

COT 5405 : ANALYSIS OF ALGORITHMS
Midterm Solution

Q3: Second largest:

Design and analyze a comparison-based algorithm to find the second largest of a set of n numbers using at most $n + \lceil \log n \rceil - 2$ element-to-element comparisons.

Solution:

1. Construct a tournament with the elements of the array. At each step the larger of the two elements is advanced to the next level. We will end up with the largest element on top of the tournament tree.

2. It stands to reason that the second largest element would have beaten all but the largest element during the tournament. So the second largest element has to lose to the largest element. But we do not know where - it could have happened at the very first round or at the championship match or at middle somewhere.

3. We scan the path from the root to the leaf, where the match (comparison) involves the largest number. During this scan, we must come across all the elements which have lost to the largest element. The second largest element must be one of these.

The number of comparisons used in the method is $n + \lceil \log n \rceil - 2$: $n-1$ comparisons during the construction of the tournament tree since each match (or comparison) one player (item) is getting eliminated, and because the tree has n elements, the height of the tree is $\lceil \log n \rceil$, and thus it takes $\lceil \log n \rceil - 1$ comparisons to find the second largest element on the path from the root to the leaf.

Thus, a total of $(n - 1) + (\lceil \log n \rceil - 1) = n + \lceil \log n \rceil - 2$ comparisons are needed.

5 points for finding first largest number with $n-1$ comparison

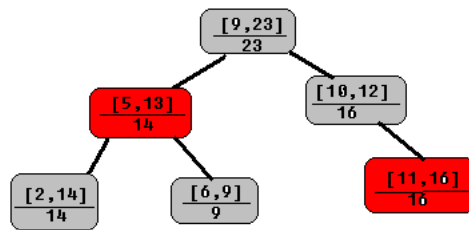
5 points for finding the second largest number with $\lceil \log n \rceil - 1$ comparison

Q4: Interval Trees

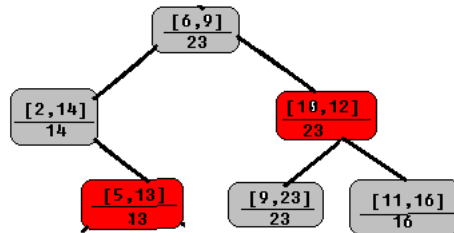
Let $A = \{[6,9],[2,14],[10,12],[5,13],[9,23],[11,16]\}$ be a set of intervals

- Construct an interval tree of A . Write down and verify the properties of a red-black tree on your construction.
- Prove that the tree is not unique
- Describe an efficient search algorithm, that given a query interval q , returns an interval from an interval tree T overlapping q that has the minimum low endpoint or nil if no such interval exists

Solution:



a)



b)

As the interval tree are built according to the low end point, i.e. any node's left low endpoint is greater than its left sub-tree's and smaller than its right sub-tree's, **so if we have found a node overlapping i using INTERVAL-SEARCH(T, i), only in its left sub-tree can we find an interval overlapping i and having a low endpoint smaller than it if there exists some node like this** . So we can do like this: first, we use INTERVAL-SEARCH(T, i) to find the first overlapped interval, and continue to search in the left sub-tree of the node(call INTERVAL-SEARCH) until it can't find any overlapping interval. In this way we can get the node who overlaps i and has the minimum low endpoint.

INTERVAL-SEARCH-SUBTREE(x, i)

```

1 while  $x \neq nil[T]$  and  $i$  does not overlap  $int[x]$ 
2   do if  $left[x] \neq nil[T]$  and  $max[left[x]] \geq low[i]$ 
3     then  $x \leftarrow left[x]$ 
4     else  $x \leftarrow right[x]$ 
5 return  $x$ 

```

INTERVAL-SEARCH-MIN(T, i)

```

2  $y \leftarrow$  INTERVAL-SEARCH-SUBTREE( $root[T], i$ )           > first, find the first overlapping interval
3  $z \leftarrow y$ 
4 while  $y \neq nil[T]$  and  $left[y] \neq nil[T]$            > continue to search in its left sub-tree
5   do  $z \leftarrow y$                                    > save the node before call the funciton
6      $y \leftarrow$  INTERVAL-SEARCH-SUBTREE( $y, i$ )
> if while loop exits by reason of  $y$  having no left sub-tree, we return the node  $y$ .
> otherwise we return  $z$ , which indicates it failed to find interval overlapping  $i$  in  $z$ 's left subtree
7 if  $y \neq nil[T]$  and  $i$  overlap  $int[y]$ 
8   then return  $y$ 
c) 9 else return  $z$ 

```

Since each iteration of the basic loop takes $O(1)$ time, and since the height of an n -node red-black tree is $O(\lg n)$, the procedure takes $O(\lg n)$ time.