Saving the entire subarrays in each vertex

In Normal Segment Tree we store information about the segment in compressed form (sum, minimum, maximum, ...). In current version of segment which we are going to discuss the root of the Segment Tree will store all elements of the array, the left child vertex will store the first half of the array, the right vertex the second half, and so on.

Question : Find the smallest number greater or equal to a specified number.
We want to answer queries of the following form: for three given numbers (l,r,x) we have to find the minimal number in the segment a[l…r] which is greater than or equal to x.

SOLUTION 1 : Naive Segment tree approach(sorted-list node)
In each vertex we store a sorted list of all numbers occurring in the corresponding segment.

Because of the structure of the Segment Tree and the similarities to the merge sort algorithm, the data structure is also often called "Merge Sort Tree".

Segment Tree Build Algorithm :
Merge the left-child-sorted-list and right-child-sorted-list to construct the parent node.

Segment Tree Build Algorithm Space Complexity : Since at each level we have 'n' elements and total numbers of level are logn(tree-height): So space complexity is : **O(nlogn)**

Segment Tree Build Algorithm Time Complexity : At each level we are merging effectively n, elements. Since time complexity of merge algorithm is 'n' and total numbers of level are logn(tree-height) : So time complexity is : **O(nlogn)**

Segment Tree Query Algorithm Time Complexity : Since we know for a particular range query, total number of terminating nodes are 2logn(because of #CASE1 or #CASE2). And we may have to do binary search(logn) on each terminating nodes. So time complexity : 2logn*logn = **O(logn^2)**

SOLUTION 2 : Optimized DataStructure Node Segment tree approach

In each vertex instead of sorted-list we may use some optimized data-structure which provides intrinsic sorting like TreeSet or TreeMultiSet(Guava). Becuase of intrinsic sorting modification queries on segment-tree would be easy to implement.

Now the time-complexity of range query would depend on search time taken by the optimized data structure.
Time Complexity of range query : logn*(search-time-complexiy-of-optimized-data-structure).
TreeSet or TreeMultiSet uses internally Red Black Tree , So time comlexity of range query = logn^2

SOLUTION 3 : Sorted List with Fractional Cascading approach

--Like merge-sort-tree Node of the tree contains Sorted List
--Additionally, each element contains the bridge tuple{l,r}.
--l represents position of the element in left child
--r represents position of the element in right child

Time Complexity : O(logn)

-- Single Binary Search will happen on Sorted List of Root Node. (logn)
-- Bridge traversal uisng position tuple{l,r}. Time taken at each node is Constant, but whole traversal time is logn.

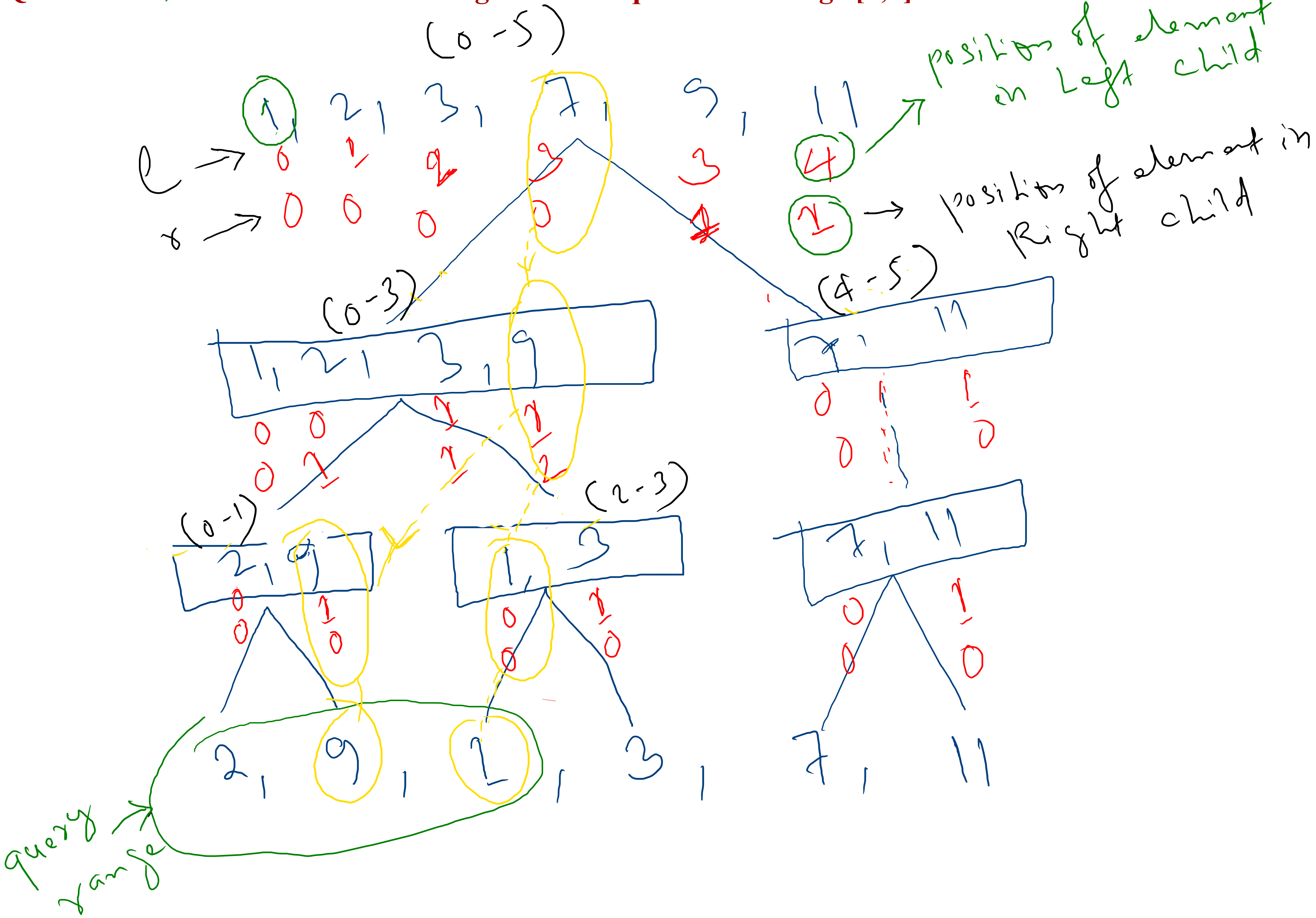
Note :

It is straightforward to apply this technique to a problem, that doesn't require any modification queries. The two positions are just integers and can easily be computed by counting when merging the two sorted sequences.

It it still possible to also allow modification queries, but that complicates the entire code. Instead of integers, you need to store the sorted array as **multiset**, and instead of indices you need to store **iterators**. And you need to work very carefully, so that you increment or decrement the correct iterators during a modification query.

Sorted List with Fractional Cascading approach : Working with example

Question : Find the smallest number greater or equal to 6 in range [0,2]


(0 - 5)
1, 2, 3, 7, 9, 11
l →
r →
position of element in Left child
position of element in Right child
(0-3)
1, 2, 3, 9
(4-5)
7, 11
(0-1)
2, 9
(2-3)
1, 3
7, 11
2, 9, 1, 3, 7, 11
query range