Name _____ Section Number _____

CS210 Exam #3 *** PLEASE TURN OFF ALL CELL PHONES*** Practice
All Sections Bob Wilson

OPEN BOOK / OPEN NOTES: You will have all 90 minutes until the start of the next class period. Spend only about one minute per point on each question to complete the exam on time.

1. Sorting and Searching (20 Points)
a. Explain a situation where we may prefer a linear search instead of a binary search. In the identified situation, give a quantitative answer based on big-O not just a qualitative answer.

b. Show the specific changes made in each pass of a Bubble Sort on the following data:

90 8 7 56 123 1

2. Tree and Heap Nomenclature (30 Points)
a. Draw a full binary tree with 7 nodes labeled A-G so that during a postorder traversal the nodes will be visited in the order A, B, C, D, E, F, and G. The letters for the nodes represent their value according to their compareTo method.

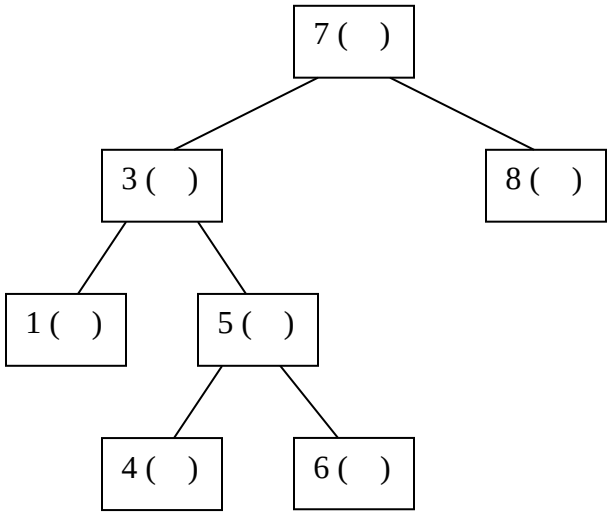
For the tree that you drew in part a:

b. The node C is located at what level? _____

c. What is the height of the tree? _____

d. Does the tree meet the requirements for being called a heap? Yes or No: _____
Explain your answer here.

3. Tree Balance Detection and Balancing Operations (20 Points)
a. Fill in the AVL Balance Factors in the following binary search tree:



b. Which rotation do the balance factors indicate you need to make and around which node?

c. Show the state of the tree including the balance factors after performing the indicated rotation.

d. Is any rotation needed now? Yes ____ No _____
Explain your answer.

4. Heap Implemented in an Array (30 Points)

The following MaxHeap class for String objects (not generic for simplicity) is provided with its given constructor, size, remove, and main methods. It is implemented in an array using the computational strategy for a binary tree and uses a comparator to compare the String objects. You need to write the code for the add method. Note: You may assume the internal array is large enough for any add operations.

```
import java.util.Comparator;

public class MaxHeap
{
  private Comparator<String> cmp; // Comparator for String objects
  private String [] heapArray;    // array for heap of Strings
  private final int ENOUGH = 10;  // large enough size for heapArray
  private final int ROOT = 0;     // index of root node is zero
                                  // computational strategy:
                                  // child indices are 2*p+1 and 2*(p+1)
                                  // all heap array elements will
                                  // automatically be contiguous from zero
  private int next;               // index of next node (the one immediately
                                  // following the "last" node)

  public MaxHeap (Comparator<String> cmp)
  {
    this.cmp = cmp;
    heapArray = new String[ENOUGH];
    next = ROOT;                  // initially the next node to be filled
  }                               // is the root node

  public int size()
  {
    return next;
  }

  public void add(String key)    // Hint: This is a lot easier than remove
  {









  }
```

```
  public String remove()    // You may tear this page off for reference
  {
    if (size() == 0)
      return null;

    // pick up and save reference to root which is max
    String s = heapArray[ROOT];

    // and swap last with root
    heapArray[ROOT] = heapArray[--next];
    heapArray[next] = null;                // clear alias

    // now heapify on remove
    int parent = ROOT;
    boolean done = false;
    while (!done) {
      done = true;
      if (2 * (parent + 1) < next) {       // parent still has two children
         int largestChild =
            cmp.compare(heapArray[2*parent+1],heapArray[2*(parent+1)]) > 0 ?
                              2*parent+1      :     2*(parent+1);
         if (cmp.compare(heapArray[parent], heapArray[largestChild]) < 0) {
           swap(parent, largestChild);
           parent = largestChild;
           done = false;
         }                                 // else now we are done
      }
      else if (2 * parent + 1 < next) {    // parent is in next to bottom level
                                           // right child is null but left is not
        int largestChild = 2 * parent + 1;
        if (cmp.compare(heapArray[parent], heapArray[largestChild]) < 0)
          swap(parent, largestChild);
      }                                    // and now we are done
      // else no children so we must be done
    }
    return s;
  }

  // helper method to swap two elements in the heapArray
  private void swap(int node1, int node2)
  {
    String temp = heapArray[node1];
    heapArray[node1] = heapArray[node2];
    heapArray[node2] = temp;
  }

  public static void main(String [] args)
  {
    MaxHeap myMaxHeap = new MaxHeap(new StringComparator());
    myMaxHeap.add("One");
    . . .
    System.out.println("Removing and printing everything");
    while (myMaxHeap.size() > 0)
      System.out.println(myMaxHeap.remove());
  }
}
```

Answer Key:

1. Sorting and Searching

a. Explain a situation where we may prefer a linear search instead of a binary search. In the identified situation, give a quantitative answer based on big-O not just a qualitative answer.
If we are updating the data very frequently and only searching infrequently, we might prefer to avoid the overhead of sorting the data after each update. Each sort is probably O(n log n) and linear search is O(n). If on average we update and resort the data more frequently than log n times per search, we would have a net loss in performance.

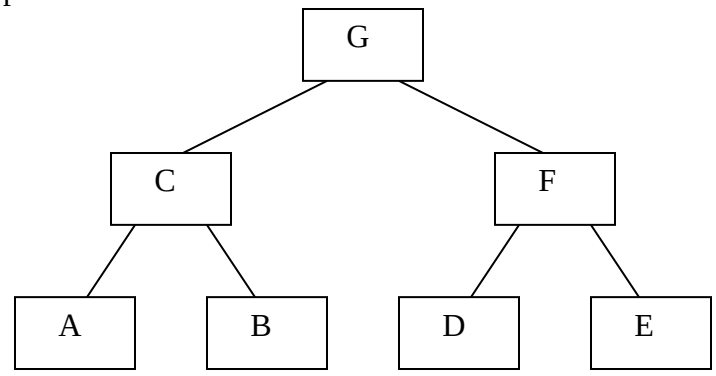
b. Show the specific changes made in each pass of a Bubble Sort on the following data:

90 8 7 56 123 1

Answer:

8	7	56	90	1	123
7	8	56	1	90	123
7	8	1	56	90	123
7	1	8	56	90	123
1	7	8	56	90	123

2. Tree and Heap Nomenclature



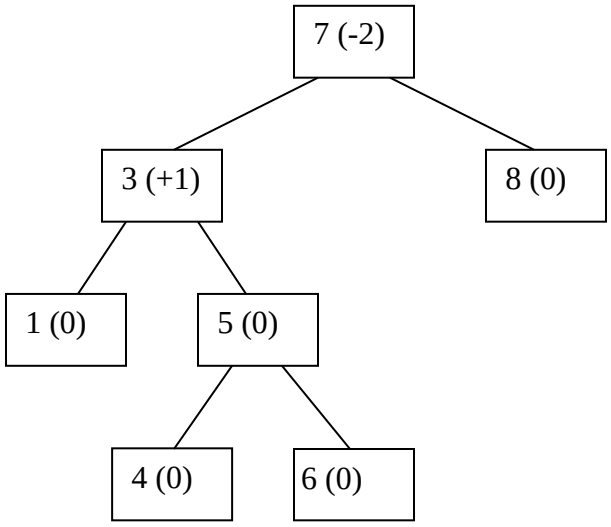
For the tree that you drew in part a:

b. The node C is located at what level? ____1_____

c. What is the height of the tree? _____2_____

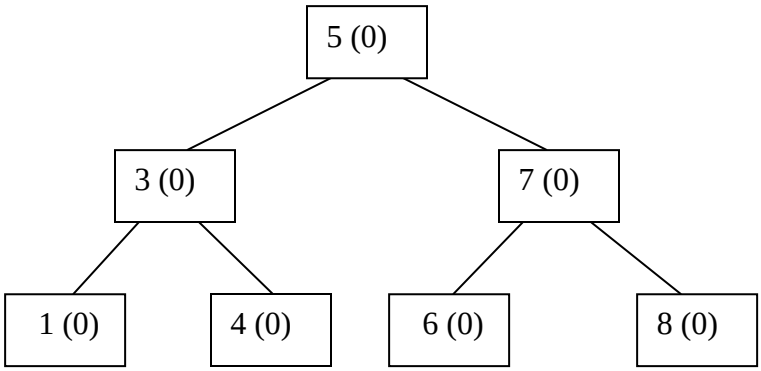
d. If the letters for the nodes represent their value according to their compareTo method, does the tree meet the requirements for being called a heap? Yes or No: ___YES_____
Explain your answer here.
It meets the requirements for being called a max-heap.
C is greater than A and B. F is greater than D and E. G is greater than C and F.

3. Tree Balance Detection and Balancing Operations
a. Fill in the AVL Balance Factors in the following binary search tree:



b. Which rotation do the balance factors indicate you need to make and around which node?
A Left-Right rotation around node 7 is indicated.

c. Show the state of the tree including the balance factors after performing the indicated rotation.



d. There is now no rotation indicated. All Balance Factors are within +1 and -1 range.

4. MaxHeap add method

```
public void add(String key)
{
  // add new node at next position in array
  heapArray[next] = key;

  // and heapify on add
  int child = next++;
  boolean done = false;
  while (child != ROOT && !done) {
    done = true;
    int parent = (child - 1) / 2;
    if (cmp.compare(heapArray[parent], heapArray[child]) < 0) {
      swap(parent, child);
      child = parent;
      done = false;
    }
  }
}
```