Points off	1	2	3	4	5		Total off	Net Score

CS 314 – Midterm 1 – Fall 2011

Your Name_____

Your UTEID _____

Circle yours TA's name: Swati Yuanzhong

Instructions:

1. There are **5** questions on this test.
2. You have 2 hours to complete the test.
3. You may not use a calculator or any other electronic devices while taking the test.
4. When writing a method assume the preconditions of the method are met.
5. When writing a method you may add helper methods if you wish.
6. When answering coding questions ensure you follow the restrictions of the question.
7. When you complete the test show the proctor your UTID and give them the test and any scratch paper. Please leave the room quietly.

1. (2 points each, 30 points total) Short answer. Place you answers on the attached answer sheet.
   a. If a question contains a syntax error or other compile error, answer "Compile error".
   b. If a question would result in a runtime error or exception answer "Runtime error".
   c. If a question results in an infinite loop answer "Infinite loop".
   d. Recall when asked for Big O your answer should be the most restrictive correct Big O function. For example Selection Sort has an average case Big O of $O(N^2)$, but per the formal definition of Big O it is correct to say Selection Sort also has a Big O of $O(N^3)$ or $O(N^4)$. I want the most restrictive, correct Big O function. (Closest without going under.)

A. What is the T(N) for method `a`? Recall, T(N) is the function that represents the *actual* number of executable statements for a function or algorithm. N = `listA.length = listB.length`

```
// pre: listA.length == listB.length
public int a(int[] listA, int[] listB) {
     int total = 0;
     for(int i = 0; i < listA.length; i++)
          for(int j = 0; j < listB.length; j++) {
               total += listA[i] * listB[j];
               total += listA[i] / 10;
               total += listB[j] / 100;
     }
     return total;
}
```

B. What is the order (Big O) of method `a`?

C. What is the best case order (Big O) of method `c`? N = `data.length`

```
public int c(int[] data, int tgt){
     int result = 0;
     for(int i = 0; i < data.length; i++)
          if(data[i] == tgt)
               result++;
     return result;
}
```

D. What is the worst case order (Big O) of method `d`? Assume `Arrays.fill` is O(N) and method `process` is O(N). N = `data.length`

```
public int[] d(int[] data, int key) {
     int[] result = new int[0];
     for(int i = 0; i < data.length; i++) {
          if(data[i] == key) {
               result = new int[data.length];
               Arrays.fill(data, i);
               process(data, i, key);
          }
     }
     return result;
}
```

E. What is the worst case order (Big O) of method `e` if the parameter `list` is a Java `ArrayList`?

```
public void e(List<E> list, Object target) {
     for(int i = 0; i < list.size(); i++)
          if(list.get(i) == target)
               list.remove(0); // position;
}
```

F. What is the worst case order (Big O) of method `e` if the parameter `list` is a Java `LinkedList`?

G. A method is $O(N^2)$. It takes 2 seconds for the method to run when N = 500,000. What is the expected time for the method to run when N = 2,500,000?

H. A method is $O(N\log_2 N)$. It takes 10 seconds for the method to run when N = 1,000,000. What is the expected time for the method to run when N = 4,000,000?

I. You do not have the source code to analyze a method so you have run a series timing experiments on it. Based on the following results, what is the most likely order (Big O) for the method?

N	Time to complete
100,000	1 second
200,000	2.1 seconds
400,000	4.0 seconds
800,000	7.8 seconds

For questions J - N consider the following classes and interfaces.

```
public interface Auctionable {
    public int getPredictedPrice();
}

public abstract class StorageUnit implements Auctionable {
    private int costPerMonth;

    public StorageUnit(int c) { costPerMonth = c; }

    public int cost() { return costPerMonth; }

    public String toString() { return "cost: " + cost(); }

    public int getPredictedSize() { return costPerMonth / 10; }
}

public class IndoorStorageUnit extends StorageUnit {
    private boolean guarded;

     public IndoorStorageUnit(int c, boolean g) {
          super(c);
          guarded = g;
     }

     public int cost() { return 1000; }

     public int getPredictedPrice() { return super.cost() * 2; }

     public String toString() { return "guard: " + guarded; }
}

public class POD implements Auctionable {
     private int size;

     public POD(int s) { size = s; }

     public int getPredictedPrice() { return size * 100; }

     public String toString() { return "POD: " + size; }
}
```

J. State if each of the following declarations is valid (meaning it will compile with no error) or invalid (meaning it causes a syntax error). (1 point each)

```
Auctionable a1 = new Auctionable(); // J.1
Auctionable a2 = new IndoorStorageUnit(100, true); // J.2
```

K. State if each of the following declarations is valid (meaning it will compile with no error) or invalid (meaning it causes a syntax error). (1 point each)

```
Object obj = new StorageUnit(); // K.1
StorageUnit st = new POD(6); // K.2
```

L. What is output by the following code?

```
IndoorStorageUnit isu1 = new IndoorStorageUnit(500, true);
System.out.print( isu1.getPredictedSize() + " " + isu1.cost() );
```

M. What is output by the following code?

```
StorageUnit st2 = new IndoorStorageUnit(500, true);
System.out.print( st2.cost() + " " + st2 );
```

N. What is output by the following code?

```
Auctionable a1 = new POD(5);
System.out.print( a1.getPredictedPrice() + " " + ((POD) a1).toString() );
```

O. Recall the IntList class we developed in lecture.

```
public IntList() // create an empty IntList
public int size() // return the size of this IntList
public void add(int val) // add val to the end of this IntList
public void insert(int pos, int val) // insert val at the specified position
public String toString() // return a String representation of this IntList
```

What is output by the following code?

```
IntList list1 = new IntList();
System.out.print(list1.size() + " "); // single space
list1.insert(0, 8);
list1.add(2);
list1.insert(1, 16);
System.out.print(list1);
```

2. The `GenericList` class. (20 points) To demonstrate encapsulation and the syntax for building a class in Java, we developed a `GenericList` class to represent a list. Recall our `GenericList` class stores the elements of the list in the first N elements of a native array. An Item's position in the list is the same as the item's position in the array. The array may be larger than the list being represented.

Complete an instance method for the `GenericList` class named `trimEqualBacks`. The method removes the back (rightmost) equal portions of two lists. The method also returns an `int` equal to the number of elements removed from the calling list.

The method header is:

```
/*   pre: other != null, other != this

     post: trim (remove) the back, rightmost portions of this list and
     other that are equal to each other. Return the number of elements
     removed from this list.
*/
public int trimEqualBacks(GenericList<E> other) {
```

Examples of calls to `trimEqualBacks`.

```
[].trimEqualBacks([]) returns 0, lists unchanged
[A].trimEqualBacks([]) returns 0, lists unchanged
[].trimEqualBacks([A]) returns 0, lists unchanged
[A].trimEqualBacks([A]) returns 1, lists now [] and []
[A, B].trimEqualBacks(B, A]) returns 0, lists unchanged
[B, A, A, A, C].trimEqualBacks(A, A, A, B, A, A, C]) returns 3,
     lists now [B, A] and [A, A, A, B]
[B, A, A, A].trimEqualBacks(A, A, A, B, A, A, A]) returns 4,
     lists now [] and [A, A, A]
```

You may not use any other methods in the `GenericList` class unless you define and implement them yourself as part of your answer. You may not use objects or methods from other Java classes except the `equals` method. Your solution shall be O(1) *space* meaning you may not use temporary arrays.

Recall that since this method is in the `GenericList` class, you have access to all `GenericLists'` instance variables.

Recall the `GenericList` class:

```
public class GenericList<E> {

     private E[] values;
     private int size; // size of list being represented
```

Complete the following instance method for the `GenericList` class.

```
/*   pre: other != null, other != this

     post: trim (remove) the back, rightmost portions of this list and
     other that are equal to each other. Return the number of elements
     removed from this list.
*/
public int trimEqualBacks(GenericList<E> other) {
```

3. (15 points total) The `MathMatrix` class. Write an instance method `getNumUniformColumns` for the `MathMatrix` class from assignment 2 that determines how many columns in the `MathMatix` consist of equal coefficients.

Consider this example with a 3 x 4 matrix

$$\begin{bmatrix} 1 & 3 & 4 & -2 \\ 1 & 2 & -1 & -2 \\ 1 & 1 & 2 & -2 \end{bmatrix}$$

Given the above `MathMatrix` the `getNumUniformColumns` method would return 2 because the first column consists of all 1's and the last column consists of all -2's

Recall the `MathMatrix` class:

```
public class MathMatrix {

    private int[][]coefficients; // no extra capacity.

    public int numRows() // the number of rows in this matrix
    public int numCols() // the number of columns in this matrix

    // pre: 0 <= r < numRows(), 0 <= c < numCols()
    // return the value at the given location
    public int getValue(int r, int c)
```

Complete the following instance method in the `MathMatrix` class. **You may not use any other methods from the `MathMatrix` class other than those shown above unless you implement them yourself as part of your answer. You may not use methods or classes from the Java standard library.**

```
/*   pre: numRows() > 1
     post: return the number of columns in this MathMatrix that
     contain all equal coefficients.
*/
public int getNumUniformColumns() {
```

Complete this method on the next page.

```
/*   pre: pre: numRows() > 1
     post: return the number of columns in this MathMatrix that
     contain all equal coefficients.
*/
public int getNumUniformColumns() {
```

4. Working with Maps, NameSurfer (20 points total) Write an instance method for the `Names` class from assignment 4 that updates the `NameRecord` objects it stores with ranks for a new decade and adds `NameRecords` for names that have not appeared in any previous decade.

For this question the `Names` class stores its `NameRecord`s in a `Map`. The keys of the `Map` are `String`s (the name such as "Olivia" or "Bob") and the values are the `NameRecord` objects themselves.

The `newDecade` method in the `Names` class accepts another `Map` as a parameter. The keys of this `Map` are `String`s (the top names for the decade being added to the `Names` object) and the values are `Integer` objects corresponding to the rank of the name for the new decade. All ranks are greater than 0.

For names that are already in `Name's` map, update the `NameRecord` object with the rank for the new decade. For names that are not present create a new `NameRecord` object with all previous ranks set to 0 indicating the name was not ranked in any previous decade plus the rank for the new decade. All `NameRecord` objects the `Names` class stores must have the same number of ranks. The number of decades is stored as an instance variable in the `Names` class.

Consider this **example**: The `Names` class stores `NameRecord`s with ranks for 8 decades. The following names and ranks for a new decade are being added to the `Names` class:

name	rank in new decade	already present in `Names`
`Olivia`	12	yes
`Trinity`	956	no

The `NameRecord` that contains `Olivia` would have the rank 12 added. The name `Trinity` did not appear in any previous decade so the method must construct a new `NameRecord` with the `String "Trinity 0 0 0 0 0 0 0 0 956"`, a 0 for each of the 8 previous decades in which `Trinity` did not appear. The new `NameRecord` is added to `Name's` map.

Your method must also update records that are present in `Names` but were not ranked in the new decade by adding a rank of 0 to them.

The `NameRecord` class for this question is:

```
public class NameRecord {
     // create a new NameRecord based on data.
     // data is space delimited with the name first and the ranks
     // following.
     public NameRecord(String data)

     // add a rank for a new decade to the end of this NameRecord.
     public void addRank(int newRank)

     /* return the number decades this NameRecord has ranks for
     including unranked decades.(0s) */
     public int numDecades()
```

Recall these methods from the `Map` interface:

- `Set<K>` **`keySet`**`()` - Returns a Set view of the keys contained in this map.
- `V` **`get`**`(Object key)` - Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
- `boolean` **`containsKey`**`(Object key)` - Returns true if this map contains a mapping for the specified key.
- `V` **`put`**`(K key, V value)` - Associates the specified value with the specified key in this map

Recall this method from the `Set` interface:

- `Iterator<E>` **`iterator`**`()` Returns an iterator over the elements in this set.

Recall these methods from the `Iterator` interface:

- `boolean` **`hasNext`**`()` – Returns true if the iteration has more elements.
- `E` **`next`**`()` - Returns the next element in the iteration.
- `void` **`remove`**`()` – Removes from the underlying collection the last element returned by the iterator

And finally the `Names` class itself:

```
public class Names {

     private Map<String, NameRecord> data;
     private int numDecades; // number of decades for each NameRecord

     // method to complete in this question:
     // pre: newRanks != null, all values in newRanks > 0
     // post: per the question description
     public void newDecade(Map<String, Integer> newRanks) {
```

You are not allowed to use any methods except those listed in this question and `String` creation (for example `String st = "something";`) and concatenation.

Complete this method on the next page.

```
// pre: newRanks != null, all values in newRanks > 0
// post: per the question description
public void newDecade(Map<String, Integer> newRanks) {
```

Scratch paper - GO ON TO QUESTION 5.

5. (Linked Lists 15 points). Write a method for a `LinkedList` class that determines the number of elements in the list "less than" some given value.

The properties of the `LinkedList` class:

- The list uses singly linked nodes that store one piece of data and a reference to the next node in the list.
- The only instance variable in the `LinkedList` class is a reference to the first node in the list.
- If the list is empty the reference to the first node is set to `null`.
- The last node's next reference is set to `null`.
- The `LinkedList` and `Node` objects are generic based on Java's generic syntax.
- All elements stored in the linked list implement the `Comparable` interface. The parameter `tgt` and all elements of the list implement the `Comparable` interface. No casting is necessary.

Your method must be O(1) ***space***, meaning you cannot use temporary arrays, lists, or other data structures whose size depends on the number of elements in the linked list.

You may not use any other classes or methods, except the `Node` class and the `compareTo` method.

Your method should be as efficient as possible given the constraints. The instance method to complete is:

```
/*   pre: tgt != null
     post: return the number of elements in this LinkedList that are
     less than tgt based on the compareTo method.
*/
public int numLessThan(E tgt) {
```

Here are some examples and the expected results for various lists of `Integer` objects.

```
[].numLessThan(12) - returns 0
[12, 12, 12, 12, 13].numLessThan(12) - returns 0
[12, 12, 12, 12, 13].numLessThan(13) - returns 4
[16, 32, 18, 14, -8, -8, -32].numLessThan(50) - returns 7
```

Recall the Node class:

```
public class Node<E> {
     public Node(E data, Node<E> next)

     public E getData()
     public Node<E> getNext()

     public void setData(E data)
     public void setNext(Node<E> next)
}
```

Recall the `Comparable` interface and its one method, `compareTo`:

```
public interface Comparable<E> {
     public int compareTo(E other);
     /* returns a negative integer, zero, or a positive integer as
     this object is less than, equal to, or greater than the specified
     object, other.*/

public class LinkedList<E extends Comparable<E>> {

     private Node<E> first;
```

Complete the following method:

```
/*   pre: tgt != null
     post: return the number of elements in this LinkedList that are
     less than tgt based on the compareTo method.
*/
public int numLessThan(E tgt) {
```

Scratch Paper

Question 1 answer Sheet.

Name_____

A. _____

B. _____

C. _____

D. _____

E. _____

F. _____

G. _____

H. _____

I. _____

1.

J. 2. _____

1.

K. 2. _____

L. _____

M. _____

N. _____

O. _____