EE 602 – Algorithm I **(UH Manoa, Fall 2020)**

Algorithms Assignment 10

November 14, 2020

Instructor: Yingfei Dong *Name: Frendy Lio Can*

Problem 1

The difference between a binary-search-tree (BST) and the min-heap property is that the BST property maintains an invariant where all nodes in the subtree are smaller, and all nodes in the right subtree are larger. The min-heap property only guarantees that the parent node is always smaller than the child node; we don't know if the left children or right children is bigger.

The min-heap property cannot be used to print out the keys of an n-node tree in sorted order in $O(n)$ time because we don't know if the left tree or right tree contains the smallest element.

Problem 2

Let x be a node that has two children, p be the predecessor of x, and, s be the successor of x.

Lets show by contradiction that s has no left child. Suppose s has a left child. Then the values of s is greater than the value of s->left. This imply that the value of s->left is bigger than the value of x node. Therefore:

$$value[s] \geq value[s \rightarrow left] \geq value[x]$$

This is a contradiction as s should be the successor of x. Therefore, the successor of x cannot have a left child.

Lets show by contradiction that p has no right child. Suppose p has a right child. Then the value of p is less than that value of p->right. This imply that the value of p->right is less than the value of x node. Therefore:

$$value[p] \leq value[p \rightarrow right] \leq value[x]$$

This is a contradiction as p should be the predecessor of x. Therefore, the predecessor of p cannot have a right child.

Problem 3

Worst case (Input is already sorted): $\theta(n^2)$

Best case (When the tree formed is balanced): $O(nlgn)$

Problem 4

The resulting tree is a red-black tree since the properties 1,3,4, and 5 for red-black tree are satisfied.

Problem 5

If both children are black, then the degree is 2. If one of the children is red and the others is black, then the degree is 3. If both children are red, then the degree is 4. We can observe that the depth of the leaves of the resulting tree are the same.

Problem 6

For a red-black tree, we know that every simple path from node x to a descendant leaf, the same number of black nodes and red nodes do not happen next to each other.

Thus, we can conclude that the shortest path from node n to a descendant leaf will have all nodes of color black. We can also conclude that the longest path will be a path with alternating black and red nodes.

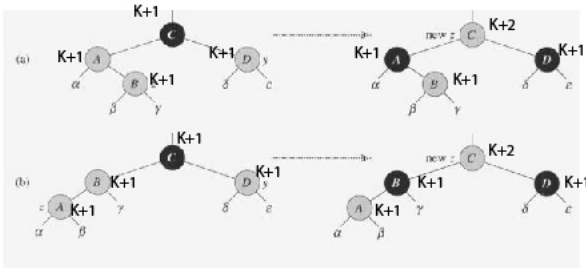
Since all leaf must be black, the longest path will have the same number of red and black nodes. It will have $2 *$ (number of black nodes), where the number of black nodes is equal to the shortest path.

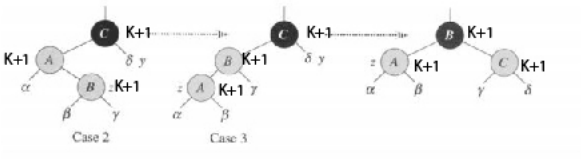
Problem 7

Lets consider a n-node binary tree where will be transform it into a right-going chain. Let the root and all its right children be elements of the chain. For any node x which is a left child of a node on the chain, a single right rotation on the parent of x will add that node to the chain and not remove any elements from the chain. Therefore, we can transform the arbitrary $n-$node binary search tree to a right-going chain with at most $n-1$ right rotations.

In order to convert the right-going chain into any other arbitrary $n-$ node binary search tree will take also $n-1$ rotations. Therefore, the time complexity is $O(2n-2) = O(n)$.

Problem 8





Problem 9

Using $OS - SELECT(x, i)$ and $OS - RANK(T, x)$ from the CLRS book page 341 and 342. The desired result is $OS - SELECT(T, OS - RANK(T, x) + i)$. This has runtime $O(h)$, where h is the height of the tree. We know by the properties of red black trees that $h = lgn \Rightarrow O(lgn)$

Problem 10

We can use the $INTERVAL - SEARCH(T, i)$ algorithm from the CLRS book (page 351), but, instead of breaking out of the loop as soon as we have an overlap, we just keep track of the overlap that has the minimum low endpoint, and continue the loop. After the loop terminates, we return the overlap stored.

Thus, it would look something like this:

```
INTERVAL-SEARCH-MODIFIED(T, i)
x = T.root
interval = T.nil
while x ≠ T.nil:
  \\ keep track of overlap
  if not (i.high ≤ x.left or x.right ≤ i.low)
    if interval == T.nil or interval.right > x.right
      interval = x
  if x.left ≠ T.nil and x.left.max ≥ i.low
    x = x.left
  else
    x = x.right
return interval
```