

Family Name:

Other Names:

ID Number:

COMP103 Test

11 Sept, 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 is java documentation at the end of the exam paper that you may find useful.

	Marks		
1. Various topics	[12]	1	<input type="text"/>
2. Using and Implementing Collections	[23]	2	<input type="text"/>
3. Recursion and Sorting	[10]	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

[14 marks]

(a) [2 marks] What properties distinguish a Set from a List?

(b) [2 marks] Give two advantages of using ArrayList over just using an array for storing items.

- 1.
- 2.

(c) [2 marks] Suppose we have a method that operates on a collection, and by looking at the code we find that the method takes $3.1n^2 + 16n + 2 \log n + 116000$ steps to complete, where n is the number of items in the collection. What is its asymptotic ('big-O') cost?

(d) [2 marks] Provided the array does not fill up completely, the per-item (*amortised*) cost of adding items to the end of an ArrayList is $O(1)$. But if the array fills up we need to make a new array of double the size, and copy all the current items into it. What is the amortised cost when we take this double-and-copy cost into account as well?

(e) [2 marks] Suppose you have a set of items stored *in sorted order* in an array. The simplest way to search the set for a given item is to go through the items in the order they appear in the array. Binary search is more efficient than this. For a set of $n = 1024$ items, how much faster is binary search, on average? (e.g. 2 times? 100 times? ...)

(f) [2 marks] Consider sorting items stored an array, when the items are already almost perfectly sorted. Which of InsertionSort and MergeSort is likely to be faster, and why?

Question 2. Using and Implementing Collections

[23 marks]

(a) [12 marks]

Write a method called `reverseAllQueues`, that takes a `Set` of `Queues` of items of type `Face`. The method has to return a `Set` of `Queues`, but now each queue must have its order reversed.

Hint. Think about how to reverse a single queue first, as this alone will gain partial marks.

```
public Set <Queue <Face>> reverseAllQueues (Set <Queue <Face>> qset) {
```

```
}
```

The next two questions are about iterating through a list. Consider `mytasks`, a List of objects of type `Task`. Since Lists are Iterable, we could iterate over `mytasks` using a for each loop as follows:

```
for (Task task : mytasks )  
    System.out.println(task.toString ());
```

(b) [3 marks] Give code which has the same effect but uses a simple for loop with an integer index instead.

(c) [4 marks] Give code which does the same but uses an iterator.

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 3. Recursion and Sorting

[10 marks]

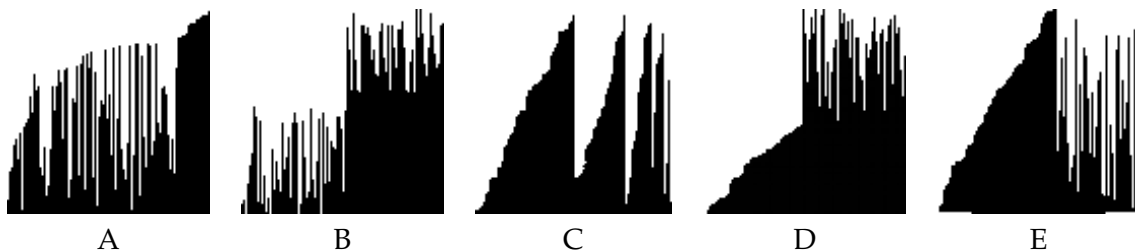
(a) [4 marks] What is the output of the following code, when called with `recurse(1)`?

```
public void recurse(int L) {
    System.out.print(" ( ");
    if (L < 8) recurse(2*L);
    else System.out.print(" * ");
    System.out.print(") ");
}
```

(b) [1 mark] Is the recursion used in MergeSort *multiple recursion* (i.e. more than one recursive call per method call)?

(c) [1 mark] Is the recursion used in MergeSort *tail-recursion*? That is, in the MergeSort method is the recursive call (or calls) done last?

(d) [2 marks] The performance of QuickSort is sensitive to the choice of pivot item. What is the "perfect" pivot item to use, and why is it infeasible to use it?



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

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();
```