Competitive Programming

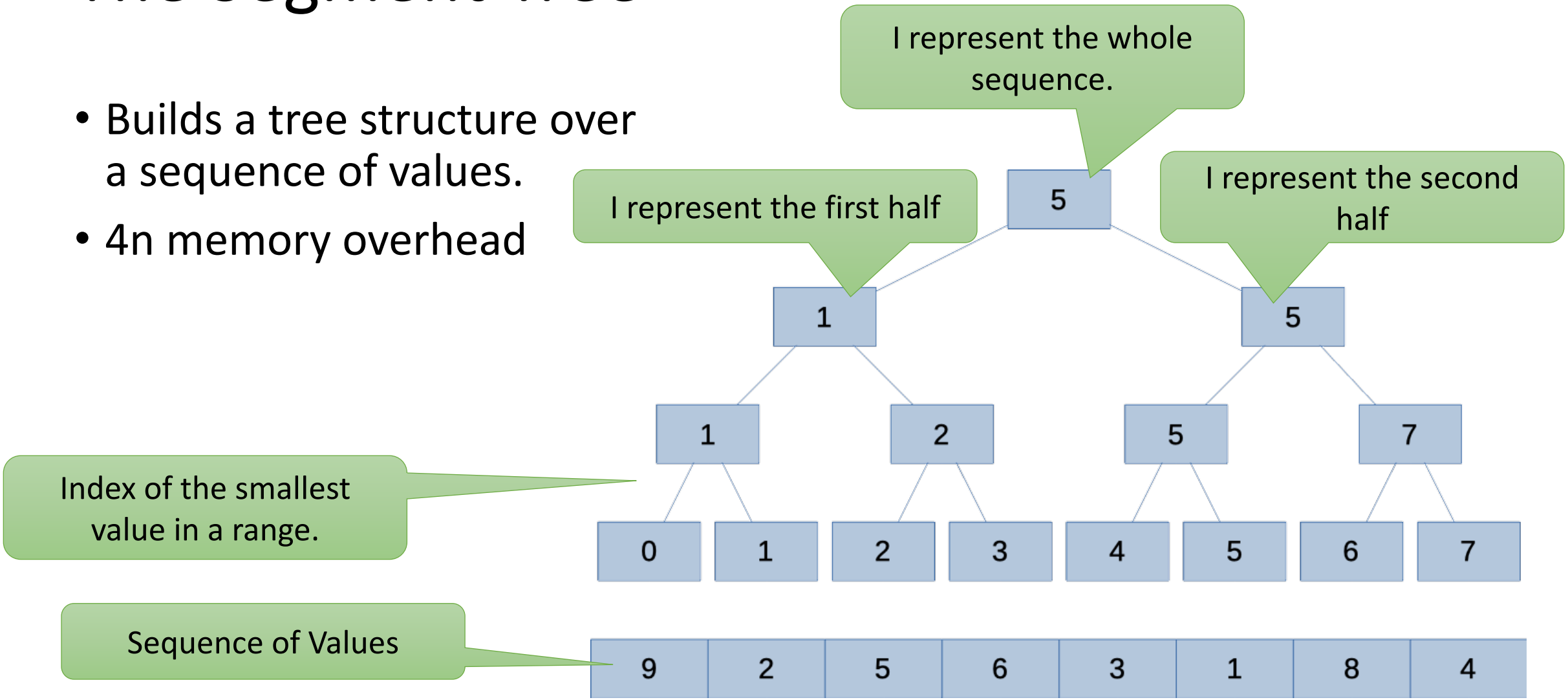
Segment Tree Data Structure

The Segment Tree

- It’s for Range Minimum Queries (RMQ)
  - Or similar range queries
  - Maximum over a range
  - Sum over a range
- Operations
  - Linear time to build the structure
  - Log time to respond to a range query
  - Log time to modify an element.

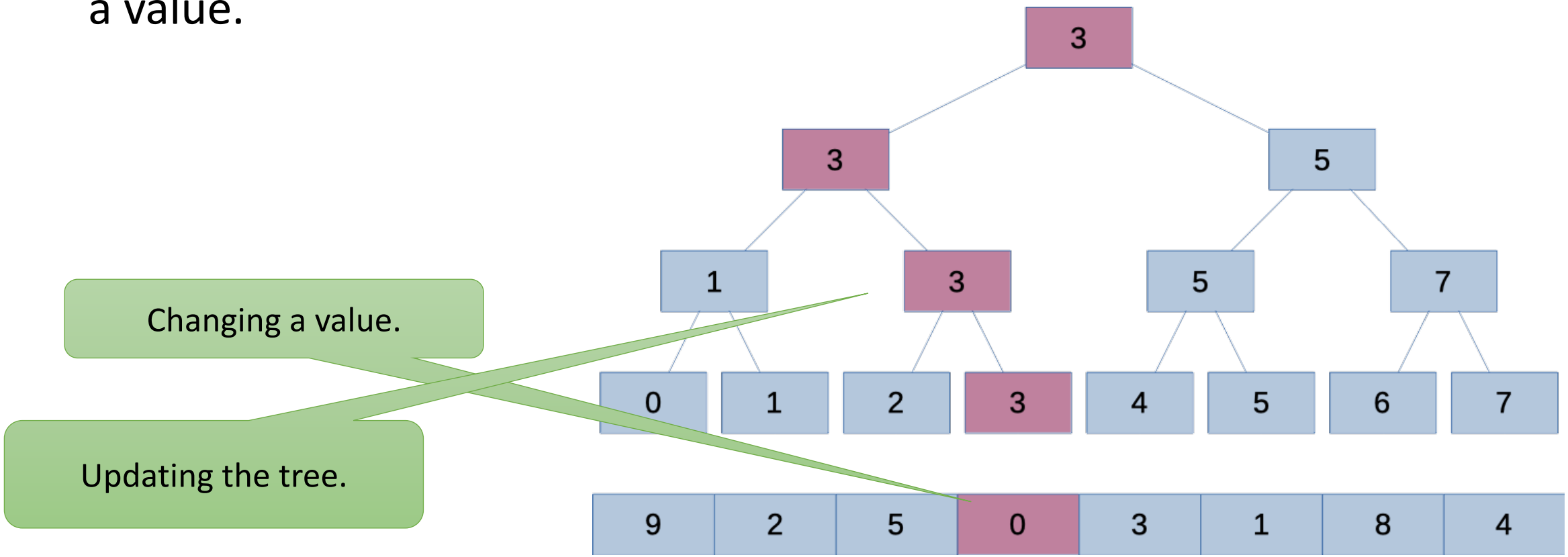
The Segment Tree

- Builds a tree structure over a sequence of values.
- 4n memory overhead



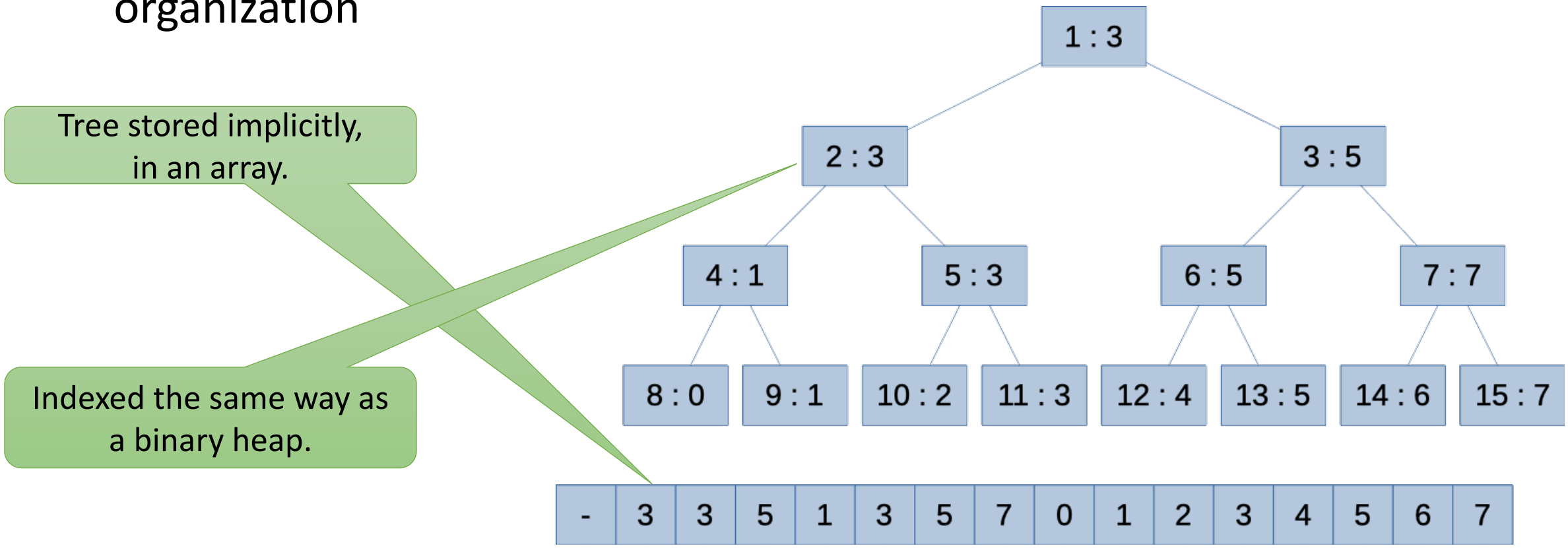
The Segment Tree

- Log-time overhead for changing a value.



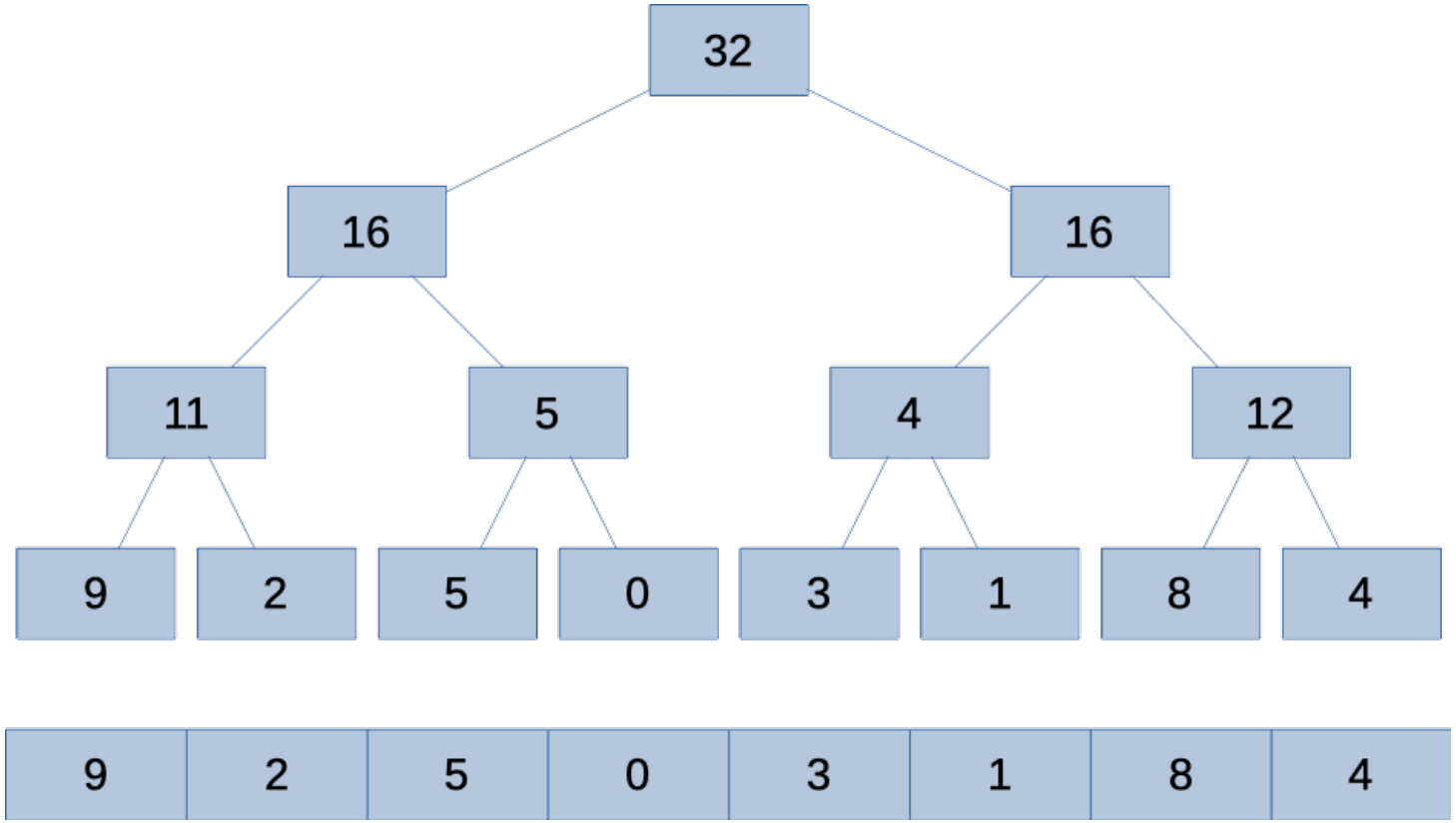
The Segment Tree

- Heap-style implicit tree organization



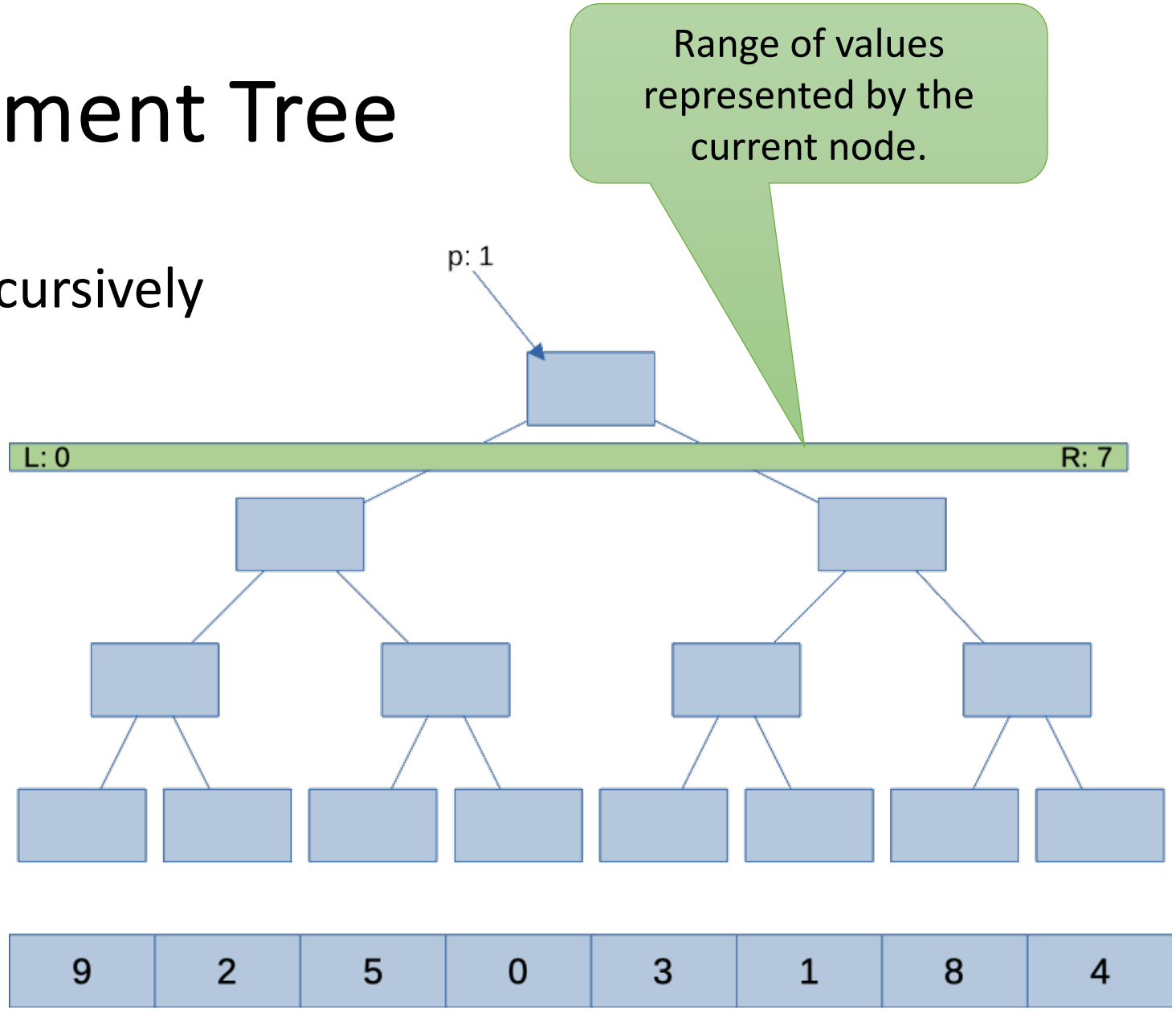
The Segment Tree

- Lots of related operations
  - E.g., range sum.



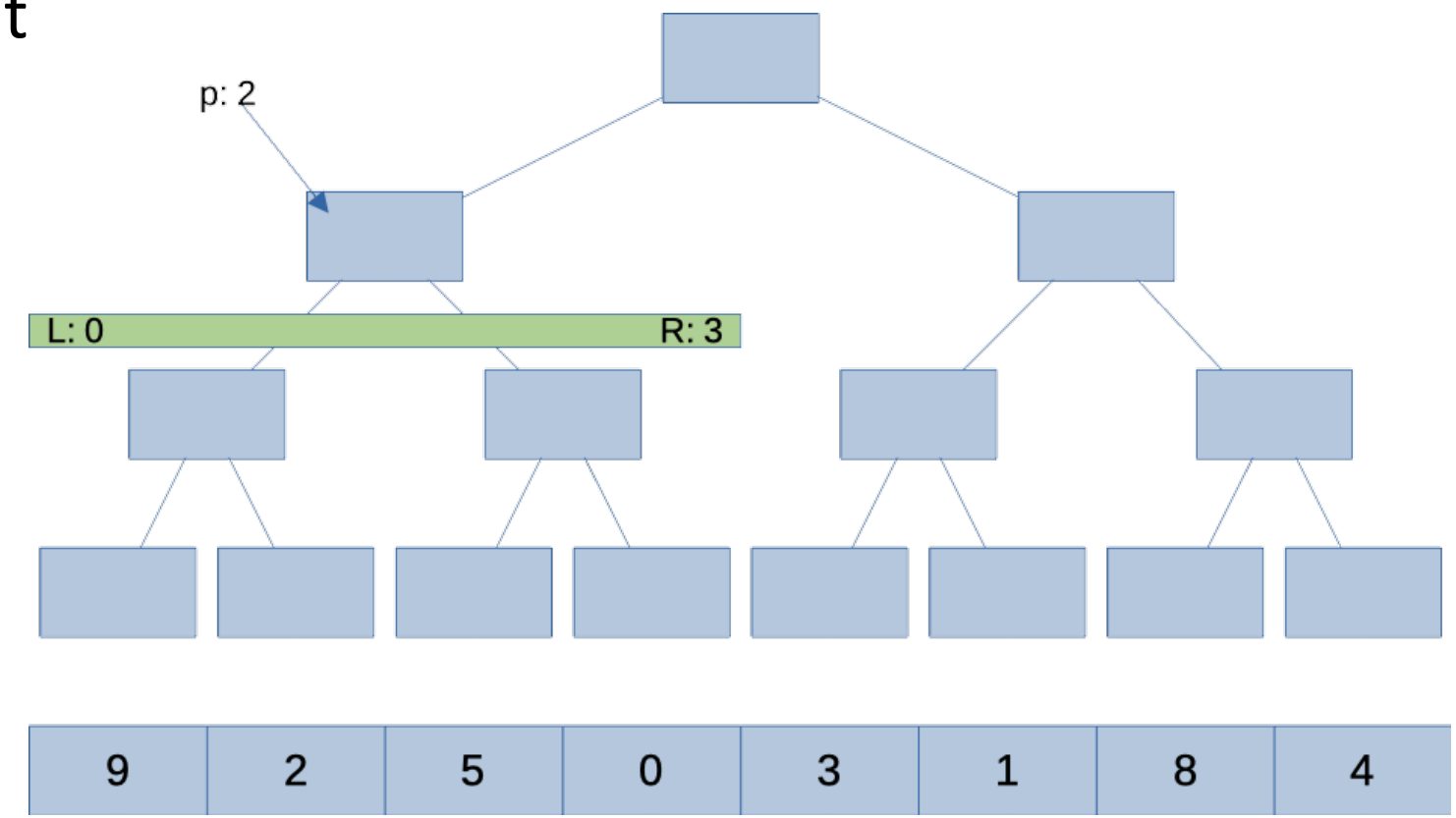
Building the Segment Tree

- We can build the tree recursively starting at the root.



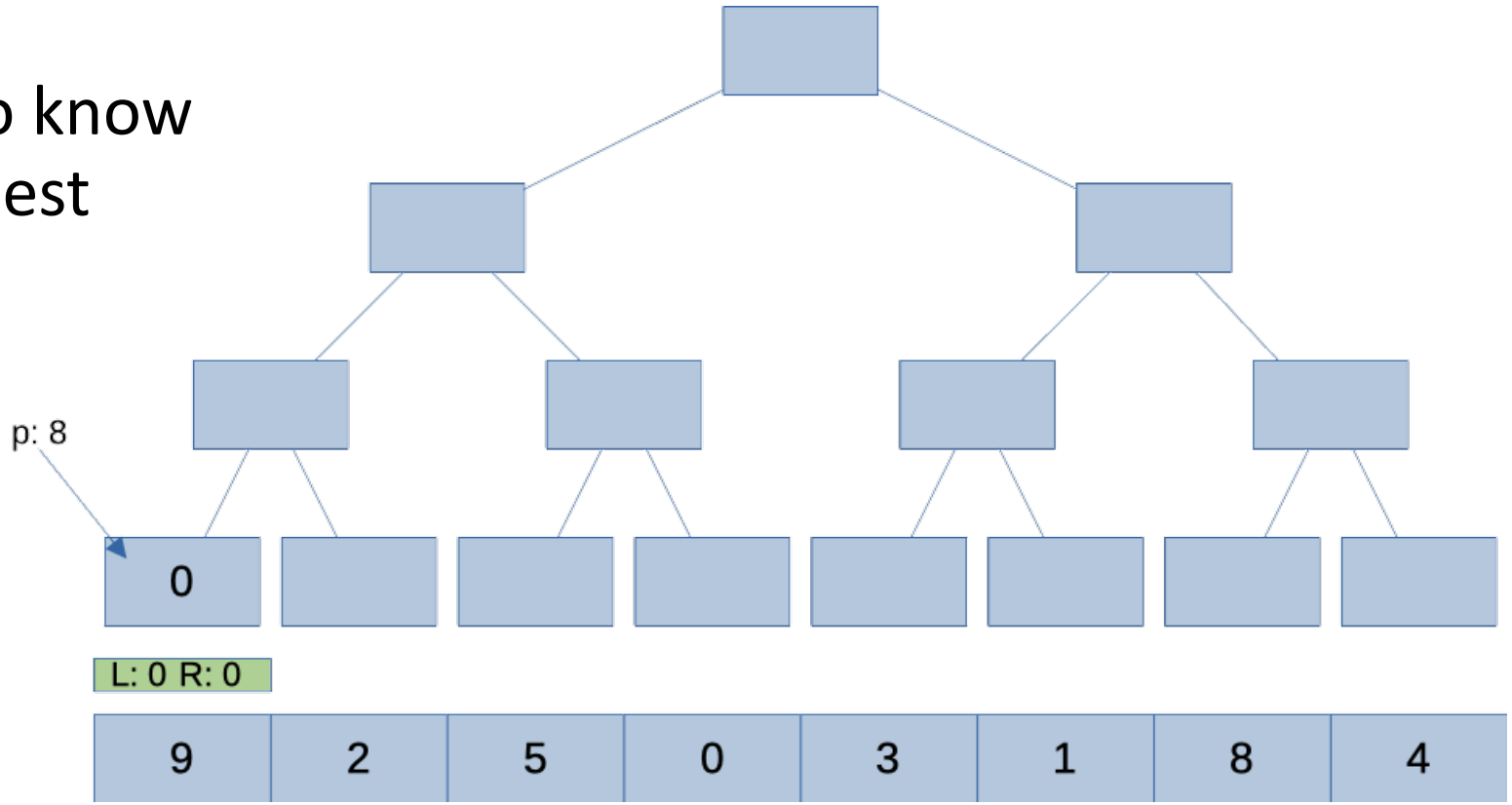
Building the Segment Tree

- As we descend the tree, keep up with the current range.



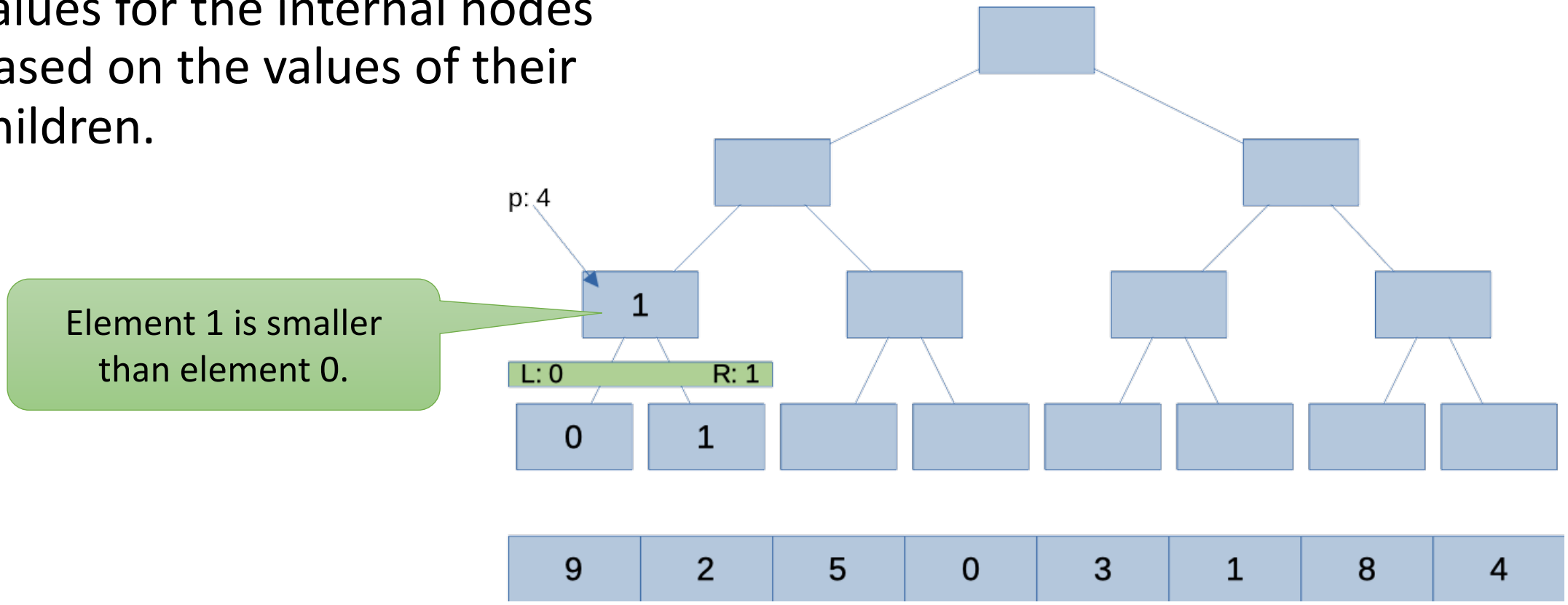
Building the Segment Tree

- A leaf covers a single sequence value.
- For a leaf, it's trivial to know the index of the smallest value.



Building the Segment Tree

- On the way back up, we can compute values for the internal nodes based on the values of their children.



Building the Segment Tree

```
void build( int p, int L, int R ) {
  if ( L == R )
    st[ p ] = L;
  else {
    build( left(p), L, (L + R) / 2 );
    build( right(p), (L + R) / 2 + 1, R );
    int p1 = st[ left(p) ], p2 = st[ right(p) ];

    st[ p ] = A[ p1 ] <= A[ p2 ] ? p1 : p2;
  }
}
```

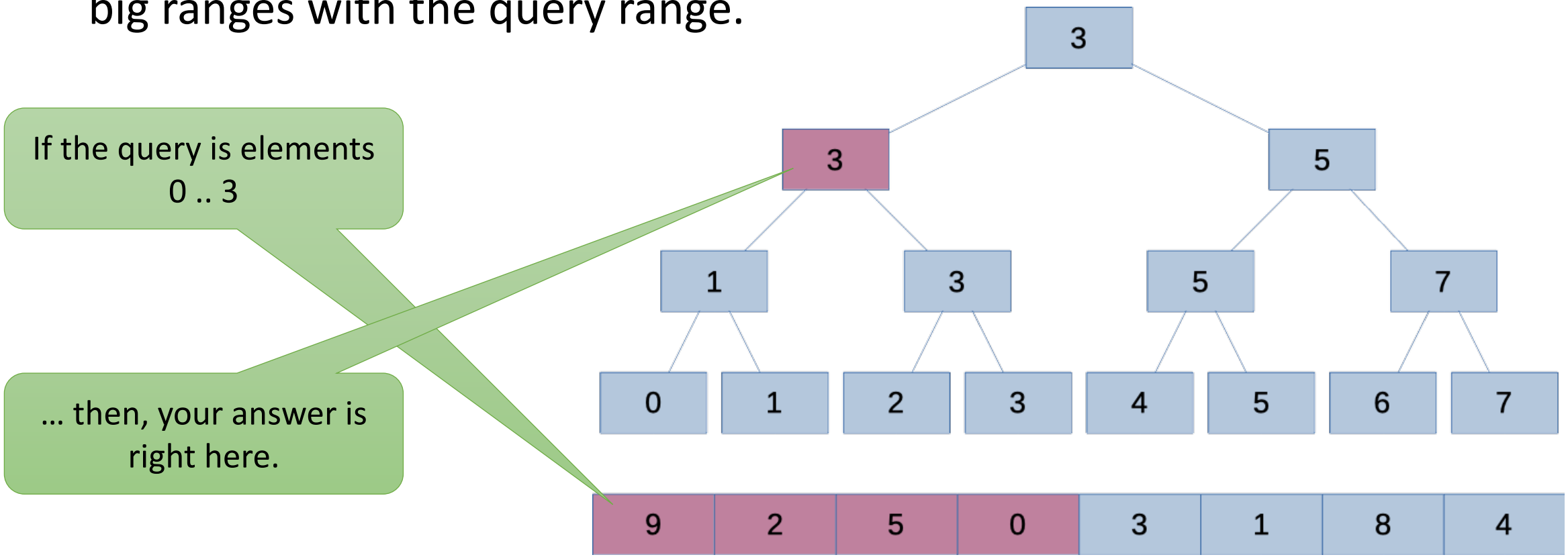
Am I at a leaf?

If not, recursively build the two subtrees.

Then, take the smaller value for the root of the subtree.

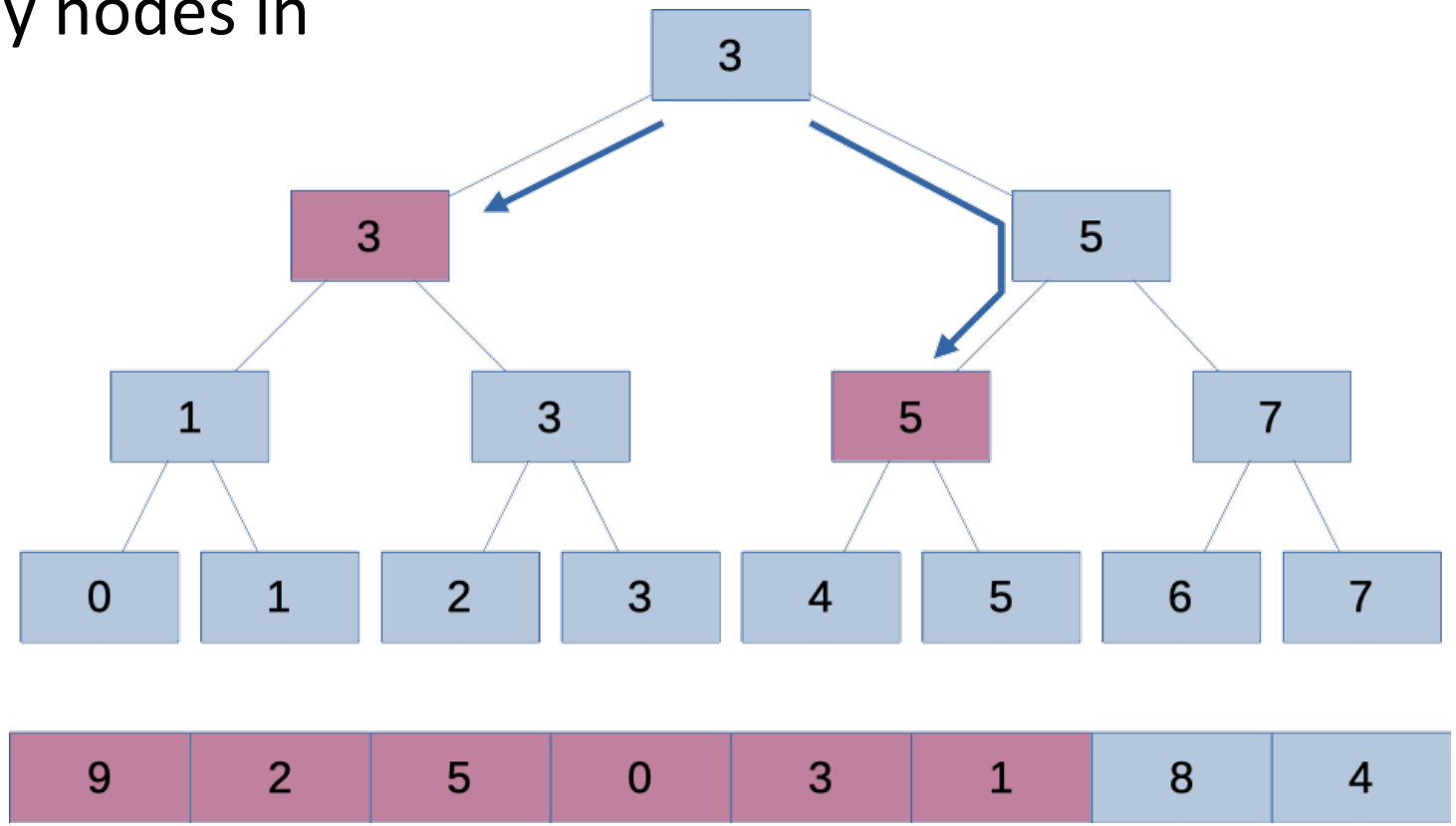
Answering a Query

- The internal nodes will let us handle big ranges with the query range.



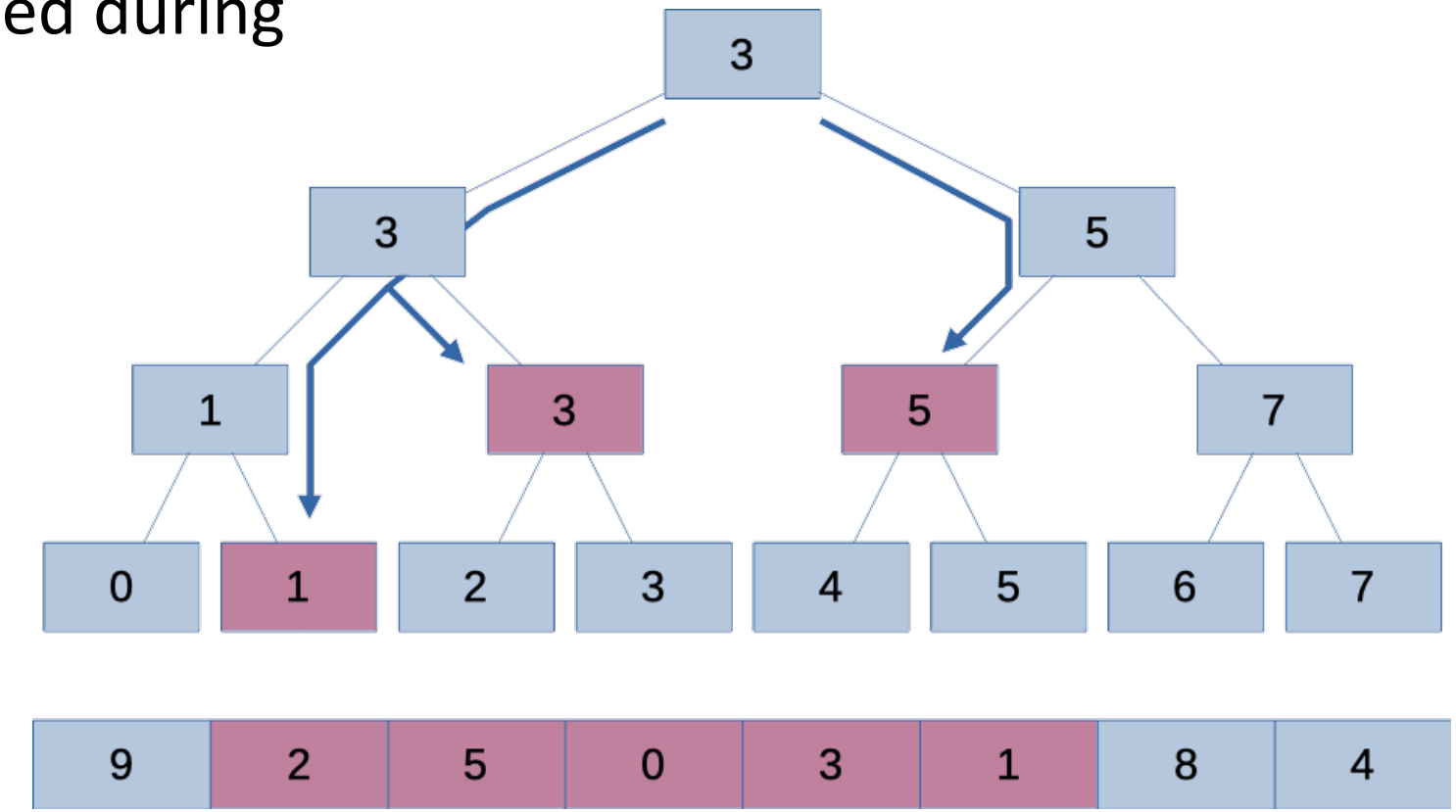
Answering a Query

- A typical query might require combining several ranges covered by nodes in the tree.



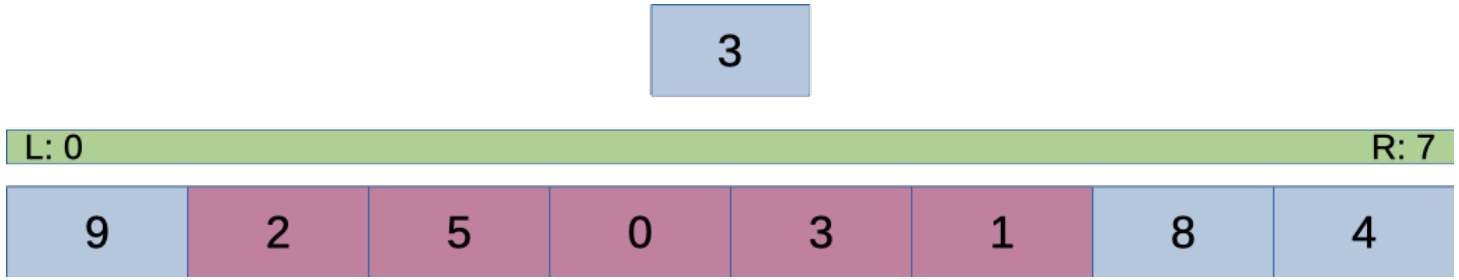
Answering a Query

- This sounds difficult, but we can pickup the nodes we need during a traversal.



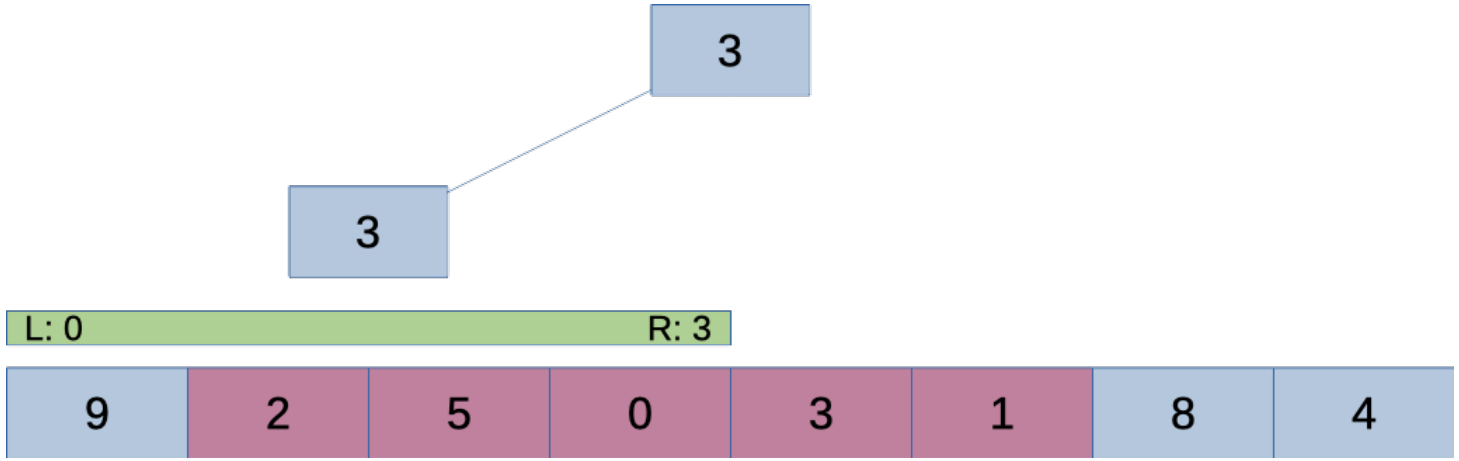
Answering a Query

- As we traverse, keep up with the range of values covered in each subtree.
- If the subtree range contains some of the query range …



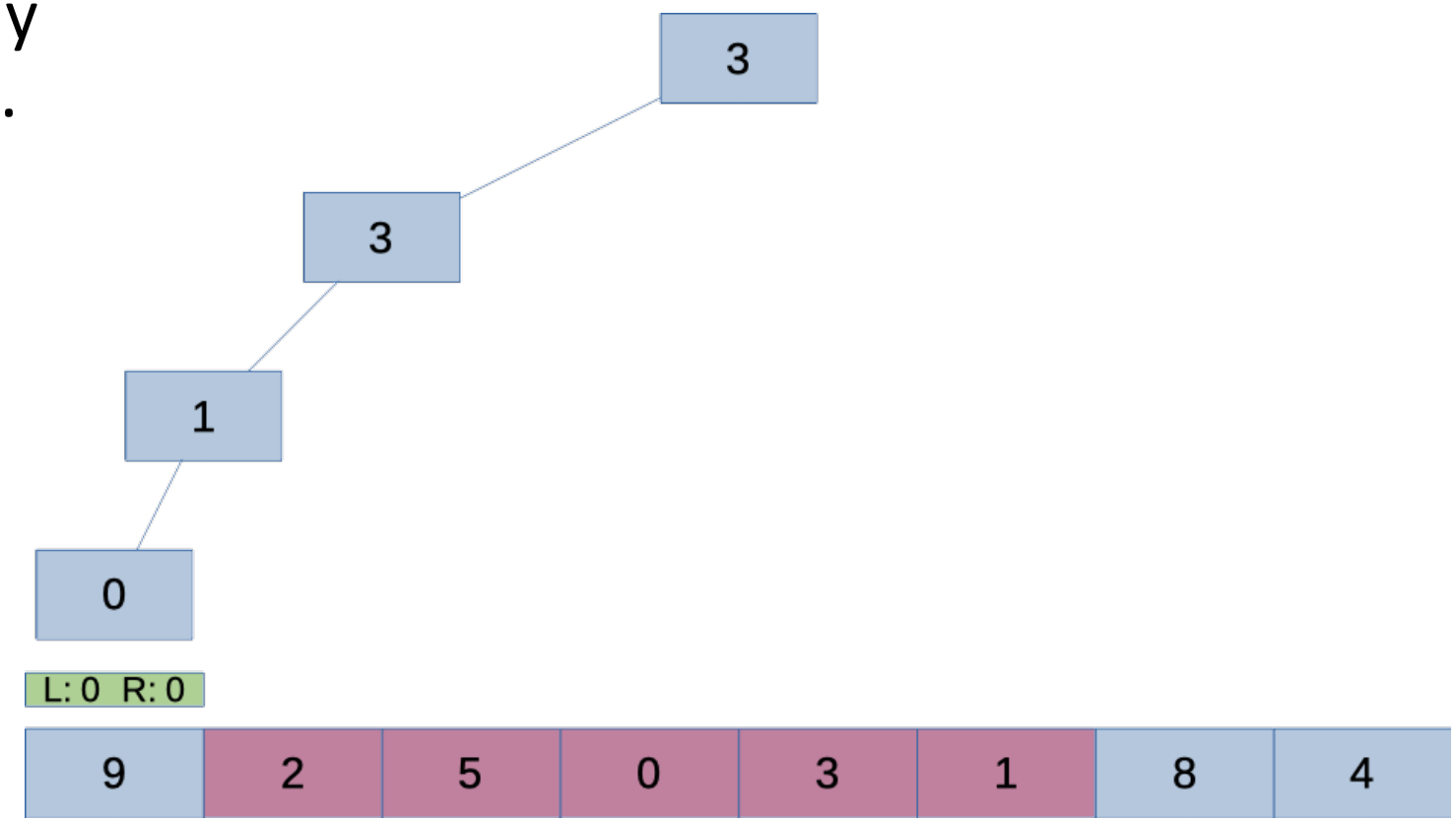
Answering a Query

- … traverse the subtrees.
- Again, the subtree range contains some of the query range.



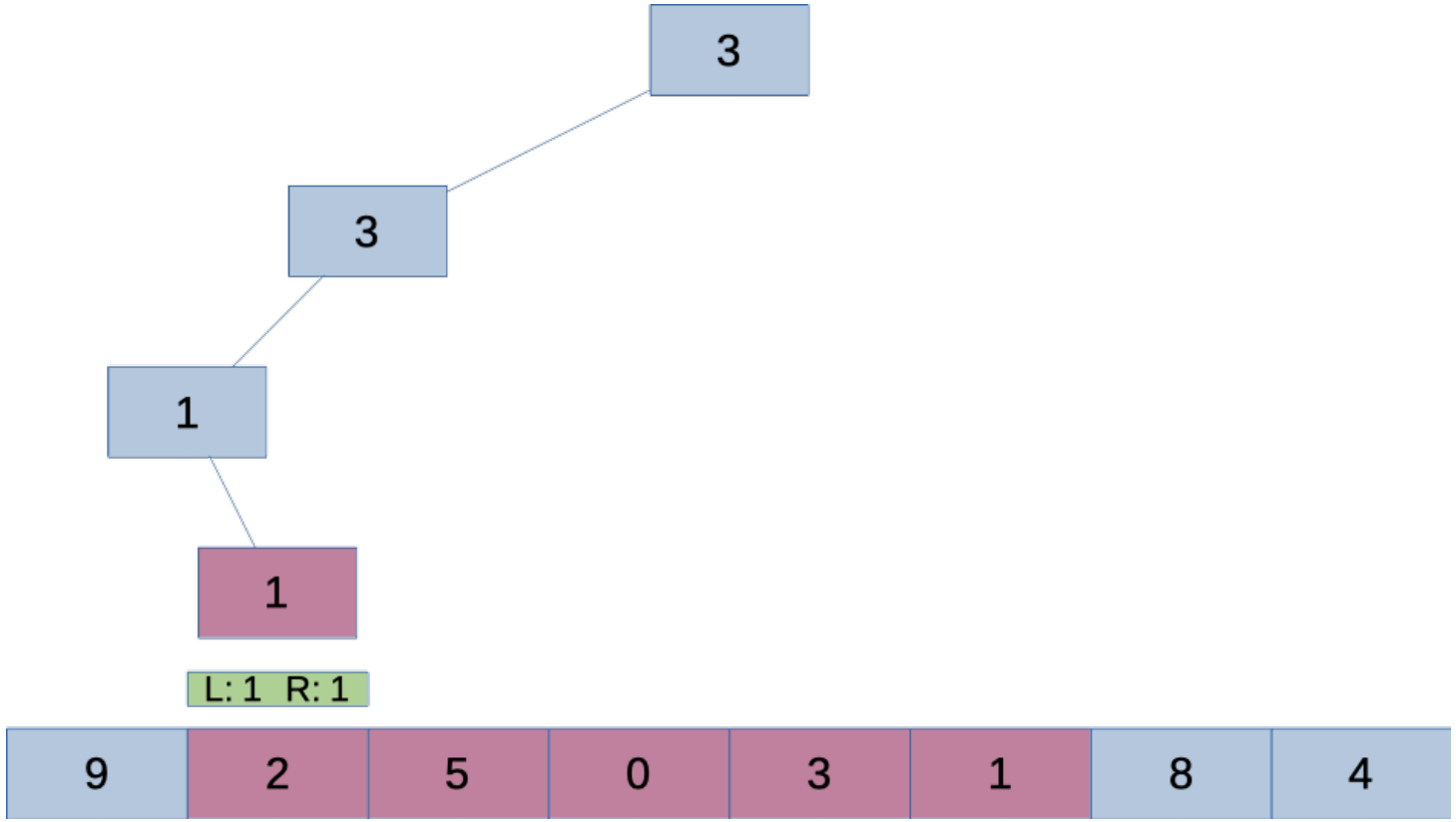
Answering a Query

- Eventually, you will reach subtrees that are entirely outside the query range.
  - You can skip these.



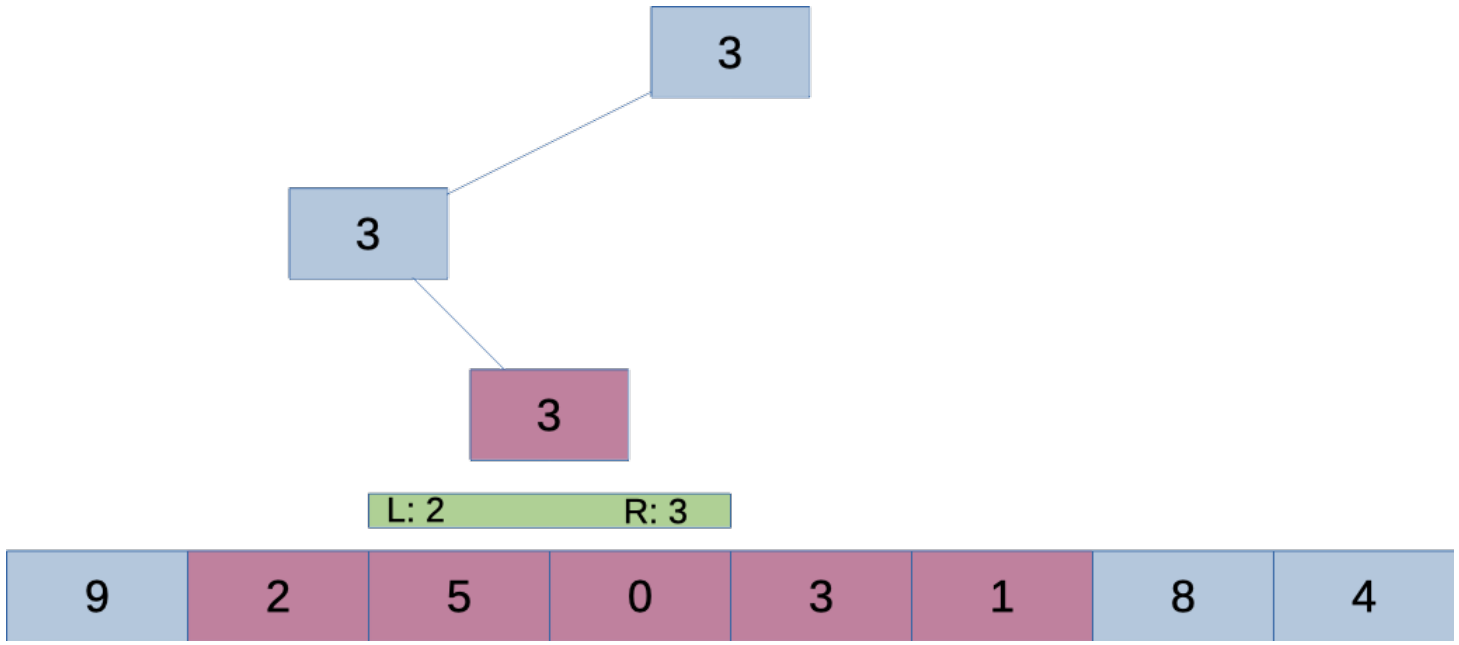
Answering a Query

- And subtrees entirely inside the query range.
  - You can handle these via their root.



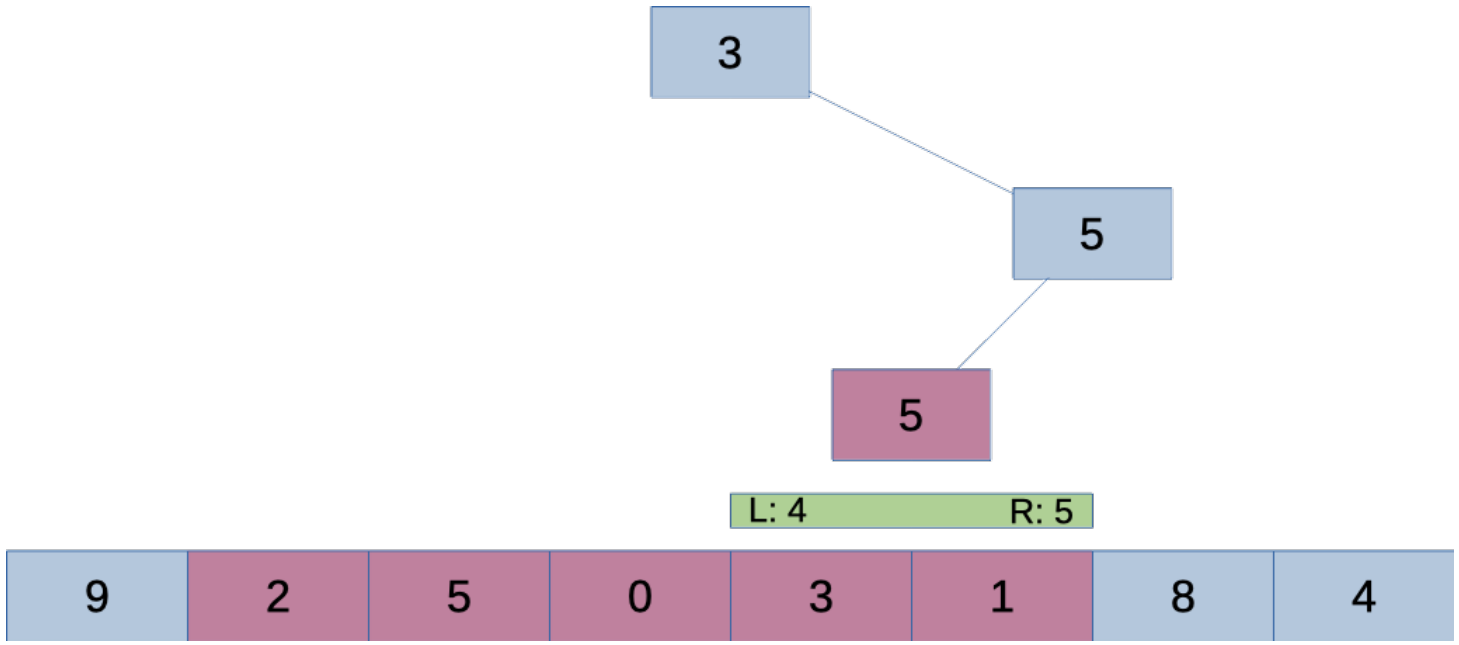
Answering a Query

- Collect results for all subtrees inside the query range …
- … as you return back up the tree.



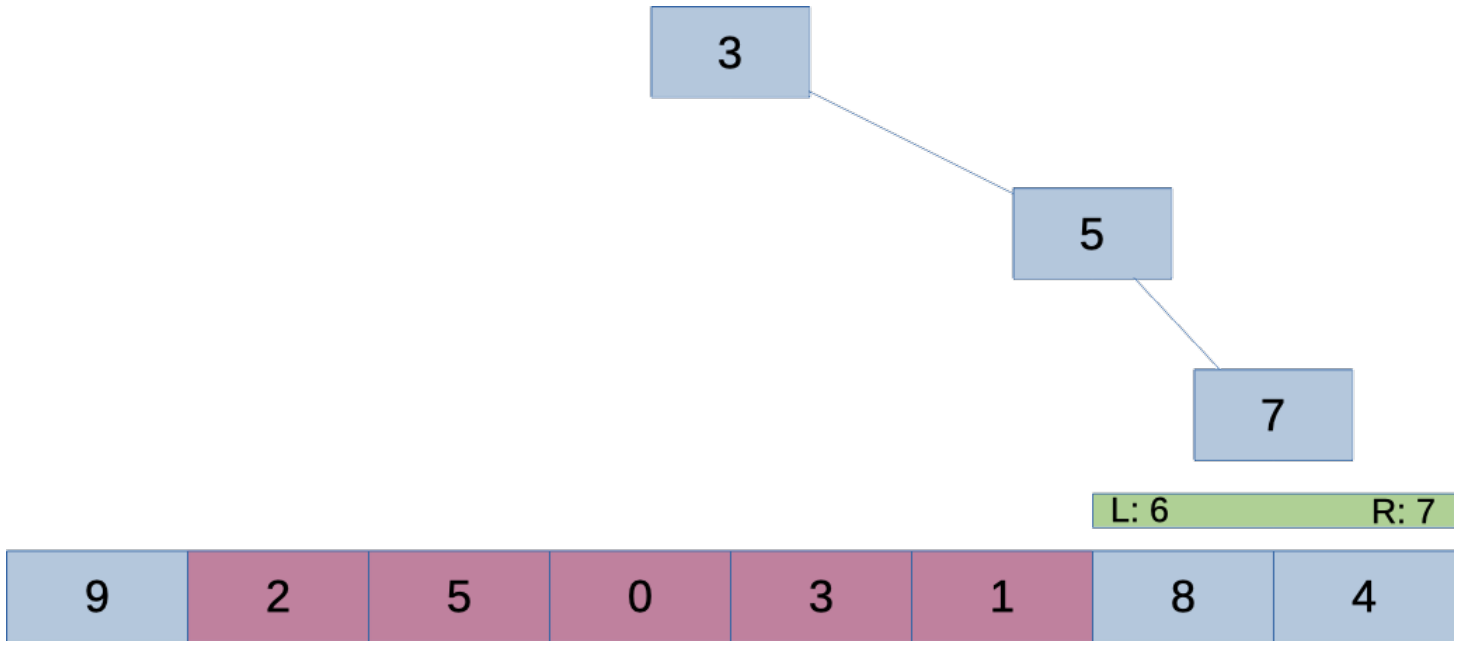
Answering a Query

- Another subtree inside the query range.



Answering a Query

- Another subtree outside the query range.



Range Query Implementation

```
int rmq( int p, int L, int R, int i, int j ) {
  if ( i > R || j < L ) return -1;
  if ( L >= i && R <= j ) return st[ p ];

  int p1 = rmq( left( p ), L, (L+R) / 2, i, j );
  int p2 = rmq( right( p ), (L+R) / 2 + 1, R, i, j );

  if ( p1 < 0 )
    return p2;
  if ( p2 < 0 )
    return p1;

  return A[ p1 ] <= A[ p2 ] ? p1 : p2;
}
```

Irregular Trees

- The sequence may not be a power of 2 in length.
- That's OK, it may waste some tree nodes.
- But, as long as we keep up with the range covered by each subtree …
- … we can handle an irregular tree structure.