**Trees Worksheet**

(preparation for TreeLab)

1.

|  |  |
| --- | --- |
| number of Nodes | 6 |
| Height | 2 |
| Depth of “B” | 1 |
| Length of longest path | 4 |
| Root | A |
| number of Leaves | 5 |
| number of Parents | 3 |
| number of Only Children | 1 |
| number of nodes at Level 0 | 1 |
| Preorder Traverse | ABDECF |
| Inorder Traverse | DBEAFC |
| Postorder Traverse | DEBFCA |
| Levelorder Traverse | ABCDEF |

|  |  |
| --- | --- |
| number of Nodes | 7 |
| Height | 2 |
| Level of “5” | 0 |
| Length of longest path | 3 |
| Root | 5 |
| number of Leaves | 6 |
| number of Parents | 3 |
| number of Only Children | 0 |
| number of nodes at Level 2 | 4 |
| Preorder Traverse | 5314869 |
| Inorder Traverse | 1345689 |
| Postorder Traverse | 1436985 |
| Levelorder Traverse | 5381469 |

2.

3. Trees are naturally recursive structures. Trace this code to search the BST above for “6”.

If the tree is empty  
 return false  
 else if the item in the root equals the target

return true

else if the item in the target is less than the item

return the result of searching the root’s left subtree

else

return the result of searching the root’s right subtree

Then trace the code to search the tree above for “7”.

5.

4.

|  |  |
| --- | --- |
| number of Nodes | 5 |
| Height of Tree | 2 |
| Depth of “\*” | 1 |
| Width | 3 |
| Root | + |
| number of Leaves | 4 |
| number of Parents | 2 |
| number of Only Children | 0 |
| number of nodes at Depth 1 | 2 |
| Preorder Traverse | +3\*54 |
| Inorder Traverse | 3+5\*4 |
| Postorder Traverse | 3 5 4 \* + |
| Levelorder Traverse | + 3 \* 5 4 |

|  |  |
| --- | --- |
| number of Nodes | 10 |
| Height | 3 |
| Depth of “D” | 2 |
| Width | 6 |
| Root | F |
| number of Leaves | 9 |
| number of Parents | 5 |
| number of Only Children | 1 |
| number of nodes at Level 3 | 3 |
| Preorder Traverse | FBADCEHGJI |
| Inorder Traverse | ABDCEFGHJI |
| Postorder Traverse | ACEDBGIJHF |
| Levelorder Traverse | FBHADGJCEI |

5.

|  |  |
| --- | --- |
| number of Nodes | 5 |
| Height | 2 |
| Depth of “\*” | 0 |
| Width | 3 |
| Root | \* |
| number of Leaves | 4 |
| number of Parents | 2 |
| number of Only Children | 0 |
| number of nodes at Level 2 | 2 |
| Preorder Traverse | \*+354 |
| Inorder Traverse | 3 + 5 \* 4 |
| Postorder Traverse | 1. 5 + 4\* |
| Levelorder Traverse | \*+ 4 3 5 |

6.

1. Which of the trees above are complete? \_\_\_\_\_\_1, 2, 6\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Which of the trees above are full? \_\_\_\_\_\_\_\_\_1, 2\_\_\_\_\_\_\_\_\_\_\_
3. Which of the trees above have the Binary Search Tree property? \_\_\_\_2, 4\_\_\_\_
4. Which of the trees above have the heap (either max or min) property? \_\_\_\_1\_\_\_\_
5. Which of the trees above are expression trees? \_\_\_\_\_\_\_\_\_\_5, 6\_\_\_\_\_\_\_\_\_\_\_
6. Draw some **full binary trees** and fill out the table:

|  |  |  |
| --- | --- | --- |
| number of nodes | depth (level) of the last node | height |
| 1 | 0 | 0 |
| 3 | 1 | 1 |
| 7 | 2 | 2 |
| 15 | 3 | 3 |
| 31 | 4 | 4 |
| 63 | 5 | 5 |
| 2^(d+1) - 1 | d | d |
| n | log2(n+1) -1 | Log2(n+1)-1 |

13. A full binary tree with *d* levels can have a maximum of \_2^(d + 1) - 1\_ nodes.

write formulas

write formulas

14. If a full binary tree has *n* nodes, then it has \_\_\_1\_\_\_\_\_\_\_ levels.

15. What is the smallest height needed to store 100,000 nodes in a binary tree?\_\_\_\_\_\_\_\_\_\_\_

16. Draw an expression tree for the prefix expression +\*123

Give its infix expression \_\_\_\_1 \* 2 + 3\_\_\_

and its postfix expression\_\_\_32 1 + \*\_\_\_\_

17. Draw an expression tree for the infix expression 2\*8+9/3

Give its prefix expression + \*2 8 /9 3\_

and its postfix expression\_\_2 8 \* 9 3 \* +\_\_

18. Draw an expression tree for

yr % 4 == 0 && (yr % 100 != 0 || yr % 400 == 0)

Questions 19-21 use recursion on ListNode linked lists. Think recursively!

9

7

2

head

3

19. **public int** countNodes(ListNode head)

{

if(head.getValue()== null)

return 0;

else

return 1 + countNodes(head.getNext());

}

20. **public void** display(ListNode head)

{

if(head.getNext() = null)

system.out.println(head);

else

system.out.println(head);

display(head.next());

}

21. **public int** findMax(ListNode head)

{

if(!head.getNext().hasNext())

return(Math.max(head.getValue, head.getNext());

else

return Math.max(head.getValue, findMax(head.getNext());

}

22. Trees are naturally recursive structures. Print the preorder expression on a tree made of TreeNodes. (What is a TreeNode? Look at its API.) Use recursion.

**public static void** preorderTraverse( TreeNode t )

{

23. Open the TreeLab shell. Complete the code.