Name:- Chad Huntebrinker

Answer the following Questions

**Q1: [6 points]** Draw an expression tree corresponding to each of the following:

a. In order traversal is x / (y + 3) \* b / c

b. Post order traversal is  **a b - c d e f g + - + \***

c. Pre order traversal is  **\* + - a – x y / c d**

|  |
| --- |
| Use this space to answer the question  In order:    Post Order:    Pre Order: |

**Q2: [4 points]** build a BST using the following data items.

a. happy, depressed, manic, sad, ecstatic, crazy

b. 15, 20, 45, 70, 10, 33, 25, 100

|  |
| --- |
| Use this space to answer the question  a)    b) |

**Q3: [5 points]** Write a recursive member method to find out if the binary tree is a full tree or not.

|  |
| --- |
| Use this space to answer the question  bool BinaryTree::returnStatus(bool treeFull) {  isFullTree(root, treeFull);  return treeFull;  }  bool BinaryTree::isFullTree(Node\* r, bool& treeFull) {  if (r != NULL)  //if left sub-tree and right sub-tree are both NULL, then this is a leaf and is still a full tree  if (r->left == NULL && r->right == NULL) {  treeFull = true;  }  else {  //if only the left sub-tree or only the right sub-tree is NULL, than this Node has  //only one child and means the tree is not a full tree.  if (r->left == NULL || r->right == NULL) {  return treeFull = false;  }  else {  //if it still hasn't been proven to not be a full tree  if (treeFull != false)  treeFull = isFullTree(r->left, treeFull);  //if it still hasn't been proven to not be a full tree  if (treeFull != false)  treeFull = isFullTree(r->right, treeFull);  }  }  return treeFull;  } |

**Q4: [4 points]** Draw the BST that results when you insert items with keys:

2 4 6 8 10 13 15 17 20 27 33

in that order, into an initially empty tree.

How accurate is to state that the Big O notation for the search operation is log(n) using the BST you created using the data set listed above. Discuss that.

|  |
| --- |
| Use this space to answer the question    I would say that it is not accurate to say this has a Big O of log n because of how the BST is set up. It basically is an array or vector. Because each numeric value is greater than the next, you put it as the right child. What you end up with is a BST that is linear (because, in order to get to 10, you have to go through 8, 6, 4, and 2). In conclusion, you get a Big O of n. |

**Q5: [6 points]**

1. build a mini heap using the following data set: 15, 20, 45, 70, 10, 33, 25, 100

|  |
| --- |
| Use this space to answer the question |

1. Insert the element 5 into this heap and show how the heap will maintain the properties of mini heap.

|  |
| --- |
| Use this space to answer the question    5 is inserted at the bottom. It is then compared with 45 and it is found that 5 < 45. So 5 is swapped with 45 and we are given:    5 < 33, so we swap those two numbers and are left with:    5 < 10, so we swap that with the root and are left with:    And this is our final answer. |

1. After that, remove the root and show how the heap will maintain the properties of mini heap.

|  |
| --- |
| Use this space to answer the question  We are given:    And we must remove the root (which is 5). So first, we delete 5 and have 45 be the new root. That looks like:    Next, we compare 45 with the left and right child. Both children are less than 45 but 10 < 15. So, we swap 45 and 10.    Now, we compare 45 with the left and right child. Only one child is less than 45 (the left child 33), so we swap them.    100 is not less than 45, so 45 is in the correct position. We are done. |