Family Name: . . . . . **Other Names:** . . . . . .

ID Number: . . . . . .

COMP103 Test

23 April, 2010

Instructions

- Time: **45 minutes**.
- Answer **all** the questions.
- There are 45 marks in total.
- Write your answers in the boxes in this test paper and hand in all sheets.
- Every box with a heavy outline requires an answer.
- If you do not understand a question, ask for clarification.
- There is java documentation at the end of the exam paper that you may find useful.

		Marks		
1.	Various topics	[8]	1	
2.	Using and Implementing Collections	[27]	2	
3.	Recursion and Sorting	[10]	3	
			Total:	

Student ID: .....

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 1. Various topics [8 marks]

(a) [2 marks] What is the main property distinguishing a Queue from a Stack?

(b) [2 marks] What is a natural data structure to use to reverse the order of items in a collection?

Consider the following definition for AbstractList:

```
public abstract class AbstractList <E> implements List <E> {
    public abstract int size ();
    public boolean isEmpty() {
        return (size () == 0);
    }
    :
}
```

(c) [2 marks] Why does size need to be declared abstract (but not isEmpty)?

(d) [2 marks] Removing an item from an ArraySet is easier than from an ArrayList, in the sense that the removed-from site can be filled by moving the last item in the array into it. So why is the cost of remove in ArraySet actually O(n)?

Question 2. Using and Implementing Collections [27 marks]

(a) [10 marks] Suppose you have a Set of objects of class Item, and need to have these items put into a fixed number of Queues. Assume that the order that they are added to the queues isn't actually important - all that matters is that they are put into queues, and that the queues end up about the same size as one another.

Complete the following splitIntoQueues method, which takes a Set of Items, together with the number of queues, and returns a List of Queues of Items.

You will need to specify an implementation for Queue, for which you may use LinkedList.

```
public List <Queue <Item>> splitIntoQueues (Set <Item> setOfItems, int n) {

}
```

A benefit of having a *sorted* Array Bag is that it can be searched efficiently by the **binary search** algorithm. In Assignment 4 you looked at an implementation of a collection type we called SortedArrayBag, which begins:

```
public class SortedArrayBag <E> extends AbstractCollection <E> {
  private static int INITIALCAPACITY = 10;
  private int count = 0;
  private E[] data;
  private Comparator<E> cmp = (Comparator<E>) new ComparableComparator();
  :
```

(b) [9 marks] Complete the following findIndex method, that uses binary search to find the index of where an element is in the array (or at least where it *ought* to be). You may assume that the item being searched for is not null. The method should use cmp (the Comparator declared at the bottom of the previous page), and should return a value between 0 and count.

```
private int findIndex(Object itm){
  E item = (E) itm;
  int low = 0;
  int high = count;









}
```

(c) [8 marks] Now complete the remove method, which uses findIndex to locate the item to be removed (see the 3rd line, below). Your method should leave the array in a correct state with the item removed. It should return true if the collection changes, and false if it did not change.

```
public boolean remove (Object item) {

  if (item==null) return false;
  int index = findIndex(item);









}
```

Question 3. Recursion and Sorting [10 marks]

(a) [5 marks] n factorial (written 'n!') is $n * (n-1) * (n-2) * \ldots * 3 * 2 * 1$.
For example, $4! = 4 * 3 * 2 * 1 = 24$.

Write a recursive method factorial that takes an integer n and returns n!

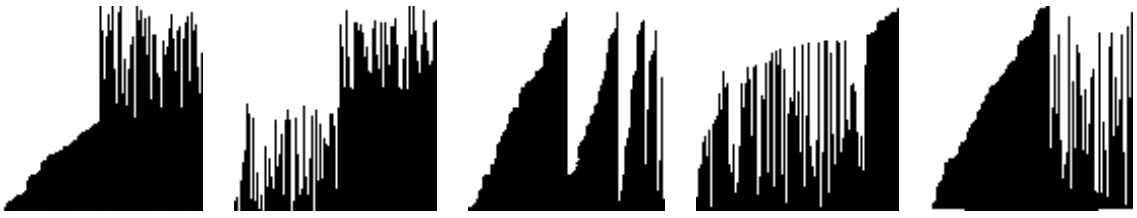
```
public static int factorial ( int n) {




}
```

(b) [1 mark] What is the asymptotic ('big-O') cost of InsertionSort on lists that are already almost sorted?

(c) [2 marks] Again, consider the case of lists that are already almost sorted. What would the 'big-O' cost of QuickSort be, on almost-sorted lists, if it were implemented by using the *final* point in the array as the pivot?



A B C D E

(d) [2 marks] Each of the above pictures shows a different sorting algorithm part of the way through its operations. Which one is MergeSort?

****

Appendix (may be removed)

Brief (and simplified) specifications of some relevant interfaces and classes.

```
public interface Iterator<E>
    public boolean hasNext();
    public E next();
    public void remove();

public interface Iterable<E>                    // Can use in the "for each" loop
    public Iterator<E> iterator();

public interface Comparable<E>                  // Can compare this to another E
    public int compareTo(E o);

public interface Comparator<E>                  // Can use this to compare two E's
    public int compare(E o1, E o2);
```

```
public interface Collection<E>
    public boolean isEmpty();
    public int size ();
    public boolean add();
    public Iterator <E> iterator();

public interface List<E> extends Collection<E>
    // Implementations: ArrayList
    public E get( int index);
    public void set( int index, E element);
    public void add(E element);
    public void add(int index, E element);
    public void remove(int index);
    public void remove(Object element);

public interface Set extends Collection<E>
    // Implementations: ArraySet, SortedArraySet, HashSet
    public boolean contains(Object element);
    public boolean add(E element);
    public boolean remove(Object element);

public interface Queue<E> extends Collection<E>
    // Implementations: ArrayQueue, LinkedList
    public E peek ();                    // returns null if queue is empty
    public E poll ();                    // returns null if queue is empty
    public boolean offer (E element);

public class Stack<E> implements Collection<E>
    public E peek ();                    // returns null if stack is empty
    public E pop ();                     // returns null if stack is empty
    public E push (E element);

public interface Map<K, V>
    // Implementations: HashMap, TreeMap, ArrayMap
    public V get(K key);                 // returns null if no such key
    public void put(K key, V value);
    public void remove(K key);
    public Set<Map.Entry<K, V>> entrySet();
```