

Name:

ID Number:

COMP103: Test: Model Answers

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 `java` 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
<code>Indexed</code>	Collection of values Order of values is not meaningful Duplicate values not allowed. SET
<code>Set</code>	Collection of key – value pairs Order of values is the order in which their keys were added. Duplicate values allowed
<code>Set</code>	Collection of values Order of values is the order in which the values were added. Duplicate values not allowed
<code>Set</code>	Collection of key – value pairs Order of values and keys is not meaningful Only one value may be associated with any key MAP
<code>Bag</code>	Collection of values Order of values is the “natural” sorting order of the values Duplicate values not allowed
<code>Bag</code>	Collection of values Order of values is not meaningful Duplicate values allowed BAG
<code>Bag</code>	Collection of values INDEXED Order of values is the order specified when the values were added. Duplicate values allowed
<code>Map</code>	Collection of key – value pairs Order of values is “natural” sorting order of the keys. Duplicate keys allowed, but values must be unique.

Question 2. Implementing ArrayBag

[9 marks]

The ArrayBag class implements the Bag interface using an unsorted array. It contains two fields:

<i>int</i> count,	the number of items in the Bag.
<i>Object</i> [] data	the array of items in the Bag.

(a) [4 marks] Complete the following `addElement` method that adds a new item to an ArrayBag. You may assume that the method `ensureCapacity()` will ensure that there is sufficient space in the `data` array.

```
public void addElement (Object item) {  
    if (item == null) throw new NoSuchElementException();  
    ensureCapacity();  
    data[count] = item;  
    count++;  
}
```

(b) [5 marks] Complete the following `containsElement` method that returns `true` if the item is in the ArrayBag, and returns `false` otherwise.

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

Question 3. Programming with Collections

[11 marks]

Suppose you are writing a program `CourseReporter` to write various reports on the courses and lecturers 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");  
    for (Enumeration e = allCourses.elements(); e.hasMoreElements();)  
        System.out.println(e.nextElement());  
}
```

(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);  
    for (Enumeration e = allCourses.elements(); e.hasMoreElements();){  
        Object course = e.nextElement();  
        if (lecturer.equals(organisers.get(course)))  
            System.out.println(course);  
    }  
}
```

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)

$O(n)$

- **SortedArraySet**
(ordered, array)

$O(\log(n))$

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

$O(n \text{ or } n/k)$

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

$O(1)$

(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:	faster with a sorted array Yes faster with an unsorted array about the same	
Justification: because we can use binary search.		
AddElement:	faster with a sorted array faster with an unsorted array Yes about the same	
Justification: because can simply put at end; no need to look at or touch other items		
RemoveElement:	faster with a sorted array faster with an unsorted array about the same Yes	
Justification: with sorted, can find quickly, but have to move everything up with unsorted, slow to find, but can quickly move last item down		

(Question 4 continued on next page)

(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 200,000 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: **80 seconds**

Justification: $O(n)$ **means when the size doubles, the time should double**

When the size increases by 10 (2,000,000/200,000), the time should increase by 10 ($8 \times 10 = 80$).gv

(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: **1200 seconds = 20 minutes.**

Justification: $O(n^2)$ **means that every time the size doubles, the time should increase by a factor of four.**

When the size increased from 5,000 to 1,000,000, the time should increase by 200^2 .

Therefore the time should be $30 \times 200 \times 200$ milliseconds.

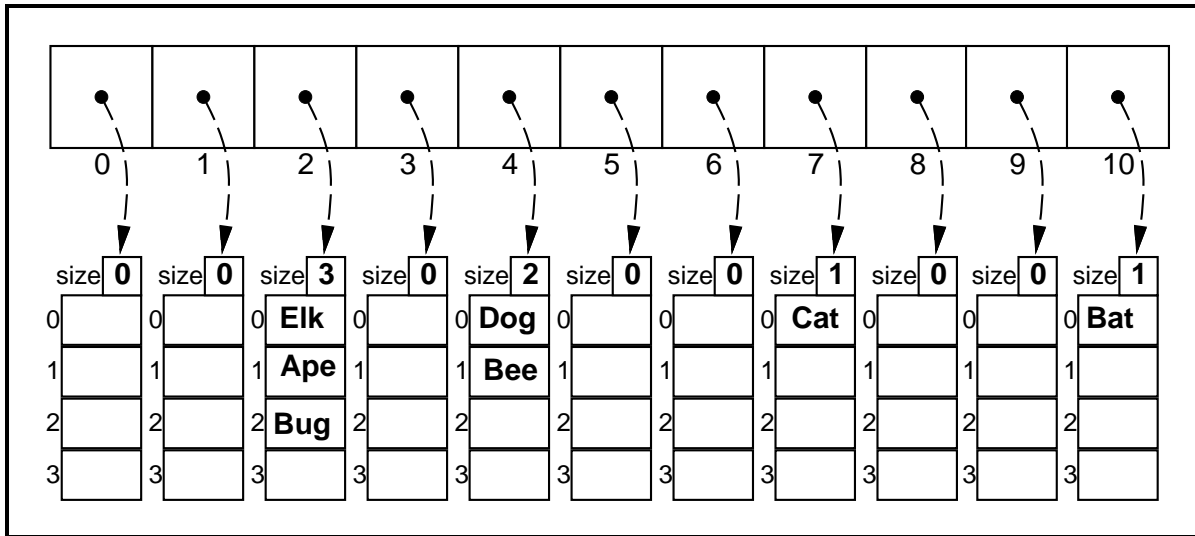
Question 5. Hash Tables

[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”, etc.

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



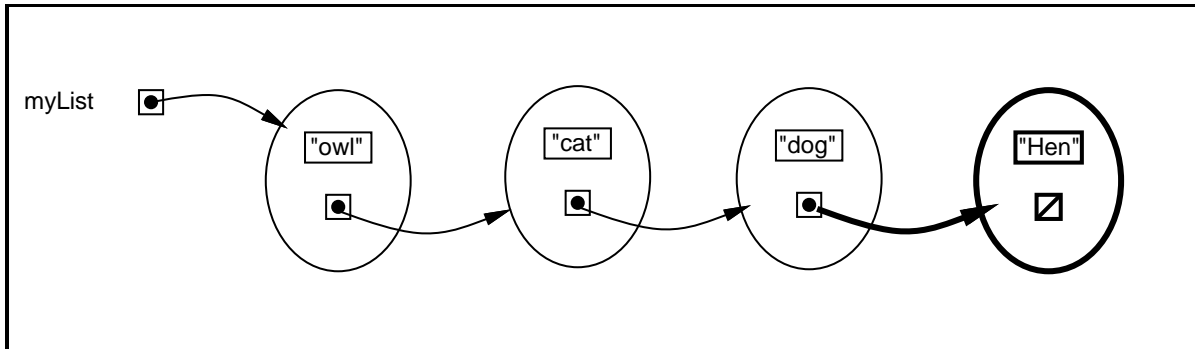
SPARE PAGE FOR EXTRA ANSWERS

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

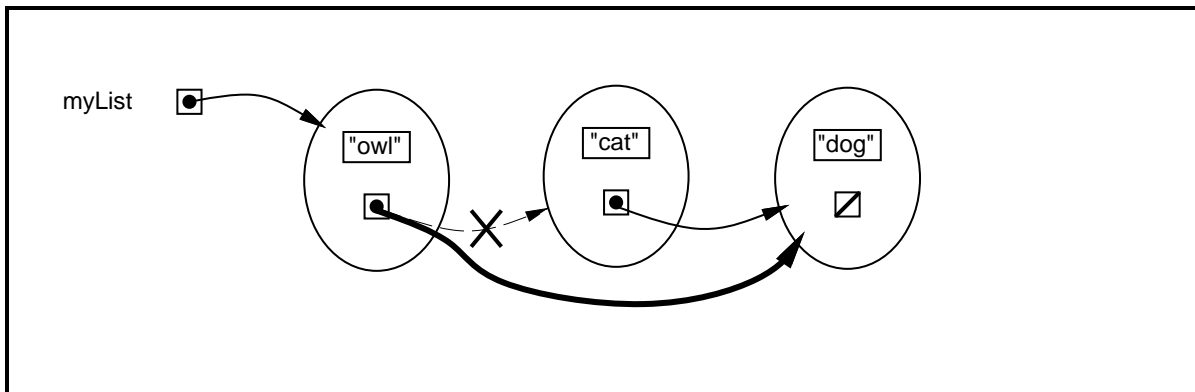
Question 6. Linked Lists

[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.



(Question 6 continued on next page)

(Question 6 continued)

(c) [6 marks] Consider the following declarations for `LispList` — a lisp style linked list (as in the diagrams for (a) and (b)).

```
public class LispList {  
    private Object value;  
    private LispList next;  
    public LispList(Object v, LispList n){  
        value = v;  
        next = n;  
    }  
    public int size () {  
        :  
    }
```

Complete the following code for the `size` method that returns the number of nodes in the `LispList`. You may use an iterative or recursive version.

```
public int size () {  
    if (next == null)  
        return 1;  
    else  
        return 1 + next.size();  
  
    or  
  
    int ans = 1;  
    for (LinkedList rest = this.next; rest != null; rest = rest.next)  
        ans++;  
    return ans;
```

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

(Question 7 continued on next page)

(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;                                should be: j = i+1
        while (j < data.length && (test.compare(item, data[ j ]) > 0)) {
            data[ j-1 ] = data[ j ];
            j++;
        }
        data[ j ] = item;                            should be: data[j-1] = 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

egg	egg	egg	egg	hen	hen	kit
0	1	2	3	4	5	6

It then throws an array out of bounds exception

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

Question 8. Linked Lists

[14 marks]

This question concerns a `SingleList` class that uses a linked list data structure to store a collection of items.

The `SingleList` class uses a singly linked list with a header containing references to the first and last nodes and the number of items in the list. Part of the implementation is given below, along with a diagram of a `SingleList` object containing three items.

```
public class SingleHeadList implements Collection {  
    private Node first;  
    private Node last;  
    private int count = 0;  
    :  
    private class Node {  
        public Object value;  
        public Node next;  
        public Node(Object o, Node n) {  
            value = o;  
            next = n;  
        }  
    }  
}
```

The diagram illustrates a singly linked list structure. On the left, a header box contains three fields: 'first' with a pointer to the 'Dog' node, 'last' with a pointer to the 'Cat' node, and 'count' with the value '3'. To the right, three nodes are shown in a vertical stack. Each node is a circle containing a rectangular box with the node's name ('Dog', 'Kea', or 'Cat') and a small square icon representing a pointer to the next node. Dashed arrows show the 'first' pointer from the header pointing to the 'Dog' node, the 'last' pointer pointing to the 'Cat' node, and the 'next' pointer from the 'Dog' node pointing to the 'Kea' node, and from the 'Kea' node pointing to the 'Cat' node.

You are to complete two methods for the `SingleList` class.

(a) [7 marks] Complete the following definition of the `popFirst` method that removes the first item from the list and returns that item. It throws a `NoSuchElementException` exception if there are no items in the collection.

```
public Object popFirst(){  
    if (first == null)  
        throw new NoSuchElementException();  
    Object ans = first.value;  
    first = first.next;  
    if (first == null)  
        last = null;  
    count--;  
    return ans;  
}
```

(Question 8 continued on next page)

(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(){
    if (last == null) throw new NoSuchElementException();
    Object ans = last.value;
    if (first == last){
        first = null;
        last = null;
    }
    else{
        Node nd = first;
        while (nd.next != last) nd = nd.next;
        last = nd;
        last.next = null;
    }
    count--;
    return ans;
}
```

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.

Because you tend to build up long runs of adjacent cells, and the runs start to grow fast, because any new item that hashes to anywhere in the run will extend the run. Long runs are bad because searching for an item in a run turns into linear search.

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.

This method will only consider two places to put an item. If it collides with an item at the first place, it looks at the other place. But if that also contains an item, the method will overwrite (and therefore lose) the item at the second place. Note that the items that collide on the first hash are unlikely to collide with each other on the second hash, but they may collide with other items.

(Question 9 continued on next page)

(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();  
    int index = hash1(val);  
    while (data[index] != null)  
        index = hash2(val, index);  
    data[index] = val;  
  
}
```

Explanation:

It uses the second hash function to determine where to probe next. Since this depends on both on the last place it looked and on the item, every item will follow a different path. Therefore, the table cannot build up “runs” in which all the items in the run follow the same path and therefore make the run grow.

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:

$n=16,000$: $600 + 3 \times 100 = 900$ milliseconds.

$n=1,000,000$: $600 + 9 \times 100 = 1.5$ seconds.

Justification:

$O(\log(n))$ means that every time the size doubles, the time should increase by the same amount. When the size doubled from 1000 to 2000, the time increased by 100 milli seconds. Each time the size doubles from 2000 to 4000 to 8000 and then to 16,000, the time should increase by 100 milliseconds. The size doubles 9 times to get from 2000 to 1,024,000

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 on next page)

(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: **0**
`computeVal(data)` \Rightarrow `computeVal(data, 0, 7)` \Rightarrow `cv(4,7) \times cv(0,3)` \Rightarrow
`[cv(6,7) \times cv(4,5)] \times [cv(2,3) \times cv(1,0)]` \Rightarrow
`[[cv(7,7) \times cv(6,6)] \times [cv(5,5) \times cv(4,4)]]` \times
`[[cv(3,3) \times cv(2,2)] \times [cv(1,1) \times cv(0,0)]]` \Rightarrow
`[[(7 \times 7) \times (2 \times 6)] \times [(5 \times 5) \times (6,4)]] \times [[(4 \times 3) \times (8 \times 2)] \times [(10 \times 1) \times (9 \times 0)]]` \Rightarrow
`[[49 \times 12] \times [25 \times 24]] \times [[12 \times 16] \times [10 \times 0]]` \Rightarrow (8 multiplications)
`[[588] \times [600]] \times [[192] \times [0]]` \Rightarrow (4 multiplications)
`[352800] \times [0]` \Rightarrow (2 multiplications)
`0` \Rightarrow (1 multiplication)
total of 15 multiplications
Note that it isn't necessary to do all the multiplications - if there is a 0 anywhere in the product, then the answer is 0!!!

(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.

$2n-1$.
There is one multiplication per call.
Each call, except for the base case will call `computeVal` twice
There is one call on an array of size n ,
two calls on size $n/2$,
4 calls on size $n/4$,
8 calls on size $n/8, \dots$
 $2^k = n$ calls on size $\frac{n}{2^k} = 1$.
 $1 + 2 + 4 + 8 + \dots + n = 2n - 1$
