Laboratory 9: Cover Sheet

Name Terence Henriod Date 10/30/2013

Section 1101

Place a check mark in the *Assigned* column next to the exercises your instructor has assigned to you. Attach this cover sheet to the front of the packet of materials you submit following the laboratory.

Activities	**Assigned:** Check or list exercise numbers	**Completed**
Implementation Testing	✓	
Programming Exercise 1		
Programming Exercise 2		
Programming Exercise 3		
Analysis Exercise 1		
Analysis Exercise 2		
	Total	

Laboratory 9: Analysis Exercise 1

Name Terence Henriod Date 10/30/2013

Section 1101

What are the heights of the shortest and tallest binary search trees that can be constructed from a set of *N* distinct keys? Give examples that illustrate your answer.

The shortest tree of N keys will have a height of the ceiling of 1 + log(N). This sort of tree would be a complete binary tree, that is, all levels above the lowest node will be completely filled. A pictorial example is shown below:

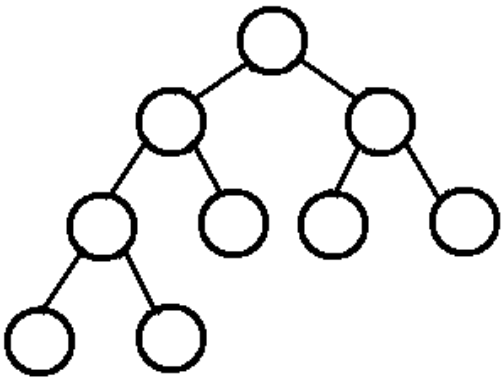

Image Source: http://www.andrew.cmu.edu/course/15-121/lectures/Trees/trees.html

The longest tree of N keys would have a height of N. This tree would be as unbalanced as possible, that is, each node would have one and only one node.

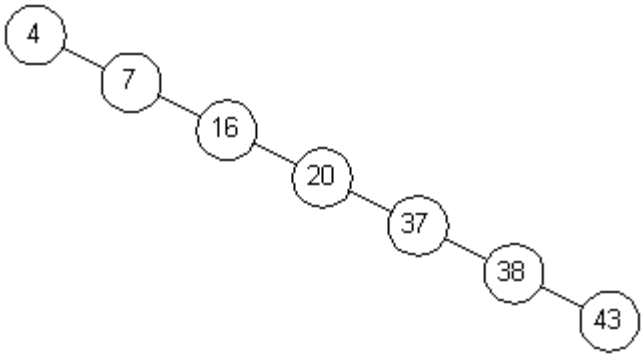

Image Source:
http://www.oopweb.com/Algorithms/Documents/Sman/Volume/BinarySearchTrees.html

Laboratory 9: Analysis Exercise 2

Name Terence Henriod Date 10/30/2013

Section 1101

Given the shortest possible binary search tree containing *N* distinct keys, develop worst-case, order-of-magnitude estimates of the execution time of the following Binary Search Tree ADT operations. Briefly explain your reasoning behind each of your estimates.

retrieve $O(\log_2(n))$

Explanation: Retrieve conducts a binary search until the sought item is found. Half of the nodes are disregarded in each operation until the sought node is found. This halving occurs until the sought node is found, which if we assume worst case scenario, will be a leaf on the lowest level. The retrieval operation has a constant complexity, as the retrieval operation happens once, no matter the size of the tree.

insert $O(\log_2(n))$

Explanation: Again, a binary search to find the correct placement for the item to be inserted is conducted. The actual insertion operation is a constant (a single node creation operation, no matter the size of the tree), so it is ignored when computing the Bog O bound.

remove O($\log_2$(n))

Explanation: Once again, a binary search operation is conducted to find the sought node for removal. However, to find a replacement for the removed node, a second "binary search" operation must be conducted to find the replacement data (called the in-order successor, found in the rightmost element of the left sub-tree corresponding to the node to be removed). This "doubles" the complexity of the removal operation, but this is not a change in order of magnitude, so the operation remains O(log(n)).

writeKeys O(n)

Explanation: Literally every node must be visited, so the operation takes n visit operations. Note, that I did not consider any tracing operations, although this would not change the complexity because when considering operations such as seeking the next node, one such operation is performed for each node. Thus this would be a constant scale factor that would be paired with n, which could be ignored for purposes of Big O notation.