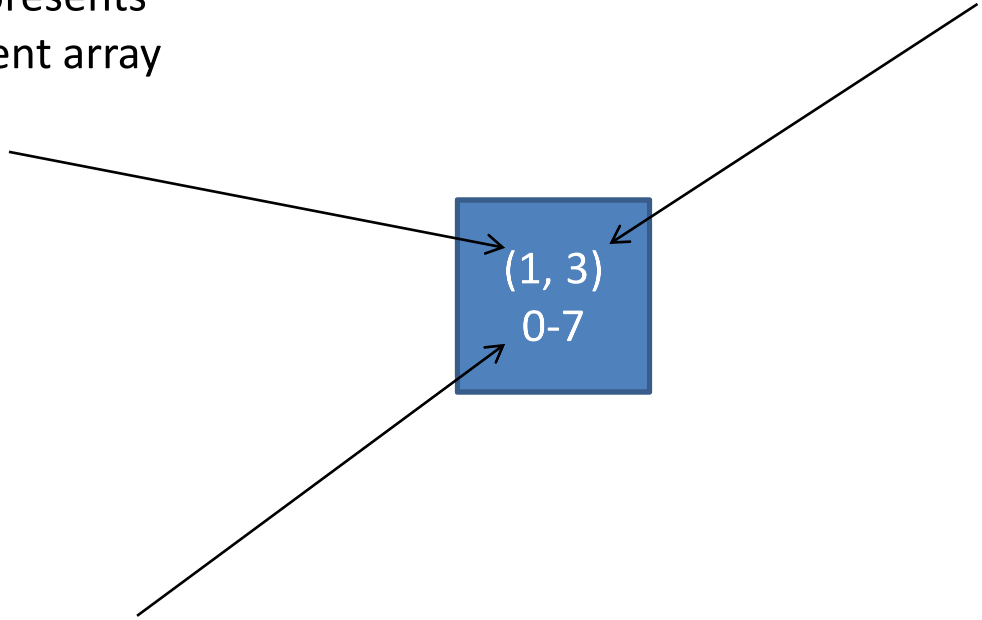
Segment Tree

- Assume N = 8 → values [0-7]
- Then we will need 3 levels (2^3). Total nodes 15 (-1+2^4). We build an array for these nodes.
- For an easier code, we start indexing in the segement tree array from 1.
- Assume initial input is array: 2 5 6, the leaves of tree

Let’s represent tree node

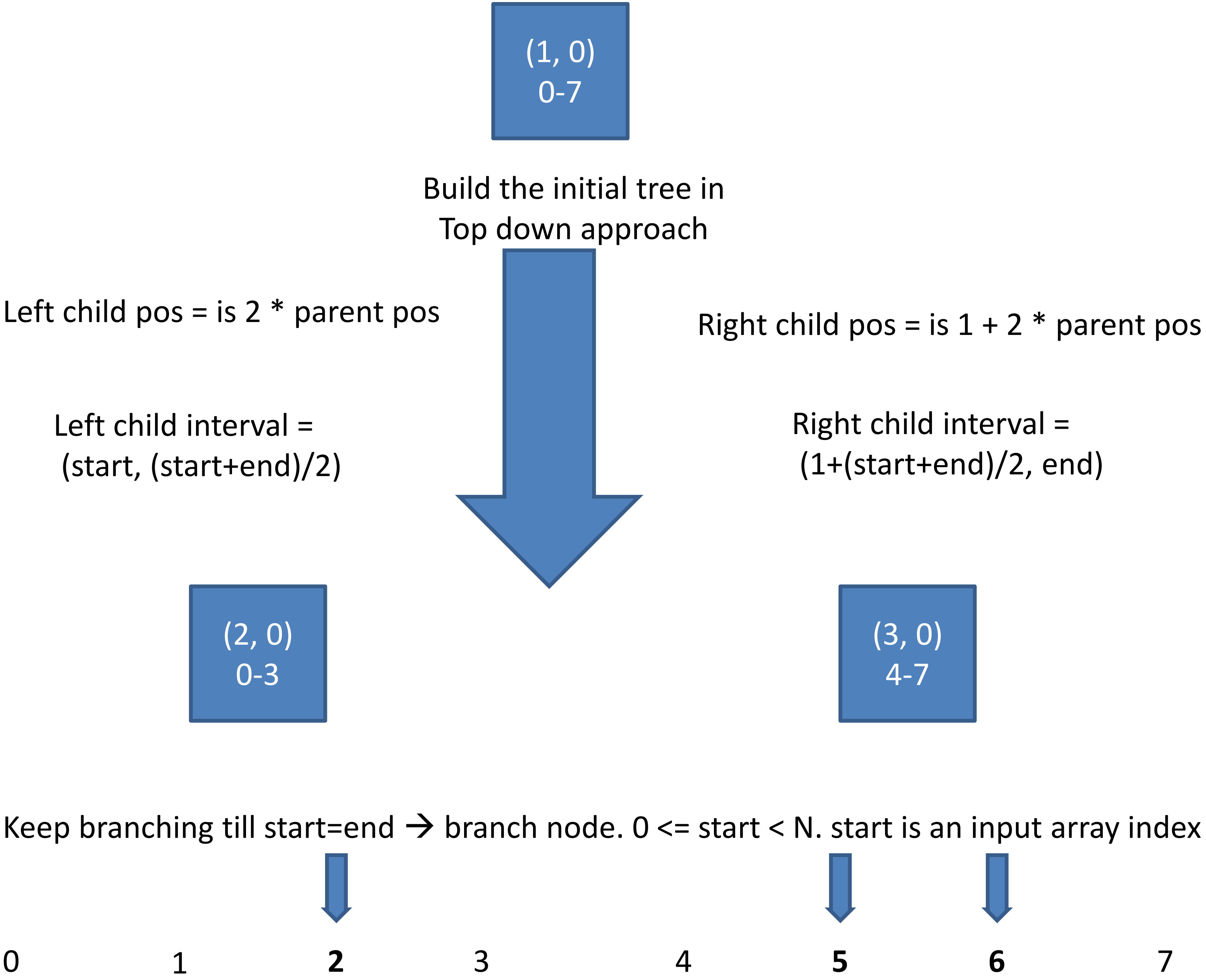
Second is the # of numbers
in this interval = 3

First number represents
Position in segment array
Root start from 1



(1, 3)
0-7

Each node has an interval that it covers.
0-7 is this node interval
Typically root node is **whole range**
and leaf node is interval of **a specific index**



(1, 0)
0-7

Build the initial tree in
Top down approach

Left child pos = is 2 * parent pos

Right child pos = is 1 + 2 * parent pos

Left child interval =
(start, (start+end)/2)

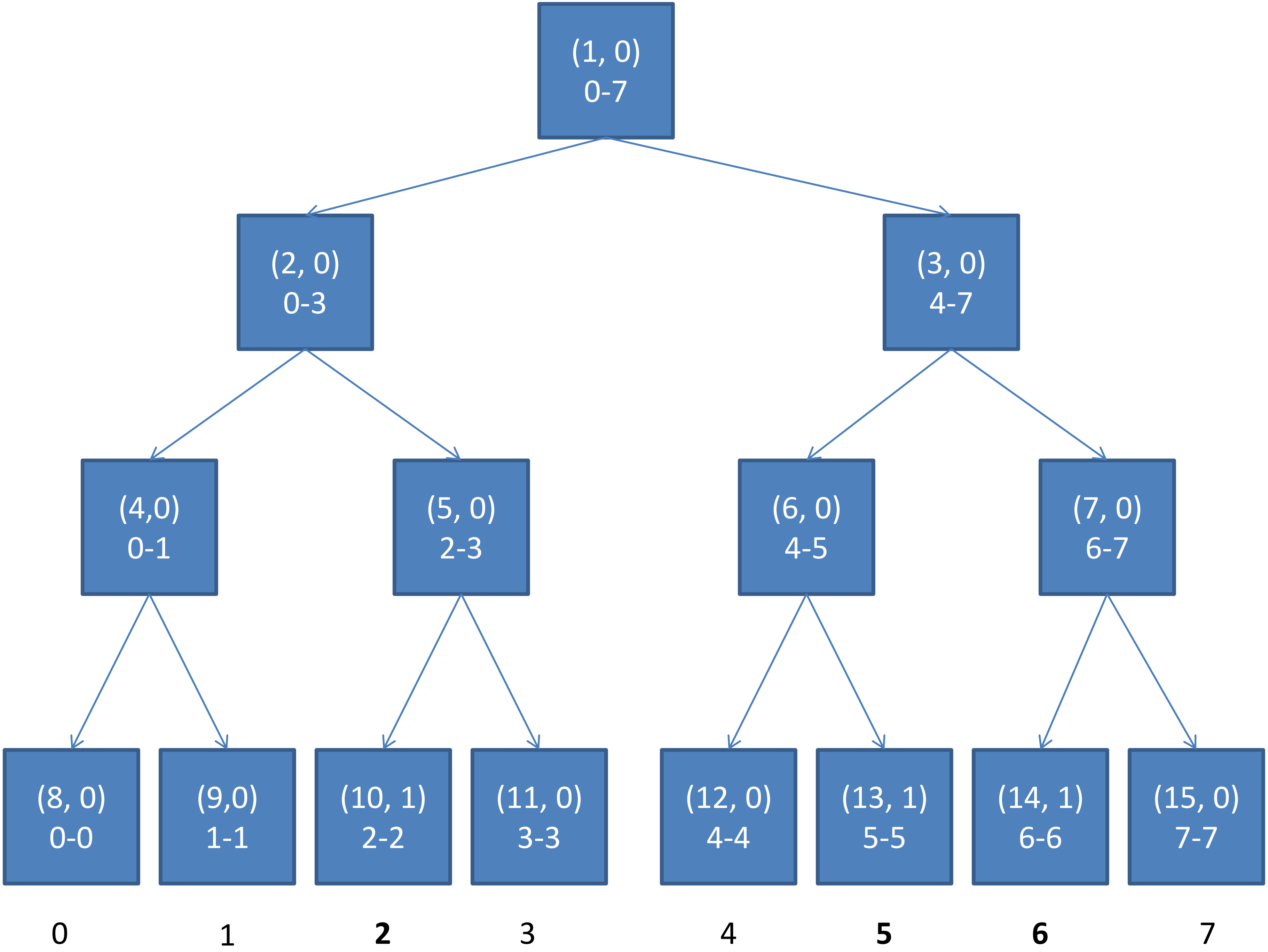
Right child interval =
(1+(start+end)/2, end)

(2, 0)
0-3

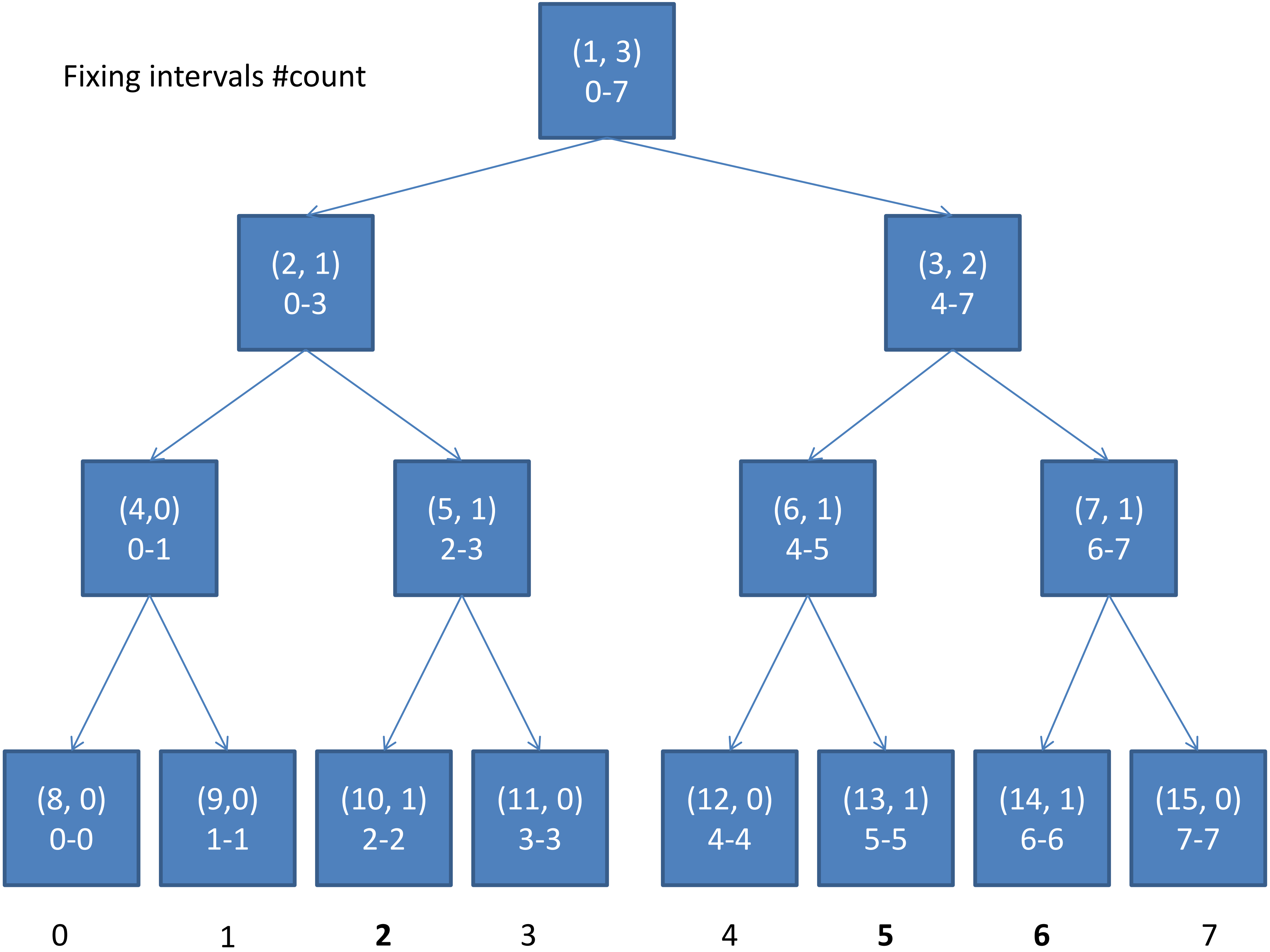
(3, 0)
4-7

Keep branching till start=end → branch node. 0 <= start < N. start is an input array index

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


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

Fixing intervals #count



(1, 3)
0-7

(2, 1)
0-3

(3, 2)
4-7

(4,0)
0-1

(5, 1)
2-3

(6, 1)
4-5

(7, 1)
6-7

(8, 0)
0-0

(9,0)
1-1

(10, 1)
2-2

(11, 0)
3-3

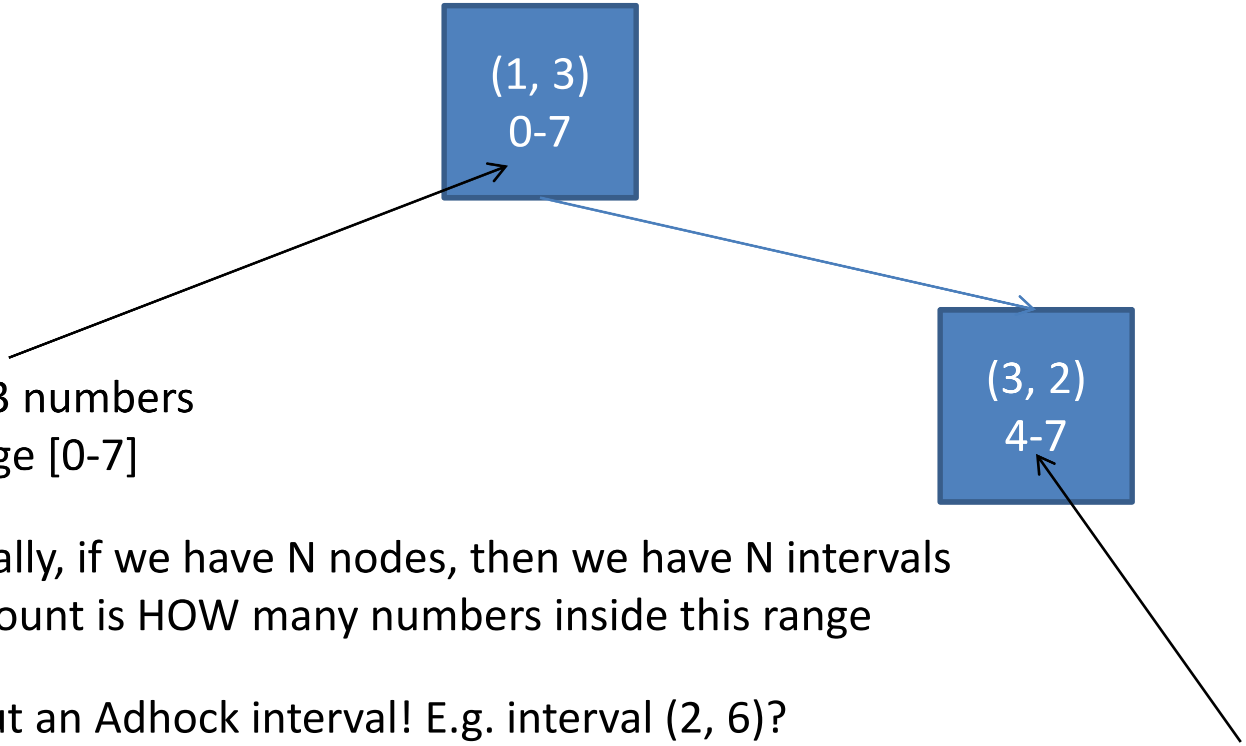
(12, 0)
4-4

(13, 1)
5-5

(14, 1)
6-6

(15, 0)
7-7

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



(1, 3)
0-7

(3, 2)
4-7

We have 3 numbers
in range [0-7]

Generally, if we have N nodes, then we have N intervals
Each count is HOW many numbers inside this range

What about an Adhock interval! E.g. interval (2, 6)?

We have 2 numbers
in range [4-7]

Any interval could be constructed from merge of others!

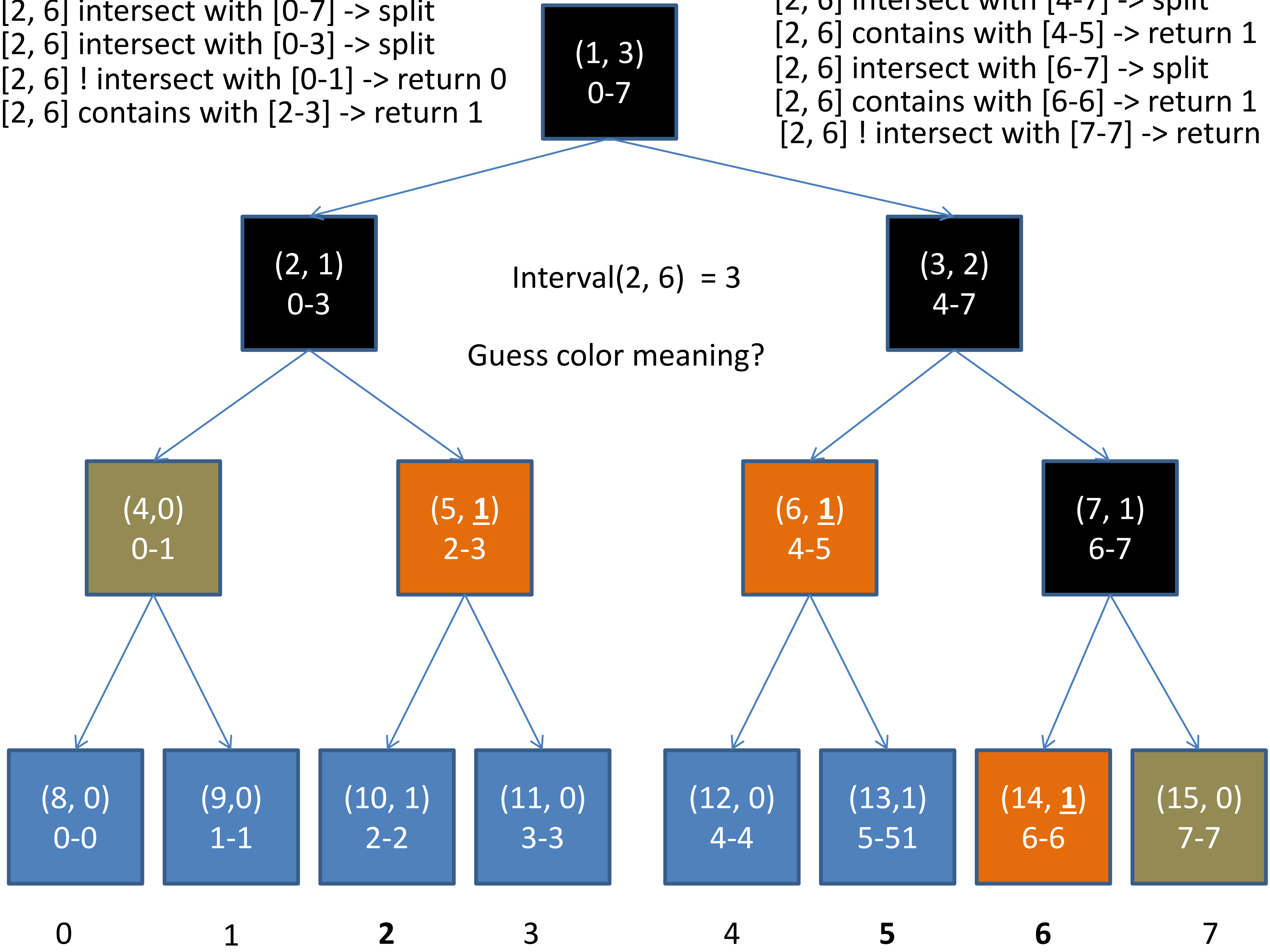
(2, 6) = (2, 3) + (4, 5) + (6, 6)

So smartly, move top down to sum these ranges
If no intersect, return 0
If interval is part of me, return its count.
If not, call left and call right

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

[2, 6] intersect with [0-7] -> split
[2, 6] intersect with [0-3] -> split
[2, 6] ! intersect with [0-1] -> return 0
[2, 6] contains with [2-3] -> return 1

[2, 6] intersect with [4-7] -> split
[2, 6] contains with [4-5] -> return 1
[2, 6] intersect with [6-7] -> split
[2, 6] contains with [6-6] -> return 1
[2, 6] ! intersect with [7-7] -> return 0

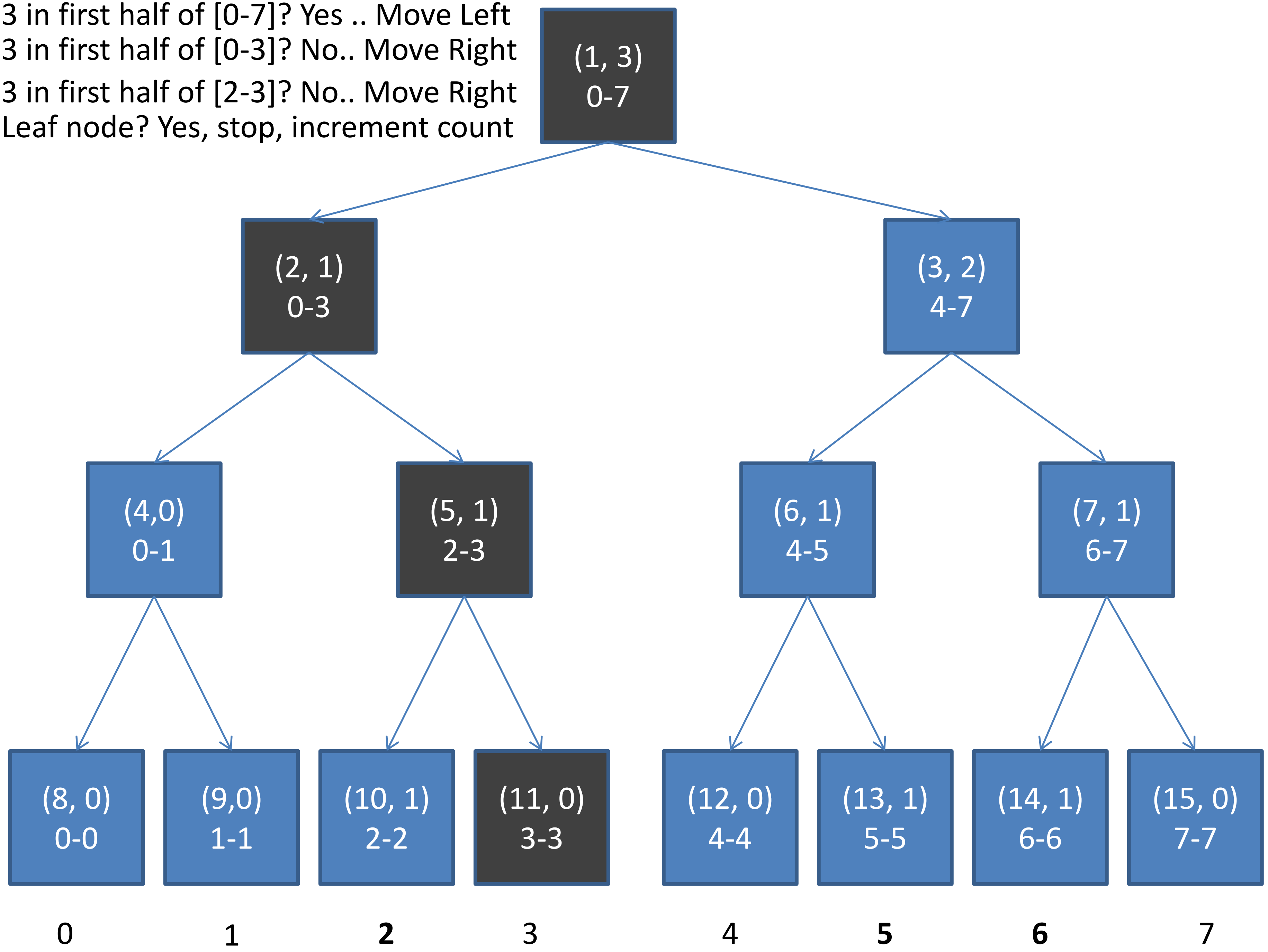


Interval(2, 6) = 3

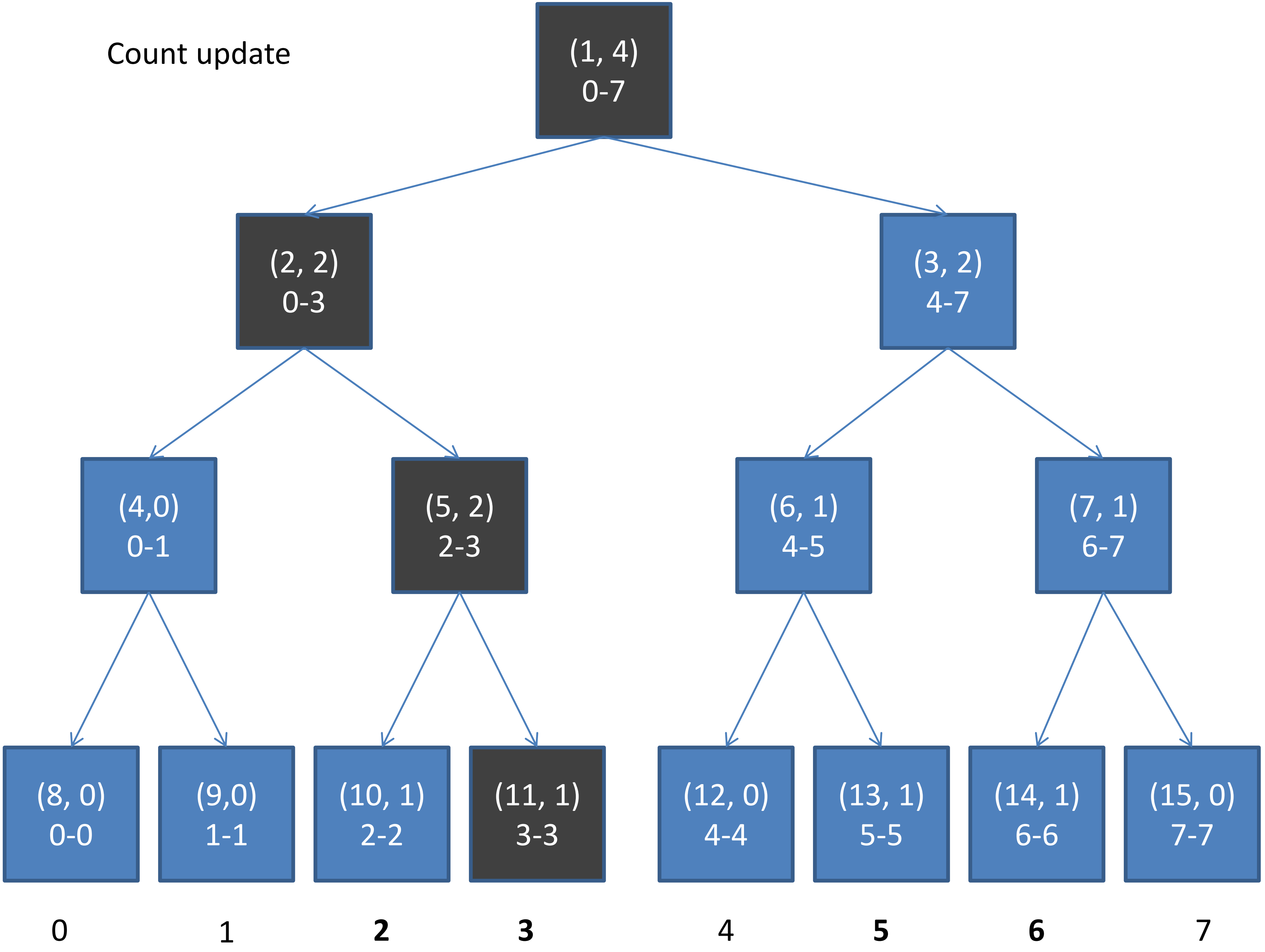
Guess color meaning?

- In last Interval **Query**, we have to **split** to 2 calls, but we stop once found covering interval → O(n)
- Note that, building the initial segment array needs accessing all nodes, which are nlogn, So Order O(nlogn).
- What about inserting new number? E.g. 3
- We just need to find the interval that contains 3, and update it. So O(logn) only

3 in first half of [0-7]? Yes .. Move Left
3 in first half of [0-3]? No.. Move Right
3 in first half of [2-3]? No.. Move Right
Leaf node? Yes, stop, increment count



Count update


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