Group Leader: Elizabeth McClellan

Group Tester: Drew Aaron

Group Requirement Leader: Andrew Hamilton

Group Documenter: Michael Beaver

Course: CS 355

Semester: Fall 2012

Assignment Number: 6

Assignment Type: Homework 4 – Group 2

Assignment Description: Implement a template Binary Search Tree.

Assignment Due Date: Thursday, October 4, 2012 (precisely at 12:30 p.m.)

To Be Included in Portfolio: YES

Total Grade: Coding Requirements Grade (60), Test cases Grade (20), Analysis Grade (20)

Coding Requirements:

1. \_\_\_\_\_ BNode class complete
   1. \_\_\_\_3 Node Constructors
   2. \_\_\_\_ GetData
2. \_\_\_\_\_ BST class complete
   1. \_\_\_\_ BST constructor
   2. \_\_\_\_ BST copy constructor
   3. \_\_\_\_Destructor
   4. \_\_\_\_Assignment operator
   5. \_\_\_\_Insert (in order, return true or false, place cursor at new item)
   6. \_\_\_\_Remove (given item to remove, return true or false, place cursor at parent of removed item or at root
   7. \_\_\_\_Search (return address if found from cursor to end of list, return NULL if not found, place cursor at found location or at end of list (rightmost node) if not found)
   8. \_\_\_\_Test Print Routine (Print the list (in order) separated by tabs, print square brackets around the value at cursor)
   9. \_\_\_\_AtCursor (return data item at the cursor)
   10. \_\_\_\_GoToBeginning (move cursor to beginning of the list, NULL if empty)
   11. \_\_\_\_GoToEnd (move cursor to last item in list, NULL if empty)
   12. \_\_\_\_GoToNext (move cursor to next slot, if on last item, move to first item)
   13. \_\_\_\_GotToPrev (move cursor to the previous slot. If on first item, move to last)
   14. \_\_\_\_ClearList (deallocate space, set head and cursor to NULL, can be called from destructor)
   15. \_\_\_\_EmptyList (return true if empty, false otherwise)
   16. \_\_\_\_PrintPre (print preorder)
   17. \_\_\_\_PrintPost (Print postorder)

Test Case Requirements Met:  
\_\_\_\_ created at least one test case for each method  
\_\_\_\_ test cases showed methods were correct  
Analysis Requirements Met:  
\_\_\_\_ Clear and correct communication  
\_\_\_\_ Reasonable/correct answers and justifications

Names: Drew Aaron, Michael Beaver, Andrew Hamilton, Elizabeth McClellan

Course: CS 355

Semester: Fall 2012

Assignment Number: 6

Assignment Type: Homework 6 – BST – Test Cases

Assignment Description: Create Test ensure your data structure is correct and robust.

Assignment Due Date: Thursday, October 4, 2012 (precisely at 12:30 p.m.)

To Be Included in Portfolio: YES

Test Case 1 – Testing GoToNext

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(15) | [15] | [15] | No |
| Insert(7) | [7] 15 | [7] 15 | No |
| Insert(12) | 7 [12] 15 | 7 [12] 15 | No |
| Insert(3) | [3] 7 12 15 | [3] 7 12 15 | No |
| Insert(20) | 3 7 12 15 [20] | 3 7 12 15 [20] | No |
| Insert(16) | 3 7 12 15 [16] 20 | 3 7 12 15 [16] 20 | No |
| Insert(17) | 3 7 12 15 16 [17] 20 | 3 7 12 15 16 [17] 20 | No |
| Insert(14) | 3 7 12 [14] 15 16 17 20 | 3 7 12 [14] 15 16 17 20 | No |
| Insert(13) | 3 7 12 [13] 14 15 16 17 20 | 3 7 12 [13] 14 15 16 17 20 | No |
| GoToBeginning | [3] 7 12 13 14 15 16 17 20 | [3] 7 12 13 14 15 16 17 20 | No |
| GoToNext | 3 [7] 12 13 14 15 16 17 20 | 3 [7] 12 13 14 15 16 17 20 | No |
| GoToNext | 3 7 [12] 13 14 15 16 17 20 | 3 7 [12] 13 14 15 16 17 20 | No |
| GoToNext | 3 7 12 [13] 14 15 16 17 20 | 3 7 12 [13] 14 15 16 17 20 | No |
| GoToNext | 3 7 12 13 [14] 15 16 17 20 | 3 7 12 13 [14] 15 16 17 20 | No |
| GoToNext | 3 7 12 13 14 [15] 16 17 20 | 3 7 12 13 [14] 15 16 17 20 | **Yes** |
| GoToNext | 3 7 12 13 14 15 [16] 17 20 | 3 7 12 13 [14] 15 16 17 20 | **Yes** |

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(15) | [15] | [15] | No |
| Insert(7) | [7] 15 | [7] 15 | No |
| Insert(12) | 7 [12] 15 | 7 [12] 15 | No |
| Insert(3) | [3] 7 12 15 | [3] 7 12 15 | No |
| Insert(20) | 3 7 12 15 [20] | 3 7 12 15 [20] | No |
| Insert(16) | 3 7 12 15 [16] 20 | 3 7 12 15 [16] 20 | No |
| Insert(17) | 3 7 12 15 16 [17] 20 | 3 7 12 15 16 [17] 20 | No |
| Insert(14) | 3 7 12 [14] 15 16 17 20 | 3 7 12 [14] 15 16 17 20 | No |
| Insert(13) | 3 7 12 [13] 14 15 16 17 20 | 3 7 12 [13] 14 15 16 17 20 | No |
| GoToBeginning | [3] 7 12 13 14 15 16 17 20 | [3] 7 12 13 14 15 16 17 20 | No |
| GoToNext | 3 [7] 12 13 14 15 16 17 20 | 3 [7] 12 13 14 15 16 17 20 | No |
| GoToNext | 3 7 [12] 13 14 15 16 17 20 | 3 7 [12] 13 14 15 16 17 20 | No |
| GoToNext | 3 7 12 [13] 14 15 16 17 20 | 3 7 12 [13] 14 15 16 17 20 | No |
| GoToNext | 3 7 12 13 [14] 15 16 17 20 | 3 7 12 13 [14] 15 16 17 20 | No |
| GoToNext | 3 7 12 13 14 [15] 16 17 20 | 3 7 12 13 14 [15] 16 17 20 | No |
| GoToNext | 3 7 12 13 14 15 [16] 17 20 | 3 7 12 13 14 15 [16] 17 20 | No |

Test Case 2 – Testing GoToPrev and GoToRoot

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(15) | [15] | [15] | No |
| Insert(7) | [7] 15 | [7] 15 | No |
| Insert(12) | 7 [12] 15 | 7 [12] 15 | No |
| Insert(3) | [3] 7 12 15 | [3] 7 12 15 | No |
| Insert(20) | 3 7 12 15 [20] | 3 7 12 15 [20] | No |
| Insert(16) | 3 7 12 15 [16] 20 | 3 7 12 15 [16] 20 | No |
| Insert(17) | 3 7 12 15 16 [17] 20 | 3 7 12 15 16 [17] 20 | No |
| Insert(25) | 3 7 12 15 16 17 20 [25] | 3 7 12 15 16 17 20 [25] | No |
| Insert(500) | 3 7 12 15 16 17 20 25 [500] | 3 7 12 15 16 17 20 25 [500] | No |
| GoToRoot | 3 7 12 [15] 16 17 20 25 500 | 3 7 12 [15] 16 17 20 25 500 | No |
| Find(25) | 3 7 12 15 16 17 20 [25] 500 | 3 7 12 15 16 17 20 [25] 500 | No |
| GoToPrev | 3 7 12 15 16 17 [20] 25 500 | 3 7 12 15 16 17 [20] 25 500 | No |
| GoToPrev | 3 7 12 15 16 [17] 20 25 500 | 3 7 12 15 16 17 20 [25] 500 | **Yes** |
| GoToPrev | 3 7 12 15 [16] 17 20 25 500 | 3 7 12 15 16 17 [20] 25 500 | **Yes** |

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(15) | [15] | [15] | No |
| Insert(7) | [7] 15 | [7] 15 | No |
| Insert(12) | 7 [12] 15 | 7 [12] 15 | No |
| Insert(3) | [3] 7 12 15 | [3] 7 12 15 | No |
| Insert(20) | 3 7 12 15 [20] | 3 7 12 15 [20] | No |
| Insert(16) | 3 7 12 15 [16] 20 | 3 7 12 15 [16] 20 | No |
| Insert(17) | 3 7 12 15 16 [17] 20 | 3 7 12 15 16 [17] 20 | No |
| Insert(25) | 3 7 12 15 16 17 20 [25] | 3 7 12 15 16 17 20 [25] | No |
| Insert(500) | 3 7 12 15 16 17 20 25 [500] | 3 7 12 15 16 17 20 25 [500] | No |
| GoToRoot | 3 7 12 [15] 16 17 20 25 500 | 3 7 12 [15] 16 17 20 25 500 | No |
| Find(25) | 3 7 12 15 16 17 20 [25] 500 | 3 7 12 15 16 17 20 [25] 500 | No |
| GoToPrev | 3 7 12 15 16 17 [20] 25 500 | 3 7 12 15 16 17 [20] 25 500 | No |
| GoToPrev | 3 7 12 15 16 [17] 20 25 500 | 3 7 12 15 16 [17] 20 25 500 | No |
| GoToPrev | 3 7 12 15 [16] 17 20 25 500 | 3 7 12 15 [16] 17 20 25 500 | No |

Test Case 3 – Testing GoToBeginning and GoToEnd

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(15) | [15] | [15] | No |
| Insert(7) | [7] 15 | [7] 15 | No |
| Insert(12) | 7 [12] 15 | 7 [12] 15 | No |
| Insert(3) | [3] 7 12 15 | [3] 7 12 15 | No |
| Insert(20) | 3 7 12 15 [20] | 3 7 12 15 [20] | No |
| Insert(16) | 3 7 12 15 [16] 20 | 3 7 12 15 [16] 20 | No |
| Insert(17) | 3 7 12 15 16 [17] 20 | 3 7 12 15 16 [17] 20 | No |
| Insert(8) | 3 7 [8] 12 15 16 17 20 | 3 7 [8] 12 15 16 17 20 | No |
| GoToBeginning | [3] 7 8 12 15 16 17 20 | [3] 7 8 12 15 16 17 20 | No |
| GoToEnd | 3 7 8 12 15 16 17 [20] | 3 7 8 12 15 16 17 [20] | No |
| GoToNext | [3] 7 8 12 15 16 17 20 | [3] 7 8 12 15 16 17 20 | No |
| GoToPrev | 3 7 8 12 15 16 17 [20] | 3 7 8 12 15 16 17 [20] | No |
| GoToBeginning | [3] 7 8 12 15 16 17 20 | [3] 7 8 12 15 16 17 20 | No |
| GoToEnd | 3 7 8 12 15 16 17 [20] | 3 7 8 12 15 16 17 [20] | No |

Test Case 4 – Testing Print In Order, Print Pre-Order, Print Post-Order (Unbalanced BST)

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(12) | [12] | [12] | No |
| Insert(25) | 12 [25] | 12 [25] | No |
| Insert(80) | 12 25 [80] | 12 25 [80] | No |
| Insert(62) | 12 25 [62] 80 | 12 25 [62] 80 | No |
| Insert(82) | 12 25 62 80 [82] | 12 25 62 80 [82] | No |
| Insert(17) | 12 [17] 25 62 80 82 | 12 [17] 25 62 80 82 | No |
| Insert(19) | 12 17 [19] 25 62 80 82 | 12 17 [19] 25 62 80 82 | No |
| Insert(21) | 12 17 19 [21] 25 62 80 82 | 12 17 19 [21] 25 62 80 82 | No |
| Insert(18) | 12 17 [18] 19 21 25 62 80 82 | 12 17 [18] 19 21 25 62 80 82 | No |
| Insert(20) | 12 17 18 19 [20] 21 25 62 80 82 | 12 17 18 19 [20] 21 25 62 80 82 | No |
| Insert(16) | 12 [16] 17 18 19 20 21 25 62 80 82 | 12 [16] 17 18 19 20 21 25 62 80 82 | No |
| Insert(13) | 12 [13] 16 17 18 19 20 21 25 62 80 82 | 12 [13] 16 17 18 19 20 21 25 62 80 82 | No |
| Insert(90) | 12 13 16 17 18 19 20 21 25 62 80 82 [90] | 12 13 16 17 18 19 20 21 25 62 80 82 [90] | No |
| Insert(84) | 12 13 16 17 18 19 20 21 25 62 80 82 [84] 90 | 12 13 16 17 18 19 20 21 25 62 80 82 [84] 90 | No |
| Insert(500) | 12 13 16 17 18 19 20 21 25 62 80 82 84 90 [500] | 12 13 16 17 18 19 20 21 25 62 80 82 84 90 [500] | No |
| Print In Order | 12 13 16 17 18 19 20 21 25 62 80 82 84 90 [500] | 12 13 16 17 18 19 20 21 25 62 80 82 84 90 [500] | No |
| Print Pre-Order | 12 25 17 16 13 19 18 21 20 80 62 82 90 84 [500] | 12 25 17 16 13 19 18 21 20 80 62 82 90 84 [500] | No |
| Print Post-Order | 13 16 18 20 21 19 17 62 84 [500] 90 82 80 25 12 | 13 16 18 20 21 19 17 62 84 [500] 90 82 80 25 12 | No |

Test Case 5 – Testing Print In Order, Print Pre-Order, Print Post-Order, GoToBeginning, GoToEnd, GoToNext, GoToPrev, Copy Constructor, Assignment Operator (Unbalanced BST)

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(50) | [50] | [50] | No |
| Insert(42) | [42] 50 | [42] 50 | No |
| Insert(23) | [23] 42 50 | [23] 42 50 | No |
| Insert(19) | [19] 23 42 50 | [19] 23 42 50 | No |
| Insert(43) | 19 23 42 [43] 50 | 19 23 42 [43] 50 | No |
| Insert(18) | [18] 19 23 42 43 50 | [18] 19 23 42 43 50 | No |
| Insert(2) | [2] 18 19 23 42 43 50 | [2] 18 19 23 42 43 50 | No |
| Insert(20) | 2 18 19 [20] 23 42 43 50 | 2 18 19 [20] 23 42 43 50 | No |
| Insert(22) | 2 18 19 20 [22] 23 42 43 50 | 2 18 19 20 [22] 23 42 43 50 | No |
| Print In Order | 2 18 19 20 [22] 23 42 43 50 | 2 18 19 20 [22] 23 42 43 50 | No |
| Print Pre-Order | 50 42 23 19 18 2 20 [22] 43 | 50 42 23 19 18 2 20 [22] 43 | No |
| Print Post-Order | 2 18 [22] 20 19 23 43 42 50 | 2 18 [22] 20 19 23 43 42 50 | No |
| GoToBeginning | [2] 18 19 20 22 23 42 43 50 | [2] 18 19 20 22 23 42 43 50 | No |
| GoToEnd | 2 18 19 20 22 23 42 43 [50] | 2 18 19 20 22 23 42 43 [50] | No |
| GoToNext | [2] 18 19 20 22 23 42 43 50 | [2] 18 19 20 22 23 42 43 50 | No |
| GoToNext | 2 [18] 19 20 22 23 42 43 50 | 2 [18] 19 20 22 23 42 43 50 | No |
| GoToPrev | [2] 18 19 20 22 23 42 43 50 | [2] 18 19 20 22 23 42 43 50 | No |
| GoToPrev | 2 18 19 20 22 23 42 43 [50] | 2 18 19 20 22 23 42 43 [50] | No |
| Copy Constructor | New: 2 18 19 20 22 23 42 43 [50]  Modified New: [-10000] 2 18 19 20 22 23 42 43 50  Original: 2 18 19 20 22 23 42 43 [50] | New: 2 18 19 20 22 23 42 43 [50]  Modified New: [-10000] 2 18 19 20 22 23 42 43 50  Original: 2 18 19 20 22 23 42 43 [50] | No |
| Assignment Operator | Modified New: [-100000] 2 18 19 20 22 23 42 43 50  Original: 2 18 19 20 22 23 42 43 [50]  <destroy new copy>  Original: 2 18 19 20 22 23 42 43 [50] | Modified New: [-100000] 2 18 19 20 22 23 42 43 50  Original: 2 18 19 20 22 23 42 43 [50]  <destroy new copy>  Original: 2 18 19 20 22 23 42 43 [50] | No |

Test Case 6 – Testing Insert, Empty, and ClearList

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(50) | [50] | [50] | No |
| Insert(42) | [42] 50 | [42] 50 | No |
| Insert(23) | [23] 42 50 | [23] 42 50 | No |
| Insert(19) | [19] 23 42 50 | [19] 23 42 50 | No |
| Insert(43) | 19 23 42 [43] 50 | 19 23 42 [43] 50 | No |
| Insert(65) | 19 23 42 43 50 [65] | 19 23 42 43 50 [65] | No |
| Insert(63) | 19 23 42 43 50 [63] 65 | 19 23 42 43 50 [63] 65 | No |
| Insert(66) | 19 23 42 43 50 63 65 [66] | 19 23 42 43 50 63 65 [66] | No |
| Insert(70) | 19 23 42 43 50 63 65 66 [70] | 19 23 42 43 50 63 65 66 [70] | No |
| Empty | Tree is Not Empty | Tree is Not Empty | No |
| ClearList | Tree is Empty | Tree is Empty | No |
| Empty | Tree is Empty | Tree is Empty | No |
| Insert(5000) | [5000] | [5000] | No |
| Empty | Tree is Not Empty | Tree is Not Empty | No |
| ClearList | Tree is Empty | Tree is Empty | No |

Test Case 7 – General test

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(50) | [50] | [50] | No |
| Insert(42) | [42] 50 | [42] 50 | No |
| Insert(65) | 42 50 [65] | 42 50 [65] | No |
| Insert(20) | [20] 42 50 65 | [20] 42 50 65 | No |
| Insert(70) | 20 42 50 65 [70] | 20 42 50 65 [70] | No |
| Empty | Tree is not Empty | Tree is not Empty | No |
| GoToBeginning | [20] 42 50 65 70 | [20] 42 50 65 70 | No |
| Find(20) | [20] 42 50 65 70 | [20] 42 50 65 70 | No |
| Find(50) | 20 42 50 65 [70] | 20 42 50 65 [70] | No |
| Remove(42) | 20 [50] 65 70 | 20 [50] 65 70 | No |
| GoToNext | 20 50 [65] 70 | 20 50 [65] 70 | No |
| GoToEnd | 20 50 65 [70] | 20 50 65 [70] | No |
| Remove(50) | 20 [65] 70 | 20 [65] 70 | No |
| ClearList | Tree is empty | Tree is empty | No |
| Empty | Tree is empty | Tree is empty | No |
| GoToRoot | Tree is empty | Tree is empty | No |
| Insert(100) | [100] | [100] | No |
| GoToRoot | [100] | [100] | No |
| Remove(50) | [100] | [100] | No |
| Remove(100) | Tree is empty | Tree is empty | No |
| Copy Constructor | New: Tree is empty  Modified New: [-10000]  Original: Tree is empty | New: Tree is empty  Modified New: [-10000]  Original: Tree is empty | No |
| Assignment Operator | Modified New: [-10000]  Original: Tree is empty <destroy new copy>  Original: Tree is empty | Modified New: [-10000]  Original: Tree is empty <destroy new copy>  Original: Tree is empty | No |

Test Case 8 – Rigorously Testing Everything – Codename: The Behemoth (see accompanying BST graph)

|  |  |  |  |
| --- | --- | --- | --- |
| Date/Time: | Expected Result | Actual Result | Action Needed |
| Insert(50) | [50] | [50] | No |
| Insert(48) | [48] 50 | [48] 50 | No |
| Insert(21) | [21] 48 50 | [21] 48 50 | No |
| Insert(63) | 21 48 50 [63] | 21 48 50 [63] | No |
| Insert(13) | [13] 21 48 50 63 | [13] 21 48 50 63 | No |
| Insert(31) | 13 21 [31] 48 50 63 | 13 21 [31] 48 50 63 | No |
| Insert(45) | 13 21 31 [45] 48 50 63 | 13 21 31 [45] 48 50 63 | No |
| Insert(71) | 13 21 31 45 48 50 63 [71] | 13 21 31 45 48 50 63 [71] | No |
| Insert(61) | 13 21 31 45 48 50 [61] 63 71 | 13 21 31 45 48 50 [61] 63 71 | No |
| Insert(55) | 13 21 31 45 48 50 [55] 61 63 71 | 13 21 31 45 48 50 [55] 61 63 71 | No |
| Insert(62) | 13 21 31 45 48 50 55 61 [62] 63 71 | 13 21 31 45 48 50 55 61 [62] 63 71 | No |
| Insert(19) | 13 [19] 21 31 45 48 50 55 61 62 63 71 | 13 [19] 21 31 45 48 50 55 61 62 63 71 | No |
| Insert(65) | 13 19 21 31 45 48 50 55 61 62 63 [65] 71 | 13 19 21 31 45 48 50 55 61 62 63 [65] 71 | No |
| Insert(69) | 13 19 21 31 45 48 50 55 61 62 63 65 [69] 71 | 13 19 21 31 45 48 50 55 61 62 63 65 [69] 71 | No |
| Insert(80) | 13 19 21 31 45 48 50 55 61 62 63 65 69 71 [80] | 13 19 21 31 45 48 50 55 61 62 63 65 69 71 [80] | No |
| Insert(32) | 13 19 21 31 [32] 45 48 50 55 61 62 63 65 69 71 80 | 13 19 21 31 [32] 45 48 50 55 61 62 63 65 69 71 80 | No |
| Insert(60) | 13 19 21 31 32 45 48 50 55 [60] 61 62 63 65 69 71 80 | 13 19 21 31 32 45 48 50 55 [60] 61 62 63 65 69 71 80 | No |
| Insert(9) | [9] 13 19 21 31 32 45 48 50 55 60 61 62 63 65 69 71 80 | [9] 13 19 21 31 32 45 48 50 55 60 61 62 63 65 69 71 80 | No |
| Insert(66) | 9 13 19 21 31 32 45 48 50 55 60 61 62 63 65 [66] 69 71 80 | 9 13 19 21 31 32 45 48 50 55 60 61 62 63 65 [66] 69 71 80 | No |
| Insert(40) | 9 13 19 21 31 32 [40] 45 48 50 55 60 61 62 63 65 66 69 71 80 | 9 13 19 21 31 32 [40] 45 48 50 55 60 61 62 63 65 66 69 71 80 | No |
| Insert(64) | 9 13 19 21 31 32 40 45 48 50 55 60 61 62 63 [64] 65 66 69 71 80 | 9 13 19 21 31 32 40 45 48 50 55 60 61 62 63 [64] 65 66 69 71 80 | No |
| Insert(46) | 9 13 19 21 31 32 40 45 [46] 48 50 55 60 61 62 63 64 65 66 69 71 80 | 9 13 19 21 31 32 40 45 [46] 48 50 55 60 61 62 63 64 65 66 69 71 80 | No |
| Insert(75) | 9 13 19 21 31 32 40 45 46 48 50 55 60 61 62 63 64 65 66 69 71 [75] 80 | 9 13 19 21 31 32 40 45 46 48 50 55 60 61 62 63 64 65 66 69 71 [75] 80 | No |
| Insert(77) | 9 13 19 21 31 32 40 45 46 48 50 55 60 61 62 63 64 65 66 69 71 75 [77] 80 | 9 13 19 21 31 32 40 45 46 48 50 55 60 61 62 63 64 65 66 69 71 75 [77] 80 | No |
| Insert(49) | 9 13 19 21 31 32 40 45 46 48 [49] 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 [49] 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| Print In Order | 9 13 19 21 31 32 40 45 46 48 [49] 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 [49] 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| Print Pre-Order | 50 48 21 13 9 19 31 45 32 40 46 [49] 63 61 55 60 62 71 65 64 69 66 80 75 77 | 50 48 21 13 9 19 31 45 32 40 46 [49] 63 61 55 60 62 71 65 64 69 66 80 75 77 | No |
| Print Post-Order | 9 19 13 40 32 46 45 31 21 [49] 48 60 55 62 61 64 66 69 65 77 75 80 71 63 50 | 9 19 13 40 32 46 45 31 21 [49] 48 60 55 62 61 64 66 69 65 77 75 80 71 63 50 | No |
| GoToBeginning | [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 [13] 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 [13] 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 [19] 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 [19] 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 [21] 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 [21] 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 [31] 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 [31] 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 [32] 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 [32] 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 [40] 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 [40] 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 [45] 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 [45] 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 [46] 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 [46] 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 [48] 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 [48] 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 [49] 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 [49] 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 [55] 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 [55] 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 [60] 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 [60] 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 [61] 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 [61] 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 [62] 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 [62] 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 [63] 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 [63] 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 [64] 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 [64] 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 [65] 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 [65] 66 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 [66] 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 [66] 69 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 [69] 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 [69] 71 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 [71] 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 [71] 75 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 [75] 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 [75] 77 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 [77] 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 [77] 80 | No |
| GoToNext | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | No |
| GoToNext | [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 [77] 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 [77] 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 [75] 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 [75] 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 [71] 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 [71] 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 [69] 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 [69] 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 [66] 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 [66] 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 [65] 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 [65] 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 [64] 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 [64] 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 [63] 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 [63] 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 [62] 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 [62] 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 [61] 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 [61] 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 [60] 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 55 [60] 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 [55] 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 50 [55] 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 [49] 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 [49] 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 [48] 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 [48] 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 [46] 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 [46] 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 [45] 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 [45] 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 [40] 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 [40] 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 [32] 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 [32] 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 [31] 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 [31] 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 [21] 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 [21] 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 [19] 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 [19] 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 [13] 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 [13] 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | No |
| GoToRoot | 9 13 19 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| AtCursor | 50 | 50 | No |
| Find(31) | 9 13 19 21 [31] 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 [31] 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| AtCursor | 31 | 31 | No |
| Find(46) | 9 13 19 21 31 32 40 45 [46] 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 [46] 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| AtCursor | 46 | 46 | No |
| Find(40) | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | No |
| AtCursor | 80 | 80 | No |
| Assignment Operator | Original: 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80]  New: 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80]  Original: 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80]  Modified New: [-5000] 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 5000  Original: 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | Original: 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80]  New: 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80]  Original: 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80]  Modified New: [-5000] 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 5000  Original: 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | No |
| GoToNext | [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| Copy Constructor | Original: [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80  New: [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80  Original: [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80  Modified New: -3000 9 13 19 21 31 32 40 45 46 [47] 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 3000  Original: [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | Original: [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80  New: [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80  Original: [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80  Modified New: -3000 9 13 19 21 31 32 40 45 46 [47] 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 3000  Original: [9] 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToPrev | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | No |
| GoToRoot | 9 13 19 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 19 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| Find(20) | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | 9 13 19 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 [80] | No |
| Remove(19) | 9 [13] 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 [13] 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| GoToNext | 9 13 [21] 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | 9 13 [21] 31 32 40 45 46 48 49 50 55 60 61 62 63 64 65 66 69 71 75 77 80 | No |
| Remove(69) | 9 13 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 [65] 66 71 75 77 80 | 9 13 21 31 32 40 45 46 48 49 50 55 60 61 62 63 64 [65] 66 71 75 77 80 | No |
| Remove(59) | 9 13 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 71 75 77 80 | 9 13 21 31 32 40 45 46 48 49 [50] 55 60 61 62 63 64 65 66 71 75 77 80 | No |
| Remove(50) | 9 13 21 31 32 40 45 46 48 49 [55] 60 61 62 63 64 65 66 71 75 77 80 | 9 13 21 31 32 40 45 46 48 49 [55] 60 61 62 63 64 65 66 71 75 77 80 | No |
| Remove all but root | [55] | [55] | No |
| Insert(69) | 55 [69] | 55 [69] | No |
| Insert(10) | [10] 55 69 | [10] 55 69 | No |
| ClearList | Tree is empty | Tree is empty | No |

**CODENAME: The Behemoth**

Input: 50, 48, 21, 63, 13, 31, 45, 71, 61, 55, 62, 19, 65, 69, 80, 32, 60, 9, 66, 40, 64, 46, 75, 77, 49

60

50

63

71

61

55

62

65

69

80

66

64

75

77

48

21

13

31

45

19

32

9

40

46

49

Name: Michael Beaver

Course: CS 355

Semester: Fall 2012

Assignment Number: 6

Assignment Type: Homework 3 - Analysis

Assignment Description: Carefully answer the questions below. Be sure you answer in complete sentences and with correct grammar. The space provided is not an indicator for the space needed to answer the question. Your responses MUST be typed and printed before you arrive to class.

Assignment Due Date: Tuesday, October 9, 2012 (precisely at 12:30 p.m.)

To Be Included in Portfolio: YES

Question 1: Consider the runtime of GoToNext vs. GoToPrevious routines. State the runtime of each. Follow the statement of each with a justification.

O ( N ) Go To Next

Justification:

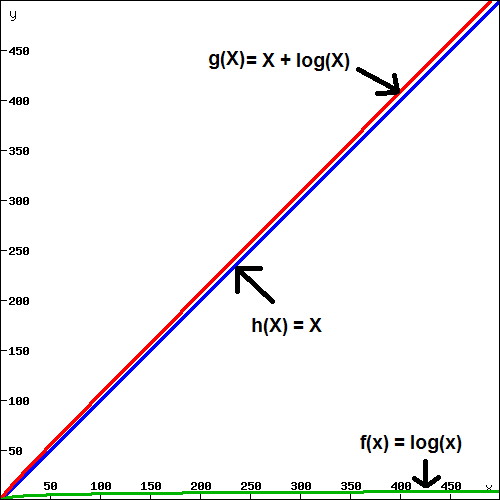
|  |  |  |
| --- | --- | --- |
| Line # | Code | Runtime |
| 1 | void BST::GoToNextHelp(int e, BNode \* n, bool & flag) { |  |
| 2 |  |  |
| 3 | *// Recursively traverse the tree until the proper value is reached or the end of the tree* |  |
| 4 | if ((n != NULL) && (flag == false)) { | 3 |
| 5 |  |  |
| 6 | *// Branch left* |  |
| 7 | GoToNextHelp(e, n->left, flag); | at most N-1 |
| 8 |  |  |
| 9 | *// Value found, set the cursor* |  |
| 10 | if ((n->GetData() > e) && (flag == false)) { | 4 |
| 11 |  |  |
| 12 | cursor = n; | 1 |
| 13 | flag = true; | 1 |
| 14 | return; | 1 |
| 15 |  |  |
| 16 | } |  |
| 17 |  |  |
| 18 | *// Branch right* |  |
| 19 | GoToNextHelp(e, n->right, flag); | at most N-1 |
| 20 |  |  |
| 21 | } |  |
| 22 |  |  |
| 23 | *// The next Node was not found so next must be the root* |  |
| 24 | if (flag == false) | 1 |
| 25 | cursor = root; | 1 |
| 26 | } | Total: O(N) |
| Line # | Code | Runtime |
| 1 | void BST::GoToNext() { |  |
| 2 |  |  |
| 3 | *// No tree, so return* |  |
| 4 | if (Empty()) | 1 |
| 5 | return; | 1 |
| 6 |  |  |
| 7 | *// Only root, so return* |  |
| 8 | else if ((root->left == NULL) && (root->right == NULL)) | 3 |
| 9 | return; | 1 |
| 10 |  |  |
| 11 | *// If at last value in tree, just go to the beginning* |  |
| 12 | else if (cursor == findMax()) | H + 1 |
| 13 | GoToBeginning(); | H (BST Height) |
| 14 |  |  |
| 15 | *// Let's look for the next Node and move the cursor to it* |  |
| 16 | else { |  |
| 17 |  |  |
| 18 | bool flag = false; | 1 |
| 19 | int cursorData = AtCursor()->GetData(); | 3 |
| 20 | BNode \* n = root; | 1 |
| 21 |  |  |
| 22 | *// Just search the left sub-tree* |  |
| 23 | if (cursorData < root->GetData()) | 1\ |
| 24 | n = n->left; | 1 \ |
| 25 |  | \ O(logN) |
| 26 | *// Just search the right sub-tree* | / |
| 27 | else if (cursorData > root->GetData()) | 1 / |
| 28 | n = n->right; | 1/ |
| 29 |  |  |
| 30 | *// Recursively move the cursor* |  |
| 31 | GoToNextHelp(cursorData, n, flag); | O(N) |
| 32 |  |  |
| 33 | } |  |
| 34 |  |  |
| 35 | } | Total: O(N) |

The runtime of this implementation of GoToNext is O(N). Lines 4-9 have no effect on the runtime because they occur in constant time. Lines 12-13 have a runtime of 2H + 1, or O(H) where H is the height of the BST. There may be better ways to write these lines so as to avoid the overhead of method calls. Regardless, a runtime of O(H) is insignificant compared to the runtime of O(N) since H ≤ N.

The lines of GoToNext that directly influence its runtime are lines 16-33. Lines 18-20 run in constant time, so they may be safely ignored. Lines 23-28 effectively halve the problem, so they happen in logarithmic time. If the value at the cursor is less than the root, there is a good chance the next value in the BST will be in the left sub-tree. If the value is not there, then the next value must be the root. Likewise, if the value at the cursor is greater than the root, then there is a good chance the next value in the BST will be in the right sub-tree. If not, then the next value will either be the root or the beginning BNode (hence lines 12-13). Therefore, lines 23-24 and lines 27-28 run in logarithmic time. Of course, only one of these conditional statements will ever be executed.

GoToNextHelp performs a linear in order traversal of one of the sub-trees of the BST. Lines 7 and 19 will at most have to traverse N-1 BNodes because there will only ever be at most N-1 BNodes in either sub-tree of the BST. Hence, the runtime of GoToNext will be the linear function 2N - 2, or O(N). While lines 10-16 and 24-25 affect the cursor, they run in constant time, so they have no significant impact on the overall runtime of GoToNextHelp.

Because GoToNextHelp runs in linear time and lines 23-28 of GoToNext run in logarithmic time, the combined runtime of lines 16-33 of GoToNext is N + logN. However, because the linear function N grows at a greater rate than the function logN, the method GoToNext is O(N). The lower term logN can be dropped because the linear growth of N (GoToNextHelp) will always grow faster than the logarithmic time of lines 23-28 of GoToNext. See the following graph where N is substituted for X on the xy-plane.



Graph of the functions f(x) = log(X), g(X) = X + log(X), and h(X) = X. Clearly g(X) grows at a relatively linear rate over time; hence, GoToNext is O(N), where the variable N is substituted for the variable X.

O( N ) GoToPrevious

Justification:

|  |  |  |
| --- | --- | --- |
| Line # | Code | Runtime |
| 1 | void BST::GoToPrevHelp(int e, BNode \* n, bool & flag) { |  |
| 2 |  |  |
| 3 | *// Recursively traverse the tree until the proper value is reached or the end of the tree* |  |
| 4 | if ((n != NULL) && (flag == false)) { | 3 |
| 5 |  |  |
| 6 | *// Branch right* |  |
| 7 | GoToPrevHelp(e, n->right, flag); | at most N-1 |
| 8 |  |  |
| 9 | *// Value found, set the cursor* |  |
| 10 | if ((n->GetData() < e) && (flag == false)) { | 4 |
| 11 |  |  |
| 12 | cursor = n; | 1 |
| 13 | flag = true; | 1 |
| 14 | return; | 1 |
| 15 |  |  |
| 16 | } |  |
| 17 |  |  |
| 18 | *// Branch left* |  |
| 19 | GoToPrevHelp(e, n->left, flag); | at most N-1 |
| 20 |  |  |
| 21 | } |  |
| 22 |  |  |
| 23 | *// The next Node was not found so previous must be the root* |  |
| 24 | if (flag == false) | 1 |
| 25 | cursor = root; | 1 |
| 26 | } | Total: O(N) |

(Continued on next page)

|  |  |  |
| --- | --- | --- |
| Line # | Code | Runtime |
| 1 | void BST::GoToPrev() { |  |
| 2 |  |  |
| 3 | *// No tree, so return* |  |
| 4 | if (Empty()) | 1 |
| 5 | return; | 1 |
| 6 |  |  |
| 7 | *// Only root, so return* |  |
| 8 | else if ((root->left == NULL) && (root->right == NULL)) | 3 |
| 9 | return; | 1 |
| 10 |  |  |
| 11 | *// If at first value in tree, just go to the end* |  |
| 12 | else if (cursor == findMin()) | H + 1 |
| 13 | GoToEnd(); | H (BST Height) |
| 14 |  |  |
| 15 | *// Let's look for the next Node and move the cursor to it* |  |
| 16 | else { |  |
| 17 |  |  |
| 18 | bool flag = false; | 1 |
| 19 | int cursorData = AtCursor()->GetData(); | 3 |
| 20 | BNode \* n = root; | 1 |
| 21 |  |  |
| 22 | *// Just search the left sub-tree* |  |
| 23 | if (cursorData < root->GetData()) | 1\ |
| 24 | n = n->left; | 1 \ |
| 25 |  | \ O(logN) |
| 26 | *// Just search the right sub-tree* | / |
| 27 | else if (cursorData > root->GetData()) | 1 / |
| 28 | n = n->right; | 1/ |
| 29 |  |  |
| 30 | *// Recursively move the cursor* |  |
| 31 | GoToPrevHelp(cursorData, n, flag); | O(N) |
| 32 |  |  |
| 33 | } |  |
| 34 |  |  |
| 35 | } | Total: O(N) |

This implementation of GoToPrev is almost identical to the previously discussed implementation of GoToNext, so the runtime of GoToPrev runs in the same linear time as GoToNext. Thus, GoToPrev is O(N). Like GoToNext, the lines 12-13 in GoToPrev are O(H). There may be better ways to write these lines so as to avoid the overhead of method calls. Regardless, the runtime of lines 12-13 is insignificant compared to the runtime of lines 16-33 because H ≤ N.

Like in GoToNext, lines 23-28 of GoToPrev cut the problem of finding the previous BNode value in half by eliminating half of the BST. Hence, these lines contribute a runtime of log(N).

Similarly to GoToNextHelp, GoToPrevHelp performs recursive calls to itself to find the previous BNode value in the BST. Whereas GoToNextHelp performed an in order traversal, GoToPrevHelp performs a reversed in order traversal that starts by traversing right, then processing the root, and then traverses to the left. Regardless of the traversal order, the method will still have to traverse at most N-1 BNodes to find the proper value. Hence, GoToPrevHelp is O(N).

The runtime of lines 16-33 of GoToPrev should be O(N + logN). However, log(N) is a lower term when compared to N; hence, the term log(N) may be dropped. Thus, GoToPrev is O(N). While N + log(N) is a more accurate representation of the runtime of GoToPrev, the function grows at a linear rate. Because Big-O is a loose upper bound and the actual runtime grows at a linear rate over time, GoToPrev is O(N). See the previously attached graph for a visual comparison of runtimes in the variable X. Because the methods GoToPrev and GoToNext are so similar in their implementation, it makes sense that they share similar—albeit exactly the same—runtimes.

Question 2: Consider the runtime of GoToBeginning vs. GoToEnd routines. State the runtime of each. Follow the statement of each with a justification. Discuss what change(s) could be made to the class to make GoToEnd a more efficient routine. Give a justification for each change.

O ( H ) GoToBeginning

Justification:

If the BST is not Empty, GoToBeginning sets the cursor equal to the result of findMin. Empty has a runtime of O(1) because it consists only of a few basic instructions.

|  |  |  |
| --- | --- | --- |
| Line # | Code | Runtime |
| 1 | void BST::GoToBeginning() { |  |
| 2 |  |  |
| 3 | *// Return if there is no tree* |  |
| 4 | if (Empty()) { | 1 |
| 5 |  |  |
| 6 | cursor = NULL; | 1 |
| 7 | return; | 1 |
| 8 |  |  |
| 9 | } |  |
| 10 |  |  |
| 11 | *// Otherwise, find the minimum value (the beginning)* |  |
| 12 | cursor = findMin(); | H + 1 (+1 for assignment) |
| 13 |  |  |
| 14 | } | Total: O(H) |

|  |  |  |
| --- | --- | --- |
| Line # | Code | Runtime |
| 1 | BNode \* BST::findMin() const { |  |
| 2 |  |  |
| 3 | BNode \* n = root; | 1 |
| 4 |  |  |
| 5 | *// Start at the root and go all the way to the left* |  |
| 6 | while (n->left != NULL) | H, where H  is the height  of the tree |
| 7 | n = n->left; |
| 8 |  |
| 9 | return n; | 1 |
| 10 |  |  |
| 11 | } | Total: O(H) |

The beginning of a BST is the minimum value, or leftmost BNode. This implementation of GoToBeginning makes use of a method call to findMin, which returns the address of the BNode with the minimum value. Hence, because Line 12 of GoToBeginning’s runtime is greater than the constant runtime of Lines 4-9, we are really concerned with the runtime of the method findMin.

The method findMin starts at the root rather than starting at the cursor. This means that half the BST is eliminated from traversal if the cursor was located on the right sub-tree. The left traversal (Lines 6-7) of findMin, in the worst case, will have to execute H times, where H is the height of the BST (see Question 3). It is possible that the runtime will be less than H, but the maximum runtime of findMin is H; hence, findMin is O(H).

Since findMin is O(H), and Line 12 of GoToBeginning is the most complex line, the runtime of GoToBeginning is O(H). In the best case (i.e., the root is the beginning), the runtime should be O(1). However, the linear runtime of O(H) is acceptable and more desirable than any other complex algorithms that might try to traverse the entire BST to find the beginning. By linearly traversing H times to the left in findMin, the runtime of GoToBeginning is “optimized” to O(H).

O( H ) GoToEnd

Justfication:

If the BST is not Empty, GoToEnd sets the cursor equal to the result of findMax. Empty has a runtime of O(1) because it consists only of a few basic instructions.

|  |  |  |
| --- | --- | --- |
| Line # | Code | Runtime |
| 1 | void BST::GoToEnd() { |  |
| 2 |  |  |
| 3 | *// Return if there is no tree* |  |
| 4 | if (Empty()) { | 1 |
| 5 |  |  |
| 6 | cursor = NULL; | 1 |
| 7 | return; | 1 |
| 8 |  |  |
| 9 | } |  |
| 10 |  |  |
| 11 | *// Otherwise, find the maximum value (the end)* |  |
| 12 | cursor = findMax(); | H + 1 (+1 for assignment) |
| 13 |  |  |
| 14 | } | Total: O(H) |

|  |  |  |
| --- | --- | --- |
| Line # | Code | Runtime |
| 1 | BNode \* BST::findMax() const { |  |
| 2 |  |  |
| 3 | BNode \* n = root; | 1 |
| 4 |  |  |
| 5 | *// Start at the root and go all the way to the right* |  |
| 6 | while (n->right != NULL) | H, where H  is the height  of the tree |
| 7 | n = n->right; |
| 8 |  |
| 9 | return n; | 1 |
| 10 |  |  |
| 11 | } | Total: O(H) |

This implementation of GoToEnd is very similar to GoToBeginning, except that GoToEnd calls findMax instead of findMin. The end of a BST is the maximum value, or rightmost BNode. This implementation of GoToEnd makes use of a method call to findMax, which returns the address of the BNode with the maximum value. Hence, because Line 12 of GoToEnd’s runtime is greater than the constant runtime of Lines 4-9, we are really concerned with the runtime of the method findMax.

The method findMax starts at the root rather than starting at the cursor. This means that half the BST is eliminated from traversal if the cursor was located on the right sub-tree. The right traversal (Lines 6-7) of findMax, in the worst case, will have to execute H times, where H is the height of the BST (see Question 3). It is possible that the runtime will be less than H, but the maximum runtime of findMax is H; hence, findMax is O(H).

Since findMax is O(H), and Line 12 of GoToEnd is the most complex line, the runtime of GoToEnd is O(H). In the best case (i.e., the root is the end), the runtime should be O(1). However, the linear runtime of O(H) is acceptable and more desirable than any other complex algorithms that might try to traverse the entire BST to find the end. By linearly traversing H times to the right in findMax, the runtime of GoToEnd is “optimized” to O(H).

The current implementation of the GoToEnd algorithm is about as efficient as can possibly be. There is no obviously visible way to optimize the algorithm to run in logarithmic time. However, it is possible that making changes to class data members and methods could potentially make GoToEnd more efficient.

It may be advantageous to incorporate a BNode pointer—as a private data member—named “end” to point to the last BNode in the BST. The “end” pointer would be set to NULL by default. In the Insert method, the “end” pointer would only be updated if a BNode is added to the end of the list. Similarly, if the “end” BNode is deleted via the Remove method, it would be redirected to point to the parent BNode. With the new “end” BNode pointer, GoToEnd could be rewritten to be potentially only one line (i.e., cursor = end).

If such modifications were made, the runtime of GoToEnd would be reduced to O(1), due to the one assignment operation. Of course, all of the background steps performed, such as moving the “end” pointer in Insert and Remove, would have little to no effect on the runtime of GoToEnd. What is more, it is possible that a “start” BNode pointer could be added as a private data member. The “start” pointer would serve a similar role as the “end” pointer, and it could potentially reduce the runtime of GoToBeginning to O(1).

Question 3: If you wanted to find the minimum value in the BST, what would the runtime of the algorithm be? Explain your answer.

O ( H ) GetMin, where H is the height of the BST

Justification:

The GetMin routine need only go straight (linearly) all the way down to the left of the BST. The minimum value will be the BNode, all the way to the left, with left and right pointers equal to NULL. In other words, the minimum value will always be the leftmost BNode. In the worst case, the minimum value is the lowest BNode in the BST; thus, the worst case runtime would be O(H), where H is the height of the BST. (For our purposes, the height of the BST is defined as the number of levels between the root BNode and the lowest BNode.)

If the minimum value is not the lowest BNode in the BST, then the runtime will still be O(H) because the runtimes will still grow at linear rates. If there is only one BNode (i.e., there is only a root and H = 0), then the runtime would be O(1) because there would be no left traversal of the left sub-tree. Hence, the algorithm will terminate with one basic instruction (i.e., processing the root). Special cases aside, the absolute worst case runtime for a GetMin algorithm would be O(H) because the algorithm needs to perform a linear traversal at most H times, where H is the height of the BST.

0

1

2

3

H = 3

In this generic example, the actual runtime of findMin—and GoToBeginning—is 3 because 3 BNodes are traversed. However, the actual runtime of findMax—and GoToEnd—is (H-1) = 2 because exactly 2 BNodes are traversed. Hence, the function H is not an exact upper bound for findMax and GoToEnd; rather, it is loose upper bound.