**A+ Computer Science  
Binary Tree M/C TEST**

**Directions ::** On your answer sheet, mark the letter of the best answer to each question.

1. **What is the best case for finding an item in a binary search tree?**

|  |  |
| --- | --- |
| a. | o(n) |
| b. | o(log2n) |
| c. | o(n\*n) |
| d. | o(n/2) |
| e. | o(1) |

2. **How many children can the root node of binary tree have?**

|  |  |
| --- | --- |
| a. | 2 |
| b. | 3 |
| c. | 1 |
| d. | 0 |
| e. | 4 |

3. **What type of tree traversal is this?**

private void traverse(TreeNode tree){

if(tree != null){

traverse(tree.getLeft());

out.print(tree.getValue() + " ");

traverse(tree.getRight());

}

}

|  |  |
| --- | --- |
| a. | inorder |
| b. | preorder |
| c. | postorder |
| d. | reverse |
| e. | funorder |

4. **What type of tree traversal is this?**

private void traverse(TreeNode tree){

if (tree != null){

traverse(tree.getRight());

out.print(tree.getValue() + " ");

traverse(tree.getLeft());

}

}

|  |  |
| --- | --- |
| a. | inorder |
| b. | preorder |
| c. | postorder |
| d. | reverse |
| e. | funorder |

5. **What type of tree traversal is this?**

private void traverse(TreeNode tree)

{

if(tree != null)

{

traverse(tree.getLeft());

traverse(tree.getRight());

out.print(tree.getValue() + " ");

}

}

|  |  |
| --- | --- |
| a. | inorder |
| b. | preorder |
| c. | postorder |
| d. | reverse |
| e. | funorder |

6. **What type of tree traversal is this?**

private void traverse(TreeNode tree){

if(tree != null){

out.print(tree.getValue() + " ");

traverse(tree.getLeft());

traverse(tree.getRight());

}

}

|  |  |
| --- | --- |
| a. | inorder |
| b. | preorder |
| c. | postorder |
| d. | reverse |
| e. | funorder |

7. **A complete binary tree will**

|  |  |
| --- | --- |
| a. | have all levels full |
| b. | have all levels full that can be and all partial levels shifted to the left |
| c. | have all levels shifted to the left |
| d. | never have leaves |
| e. | be balanced |

8. **How many leaves max could a tree with 3 nodes have?**

|  |  |
| --- | --- |
| a. | 3 |
| b. | 5 |
| c. | 1 |
| d. | 4 |
| e. | 2 |

9. **Which of the following is true for a full binary tree?**

|  |  |
| --- | --- |
| a. | the bottom most level has mostly leaves |
| b. | the bottom most level is almost complete |
| c. | the bottom most level is nothing but parents |
| d. | the bottom most level has the biggest nodes |
| e. | the bottom most level has nothing but leaves |

10. **Each node in a threaded tree will have**

|  |  |
| --- | --- |
| a. | a left child and a right child only |
| b. | at least two children |
| c. | a parent and at most two children |
| d. | one child. |
| e. | one parent. |

11. **Given a complete tree with 4 levels, how many leaves could this tree have?**

I. 2

II. 6

III. 12

|  |  |
| --- | --- |
| a. | I only |
| b. | II only |
| c. | III only |
| d. | I and II only |
| e. | I and III only |

12. **How many nodes would a full tree with 5 levels have?**

|  |  |
| --- | --- |
| a. | 15 |
| b. | 16 |
| c. | 32 |
| d. | 31 |
| e. | 63 |

13. **Which of the following is true for a heap that stores the minimum priority value at the root?**

|  |  |
| --- | --- |
| a. | The root will be the item with the lowest value in the tree. |
| b. | The nodes from the root down are in sorted order by level. |
| c. | The root will be the item with the greatest value in the tree. |
| d. | The nodes from the root down are in sorted order by branch. |
| e. | none of these answers are correct. |

14. **If a complete tree has 3 levels, which of the following are possible node counts for the tree?**

|  |  |
| --- | --- |
| a. | 9 |
| b. | 5 |
| c. | 3 |
| d. | 2 |
| e. | 11 |

15. **If a tree has a height of 8, how many levels does it have?**

|  |  |
| --- | --- |
| a. | 5 |
| b. | 6 |
| c. | 7 |
| d. | 8 |
| e. | 9 |

16. **What is returned by method fun?**

private int fun(TreeNode t){

if(t==null)

return 0;

return 1 + fun(t.getRight()) + fun(t.getLeft());

}

|  |  |
| --- | --- |
| a. | the number of levels in the tree |
| b. | the number of leaves in the tree |
| c. | the number of nodes in the tree |
| d. | the sum of the binary tree’s leaves only |
| e. | the sum of the binary tree’s parent nodes only |

17. **What is returned by method fun?**

private int fun(TreeNode t){

if(t==null)

return 0;

else if(t.getRight()==null && t.getLeft()==null)

return 1;

else

return fun(t.getRight()) + fun(t.getLeft());

}

|  |  |
| --- | --- |
| a. | the number of levels in the tree |
| b. | the number of leaves in the tree |
| c. | the number of nodes in the tree |
| d. | the sum of the binary tree’s leaves only |
| e. | the sum of the binary tree’s parent nodes only |

18. **What does fun do?**

private TreeNode fun(Comparable val, TreeNode tree){

if(tree == null)

tree = new TreeNode(val, null, null);

else if(val.compareTo(tree.getValue()) < 0)

tree.setLeft(fun(val, tree.getLeft()));

else if(val.compareTo(tree.getValue()) > 0)

tree.setRight(fun(val, tree.getRight()));

return tree;

}

|  |  |
| --- | --- |
| a. | counts the levels |
| b. | counts the leaves |
| c. | determines the size |
| d. | determines the height |
| e. | adds nodes |

19. **Why do you have to have return tree at the bottom of method fun?**

private TreeNode fun(Comparable val, TreeNode tree){

if(tree == null)

tree = new TreeNode(val, null, null);

else if(val.compareTo(tree.getValue()) < 0)

tree.setLeft(fun(val, tree.getLeft()));

else if(val.compareTo(tree.getValue()) > 0)

tree.setRight(fun(val, tree.getRight()));

return tree;

}

|  |  |
| --- | --- |
| a. | return methods save memory |
| b. | because JAVA passes all parameters by value |
| c. | because JAVA passes all parameters by reference |
| d. | because JAVA passes some parameters by value |
| e. | C && D only |

20. **What is the output of the following code?**

TreeNode w = new TreeNode(90,

new TreeNode(100,

new TreeNode(13,null,null),

new TreeNode(45,null,null)),

new TreeNode(200,

new TreeNode(99,null,null),

new TreeNode(13,null,null)));

out.println(w.getLeft().getLeft().getValue());

|  |  |
| --- | --- |
| a. | 45 |
| b. | 200 |
| c. | 100 |
| d. | 99 |
| e. | 13 |

21. **If t is referring to the root, what is returned by** f **?**

private String f(TreeNode t){

if(t==null)

return "";

return "" + f(t.getLeft())+ " " +

t.getValue() + " " + f(t.getRight());

}

|  |  |
| --- | --- |
| a. | 0 |
| b. | the number of nodes in the tree |
| c. | the numeric sum of all nodes |
| d. | the sum of all node values that are leaves |
| e. | the sum of all node values as one big string |

22. **If you insert the following numbers in the order listed into a binary search tree, which of the following represents the tree’s in order traversal output?**

**Items inserted in the following order :**  200 150 225 75 210 250

|  |  |
| --- | --- |
| a. | 250 225 210 200 150 75 |
| b. | 75 150 210 250 225 200 |
| c. | 75 150 200 210 225 250 |
| d. | 200 150 75 225 210 250 |
| e. | 200 150 225 210 250 75 |

23. **If you insert the following numbers in the order listed into a binary search tree, which of the following represents the tree’s pre order traversal output?**

**Items inserted in the following order:** 200 150 225 75 210 250

|  |  |
| --- | --- |
| a. | 250 225 210 200 150 75 |
| b. | 75 150 210 250 225 200 |
| c. | 75 150 200 210 225 250 |
| d. | 200 150 75 225 210 250 |
| e. | 200 150 225 210 250 75 |

24. **If you insert the following numbers in the order listed into a binary search tree, which of the following represents the tree’s post order traversal output?**

**Items inserted in the following order :** 200 150 225 75 210 250

|  |  |
| --- | --- |
| a. | 250 225 210 200 150 75 |
| b. | 75 150 210 250 225 200 |
| c. | 75 150 200 210 225 250 |
| d. | 200 150 75 225 210 250 |
| e. | 200 150 225 210 250 75 |

25. **If you insert the following numbers in the order listed into a binary search tree, which of the following represents the tree’s reverse order traversal output?**

**Items inserted in the following order:** 200 150 225 75 210 250

|  |  |
| --- | --- |
| a. | 250 225 210 200 150 75 |
| b. | 75 150 210 250 225 200 |
| c. | 75 150 200 210 225 250 |
| d. | 200 150 75 225 210 250 |
| e. | 200 150 225 210 250 75 |

26. **Assuming t does not equal null, which of the following blocks of code would fill** // blank 1  **so that method fun would return the minimum value in the tree?**

private Comparable fun(TreeNode t)

{

// blank 1

}

I.

while(t!=null)

{

t=t.getLeft();

}

return t.getValue();

II.

while(t!=null&&t.getLeft()!=null)

{

t=t.getLeft();

}

return t.getValue();

III.

while(t!=null)

{

t=t.getLeft();

}

return t;

|  |  |
| --- | --- |
| a. | I |
| b. | I and II |
| c. | III only |
| d. | II and III |
| e. | II only |

27. **Assuming t does not equal null, which of the following blocks of code would fill** // blank 1  **so that method fun would return the maximum value in the tree?**

private Comparable fun(TreeNode t)

{

// blank 1

}

I.

while(t != null)

{

t=t.getRight();

}

return t;

II.

while(t!=null && t.getRight()!=null)

{

t=t.getRight();

}

return t.getValue();

III.

while(t != null)

{

t=t.getLeft();

}

return t.getValue();

|  |  |
| --- | --- |
| a. | I |
| b. | I and II |
| c. | II only |
| d. | II and III |
| e. | III only |

28. **What is wrong with the traversal at right?**

private void traverse(TreeNode t)

{

traverse(t.getLeft());

traverse(t.getRight());

out.print(t.getValue() + " ");

}

|  |  |
| --- | --- |
| a. | there should be no print |
| b. | the print should be first |
| c. | the print should be in between the recursive calls |
| d. | there is no check for null |
| e. | the method should be public |

29. **How many leaves max could a complete tree with 6 nodes have?**

|  |  |
| --- | --- |
| a. | 3 |
| b. | 4 |
| c. | 5 |
| d. | 6 |
| e. | 7 |

30. **Assuming tree does not equal null, which of the following blocks of code would fill** //blank 1  **so that method go would return a String containing all nodes in pre order?**

private String go(TreeNode tree)

{

//blank 1

}

I.

if(tree==null)

return "";

return "" + go(tree.getLeft())+ " " +

tree.getValue() + " " + go(tree.getRight());

II.

if(tree==null)

return "";

return "" + go(tree.getLeft()) +

go(tree.getRight())+ tree.getValue() + " ";

III.

if(tree==null)

return "";

return "" + tree.getValue() + " " +

go(tree.getLeft())+ go(tree.getRight());

|  |  |
| --- | --- |
| a. | I only |
| b. | I and II only |
| c. | II only |
| d. | II and III only |
| e. | III only |

31. **What is the binary search tree property?**

|  |  |
| --- | --- |
| a. | All right child nodes have greater values than the parent node |
| b. | All left child nodes have lesser values than the parent node |
| c. | All right child nodes have lesser values than the parent node |
| d. | All left child nodes have greater values than the parent node |
| e. | A & B |

32. **Which tree traversal algorithim prints all the values of a binary search tree out in ascending order?**

|  |  |
| --- | --- |
| a. | Inorder |
| b. | Reverse order |
| c. | Preorder |
| d. | Postorder |
| e. | None |

33. **Which tree traversal algorithim prints all the values of a binary search tree out in descending order?**

|  |  |
| --- | --- |
| a. | Inorder |
| b. | Reverse order |
| c. | Preorder |
| d. | Postorder |
| e. | None |

34. **A heap is used in which sort?**

|  |  |
| --- | --- |
| a. | Quicksort |
| b. | Treesort |
| c. | Heapsort |
| d. | Mergesort |
| e. | Bubblesort |

35. **Which of the following are self balancing binary search trees?**

|  |  |
| --- | --- |
| a. | Trie |
| b. | Red-Black Tree |
| c. | AVL Tree |
| d. | A & B |
| e. | B & C |

36. **What is a tree rotation?**

|  |  |
| --- | --- |
| a. | When the nodes are inserted into the tree in such a way that it’s unbalanced |
| b. | When all the nodes are inserted into the tree on one side of the root |
| c. | An operation that swaps the left and right children |
| d. | An operation that changes the tree structure but retains the binary search tree property |
| e. | An operation that removes a tree node and adds a new node |

37. **Which Java class is an example of a heap?**

|  |  |
| --- | --- |
| a. | ArrayList |
| b. | LinkedList |
| c. | TreeMap |
| d. | TreeSet |
| e. | PriorityQueue |

38. **What is the run time of inserting into a heap?**

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N2) |
| d. | O(Log2N) |
| e. | O(N!) |

39. **What is the run time of removing from a heap?**

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N2) |
| d. | O(Log2N) |
| e. | O(N!) |

40. **What is the run time of building a heap?**

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N2) |
| d. | O(Log2N) |
| e. | O(N \* Log2N) |