

Family Name:

Other Names:

ID Number:

COMP103 Test 2

7 May, 2009

| ***** WITH SOLUTIONS *****

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	<input type="text"/>
2. Sorting	[15]	2	<input type="text"/>
3. Linked Structures	[20]	3	<input type="text"/>
		Total:	<input type="text"/>

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.

| One of: Selection, Insertion, Bubble

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

| One of: Merge, Quick

(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?

| Front: $O(1)$ add and remove. Back would be $O(n)$ for remove.

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

- contains | Contains is more efficient: $O(\log(n))$ as opposed to $O(n)$
- add
- remove

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

| A binary tree has at most two children for every node.
A general tree has an arbitrary number of children for each node.

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	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log(n))$	$O(n \log(n))$	$O(n \log(n))$
Quick Sort	$O(n \log(n))$	$O(n^2)$	$O(n \log(n))$

(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( int i = 0; i < array.length-1; i++ ) {

        int min = i;

        for ( int j = i+1; j < array.length; j++ ) {
            if (array[j] < array[min]) {
                min = j;
            }
        }

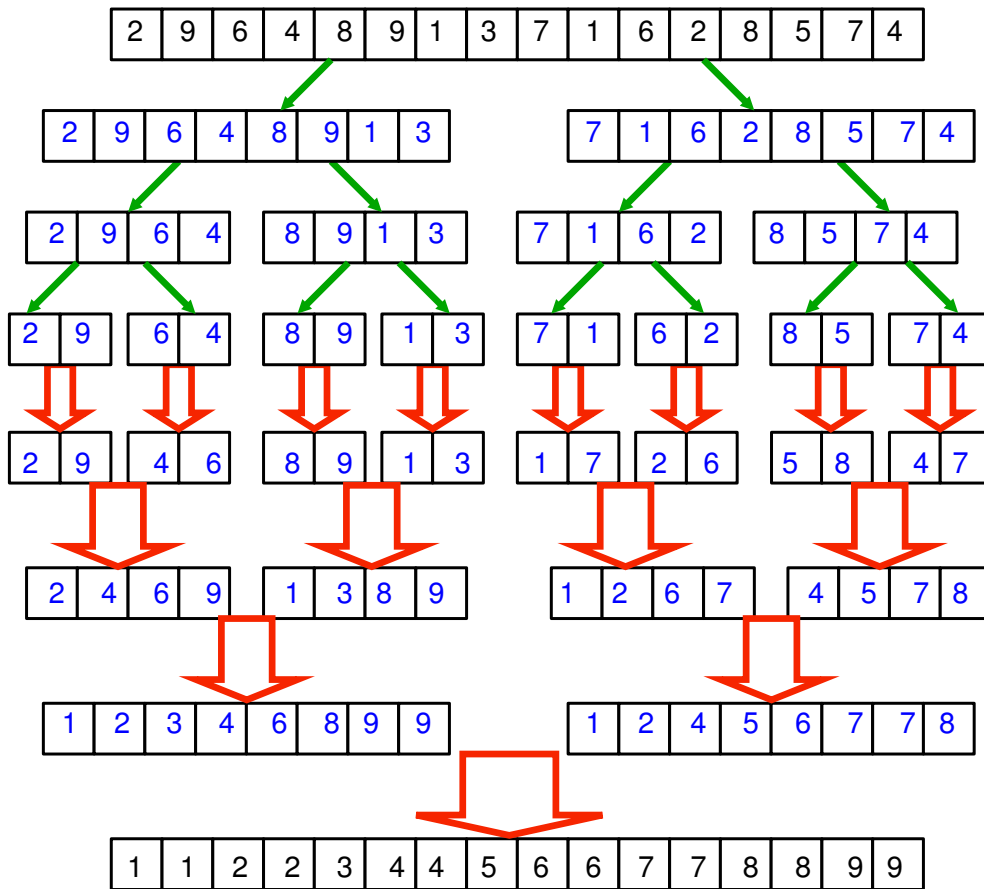
        int tmp = array[min];
        array[min] = array[i];
        array[i] = tmp;

    }

    return array;
}

```

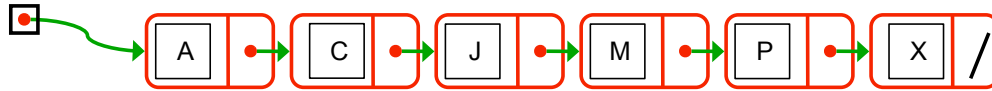
(c) [5 marks] Complete the following diagram, showing the intermediate steps that merge-sort uses to sort the following array of integers from smallest to largest.



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) {
        if (next != null) {
            next = next.remove(value);
        }
        if (this.value.equals(value)) {
            return next;
        }
        else {
            return this;
        }
    }
}

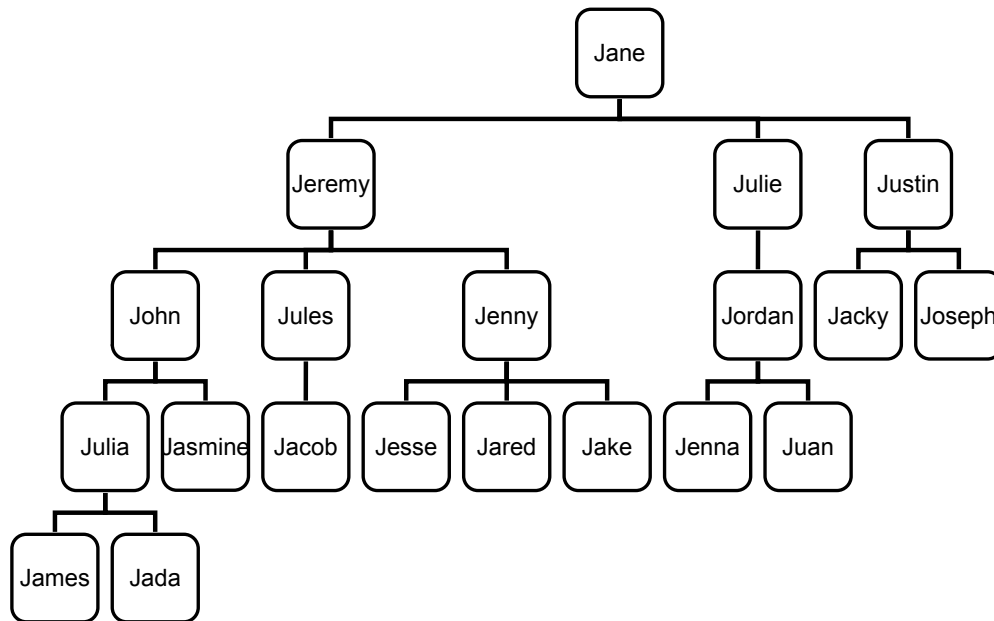
```

(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.

$O(n)$

Even though we could access the last element quickly, it would still take $O(n)$ steps to access the second last one to store it in the `tail` field.



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

| 11

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

| 4

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?

| Remove left and right and replace them with a list

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>

public Iterator <E> iterator();

// Can use in the "for each" loop

public interface Comparable <E>

public *int* compareTo(E o);

// Can compare this to another E

public interface Comparator <E>

public *int* compare(E o1, E o2);

// Can use this to compare two E's

```
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();
```