

Family Name:

Other Names:

ID Number:

COMP103 Test

5 Sept, 2007

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

Instructions

- Time: **90 minutes**.
- Answer **all** the questions.
- There are 90 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 formulas and documentation at the end of the exam paper.

	Marks		
1. Collection Types	[16]	1	<input type="text"/>
2. Implementing a Collection	[15]	2	<input type="text"/>
3. Sorting Algorithms	[19]	3	<input type="text"/>
4. Cost of Algorithms	[25]	4	<input type="text"/>
5. Linked Lists	[15]	5	<input type="text"/>
		Total:	<input type="text"/>

Question 1. Collection Types

[16 marks]

(a) [12 marks] Describe each of the following collection types by specifying any structure that the collection imposes on the items, any constraints on the items (*e.g.*, if it allows or disallows duplicates), and any constraints on how the items can be accessed.

Note, you may wish to refer to the Java API documentation at the end of the paper.

Bag: No structure on items required

| Duplicates allowed

| No constraints on access

Set: No structure on items required

| Duplicates not allowed

| No constraints on access

List: Items must be stored in a sequence/order

| Duplicates are allowed

| No constraints on access

Queue: Items must be stored in a sequence/order

| Duplicates are allowed

| Items may only be added at one end, and removed from the other end

(Question 1 continued on next page)

(Question 1 continued)

(b) [4 marks] Maps are more complex than just collections of items. Explain briefly what a Map is, and how values in a Map are added and accessed.

Map: A Map is a set of key/value pairs, so that each key is associated with one value

- | duplicate keys are not allowed, but duplicate values for different keys are allowed
- | Items may only be accessed via keys

Question 2. Implementing a Collection

[15 marks]

For this question, you are to complete some of the methods of the `ArraySet` class, which defines a Set collection that stores the items (unordered) in an array.

The header and fields of the class are given below:

```
public class ArraySet <E> extends AbstractSet <E> {
    private static int INITIALCAPACITY = 10;
    private int count = 0;
    private E[] data;
```

(a) [4 marks] Complete the following constructor that constructs an empty set. It should initialise the data field.

```
public ArraySet(){
    data = (E[] ) new Object[INITIALCAPACITY];
}
```

(b) [6 marks] Complete the add method which will insert an item into the set, unless the item is already present. add will return true if and only if it modifies the set. Before adding an item, add should call the ensureCapacity method to make sure there is sufficient room in the data array.

```
public boolean add(E item){
    for ( int i=0; i<count; i++){
        if (item.equals(data[i ]))
            return false;
    }
    ensureCapacity();
    data[count]=item;
    count++;
    return true;
}
```

(Question 2 continued on next page)

(Question 2 continued)

(c) [5 marks] Complete the following ensureCapacity method which will “double and copy” the data array if it is currently full.

```
private void ensureCapacity () {  
    if (count < data.length) return;  
    E [ ] newArray = (E[ ])(new Object[data.length*2]);  
    for (int i = 0; i < count; i++)  
        newArray[i] = data[i];  
    data = newArray;  
  
}
```

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. Sorting Algorithms

[19 marks]

The lectures described five common sorting algorithms:

- BubbleSort
- MergeSort
- InsertionSort
- QuickSort
- SelectionSort

(a) [2 marks] Which two of the algorithms had fast (*i.e.*, $O(n \log(n))$) average case costs:

| MergeSort, QuickSort.

(b) [2 marks] Which algorithm is very fast (*i.e.*, $O(n)$) if the items are already sorted:

| InsertionSort.

(c) [2 marks] Which algorithm is not “in place”, (*i.e.*, it requires a second array to hold items while sorting).

| MergeSort.

(d) [2 marks] Name one of the algorithms that is not stable (*i.e.*, it does not keep equal items in their original order).

| SelectionSort or QuickSort.

(e) [2 marks] Name one of the algorithms that used a “Divide and Conquer” strategy.

| MergeSort or QuickSort.

(Question 3 continued on next page)

(Question 3 continued)

Each diagram below shows part of the progress of one of the five sorting algorithms. Each line shows the items in the array after an assignment or a swap. The underlined items are the ones that were assigned or swapped in that step.

All the algorithms are started on the same initial array.

For each diagram, state the sorting algorithm.

(f) [3 marks]

Initial array:	c	h	e	a	f	g	d	b
step 1:	c	<u>h</u>	e	a	f	g	d	b
step 2:	c	h	<u>h</u>	a	f	g	d	b
step 3:	c	<u>e</u>	h	a	f	g	d	b
step 4:	c	e	h	<u>h</u>	f	g	d	b
step 5:	c	e	<u>e</u>	h	f	g	d	b
step 6:	c	<u>c</u>	e	h	f	g	d	b
step 7:	<u>a</u>	c	e	h	f	g	d	b
step 8:	a	c	e	h	<u>h</u>	g	d	b
step 9:	a	c	e	<u>f</u>	h	g	d	b

Algorithm: | InsertionSort.

(g) [3 marks]

Initial array:	c	h	e	a	f	g	d	b
step 1:	<u>b</u>	h	e	a	f	g	d	<u>c</u>
step 2:	b	<u>a</u>	e	<u>h</u>	f	g	d	c
step 3:	<u>a</u>	<u>b</u>	e	h	f	g	d	c
step 4:	a	b	<u>c</u>	h	f	g	d	<u>e</u>
step 5:	a	b	c	<u>d</u>	f	g	<u>h</u>	e
step 6:	a	b	c	d	<u>e</u>	g	h	<u>f</u>
step 7:	a	b	c	d	e	<u>f</u>	h	<u>g</u>
step 8:	a	b	c	d	e	f	<u>g</u>	<u>h</u>

Algorithm: | QuickSort.

(Question 3 continued)**(h)** [3 marks]

Initial array:	c	h	e	a	f	g	d	b
step 1:	<u>a</u>	h	e	<u>c</u>	f	g	d	b
step 2:	a	<u>b</u>	e	c	f	g	d	<u>h</u>
step 3:	a	b	<u>c</u>	<u>e</u>	f	g	d	h
step 4:	a	b	c	<u>d</u>	f	g	<u>e</u>	h
step 5:	a	b	c	d	<u>e</u>	g	<u>f</u>	h
step 6:	a	b	c	d	e	<u>f</u>	<u>g</u>	h
step 7:	a	b	c	d	e	f	<u>g</u>	h

Algorithm: SelectionSort.

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 4. Cost of Algorithms

[25 marks]

The following randomise method will put the items in an array into a random order.

Assume that `random.nextInt(num)` will return a random integer between 0 and `num-1`.

```
public void randomise(String[] data){
    for (int i=0; i<n; i++){
        int j = random.nextInt(data.length);
        String temp = data[i];
        data[i] = data[j];
        data[j] = temp;
    }
}
```

Assume that n is the length of the array.

(a) [3 marks] How many times will the `nextInt` method be called, as a function of n .

n

(b) [3 marks] How many assignments of String values will `randomise` perform, as a function of n .

$3n$

(c) [2 marks] What is the asymptotic ("Big-O") complexity of `randomise`, as a function of n .

$O(n)$

(Question 4 continued on next page)

(Question 4 continued)

The following countSort algorithm first copies an array to a temporary array, then, for each item in the array, counts how many other items are smaller than the item. This number is the position the item should be in, so it copies the item into that position in the data array.

```
public void countSort(String[] data){
    // Copy data to a temporary array.
    String[] tempArray = new String[data.length];
    for (int i=0; i<data.length; i++){
        tempArray[i] = data[i];

        // Copy each item back to its correct position
        for (String item : tempArray){
            int countSmaller = 0;
            for (int j=0; j<data.length; j++){
                if (tempArray[j].compareTo(item) < 0)
                    countSmaller++;
            }
            data[countSmaller] = item;
            System.out.printf("%d items smaller than %s\n",countSmaller, item);
        }
    }
}
```

(d) [3 marks] Show what the countSort method would print out if it were called with an array containing the values {"c", "a", "b"}.

```
| 2 items smaller than c
| 0 items smaller than a
| 1 items smaller than b
```

(e) [2 marks] If countSort is called on an array with n items, how many calls to compareTo will it perform, as a function of n ?

```
|  $n^2$ 
```

(f) [2 marks] How many String assignments will countSort do, as a function of n ?

```
|  $2n$  ( $n$  to copy into tempArray, and  $n$  to copy back into correct place)
```

(g) [2 marks] What is the Asymptotic ("Big-O") complexity of countSort, as a function of n ?

```
|  $O(n^2)$ 
```

(Question 4 continued)

(h) [8 marks] The `countSort` method above may not sort the items correctly if the data array contains two or more equal items. Write a correct version of the `countSort` algorithm which correctly sorts arrays of Strings, even if there are duplicates.

```

public void countSort(String[] data){
    // Copy data to a temporary array.
    String[] tempArray = new String[data.length];
    for (int i=0; i<data.length; i++)
        tempArray[i] = data[i];

    // Copy each item back to its correct position
    for
        (int index=0; index<data.length; index++){
            String item = tempArray[index];
            int countSmaller = 0;
            for (int j=0; j<data.length; j++){
                if ((tempArray[j].compareTo(item) < 0) ||
                    (tempArray[j].equals(item) && j<index))
                    countSmaller++;
            }
            data[countSmaller] = tempArray[index];
//OR
        for (int i=0; i<data.length; i++){
            tempArray[i] = data[i];
            data[i] = null;           // put null markers in the data array
        }
        for (String item : tempArray){
            int countSmaller = 0;
            for (int j=0; j<data.length; j++){
                if (tempArray[j].compareTo(item) < 0)
                    countSmaller++;
            }
            while(data[countSmaller] != null) countSmaller++;
            data[countSmaller] = tempArray[index];
        }
}

```

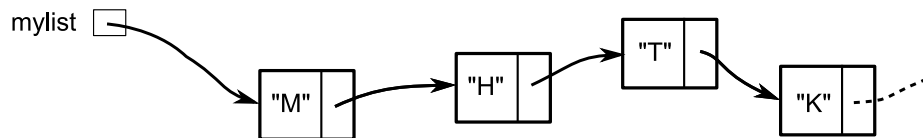
Question 5. Programming with LinkedNodes

[15 marks]

This question uses a `LinkedList` class, like the one in lectures, which defines `LinkedList` objects that contain a value and a reference to another `LinkedList`. The class is given on the facing page.

In your answers below, you should use the `getValue()`, `getNext()`, and `setNext()` methods, rather than the `value` and `next` fields.

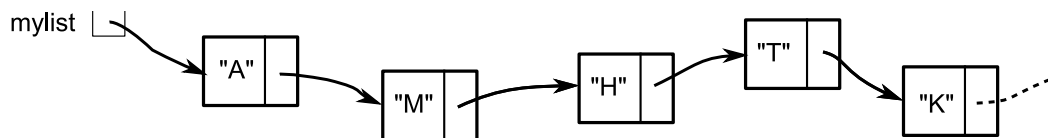
Suppose `myList` is a variable of type `LinkedList<string>`, which contains a linked list of `LinkedList`s. For example, it might contain:



(a) [3 marks] Complete the following statement so that it will print out the value contained in the second node of the list in `myList` (the value "H" in the example above).

```
System.out.println( myList . getNext().getValue() );
```

(b) [3 marks] Write a statement that will insert the string "A" at the front of the list in `myList`. For the example above, `myList` should now contain:



```
myList = new LinkedList<String>("A", myList) ;
```

(c) [4 marks] Write a statement (or a sequence of statements) that will remove the third item from the list in `myList`.

You may assume that there are at least three nodes in the list.

```
myList.getNext().setNext(myList.getNext().getNext().getNext());
//OR
LinkedList<String> scnd = myList.getNext();
scnd.setNext(scnd.getNext().getNext());
```

(Question 5 continued on next page)

(Question 5 continued)

(d) [5 marks] Complete the following `contains` method that is passed a linked list and an item. It should return true if any value in the list is equal to the item. You may write a recursive or iterative version. You may assume that neither `item` nor any of the values in the list are null.

```

public boolean contains(LinkedList<String> list, String item){
    if ( list ==null) return false;
    if ( list .getValue().equals(item)) return true;
    return contains( list .getNext(), item);
\\OR
    for(LinkedList<String> rest=list; rest!=null; rest=rest.getNext())
        if (rest.getValue().equals(item)) return true;
    return false;
}

```

The `LinkedList` class:

```

public class LinkedList <E>{
    private E value;
    private LinkedList<E> next;

    public LinkedList(E item, LinkedList<E> nextNode){
        value = item;
        next = nextNode;
    }
    public E getValue(){
        return value;
    }
    public LinkedList<E> getNext(){
        return next;
    }
    public void setNext(LinkedList<E> n){
        next = n;
    }
}

```

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.

Appendices (may be removed)

Possibly useful formulas:

- $1 + 2 + 3 + 4 + \dots + k = \frac{k(k+1)}{2}$
- $1 + 2 + 4 + 8 + \dots + 2^k = 2^{k+1} - 1$
- $a + (a + b) + (a + 2b) + \dots + (a + kb) = \frac{(2a+kb)(k+1)}{2}$
- $a + as + as^2 + as^3 + \dots + as^k = \frac{as^{k+1} - a}{s-1}$

Table of base 2 logarithms:

n	1	2	4	8	16	32	64	128	256	512	1024	1,048,576
$\log_2(n)$	0	1	2	3	4	5	6	7	8	9	10	20

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

public class Random

```
public int nextInt(int n);           // return a random integer between 0 and n-1
public double nextDouble();         // return a random double between 0.0 and 1.0
```

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();

```