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

ID Number: . . . . . .

COMP103 Test 2

7 May, 2009

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 are useful formulae and documentation at the end of the exam paper.

		Marks		
1.	Various topics	[10]	1	
2.	Sorting	[15]	2	
3.	Linked Structures	[20]	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 [10 marks]

(a) [2 marks] Give an example of a 'slow' sorting algorithm from lectures.

(b) [2 marks] Give an example of a 'fast' sorting algorithm from lectures.

(c) [2 marks] If you were implementing a Stack using a Linked List, which end of the list would you add and remove from? Why?

(d) [2 marks] Which of the methods given below are more efficient in a SortedArraySet implementation than in an unsorted ArraySet?

- contains
- add
- remove

(e) [2 marks] What is the difference between a binary tree and a general tree?

Question 2. Sorting [15 marks]

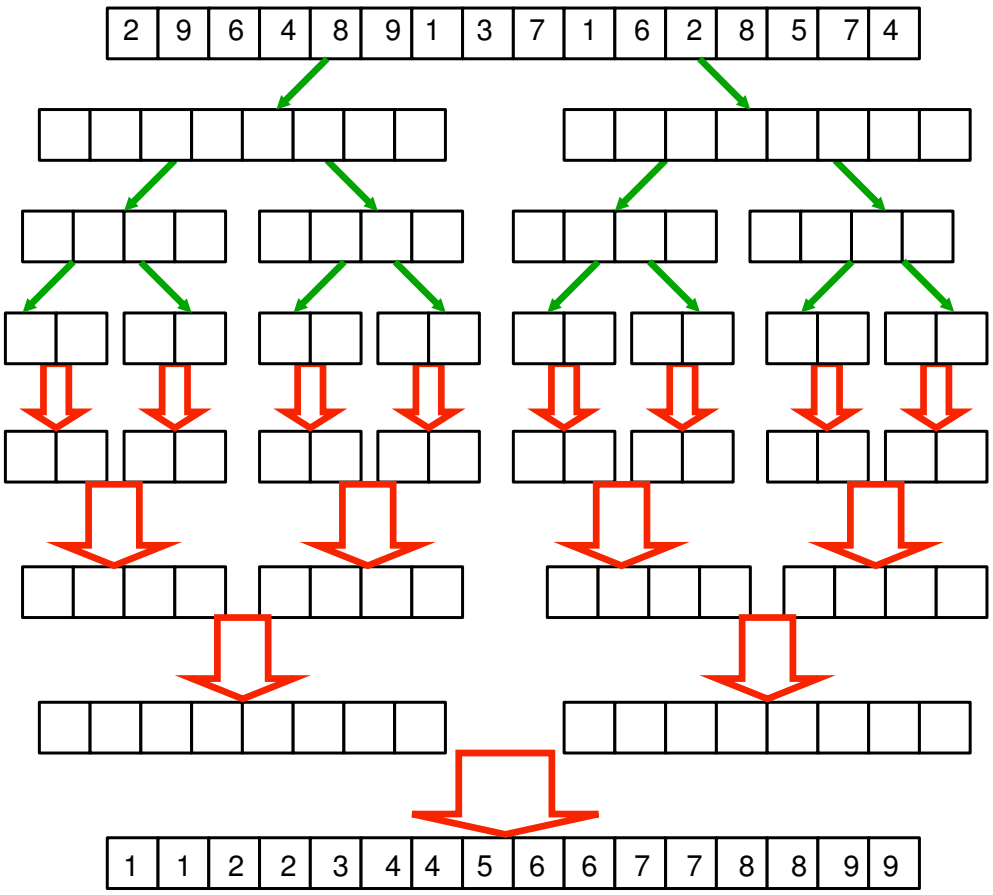
(a) [5 marks] Complete "big O" costs for each of the following sorting algorithms. Two of them are given for you.

	Best Case	Worst Case	Average Case
Insertion Sort		$O(n^2)$	$O(n^2)$
Selection Sort			
Merge Sort			
Quick Sort			

(b) [5 marks] Complete the following method to sort the array of integers from lowest to highest using one of the 'slow' sorting algorithms.

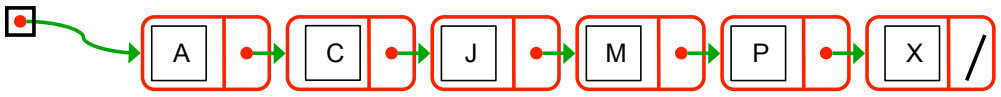
```
public int[] sort(int[] array) {

  /* This is _____ sort */

  for(                                    ) {









   }

   return array;
}
```

(c) [5 marks] Complete the following diagram, showing the intermediate steps that merge-sort uses to sort the following array of integers from smallest to largest.



Question 3. Linked Structures [20 marks]

This diagram, which you saw in lectures, represents a linked list.



(a) [3 marks] Modify the *links* in the diagram, so as to:

- Remove the last element (X)
- Add a new element Q between J and M
- Remove the first element A.

The following code implements a linked list, like the one in the diagram above.

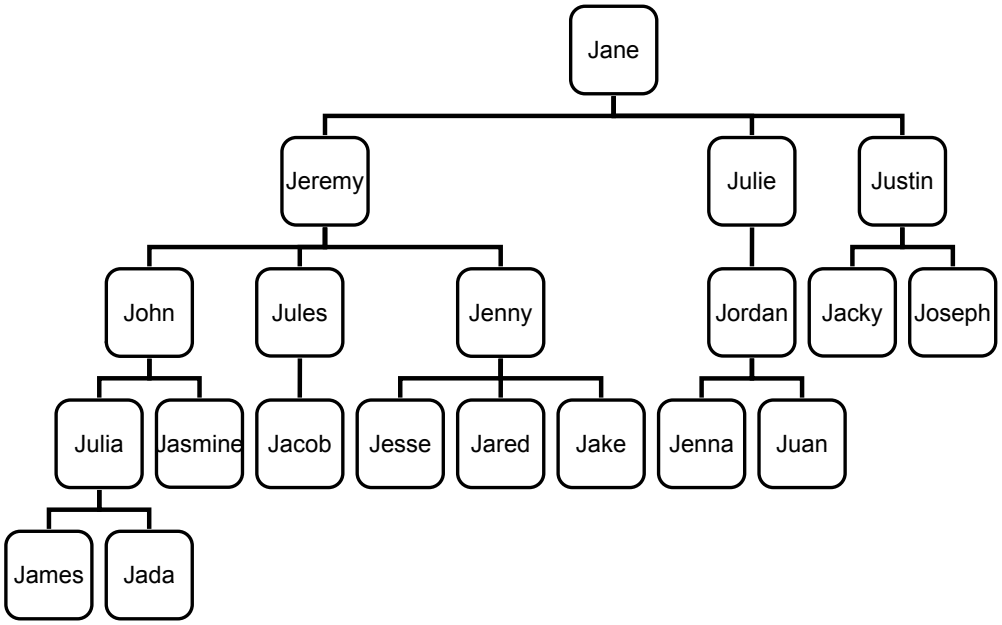
```
class LinkedList<E> {
        private LinkedListNode<E> data;

        public void addFirst(E value) {
                LinkedListNode<E> first = new LinkedListNode<E>(value);
                 first .next = data;
                data =  first ;
        }

        public E removeFirst() {
                if (data != null) {
                        E val = data.value;
                        data = data.next;
                        return val;
                }
                return null;
        }

        public void remove(E value) {
                if (data != null) {
                        data = data.remove(value);
                }
        }
}
```

(b) [8 marks] Complete the implementation of LinkedListNode below by implementing a recursive remove method to remove all occurrences of a particular value from the list.

```
class LinkedListNode<E> {
        E value;
        LinkedListNode<E> next;
        public LinkedListNode(E value) {
                this.value = value;
        }
        public LinkedListNode<E> remove(E value) {









        }
}
```

(c) [4 marks] Suppose we added a new field, tail, to LinkedList which stores the last node in the linked list so that we could add a new element to the list with O(1) cost.

If we added a new method to remove the last node in the list, what would the cost be? Justify your answer.



(d) [1 mark] How many leaves does the tree have?

(e) [1 mark] What is its depth?

Here is a simple class for representing a node in a binary tree.

```
class TreeNode<E> {
        TreeNode<E> left;
        TreeNode<E> right;
        E value;
}
```

(f) [3 marks] What would you change to make this a *general* tree?

****

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.

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();
```