Laboratory 9: Cover Sheet

Name Catherine Pollock Date 10/19/14

Section 1001

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: Implementation Testing

Name Catherine Pollock______________________________ Date 10/19/14_________________

Section 1001_______________________________________

Check with your instructor whether you are to complete this exercise prior to your lab period or during lab.

Test Plan 9-1 (Binary Search Tree ADT operations)			
Test case	**Commands**	**Expected result**	**Checked**
Empty tree	-		
1	K	Keys:	Yes
2	C	Clear the tree	Yes
3	E	Tree is empty	Yes
Single branch	+9+3		
1	?3	Retrieved : getKey = 3	Yes
2	?4	Not found	Yes
3	-0	Not found	Yes
4	-9	Removed data item	Yes
5	K	Keys: 3	Yes
Only right branch	+0+1+2+3		
1	K	Keys: 0 1 2 3	Yes
2	?0	Retrieved : getKey = 0	Yes
3	C	Clear the tree	Yes
4	K	Keys:	Yes
Single item	+8		
1	?8	Retrieved : getKey = 8	Yes
2	?18	Not found	Yes
3	-8	Removed data item	Yes
Many branches	+51+73+99+32+35+21+7+1+97+42		
1	K	Keys: 1 7 21 32 35 42 51 73 97 99	Yes
2	-7	Removed data item	Yes
3	?35	Retrieved : getKey = 35	Yes
4	-51	Removed data item	Yes
5	+101	Insert : key = 101	Yes
6	K	Keys: 1 7 21 32 35 42 73 97 99 101	Yes
7	-51	Not found	Yes
8	?1	Retrieved : getKey = 1	Yes
9	C	Clear the tree	Yes
10	K	Keys:	Yes

Laboratory 9: Programming Exercise 1

Name Catherine Pollock Date 10/19/14

Section 1001

Test Plan 9-2 (accounts database indexing program)

Test case	Expected result	Checked
6274	0 : 6274 James Johnson 415.56	Yes
2843	1 : 2843 Marcus Wilson 9217.23	Yes
4892	2 : 4892 Maureen Albright 51462.56	Yes
8337	3 : 8337 Debra Douglas 27.26	Yes
1892	4 : 1892 Bruce Gold 719.32	Yes
9523	5 : 9523 John Carlson 1496.24	Yes
3165	6 : 3165 Mary Smith 918.26	Yes
3924	7 : 3924 Simon Becker 386.85	Yes
6023	8 : 6023 John Edgar 9.65	Yes
5290	9 : 5290 George Truman 16110.68	Yes
8529	10 : 8529 Ellen Fairchild 86.77	Yes
1144	11 : 1144 Donald Williams 4114.26	Yes
9999	No record with that account ID	Yes

Laboratory 9: Programming Exercise 2

Name Catherine Pollock Date 10/19/14

Section 1001

Test Plan 9-3 (getCount operation)			
Test case	**Commands**	**Expected result**	**Checked**
Many branches	+65+23+43+87+9+2+67		
1	G	7	Yes
2	-65G	6	Yes
3	-43G	5	Yes
Empty tree	-		
1	G	0	Yes
One branch	+1+2+3+4		
1	G	4	Yes
2	+5G	5	Yes
3	-1G	4	Yes

Test Plan 9-4 (getHeight operation)			
Test case	**Commands**	**Expected result**	**Checked**
Many branches	+65+23+43+87+9+2+67		
1	H	4	Yes
2	-65	4	Yes
3	-43	3	Yes
Empty tree	-		
1	H	0	Yes
One branch	+1+2+3+4		
1	H	4	Yes
2	+5H	5	Yes
3	-1H	4	Yes

Laboratory 9: Programming Exercise 3

Name Catherine Pollock Date 10/19/14

Section 1001

Test Plan 9-5 (writeLessThan operation)

Test case	Commands	Expected result	Checked

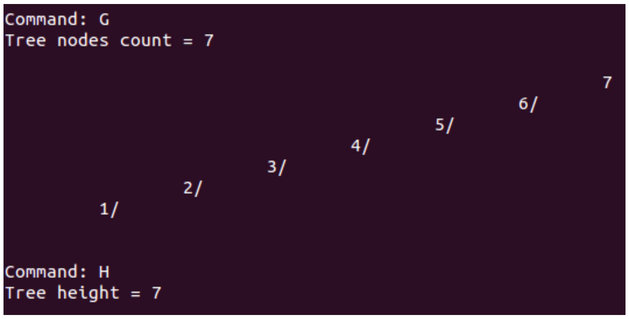
Laboratory 9: Analysis Exercise 1

Name Catherine Pollock Date 10/19/14

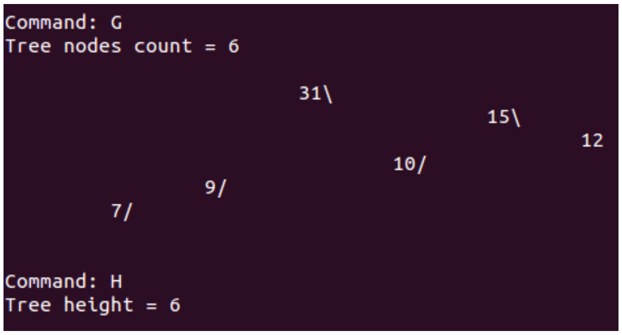
Section 1001

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.

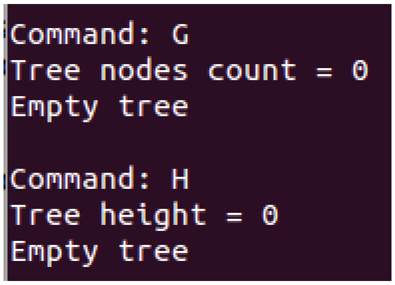
The **maximum** height for a tree of *N* keys is *N*. This is the case if each key has only one child, either right or left.
For example:



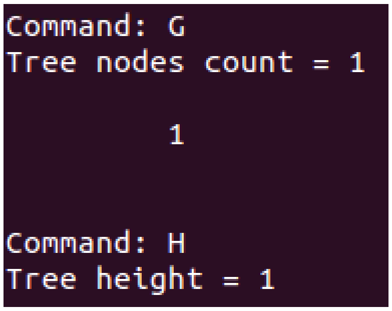
```
Command: G
Tree nodes count = 7

                                                         7
                                                 6/
                                         5/
                                 4/
                         3/
                 2/
         1/


Command: H
Tree height = 7
```



```
Command: G
Tree nodes count = 6

                         31\
                                         15\
                                                 12
                                 10/
                 9/
         7/


Command: H
Tree height = 6
```



```
Command: G
Tree nodes count = 0
Empty tree

Command: H
Tree height = 0
Empty tree
```

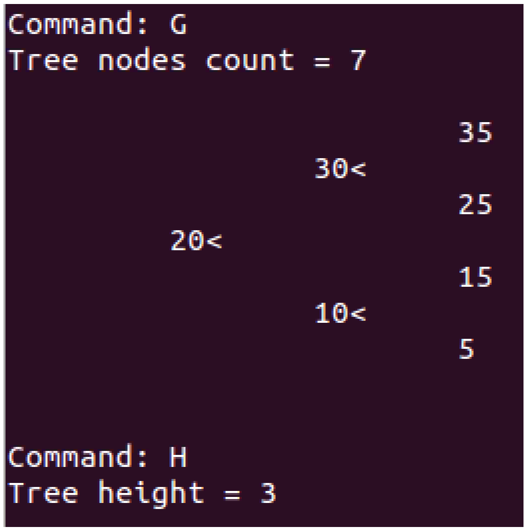


```
Command: G
Tree nodes count = 1

          1


Command: H
Tree height = 1
```

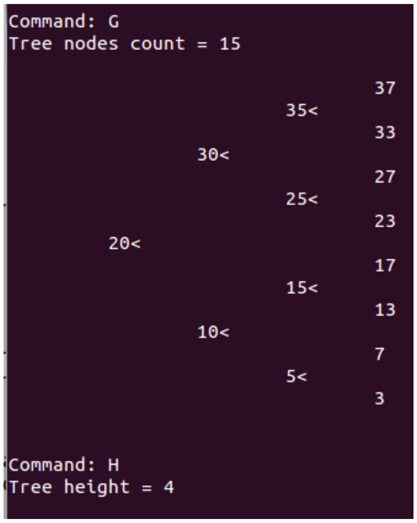
The **minimum** height for a tree of *N* keys is $log_2(N+1)$ (rounded up if decimal remainder). This means that each level of the tree (starting at root) must have two children before the next level can have children.

For example:

$log_2(7+1)=3=3$ $log_2(15+1)=4=4$

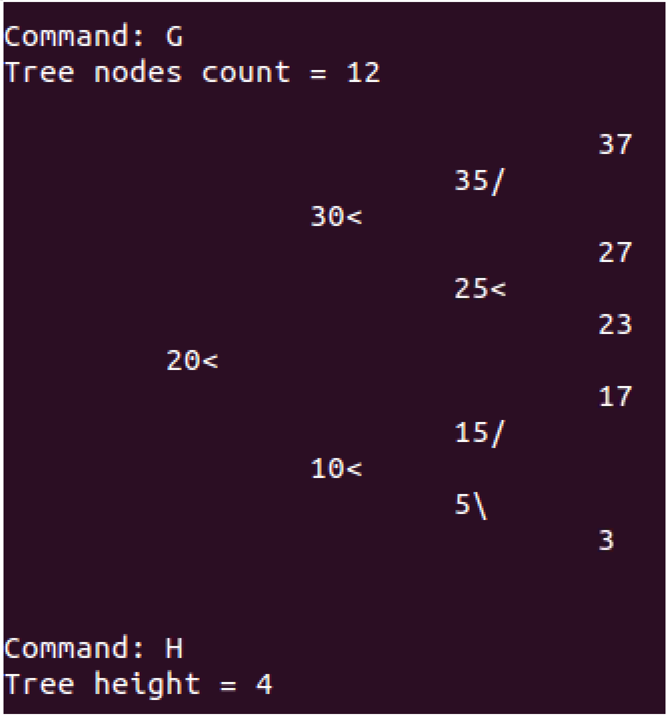


```
Command: G
Tree nodes count = 7

                   35
             30<
                   25
       20<
                   15
             10<
                   5

Command: H
Tree height = 3
```



```
Command: G
Tree nodes count = 15

                                 37
                         35<
                                 33
                 30<
                                 27
                         25<
                                 23
         20<
                                 17
                         15<
                                 13
                 10<
                                 7
                         5<
                                 3

Command: H
Tree height = 4
```

$log_2(12+1)=3.70=4$



```
Command: G
Tree nodes count = 12

                                 37
                         35/
                 30<
                                 27
                         25<
                                 23
         20<
                                 17
                         15/
                 10<
                         5\
                                 3

Command: H
Tree height = 4
```

Laboratory 9: Analysis Exercise 2

Name Catherine Pollock Date 10/19/14

Section 1001

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(N)

Explanation:

In the worst case scenario, the retrieveHelper function would search all the way from the root node to the node furthest from root, which would make the height equal to the number of nodes (as explained in previous page). In the worst case, the helper function would have to compare the value of each node, since each node only has one child (kind of like a linked list). Therefore, execution time is estimated for retrieve by O(N).

insert O(N)

Explanation:

In the worst case scenario, insertHelper would have to try to insert starting from the root, all the way until the node furthest from root, in a tree where each node only has one child (like a linked list). This would make the height equal to N, like how described above. The helper function would have to go to each child of each node, making it take N tries to insert. Therefore, execution time is estimated for insertion by O(N).

remove O(N)

Explanation:

Similar to the previous answers, the worst case scenario would feature a tree where each node has only one child, making it similar to a linked list with height of N. The removeHelper would have to check each node from root until the node furthest from root for removal in the worst case scenario, making the execution time for remove O(N).

writeKeys O(N)

Explanation:

Similar to the previous answers, the worst case scenario would feature a tree where each node has only one child, making it similar to a linked list with height of N. The writeKeysHelper would have to print each node from root until the node furthest from root for removal in the worst case scenario, making the execution time for writeKeys O(N).