

Name:

ID Number:

COMP103: Test

8 Sept, 2004.

Instructions

- Time: **2 hours**.
- Answer **all** the questions.
- There are 120 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.
- The questions in part 2 are more difficult than the questions in part 1.
- There are useful formulas and documentation at the end of the exam paper.

Part 1: (67 marks)

Marks

1. Collection Types	[12]	1	<input type="text"/>
2. Implementing ArrayBag	[9]	2	<input type="text"/>
3. Programming with Collections	[11]	3	<input type="text"/>
4. Asymptotic or “Big O” analysis	[16]	5	<input type="text"/>
5. Hash Tables	[7]	6	<input type="text"/>
6. Linked Lists	[12]	7	<input type="text"/>

Part 2: (53 marks)

7. Sorting Algorithms	[15]	8	<input type="text"/>
8. Linked Lists	[14]	9	<input type="text"/>
9. Hash Tables with probing	[14]	10	<input type="text"/>
10. Analysing cost of Algorithms	[10]	11	<input type="text"/>

Total:

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.

Part 1

Question 1. Collection Types

[12 marks]

Match each of the jds types in the left column to its best description in the right hand column by drawing lines from the types to the descriptions. Note: four descriptions do not match any type.

Types	Descriptions
Indexed	Collection of values Order of values is not meaningful Duplicate values not allowed.
Set	Collection of key – value pairs Order of values is the order in which their keys were added. Duplicate values allowed
Bag	Collection of values Order of values is the order in which the values were added. Duplicate values not allowed
Map	Collection of key – value pairs Order of values and keys is not meaningful Only one value may be associated with any key
	Collection of values Order of values is the “natural” sorting order of the values Duplicate values not allowed
	Collection of values Order of values is not meaningful Duplicate values allowed
	Collection of values Order of values is the order specified when the values were added. Duplicate values allowed
	Collection of key – value pairs Order of values is “natural” sorting order of the keys. Duplicate keys allowed, but values must be unique.

Question 3. Programming with Collections

[11 marks]

Suppose you are writing a program `CourseReporter` to write various reports on the courses and lectures in the university. Assume that `CourseReporter` includes the fields

```
private Set allCourses;  
private Map organisers;
```

where `allCourses` is a `Set` of the course codes (eg "`COMP103`") of all courses, and `organisers` is a `Map` which contains the lecturer in charge of each course (the key is the course code, and the value is the name of the lecturer).

(a) [4 marks] Complete the following `printCourses` method that should print out the course code of every course, one per line, to `System.out`.

```
private void printCourses(){  
    System.out.println("All courses");  
  
}
```

(b) [7 marks] Complete the following `selectedCourses` method that should print out the course codes of the courses organised by a particular lecturer, one per line, to `System.out`.

```
private void selectedCourses(String lecturer){  
    System.out.println("Courses organised by "+lecturer);  
  
}
```

Note: `jds` documentation is at the end of the test paper.

Question 4. Asymptotic or “Big O” analysis

[16 marks]

(a) [4 marks] What is the average case asymptotic (“Big O”) costs of searching for an item in each of the following implementations of **Set**? Assume that the size of the Set is n .

- **ArraySet**
(unordered, array)

$O(\quad)$

- **SortedArraySet**
(ordered, array)

$O(\quad)$

- **BucketHashSet**
(Hashtable with k buckets where each bucket is an **ArraySet**)

$O(\quad)$

- **OpenHashSet**
(Hashtable with open addressing, linear probing, and guaranteed less than 60% full).

$O(\quad)$

(b) [6 marks] For each of the following operations on Bags, circle the correct answer to say whether the operation will be faster in a Bag using a sorted array, faster in a Bag using an unsorted array, or about the same speed, and give a very brief justification.

FindElement:	<input type="radio"/> faster with a sorted array <input type="radio"/> faster with an unsorted array <input type="radio"/> about the same
Justification:	
AddElement:	<input type="radio"/> faster with a sorted array <input type="radio"/> faster with an unsorted array <input type="radio"/> about the same
Justification:	
RemoveElement:	<input type="radio"/> faster with a sorted array <input type="radio"/> faster with an unsorted array <input type="radio"/> about the same
Justification:	

(Question 4 continued)

(c) [3 marks] Suppose a program uses an algorithm with an average case asymptotic cost of $O(n)$ (where n is the size of the input collection).

When the program is run on a collection with 200,00 items, the measured running time is 8 seconds.

Give a reasonable estimate of the running time of the program on a collection with 2,000,000 items.

Justify your answer.

Estimated running time:

Justification:

(d) [3 marks] Suppose a program uses an algorithm with an average case asymptotic cost of $O(n^2)$, where n is the size of the input collection.

When the program is run on a collection of 5,000 items, the measured running time is 30 milliseconds.

Give a reasonable estimate of the running time of the program on a collection of 1,000,000 items.

Justify your answer.

Estimated running time:

Justification:

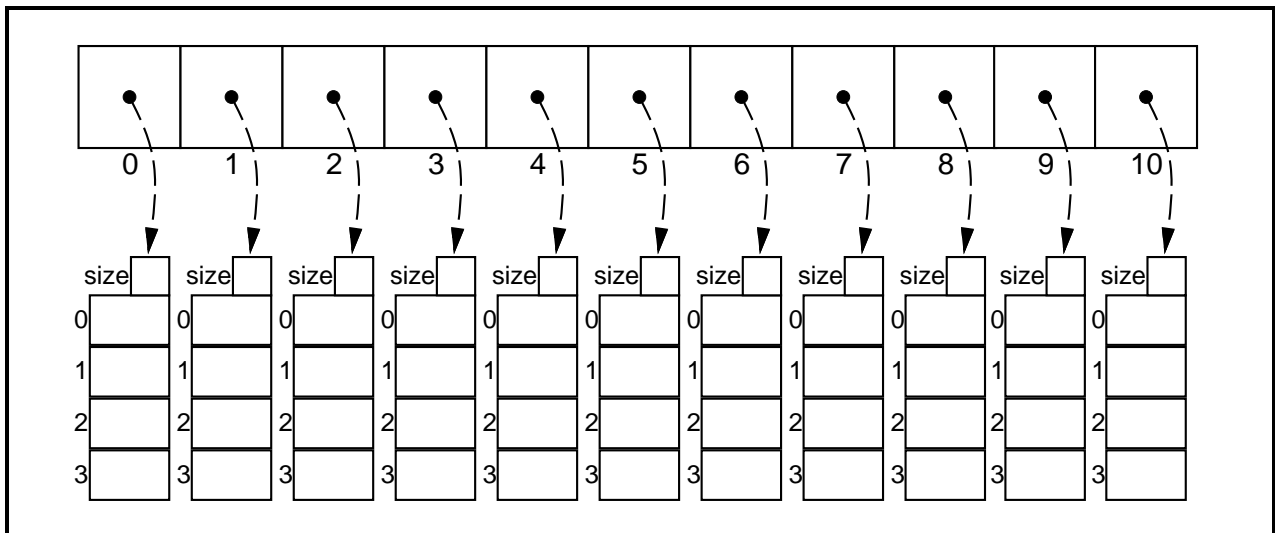
Question 5. Hash Tables

[7 marks]

The diagram below shows an instance of **BucketHashBag** which is a **Bag** implemented in a hash table using Buckets. Each bucket is implemented as an **ArrayBag**.

Show (on the diagram) the contents of the Bag when the following seven items are added to the set in the order “Elk”, “Cat”, “Dog”, “Ape”, “Bat”, “Bee”, “Bug”, etc.

item:	"Elk "	"Cat "	"Dog "	"Ape "	"Bat "	"Bee "	"Bug "
hash value:	2	7	4	2	10	4	2



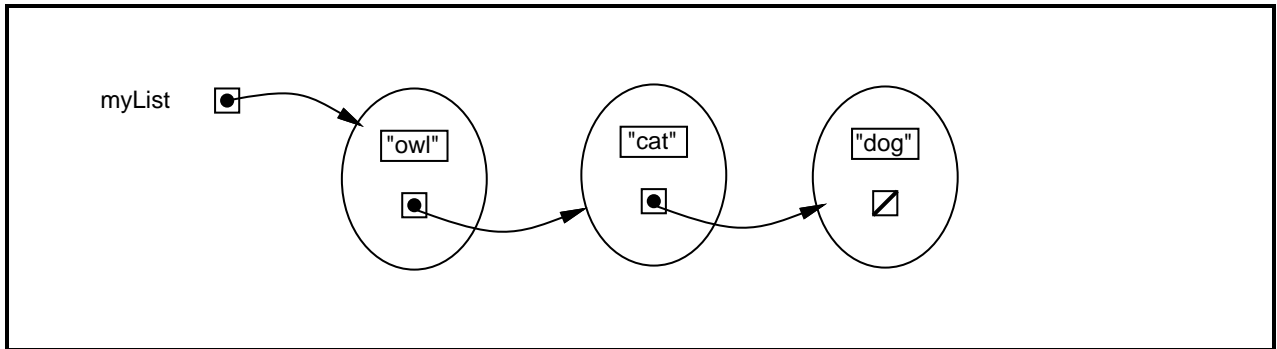
SPARE PAGE FOR EXTRA ANSWERS

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

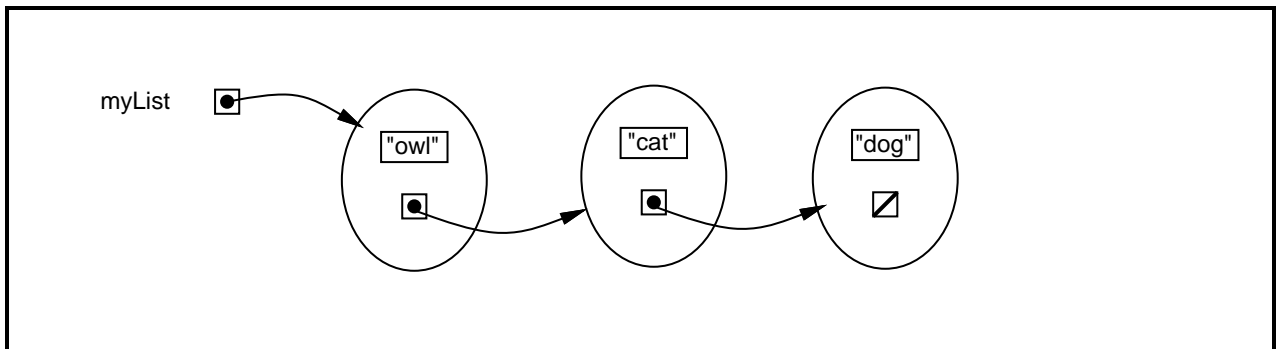
Question 6. Linked Lists

[12 marks]

(a) [3 marks] Consider the diagram below of a variable `myList` that contains a reference to a linked list of three items. On the diagram, draw the changes if a new list node containing "Hen" were added to the end of the list.



(b) [3 marks] Consider the same diagram of the same linked list. On the diagram, draw the changes if the node containing "Cat" were removed from the list.



Part 2

Question 7. Sorting Algorithms

[15 marks]

(a) [7 marks] The core of the Quicksort algorithm is the partition algorithm that separates smaller items from larger items in an array. Here is an implementation of partition that partitions the values from index `start` to index `stop` inclusive.

```
public static int partition(int start, int stop, String[] data){
    String pivot = data[start];
    int low = start - 1;
    int high = stop + 1;
    while (low < high){
        do high-- ; while (data[high].compareTo(pivot) > 0);
        do low++ ; while (data[low].compareTo(pivot) < 0);
        if (low >= high)
            break;
        String temp = data[low];
        data[low] = data[high];
        data[high] = temp;
    }
    return high;
}
```

Suppose the variable `data` contained the following array:

man	jay	ram	owl	eel	fly	kea	hen	ant	pig	tui	dab	cat	gnu	bat
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Show the contents of `data` and `mid` (the return value) after the statement

```
int mid = partition(0, 14, data);
```

`data =`

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

`mid =`

(Question 7 continued)

The usual version of insertion sort builds up a sorted region of the array from the left. The following version has been modified to build up the sorted region from the right. However, it has at least two bugs.

```
public void badInsertionSort (Object[] data) {  
    for (int i = data.length-2; i >= 0; i--) {  
        Object item = data[ i ];  
        int j = i-1;  
        while (j < data.length && (test.compare(item, data[ j ]) > 0)) {  
            data[ j-1 ] = data[ j ];  
            j++;  
        }  
        data[ j ] = item;  
    }  
}
```

(write your answers to part (c) in this box)

(b) [4 marks] Show the result of calling `badInsertionSort` on the following array:

yak	gnu	ant	egg	pig	hen	kit
0	1	2	3	4	5	6

0	1	2	3	4	5	6

(c) [4 marks] Identify and fix the bugs in `badInsertionSort`.
(Write your corrections on the code above).

(Question 8 continued)

(b) [7 marks] Complete the following definition of the `popLast` method that removes the last item in the list and returns that item. It throws a `NoSuchElementException` if the list is empty.

```
public Object popLast() {
```

```
}
```

Question 9. Hash Tables with probing

[14 marks]

The central issue in Open Address Hash tables is what to do when two items collide — ie, hash to the same index.

(a) [4 marks] Explain why linear probing is not a good method of dealing with collisions.

Consider the following method for adding an element to an open hash table that uses a second (different) hash function to resolve collisions:

```
public void addElement(Object val){  
    if (val == null) throw new NoSuchElementException();  
    int index = hash1(val);  
    if (data[index] != null)  
        index = hash2(val); // a different hash function  
    data[index] = val;  
}
```

(b) [4 marks] Explain the problems with this method.

(Question 9 continued)

(c) [6 marks] Suppose you had two hash functions:

```
public int hash1(Object item)
```

and

```
public int hash2(Object item, int i )
```

where `hash2` returned an integer that depended on both the item and the integer.

Write an effective method for using `hash2`, and explain why it is better than linear probing.

```
public void addElement(Object val){  
    if (val == null) throw new NoSuchElementException();
```

```
}
```

Explanation:

Question 10. Analysing the cost of Algorithms

[10 marks]

(a) [4 marks] Suppose a program uses an algorithm with an average case asymptotic cost of $O(\log(n))$. When the program is run on a case where $n = 1,000$, the measured running time is 500 milliseconds. When $n = 2,000$, the measured running time is 600 milliseconds. Give a reasonable estimate of the running time of the program on cases where $n = 8,000$ and $n = 1,024,000$. Justify your answer.

Estimated running times:

Justification:

Consider the following divide and conquer algorithm that computes a function of an array of numbers by dividing the array in half, computing the function on each half, and then multiplying the results.

```
public static int computeVal(int[] data){
    return computeVal(data, 0, data.length-1);
}
public static int computeVal(int[] data, int start, int stop){
    if (start == stop){
        System.out.println(data[start] + " * " + start);
        return data[start] * start;
    }
    else{
        int mid = (start+stop)/2;
        int high = computeVal(data, mid+1, stop);
        int low = computeVal(data, start, mid);
        return (high * low);
    }
}
```

(Question 10 continued)

(b) [2 marks] What value will be returned if `computeVal` is called on the following array:

9	10	8	4	6	5	2	7
0	1	2	3	4	5	6	7

value:

(c) [4 marks] If the size of the array is $n = 2^k$, exactly how many multiplications will `computeVal` perform, as a function of n ? Justify your answer.

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.

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
$\log_2(n)$	0	1	2	3	4	5	6	7	8	9	10

This page gives brief specifications of the interfaces that you may need to use in this test.

```
public interface Enumeration {
    public boolean hasMoreElements();
    public Object nextElement();
}

public interface Comparator {
    public int compare(Object o1, Object o2);
}

public interface BufferedReader{
    public String readLine();
}

public interface Collection {
    public boolean isEmpty ();
    public int size ();
    public Enumeration elements ();
}

public interface Indexed extends Collection{
    // Implementations: Vector
    public Object elementAt (int index);
    public void setElementAt (Object value, int index);
    public void addElementAt (Object value, int index);
    public void removeElementAt (int index);
}

public interface Bag extends Collection {
    // Implementations: ArrayBag, SortedArrayBag, BucketHashBag, OpenHashBag, LinkedListBag
    public void addElement (Object value);
    public boolean containsElement (Object value);
    public Object findElement (Object value);
    public void removeElement (Object value);
}

public interface Set extends Bag {
    // Implementations: ArraySet, SortedArraySet, BucketHashSet, OpenHashSet, LinkedListSet
    // same operations as Bag and the following
    public void unionWith (Bag aSet);
    public void intersectWith (Bag aSet);
    public void differenceWith (Bag aSet);
    public boolean subsetOf (Bag aSet);
}

public interface Map extends Collection {
    // Implementations: ArrayMap, SortedArrayMap, BucketHashMap, OpenHashMap, LinkedListMap
    public boolean containsKey (Object key);
    public Object get (Object key);
    public void removeKey (Object key);
    public void set (Object key, Object value);
}

public interface SortAlgorithm {
    // Implementations: MergeSort, InsertionSort, SelectionSort, BubbleSort, Partition
    // Constructors require a Comparator
    public void sort(Indexed collection);
}
```