Problem 1:
True or false: Lady Lovelace was romantically linked to Charles Babbage.

Problem 2:

It is commonly stated in the literature that a hashtable using open addressing has O(1) runtime for the get and put operations. Under what conditions is this statement false? What is the O() runtime of the get and put method in these conditions? What can you do to ensure that such conditions do not exist; thereby ensuring O(1) runtimes for hashtables?

Problem 3: Queues and Classes

You are given a class BriefQueue that correctly implements the offer, peek and poll methods of Queue (These methods are documented below.). That class is properly encapsulated. These are the only public methods. You do not have access to the source code of BriefQueue. Here is documentation for BriefQueue. "A" in the documentation is a generic class identifier.

```
/**
 * Return the first element in the queue, or null iff the queue is empty
 * @return the first element
 */
public A peek()
/**
 * Remove the first element from the queue and return it.
 * @return The first element of the queue, or null is the queue is empty
 */
public A poll()
/**
 * Add an element of the end of the queue
 * @param ana the element to be added
 */
public void offer(A ana)
```

Write a class, FullQueue, that extends BriefQueue and which provides implementations of empty(), count(), and clear(). Do not reimplement offer, peek, and poll. Do not, even temporarily, store items twice. At the completion of the empty and count methods the contents of the queue must be identical to the contents prior to their execution. The queue should be empty at the completion of clear(). (You may use iteration in these functions. You may use any data structure you think would be helpful.) Here is documentation for these functions

```
/**
 * Return true if the queue has no elements
 * @return true if the queue has no elements, false otherwise
 */
public boolean empty()
/**
 * Remove all elements from the queue.  After executing this
 * method empty() will return true.
 */
public void clear()

/**
 * Return the number of items in the queue
 * @return the number of items in the queue
 */
public int count()
```

Classes and Comparable

Write a class called OddString that is an extension of the String class. (Technically this is not allowed because String declared to be final. For this test, assume it is legal to extend String.) It should be identical to the standard String class except that the comparison of strings should be based upon only the letters in the odd numbered positions.

So, for instance AsDfG would be equal to QsWfE because there are two letters in the odd positions in both strings (s and f) and they are the same. Similarly, WsDFg would be less than ZtZFg.

Provide a complete implementation of this class.

To help you, here is an implementation of the equals method for your class.

Hint, you only need to write one method.

```
public boolean equals(Object ob) {
        if (ob instanceof OddString)
            return this.compareTo(ob) == 0;
        return false;
    }
```

Complexity

THIS QUESTION DOES NOT ASK YOU TO WRITE CODE.

You are given an ArrayList (call it U) of items that implement the comparable interface. U is in no particular order. You are asked to do the following:

1. Create a new empty ArrayList (call it S)
2. find the smallest item in U, call it M
3. remove M from U
4. add M to S
5. Repeat steps 2, 3 and 4 until U is empty

What is the computation complexity of each of steps 1-4 (in terms of the the number of items in U)?

What is the computational complexity of the entire process?

Recursion

```
/**
     * Return an Array List containing the factors of the
provided number.
     * For instance, for 6, the returned ArrayList should
contain 1,2,3,6.
     * For 12, the ArrayList would be 1,2,3,4,6,12.
     * The order of the factors is not significant.
     * @param num the number whose factors are to be determined
     * @return An ArrayList containing the factors
     */
```

```
public ArrayList<Integer> factors(int number);
```

Write a recursive implementation of this function. You may not use loops of any form. You may use a private recursive function if you determine it would be helpful.

Recursion and Code Reading

Consider the following function `cd` and its recursive helper `cdUtil`

```
public class M2_7_CD {

    public int cd(long val) {
        if (val)<0) return -1;
        if (val==0) return 0;
        return cdUtil(val, 1, 0, 5);
    }

    private int cdUtil(long v, long vv, int c, int m) {
        if (m == 2) {
            c++;
            m = 5;
        } else {
            m = 2;
        }
        if (vv > v)
            return c;
        return cdUtil(v, vv * m, c, m);
    }
}
```

PART 1: Show the entire set of function calls and return values when cd is called with the argument 64 and 122. Be sure to indicate the final return value.

PART 2: In 1 or 2 sentences state what the cd function does. NOT how, but what. If your answer, in any way, violates encapsulation, then it is not an acceptable answer. This statement should be usable as the start of the function-level comment about the cd function. (Do write the comment; just write the sentences.)

ANSWERS

Problem 1: (1 point)

True or false: Lady Lovelace was romantically linked to Charles Babbage.

FALSE

Problem 2: (7 points)

It is commonly stated in the literature that a hashtable using open addressing has O(1) runtime for the get and put operations. This is false. Under what conditions is this statement false? What is the O() runtime of the get and put method in these conditions?

In case where the runtime is not O(1) the runtime is O(n).

For put this can happen in two ways. First, if the load factor is allowed to get too high, then the put can spend O(n) time simply finding an open spot into which to put a new value (or to find a value to be updated). Second, during rehashing, This is done to prevent the load factor from getting high, but rehashing is necessarily O(n). For get, time can be O(n) when th load factor is high, which can necessitate goring through a large percentage of the hashtable to determine if an key in in the table.

Problem 3: (20 points) Queues and Classes

You are given a class BriefQueue that correctly implements the offer, peek and poll methods of Queue (These methods are documented below.). That class is properly encapsulated. These are the only public methods. You do not have access to the source code of BriefQueue. Here is documentation for BriefQueue. “A” in the documentation is a generic class identifier.

```
/**
 * Return the first element in the queue, or null iff the queue is empty
 * @return the first element
 */
public A peek()
/**
 * Remove the first element from the queue and return it.
 * @return The first element of the queue, or null is the queue is empty
 */
public A poll()
/**
 * Add an element of the end of the queue
 * @param ana the element to be added
 */
public void offer(A ana)
```

Write a class, FullQueue, that extends BriefQueue and which provides implementations of empty(), count(), and clear(). Do not reimplement offer, peek, and poll. Do not, even temporarily, store items twice. At the completion of the empty and count methods the contents of the queue must be identical to the contents prior to their execution. The queue should be empty at the completion of clear(). (You may use iteration in these functions. You may use any data structure you think would be helpful.) Here is documentation for these functions

```
/**
 * Return true if the queue has no elements
 * @return true if the queue has no elements, false otherwise
 */
public boolean empty()
/**
 * Remove all elements from the queue.  After executing this
 * method empty() will return true.
 */
public void clear()

/**
 * Return the number of items in the queue
 * @return the number of items in the queue
 */
public int count()
```

```java
public class M2_4_FQ<B> extends M2_4_BQ<B> {
    public M2_4_FQ() {
        super();
    }

    /**
     * Return true if the queue has no elements
     * @return true if the queue has no elements, false otherwise
     */
    public boolean empty() {
        return peek() == null;
    }

    /**
     * Remove all elements from the queue.  After executing this method
     * empty() will return true.
     */
    public void clear() {
        while (peek() != null) {
            poll();
        }
    }

    /**
     * Return the number of items in the queue
     * @return the number of items in the queue
     */
    public int count() {
        M2_4_BQ<B> tmp = new M2_4_BQ<>();
        int cc = 0;
        while (peek() != null) {
            cc++;
            tmp.offer(this.poll());
        }
        while (tmp.peek() != null) {
            this.offer(tmp.poll());
        }
        return cc;
    }

    public static void main(String[] args) {
        M2_4_FQ<Integer> aaa = new M2_4_FQ<>();
        for (int i=0; i<7; i++)
            aaa.offer(i);
        System.out.println(aaa);
        System.out.println(aaa.count());
        System.out.println(aaa);
        System.out.println(aaa.empty());
        aaa.clear();
        System.out.println(aaa);
        System.out.println(aaa.empty());


    }
}
```

Problem 4: (20 points) Classes and Comparable

Write a class called OddString that is an extension of the String class. (Technically this is not allowed because String declared to be final. For this test, assume it is legal to extend String.) It should be identical to the standard String class except that the comparison of strings should be based upon only the letters in the odd numbered positions.

So, for instance AsDfG would be equal to QsWfE because there are two letters in the odd positions in both strings (s and f) and they are the same. Similarly, WsDFg would be less than ZtZFg.

Provide a complete implementation of this class.

To help you, here is an implementation of the equals method for your class.

Hint, you only need to write one method.

```
public boolean equals(Object ob) {
        if (ob instanceof OddString)
            return this.compareTo(ob) == 0;
        return false;
    }
```

Two solutions here is my first one

```
public int compareTo(Object ob) {
        if (ob instanceof M2_5_OS) {
            String obs = ((M2_5_OS) ob).getString();
            String ss = this.getString();
            int locc = 1;
            while (true) {
                if (locc > ss.length() && locc > obs.length())
                    return 0;
                if (locc > ss.length())
                    return -1;
                if (locc > obs.length())
                    return 1;
                int diff = ss.charAt(locc) - obs.charAt(locc);
                if (diff != 0)
                    return diff;
                locc += 2;
            }
        } else
```

```
            return -1;
    }
```

Now my second one. Trickier.

```
private String getOddString() {
        String ss = this.getString();
        String st = "";
        for (int i = 1; i < ss.length(); i = i + 2) {
            st = st + ss.charAt(i);
        }
        return st;
    }

    @Override
    public int compareTo(Object ob) {
        if (ob instanceof M2_5_OS) {
            String obs = ((M2_5_OS) ob).getOddString();
            String obs2 = getOddString();
            return obs2.compareTo(obs);
        }
        return -1;
    }
```

Problem 4: (12 points) Complexity

THIS QUESTION DOES NOT ASK YOU TO WRITE CODE.

You are given an ArrayList (call it U) of items that implement the comparable interface. U is in no particular order. You are asked to do the following:

1. Create a new empty ArrayList (call it S)
2. find the smallest item in U, call it M
3. remove M from U
4. add M to S
5. Repeat steps 2, 3 and 4 until U is empty

What is the computation complexity of each of steps 1-4 (in terms of the the number of items in U)?

What is the computational complexity of the entire process?

Explain, briefly, each of your computational complexity answers

1. *O(1) — creating an empty ArrayList always takes the same amount of time*
2. *O(n) — to find the smallest item, you need to look at every item*
3. *O(n) — when you remove an item from an array list, you need to shift all of the items in the array list down to close the gap created by the removal.*
4. *O(n) — when adding this to ArrayLists, the List may need to increase its capacity which takes O(n) time*
5. *The repeat will occur N times. So we have n* (n+ n + n) = O(n^2).*

Problem 5: (20 points)Recursion

```
/**
     * Return an Array List containing the factors of the
provided number.
     * For instance, for 6, the returned ArrayList should
contain 1,2,3,6.
     * For 12, the ArrayList would be 1,2,3,4,6,12.
     * The order of the factors is not significant.
     * @param num the number whose factors are to be determined
     * @return An ArrayList containing the factors
     */
```

`public ArrayList<Integer> factors(int number);`

Write a recursive implementation of this function. You may not use loops of any form. You may use a private recursive function if you determine it would be helpful.

```
public class Primmer {
    private ArrayList<Integer> factorUtil(int num, ArrayList<Integer> fff,
int curr) {
        if (curr>num)
            return fff;
        if (num % curr == 0)
            fff.add(curr);
        return factorUtil(num, fff, curr + 1);
    }

    /**
     * Return an Array List containing the factors of the provided number.
     * For instance, for 6, the returned ArrayList should contain 1,2,3,6.
     * For 12, the ArrayList would be 1,2,3,4,6,12.
     * @param num the number whose factors are to be determined
     * @return An ArrayList containing the factors
     */
    public ArrayList<Integer> factors(int num) {
        return factorUtil(num, new ArrayList<Integer>(), 1);
    }

    public static void main(String[] args) {
        Primmer p = new Primmer();
        System.out.println(p.factors(128));
    }
}
```

Recursion and Code Reading

Consider the following function `cd` and its recursive helper `cdUtil`

```
public class M2_7_CD {

    public int cd(long val) {
        if (val)<0) return -1;
        if (val==0) return 0;
        return cdUtil(val, 1, 0, 5);
    }

    private int cdUtil(long v, long vv, int c, int m) {
        if (m == 2) {
            c++;
            m = 5;
        } else {
            m = 2;
        }
        if (vv > v)
            return c;
        return cdUtil(v, vv * m, c, m);
    }
}
```

PART 1: Show the entire set of function calls and return values when cd is called with the argument 64 and 122. Be sure to indicate the final return value.

PART 2: In 1 or 2 sentences state what the cd function does. NOT how, but what. If your answer, in any way, violates encapsulation, then it is not an acceptable answer. This statement should be usable as the start of the function-level comment about the cd function. (Do write the comment; just write the sentences.)

cd(64)

cdUtil(64,1,0,5)

cdUtil(64,5,0,2)

cdUtil(64,10,1,5)

```
cdUtil(64,50,1,2)

  cdUtil(64, 100, 2, 5)

        return 2;
```

Not showing the other, it follows exactly.

“cd” is short for “Count Digits” which is exactly what this function does. There are several other acceptable answers. For instance, something about integer value of the base 10 log would be OK as well.