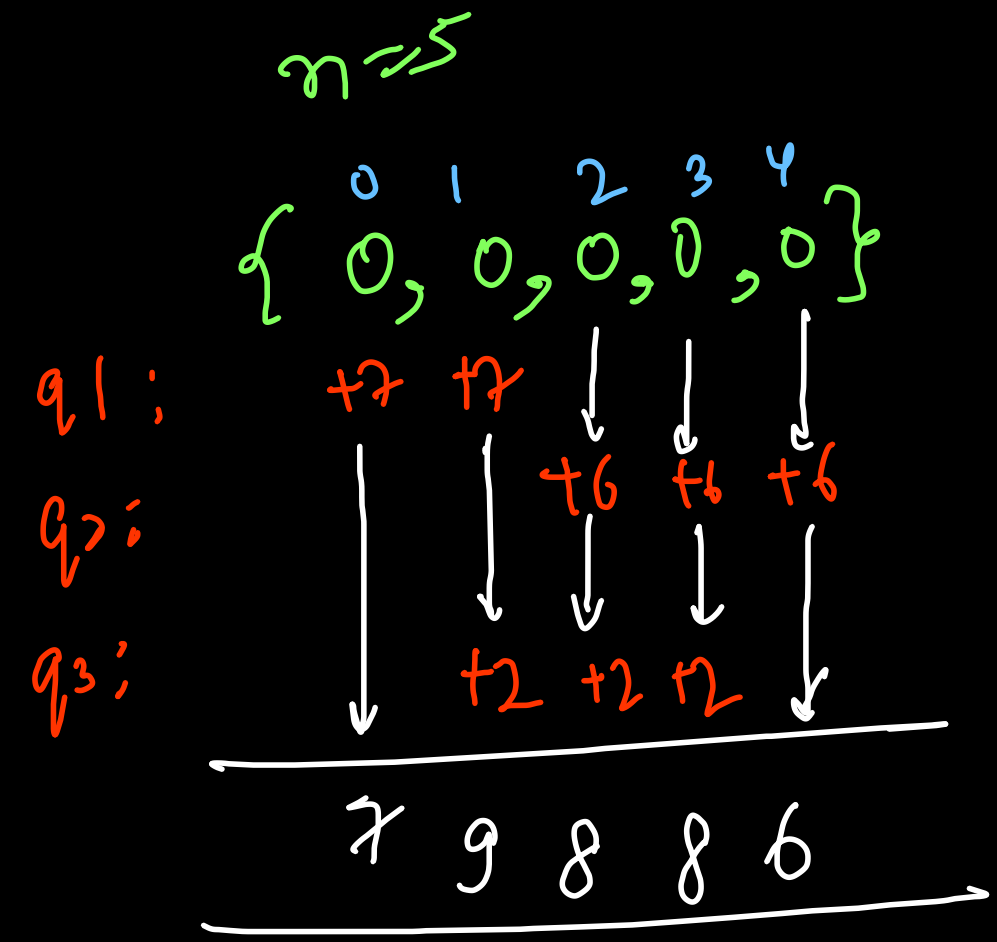
→ O(1) per query × Q

LC 304

Time
≈ $O(n^2 + q)$

Space
≈ $O(1)$

Update It Spoj



update queries

```
Input:
1
5 3
0 1 7
2 4 6
1 3 2
3
0
3
4

Output:
7
8
6
```

get ith index

Brute force

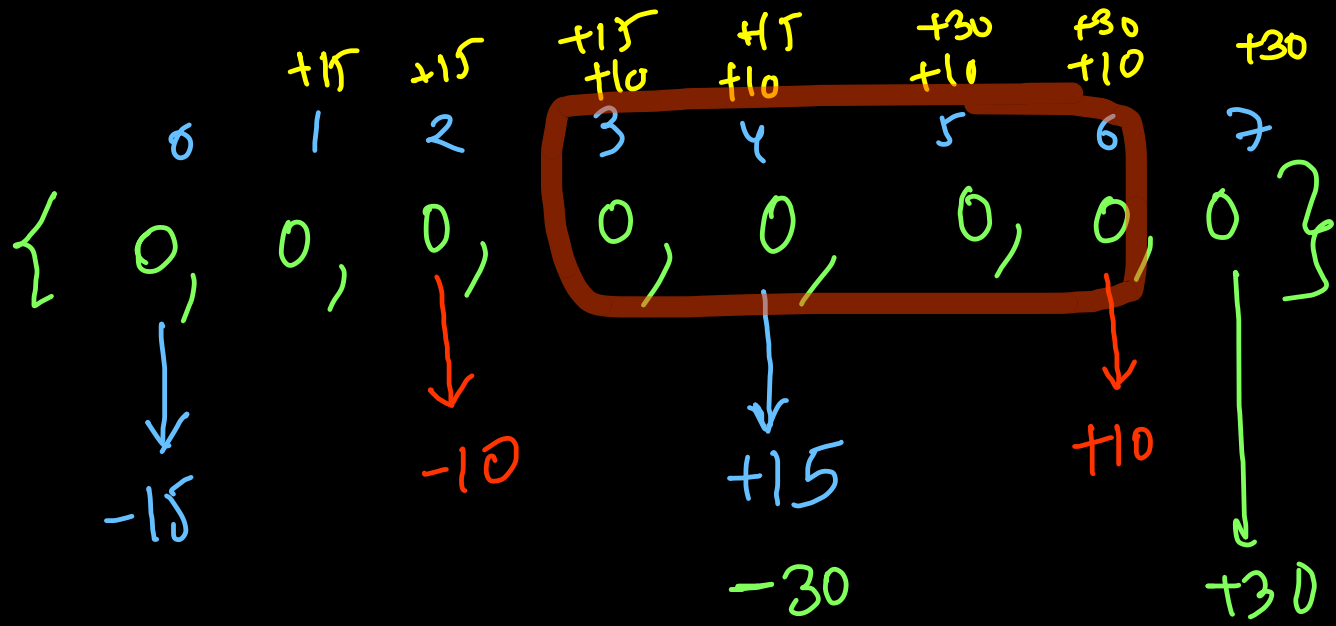
per query O(n)

Total time ≈ O(n × q)

$10^4 \times 10^5$

TLE

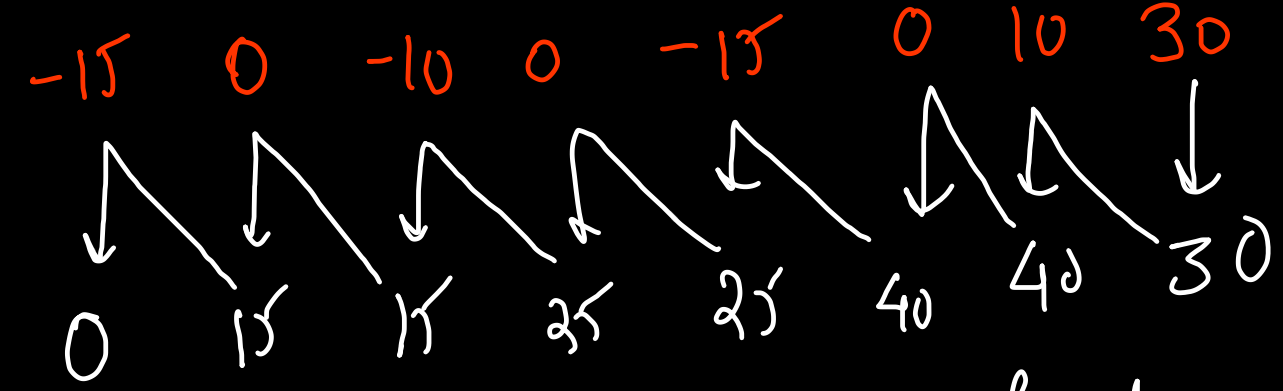
per query O(1)



q_1 : (3, 6, +10)
l r val

q_2 : (1, 4, +15)
l r val

q_3 : (5, 7, +30)
l r val



final query ⇒ O(n)

Difference array
Update It & Poj

```java
public static void solve() throws Exception {
    int t = scn.nextInt();
    while(t-- > 0){
        int n = scn.nextInt();
        int u = scn.nextInt();

        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--)
            diff[i] += diff[i + 1];

        int g = scn.nextInt();
        while(g-- > 0){
            int idx = scn.nextInt();
            out.println(diff[idx]);
        }
    }
}
```

→ length of array
→ no of update queries

→ O(1) per query × q = O(q)

+

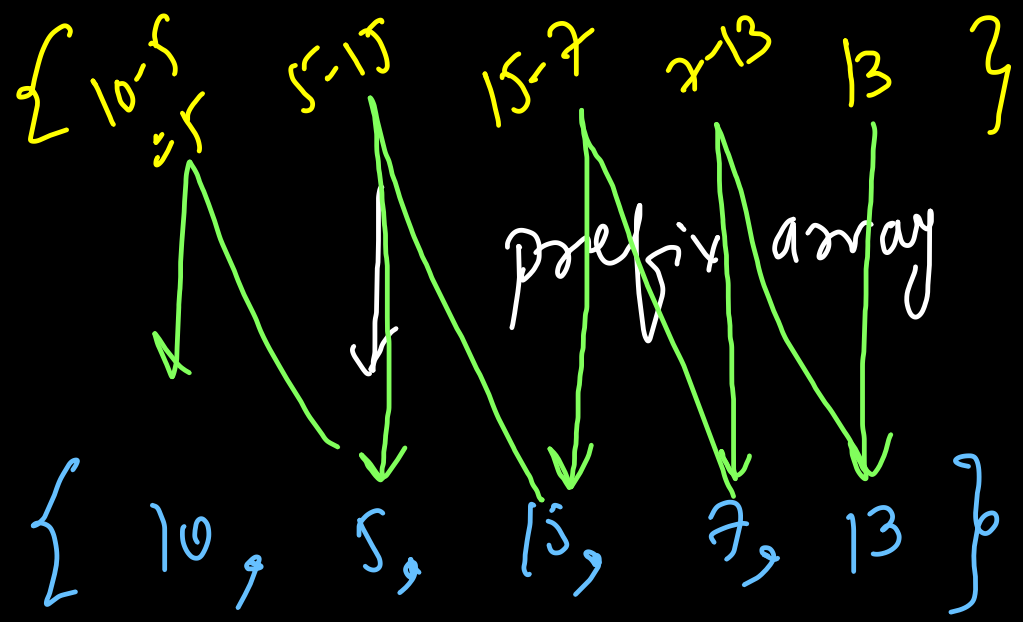
→ Array construct & O(n) = O(n)

+

→ get ith idx queries ⇒ O(1) × g = O(g)

{ 10, 5, 15, 7, 13 }

↓ Diff Array



GFG If input is not containing 0's

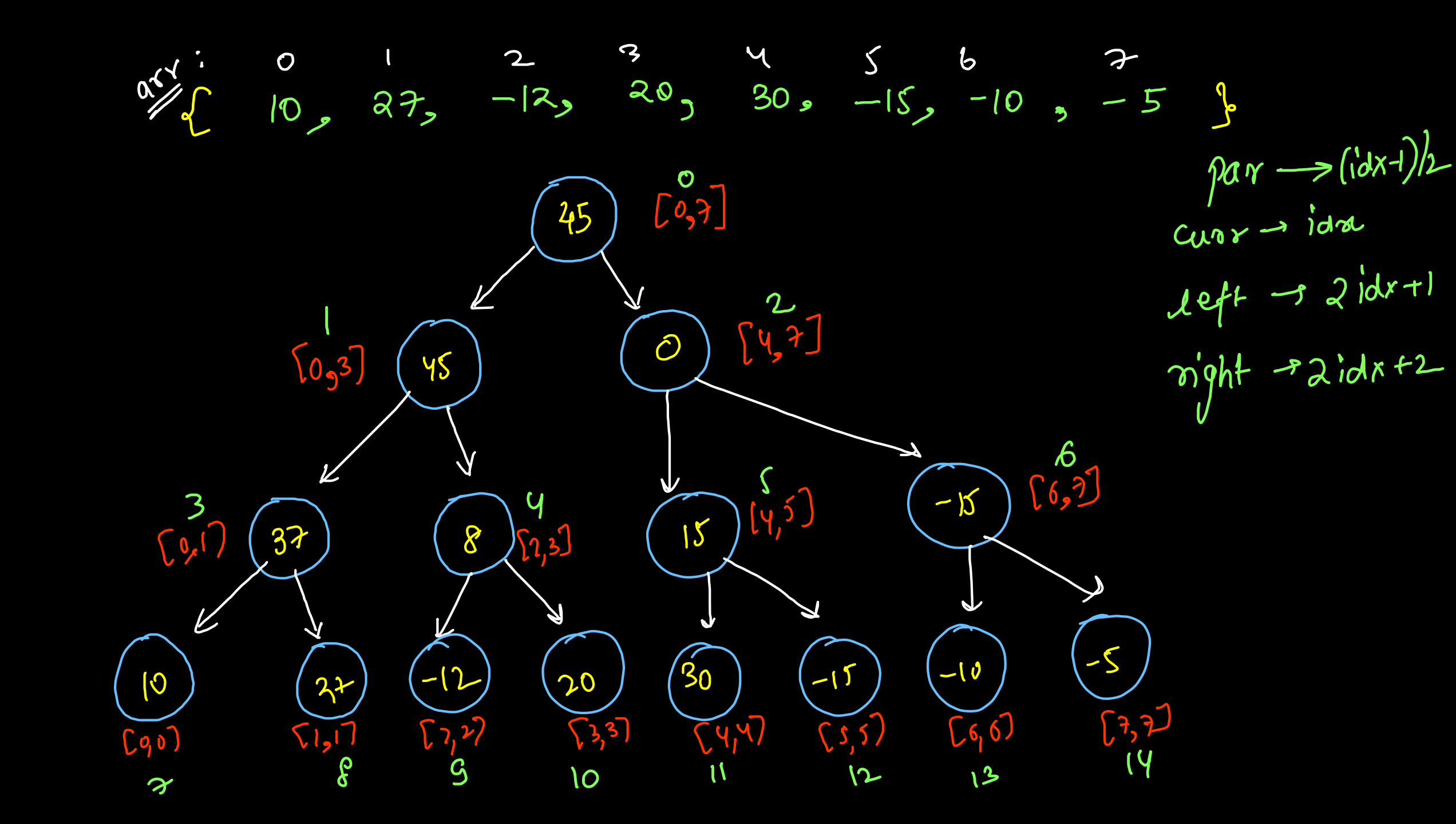
no update queries

diff[i] = Arr[i] - A[i+1]

Range Sum Query Segment Tree

⇒ Range query ⇒ Point update

	0	1	2	3	4	5	6	7
ary:	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 → (idx-1)/2
curr → idx
left → 2 idx+1
right → 2 idx+2
45
[0,7]
0
45
[0,3]
1
0
[4,7]
2
37
[0,1]
3
8
[2,3]
4
15
[4,5]
5
-15
[6,7]
6
10
[0,0]
7
27
[1,1]
8
-12
[2,2]
9
20
[3,3]
10
30
[4,4]
11
-15
[5,5]
12
-10
[6,6]
13
-5
[7,7]
14

```
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;
}
```

O(n log n)

→ constructor parameter

```
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;
    }
}
```

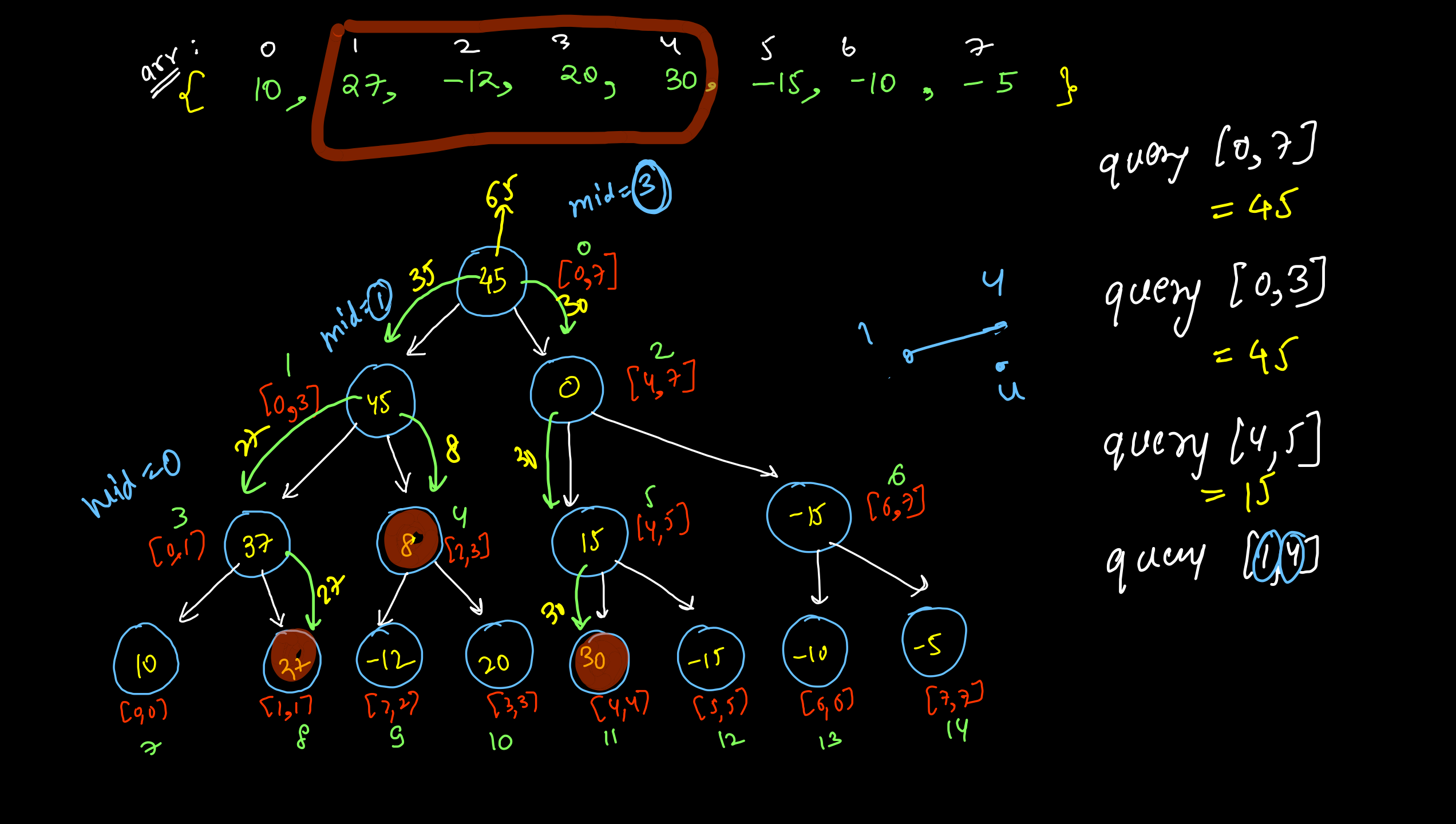
1C307

```java
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

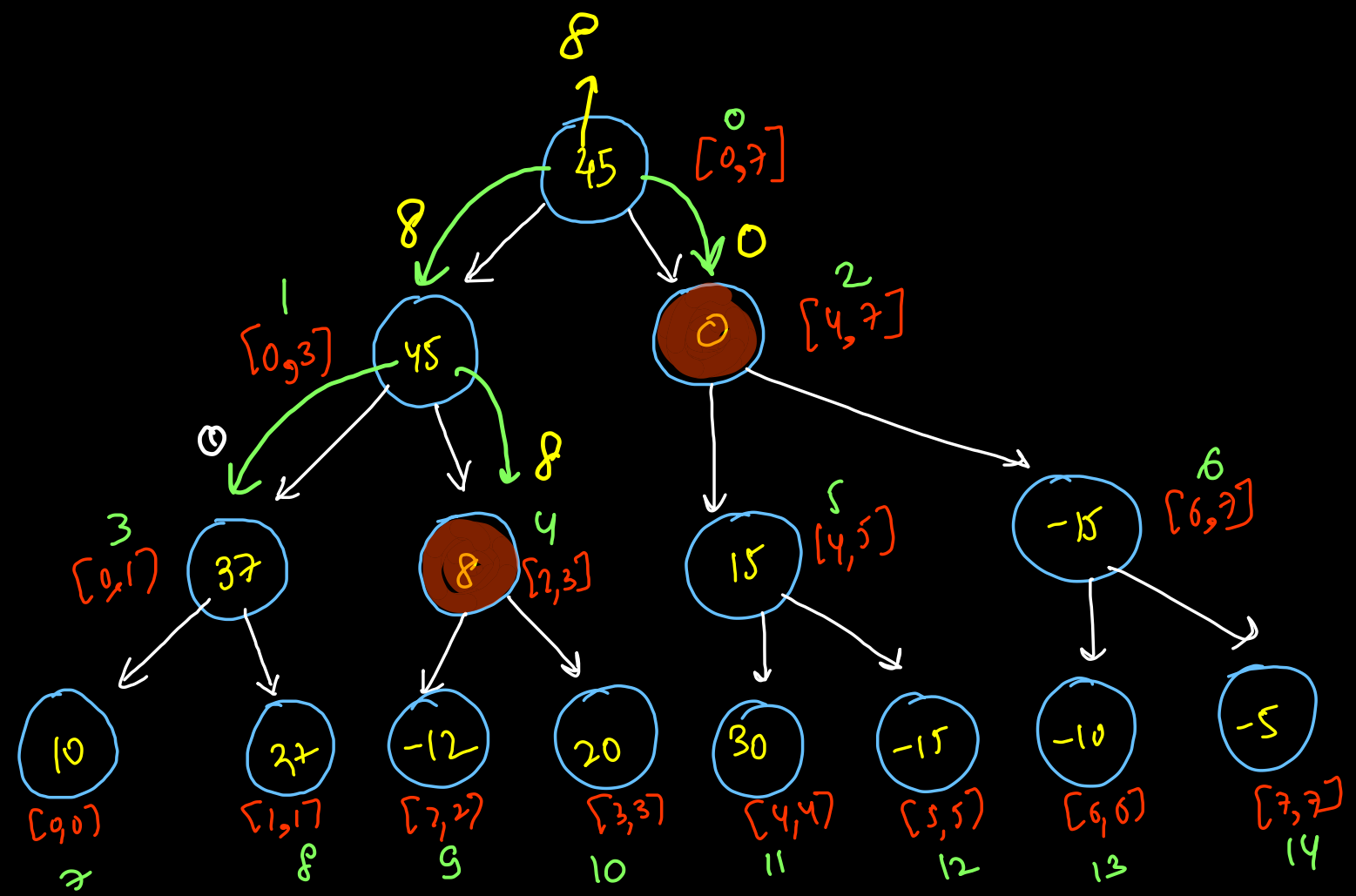
→ O(nlgn) time

O(n) space

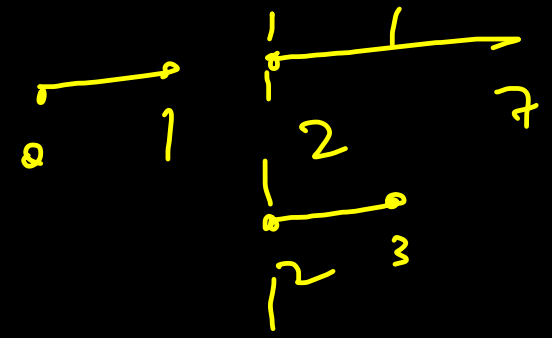

arr :
0 1 2 3 4 5 6 7
{ 10, 27, -12, 20, 30, -15, -10, -5 }
mid=3
mid=1
mid=0
65
45
35
30
[0,7]
0
45
[0,3]
1
0
[4,7]
2
37
[0,1]
3
8
[2,3]
4
15
[4,5]
5
-15
[6,7]
6
27
8
30
27
10
27
-12
20
30
-15
-10
-5
[0,0]
0
[1,1]
1
[2,2]
2
[3,3]
3
[4,4]
4
[5,5]
5
[6,6]
6
[7,7]
7
4
1
u
query [0,7] = 45
query [0,3] = 45
query [4,5] = 15
query [1,4]

arr : 0 1 2 3 4 5 6 7

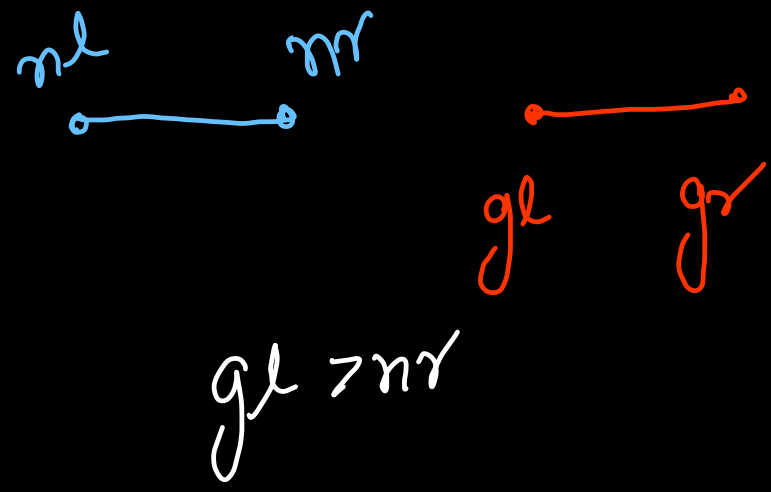
{ 10, 27, -12, 20, 30, -15, -10, -5 }



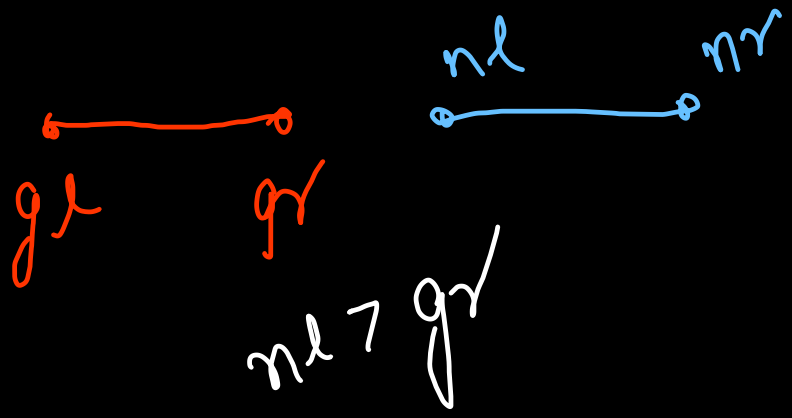
query = [2, 7]



return 0;
no overlapping

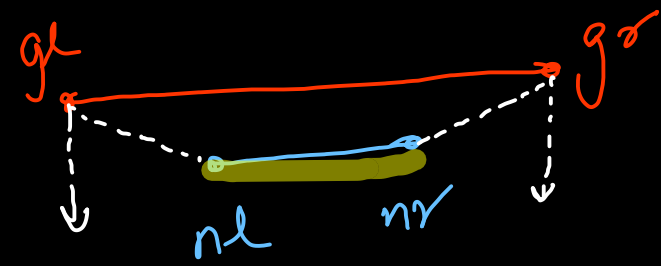


gl > nr



nl > gr

node completely lies inside the given range



nl ≥ gl && nr ≤ gr

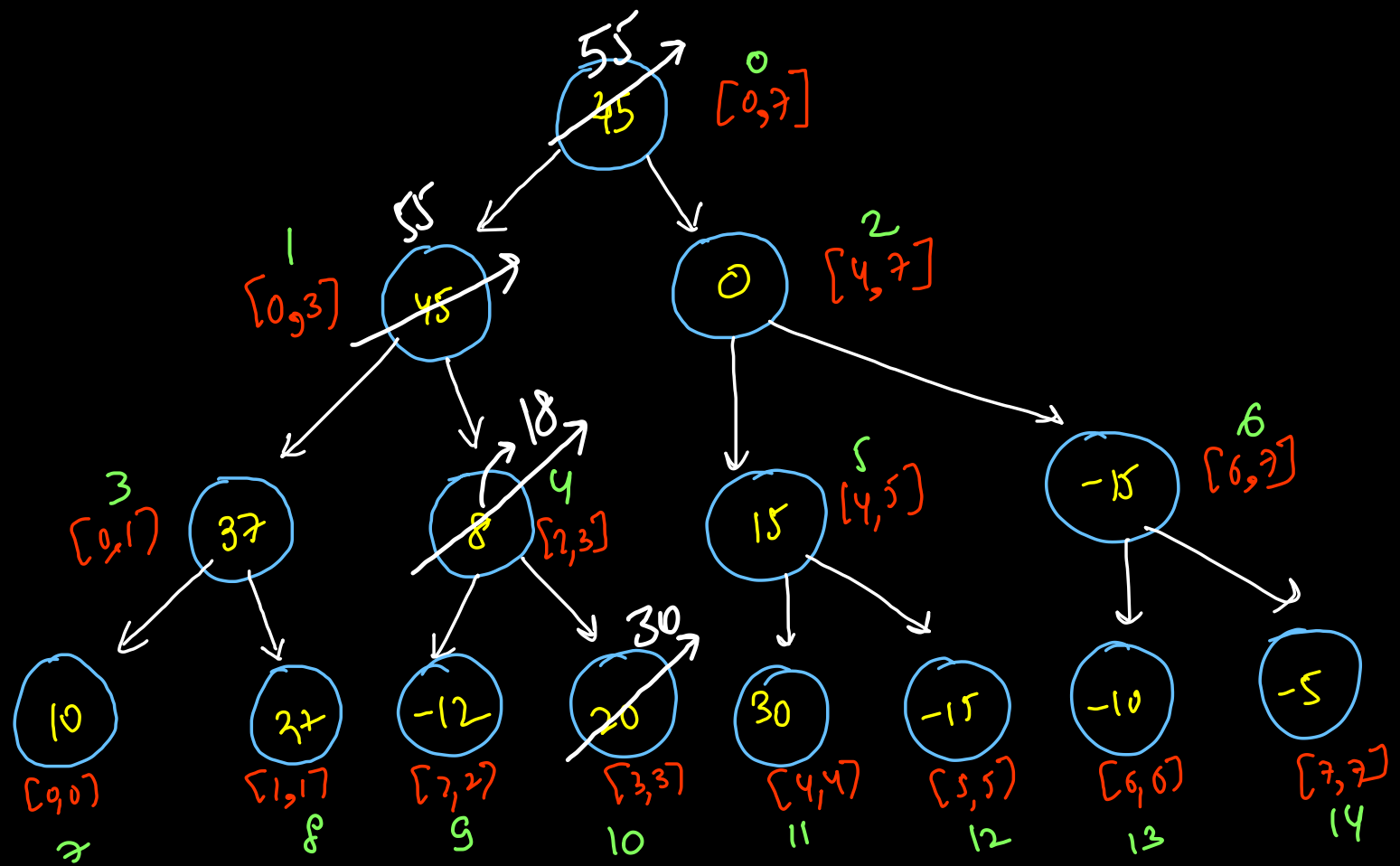
```java
public int dfs(int gl, int gr, int idx, int nl, int nr){
    if(gl > nr || nl > gr) return 0; // no overlapping
    if(nl >= gl && nr <= gr) return nodes[idx]; // completely inside

    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(logn) per query

Point update

arr :

0	1	2	3	4	5	6	7
10	27	-12	20	30	-15	-10	-5



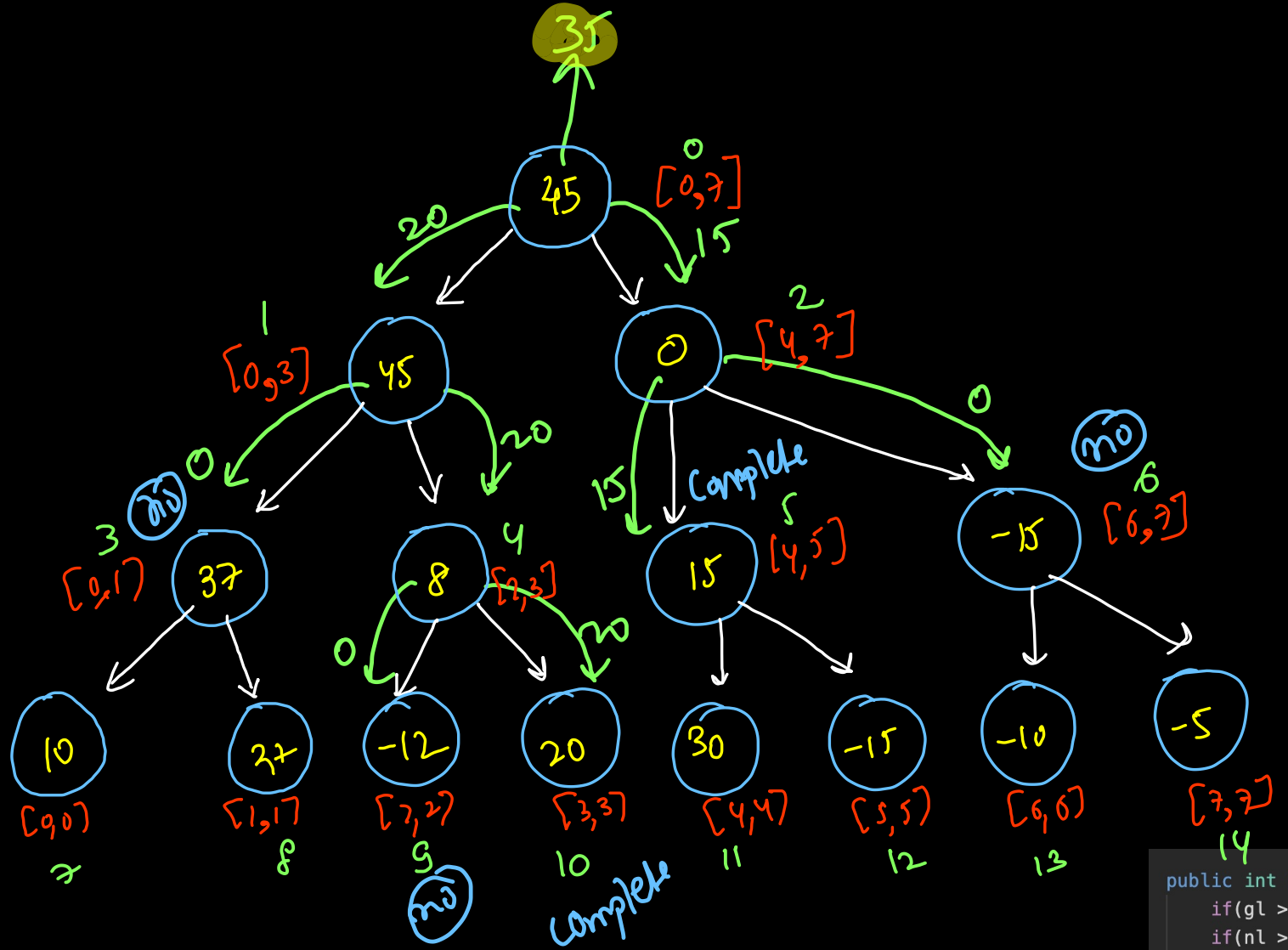
update

nums[3] → 30

diff = 30 − 20

↓ ↓

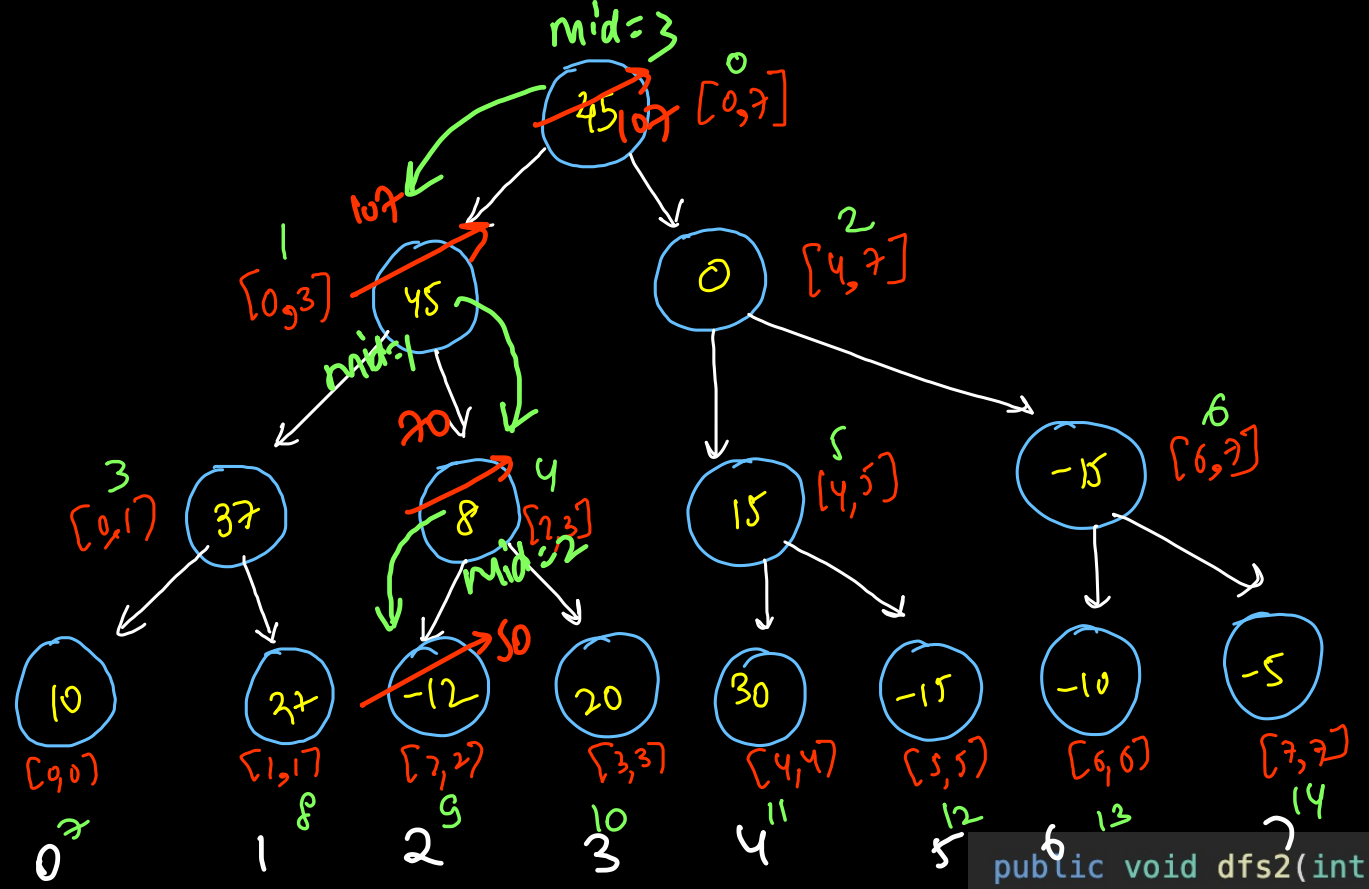
val − oldval



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

nums[2] = 50

```
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];
}
```

→ O(log n)

Range minimum/maxm query

	0	1	2	3	4	5	6	7
arr :	10	27	-12	20	30	-15	-10	-5

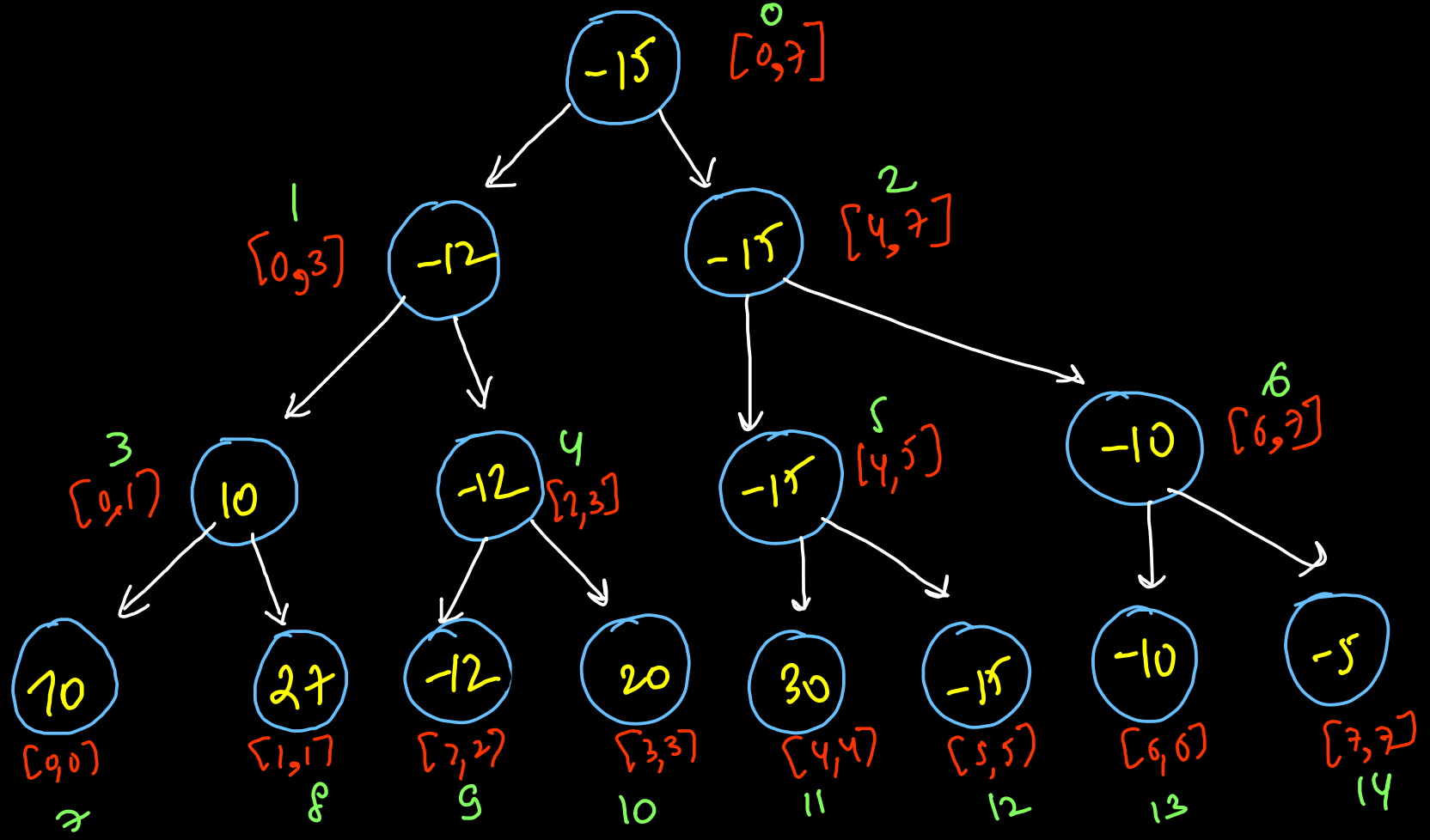
$[1,4] = -12$

$[3,7] = -15$

$[4,4] = 30$

$[0,7] = -15$

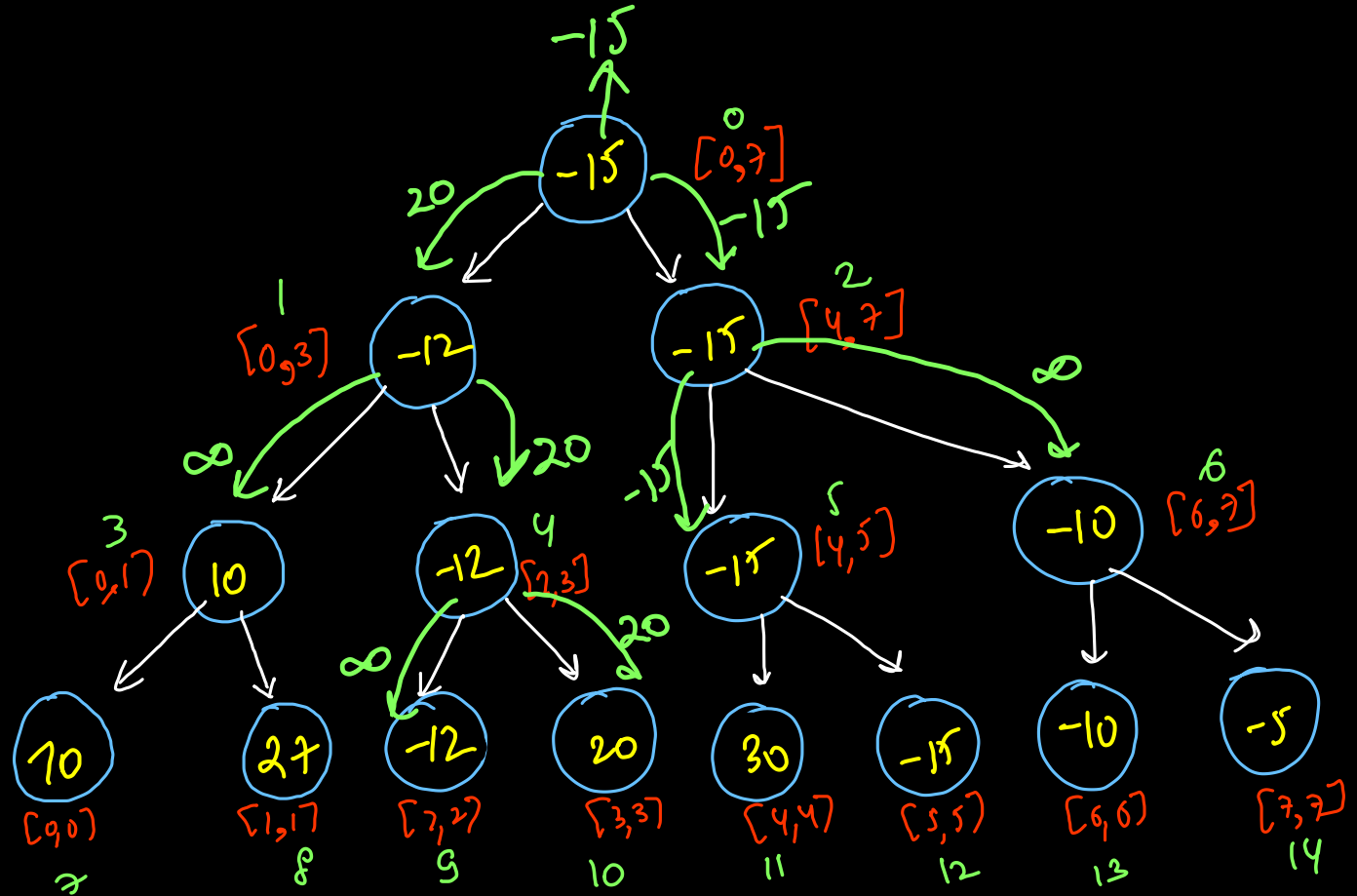
arr:	0	1	2	3	4	5	6	7
	10	27	-12	20	30	-15	-10	-5



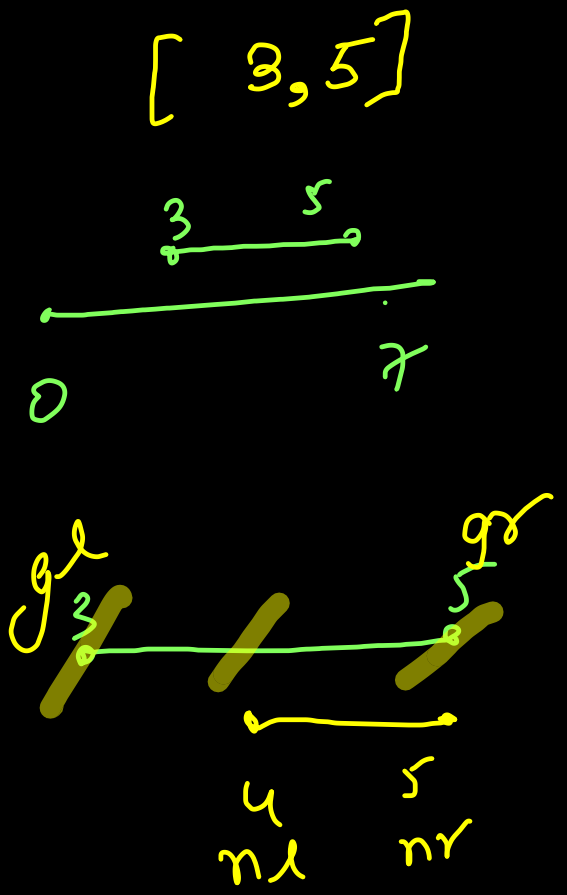
{ } ⇒ min = $+\infty$

l == r

⇒ nums[l]



range min queries



```java
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(logn)

```java
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(logn)

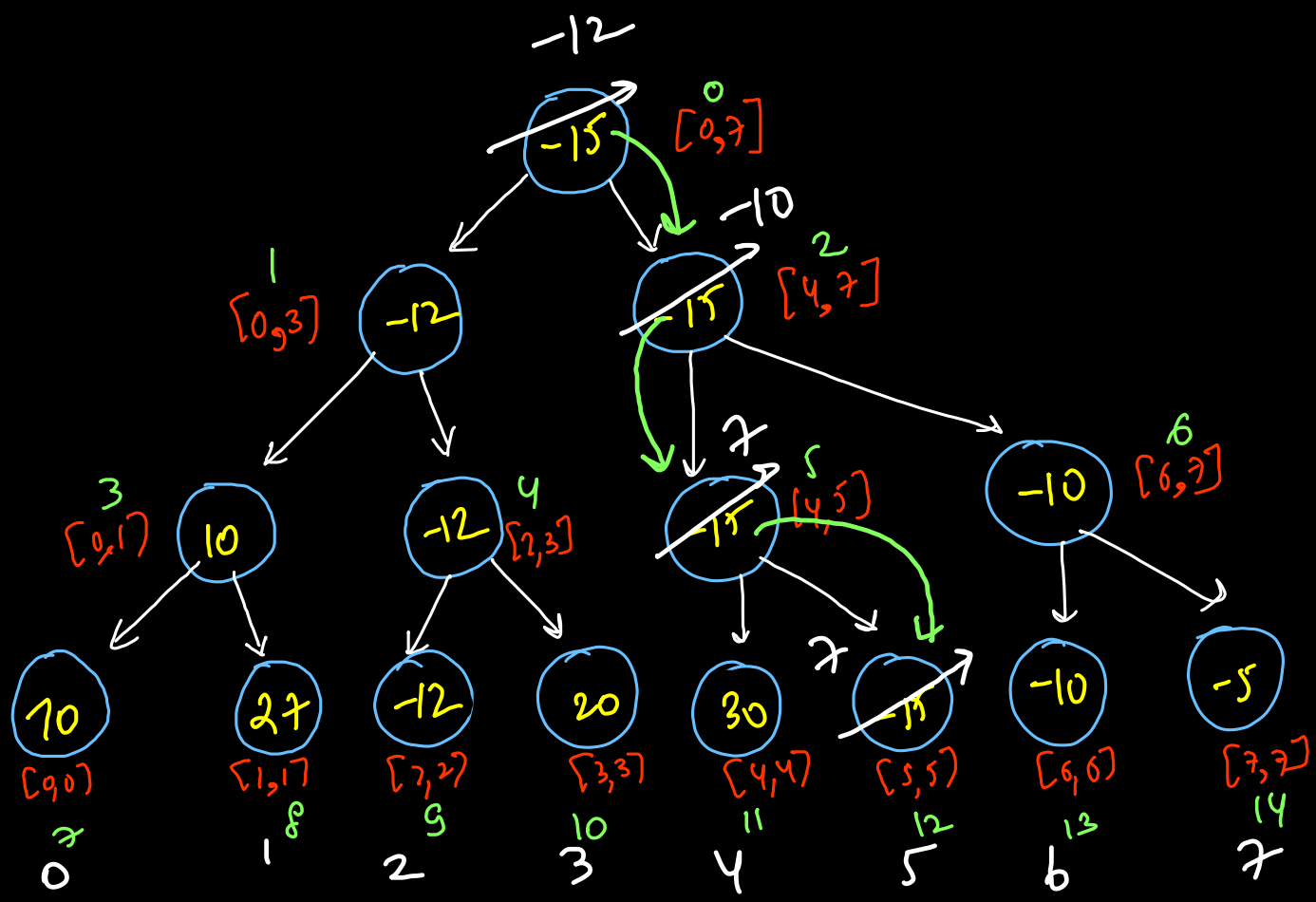
```java
public static void solve() throws Exception {
    int n = scn.nextInt();
    int q = scn.nextInt();

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

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

    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(n) → nlogn D&C

O(q)

Time
= nlogn + qlogn

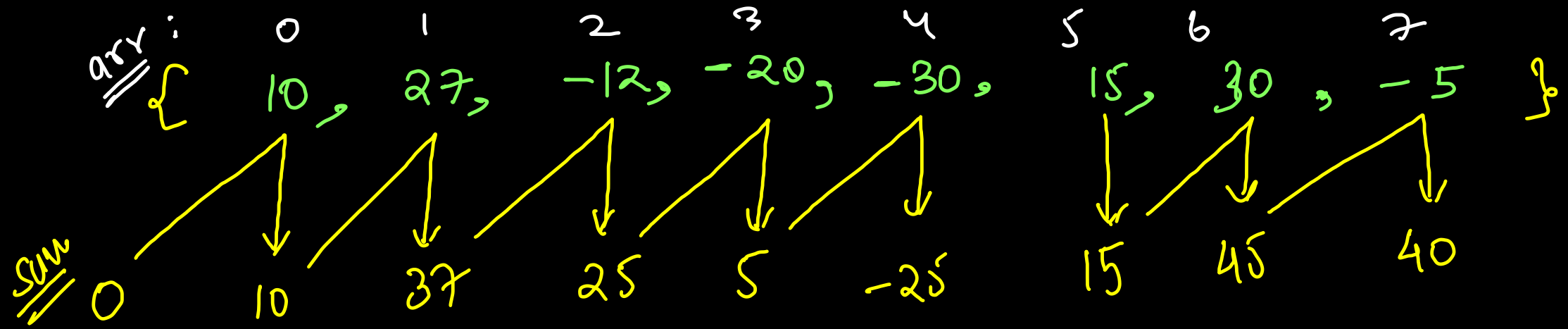
$n, q \in 10^5, 10^5$



nums[5] = +7

index

value

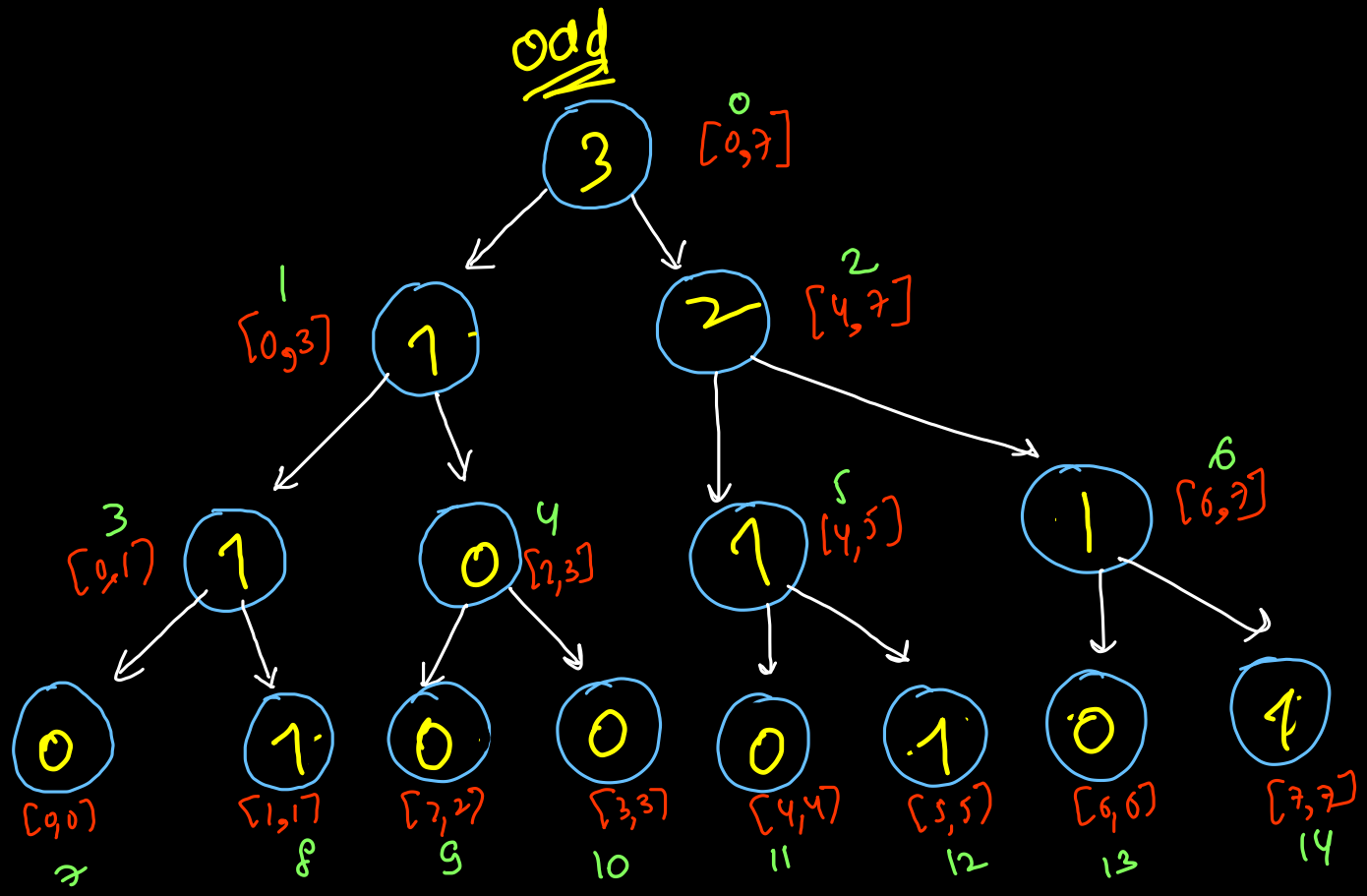


Maxm Sum subarray → Kadane's algorithm

O(n) time
O(1) space

maxm = ~~-∞~~ 10 ~~37~~ 45 ⇒

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



odd $[l, r] = 1$

even $[1,3] = 2$

odd $[4,7] = 2$

even $[4,7] = 2$

```java
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;
    }
```

```java
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;
}
```

```java
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

```java
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(logn)

```java
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;
}
```

O(nlogn)

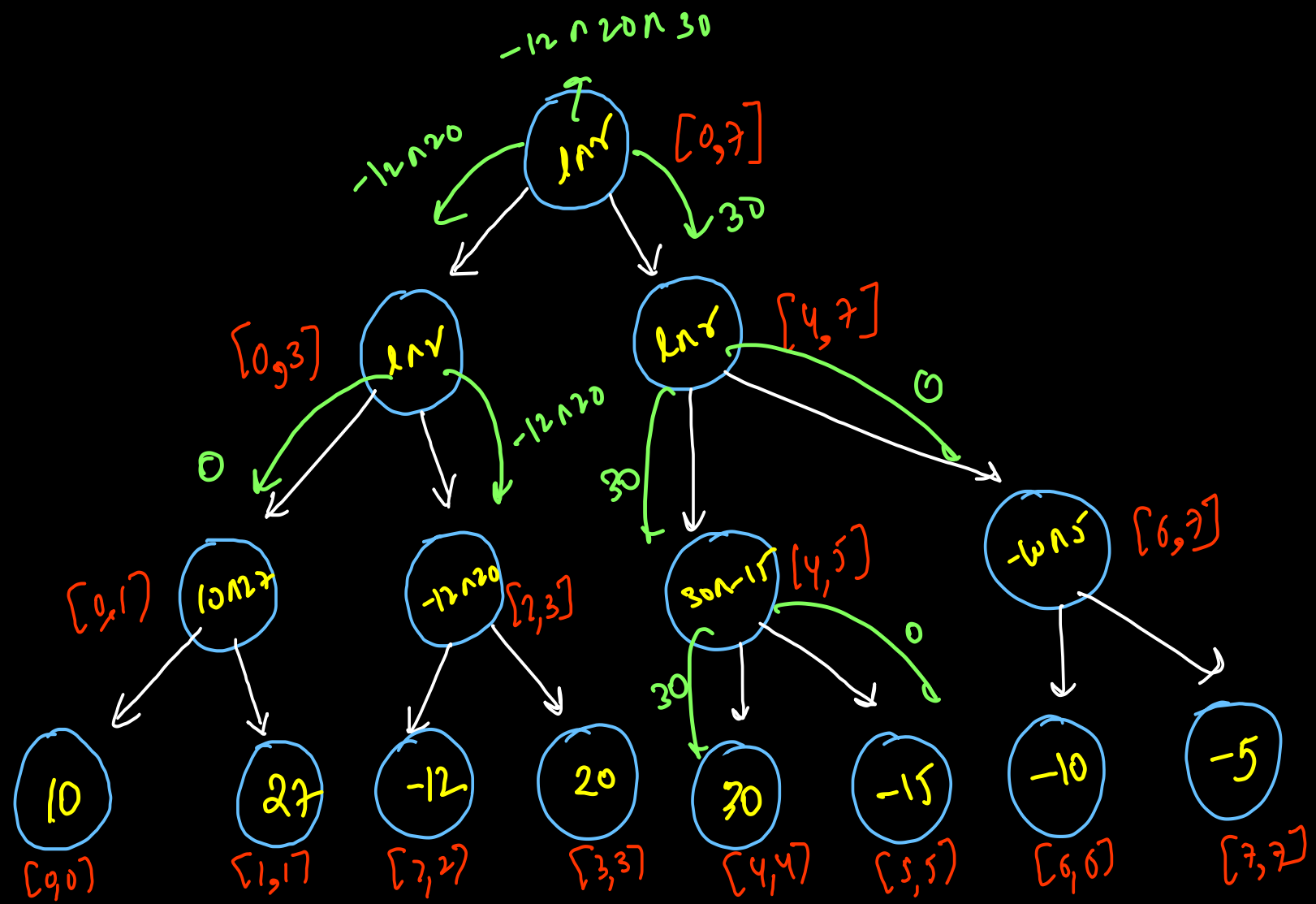
```java
static int query(Node root, int low, int high){
    if(high < root.low || root.high < low)
        return 0; // non overlap
    if(root.low >= low && root.high <= high)
        return root.sum; // completely inside
    return query(root.left, low, high)
         + query(root.right, low, high);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

O(logn) avg

```java
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); }
}
```

arr :

0	1	2	3	4	5	6	7
10	27	-12	20	30	-15	-10	-5



[2,4]

-12 ∧ 20 ∧ 30

```java
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
    }
}
```

nlogn

logn

```java
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;
    }
}
```

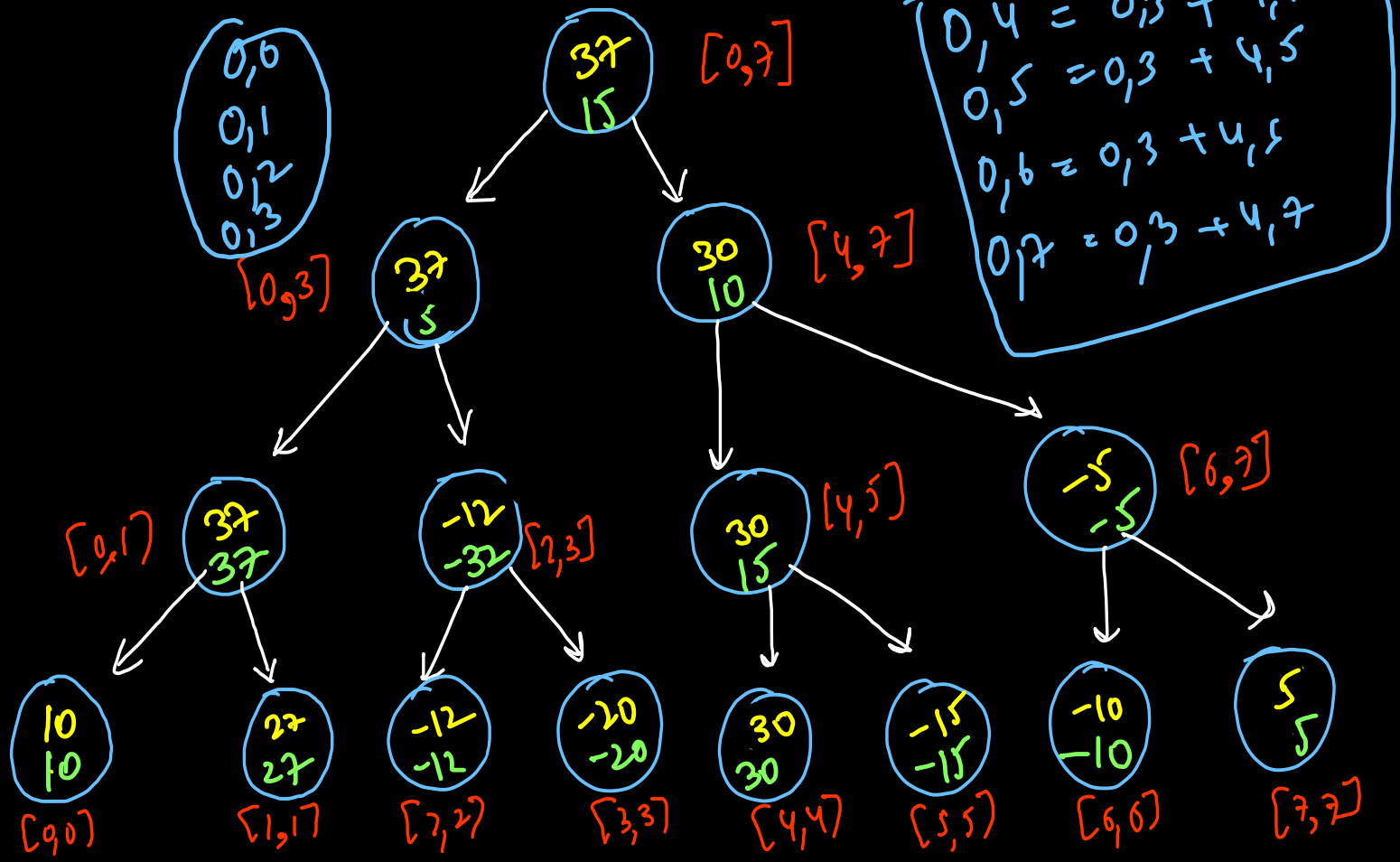
nlogn + qlogn

time = (n+q)logn

Space = O(4*n) extra space

arr :

0	1	2	3	4	5	6	7
10	27	-12	-20	30	-15	-10	5



$0,4 = 0,3 + 4,4$
$0,5 = 0,3 + 4,5$
$0,6 = 0,3 + 4,6$
$0,7 = 0,3 + 4,7$

max^m prefix sum

st end
{ 2, 5 }

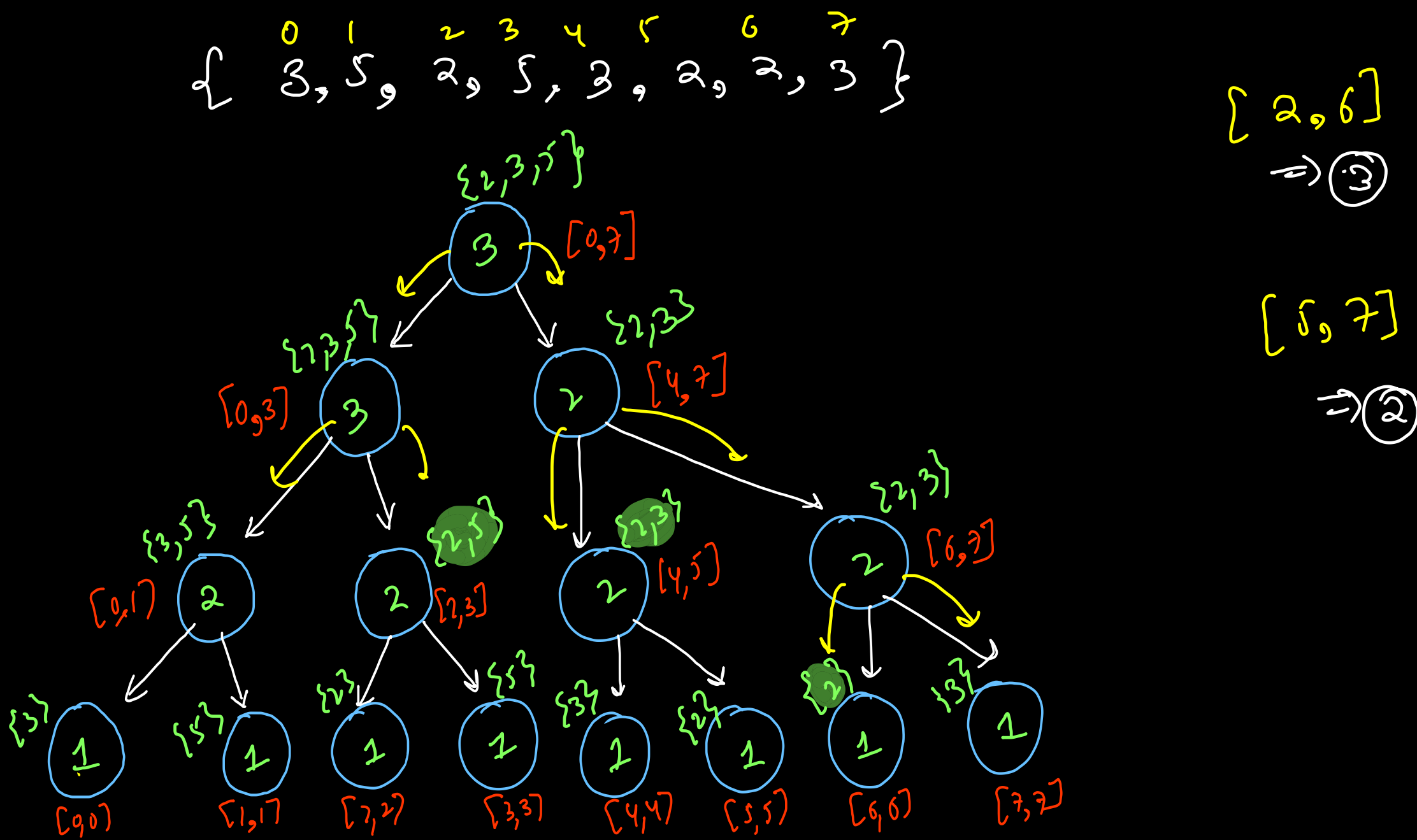
yellow → prefix sum

green → total sum

✓ root.prefix = max(root.left.prefix
root.right.prefix + root.left.total);

✓ root.total = root.left.total + root.right.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]
⇒ 3
[5,7]
⇒ 2