REVIEW FOR CIS 22C (Java) MIDTERM

Describe what a stack is and its operations.

Describe what a queue is and its operations.

Describe the difference between array implementations and linked list implementations of stacks (the stack functions in StackInterface of the textbook)

Write a Java function that transfers items from a stack to a queue, BUT keeping the original stack insertion order (first one on the stack will be first on in the queue, etc.)

Determine the worst case Big-Oh of the following function:

```
public static <E> void insertionSort(E theArray[], int n)
{
   for (int unsorted = 1; unsorted < n; unsorted++)
   {
      E nextItem = theArray[unsorted];
      int loc = unsorted;
      while ((loc > 0) && (theArray[loc - 1] > nextItem))
      {
         theArray[loc] = theArray[loc - 1];
         loc--;
      }  // end while

      theArray[loc] = nextItem; // Insert nextItem into sorted region
   }  // end for
}  // end insertionSort
```

Answer the questions below about the following function:

```
public static int gcd(int a, int b)
{
   if (a % b == 0)  // line 1
      return b;     // line 2
   else                // line3
      return gcd(b, a % b); // line4
}  // end gcd
```

- what is the base case of the function?
- what is the recursive statement?
- could you write the above iteratively (using a loop and no recursion)?

Using the following function from the SortedList (concrete LList class in which the items are inserted in order), answer the questions below. (NOTE: THIS WON'T WORK FOR HW#3)

```
    private Node getPreviousNode(T anEntry)
    {
        Node prev, curr;

        prev = null;
        curr = firstNode;
        while( curr != null && curr.getData().compareTo(anEntry) < 0 )
        {
            prev = curr;
            curr = curr.getNextNode();
        }
        return prev;
    }
```

- if the list is empty, what will happen?
- if the "anEntry" item equals the firstNode's item, what is returned?
- if the "anEntry" item equals the 2nd node's item, what is returned?
- if the "anEntry" item > than all of the list's items, describe what is returned

---

What is a binary tree?

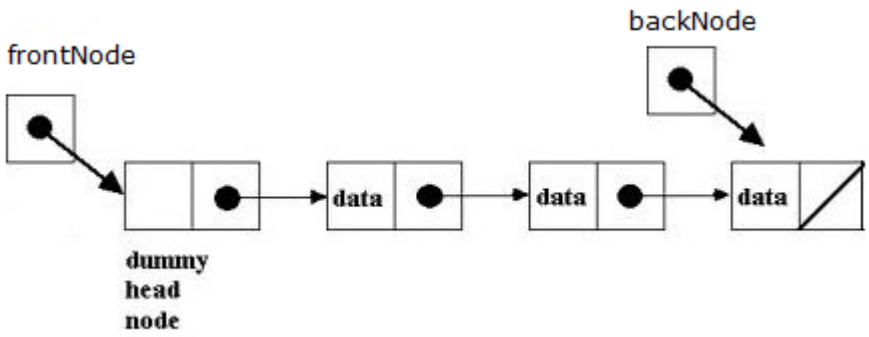
Know the terminology: height of a subtree, depth of a tree (in the textbook, it's like the level-1), leaf nodes

---

Using the ArrayQueue class from Lesson 3, add a new INSTANCE method to the ArrayQueue that returns (in a return statement) a new T[], and has all the data in the ArrayQueue in the array (just assign, don't worry about cloning here). Be sure to copy starting with front as element 0 (DON'T COPY FROM ELEMENT 0 OF THE items MEMBER ARRAY)! DON'T CHANGE ANYTHING IN THE ArrayQueue so NEVER call dequeue().

```
//YOU WRITE THIS METHOD:***********************************************
    public T[] toArray() // hint: look at the constructor for this class
    {
```

Change the LinkedQueue<T> so it has a "dummy" frontNode node as shown in the picture below, but the backNode is NOT a dummy and will ONLY refer to the dummy head node if it's empty. (fill in the BLANKS***, see p.1 for most of the original class)




```
public class LinkedQueueDummy<T> implements QueueInterface<T>
{
      private Node frontNode; // References node at front of queue
      private Node backNode;  // References node at back of queue
      private int count = 0;

    public LinkedQueueDummy()
    {
        frontNode = backNode = new Node(null); // MY CHANGE!!!
    } // end default constructor

    public boolean enqueue(T newEntry)
    {
        Node newNode = new Node(newEntry);
//  CROSS OUT WHAT LINES SHOULD BE DELETED HERE:**********************
        if( count == 0 )
            frontNode = newNode;
        else
            backNode.setNextNode(newNode);
        backNode = newNode;

        ++count;
        return true;
    } // end enqueue

    public T peekFront()
    {
        if (isEmpty())
            return null;
        else
                return ____________________________________;//**********************
    } // end getFront

    public T dequeue()
    {
       T front = peekFront();
```

```
         if( count > 0 )
         {
       Node nodeToRemove = _____________________________________;//*************
//****DON'T CHANGE frontNode, only its next reference below!

       ____________________________________________ //*******************
            if( count == 1 )
             backNode = _____________________________;//************************
           --count;
         }

         return front;
    } // end dequeue

// REST OF THE CLASS IS THE SAME AS THE REGULAR LinkedQueue
```

Change the following code for a LList so it's a SortedList (the items are also in sorted order). Assume T implements Comparable (like in HW#3)

```
    public void add(T newEntry)    // NOW add so it's in order
    {
       Node newNode = new Node(newEntry);

       if (isEmpty())
          firstNode = newNode;
       else                                // Add to end of non-empty list
       {
          Node lastNode = getNodeAt(numberOfEntries);//REPLACE THIS
             // WITH SEVERAL LINES, but don't traverse more than once!


          lastNode.setNextNode(newNode); // Make last node reference new
node
       } // end if

       numberOfEntries++;
    }  // end add
```

MORE REVIEW GIVEN IN THE CLASS NOTES!!! (esp. doubly linked lists!)