EE602: ALGORITHMS

HOMEWORK #5 Due: 11/16

Hualiang Li

(1) 12.1-2 What is the difference between the binary-search-tree property and the min-heap property (see page 153)? Can the min-heap property be used to print out the keys of an n-node tree in sorted order in O(n) time? Show how, or explain why not.

In a min-heap, a node's key is ≤ both of its children's keys. In a binary search tree, a node's key is ≥ its left child's key, but ≤ its right child's key.

The heap property, unlike the binary-search-tree property, doesn't help print the nodes in sorted order because it doesn't tell which sub-tree of a node contains the element to print before that node. In a heap, the largest element smaller than the node could be in either sub-tree.

(2) 12.2-5 Show that if a node in a binary search tree has two children, then its successor has no left child and its predecessor has no right child.

Let x be a node with two children. In an in-order tree walk, the nodes in its left sub-tree immediately precede x and the nodes in x's right sub-tree immediately follow x. Thus, x's predecessor is in its left sub-tree, and its successor is in its right sub-tree.

Let s be x's successor. Then s cannot have a left child, for a left child of s would come between x and s in the in-order walk. (It's after x because it's in x's right sub-tree, and it's before s because it's in its left sub-tree.) If any nodes were to come between x and s in an in-order walk, then s would not be x's successor, as we had supposed.

Symmetrically, x's predecessor has no right child.

(3) 12.3-3 We can sort a given set of n numbers by building a binary search tree containing these numbers (using TREE-INSERT repeatedly to insert the numbers one by one) and then printing the number by an inorder tree walk. What are the worst-case and best-case running times for this sorting algorithm?

We have to do n times insertion operation. For each insertion operation, the worst case is O(n), best case is O(lgn). Therefor,

Worst case: (n^2) occurs when a linear chain of nodes results from the repeated

TREE-INSERT operations.

Best case: (n lg n)óoccurs when a binary tree of height (lg n) results from the repeated TREE-INSERT operations.

(4) 13.1-3 Let us define a **relaxed red-black tree** as a binary search tree that satisfies red-black properties 1, 3, 4, and 5. In other words, the root may be either red or black. Consider a relaxed red-black tree T whose root is red. If we color the root of T black but make no other changes to T, is the resulting tree a red-black tree?

Yes, it is a red-black tree. The black height of each node does not change.

(5) 13.1-4 Suppose that we "absorb" every red node in a red-black tree into its black parent, so that the children of the red node become children of the black parent. (Ignore what happens to the keys.) What are the possible degrees of a black node after all its red children are absorbed? What can you say about the depths of the leaves of the resulting tree?

After absorbing each red node into its black parent, the degree of each node black node is

2, if both children were already black,

3, if one child was black and one was red,

or 4, if both children were red.

All leaves of the resulting tree have the same depth.

(6) 13.1-5 Show that the longest simple path from a node x in a red-black tree to a descendant leaf has length at most twice that of the shortest simple path from node x to a descendant leaf.

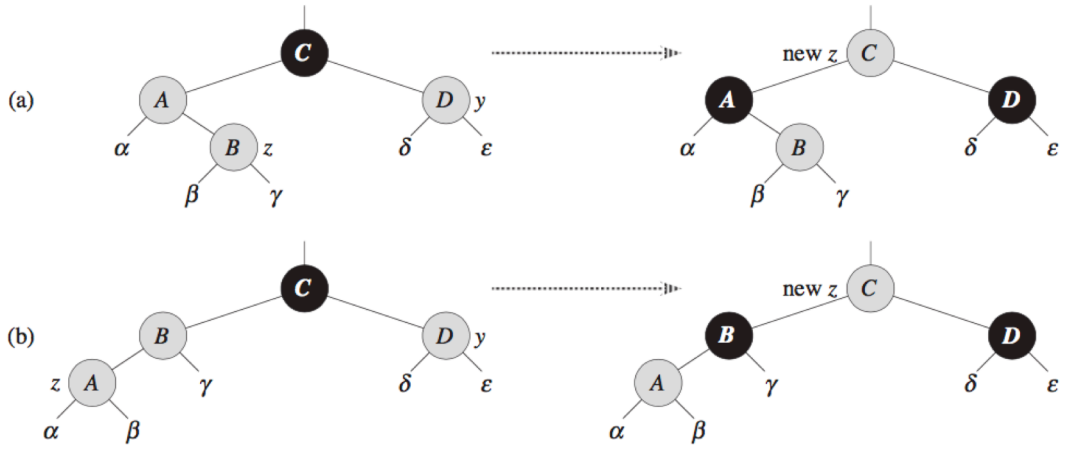
By property 5 the longest and shortest path must contain the same number of black nodes. By property 4 every other nodes in the longest path must be black and therefore the length is at most twice that of the shortest path. So length of longest path = height(x) $\leqslant$ 2 • bh(x) $\leqslant$ twice length of shortest path .

(7) 13.2-4 Show that any arbitrary n-node binary search tree can be transformed into any other arbitrary n-node binary search tree using O(n) rotations. (Hint: First show that at most n-1 right rotations suffice to transform the tree into a right-going chain.)

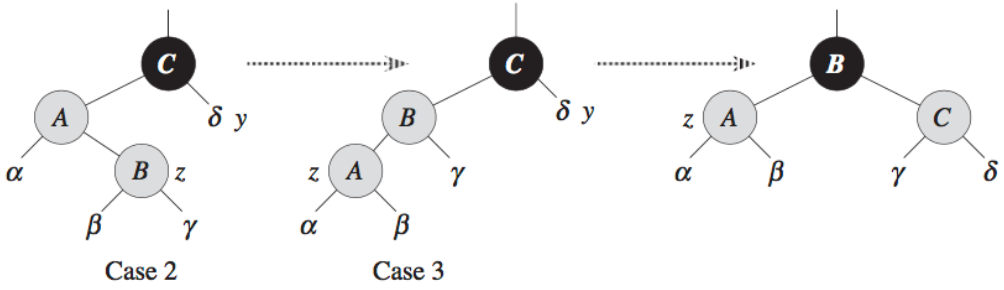
A right-going chain has all n nodes in the right side, basically it is a linked list. As long as the tree is not a right-going chain, we can just perform a right rotation on y. Therefor, at most n-1 right rotations suffice to transform the tree into a right-going chain. Then , if we knew he sequence of right rotations that

transforms an arbitrary binary search tree T to a single right-going chain T′ , then we could perform this sequence in reverse turning each right rotation into its inverse left rotation to transform T′ back into T .

(8) 13.3-3 Suppose that the black-height of each of the subtrees α, β, γ, δ, ε in Figure 13.5 and 13.6 is k. Label each node in each figure with its black-height to verify that the



indicated transformation preserves property 5.



(a). A, B, D has bh of k, C from the left path has bh of k, same with from the right path. After transform, A, B has bh of k, counting from left path, C has k+1 bh, from right path, C also has k+1 bh. (b) C has bh of k counting from both left and right path. After transforming, C has bh of k+1 counting from both paths. In 13.6, case 2, C has bh of k, so as in case 3 and after transforming.

(9) 14.1-5 Given an element x in an n-node order-statistic tree and a natural number i, how can we determine the ith successor of x in the linear order of the tree in O(lg n) time?

We can compare the value of current node, if the left node size is greater than i, we search to left, otherwise, we search to right, and search (i-x)th element instead.

(10) 14.3-3 Describe an efficient algorithm that, given an interval i, returns an interval overlapping i that has the minimum low endpoint, or T.nil if no such interval exists.

```
MIN-INTERVAL-SEARCH(T, i)

return MIN-INTERVAL-SEARCH-FROM(T, root[T ], i)

MIN-INTERVAL-SEARCH-FROM(T, x, i)

if left[x] ≠ nil[T] and max[left[x]] ≥ low[i]

        then y ← MIN-INTERVAL-SEARCH-FROM(T, left[x], i)

                if y ≠ nil[T ] then return y

                elseif i overlaps int[x] then return x

                else return nil[T ]

elseif i overlaps int[x] then return x

else return MIN-INTERVAL-SEARCH-FROM(T, right[x], i)
```