Name: ....

ID Number: ....

COMP103: Test

10 Sept, 2003.

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 think some question is unclear, ask for clarification.
- We expect the later questions to be more difficult than the earlier ones.

Questions	**Marks**	
1. Collection Types	[12]	
2. Hash Tables	[10]	
3. Asymptotic or "Big O" analysis	[18]	
4. Programming with Collections	[21]	
5. Cost of Programs	[13]	
6. Timing Performance of Bags	[16]	
7. Linked Lists	[16]	
8. Union for Sets	[14]	

There is documentation on the jds types at the end of the exam paper. Total:

Possibly useful formulas:

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

Question 1. Collection Types [12 marks]

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

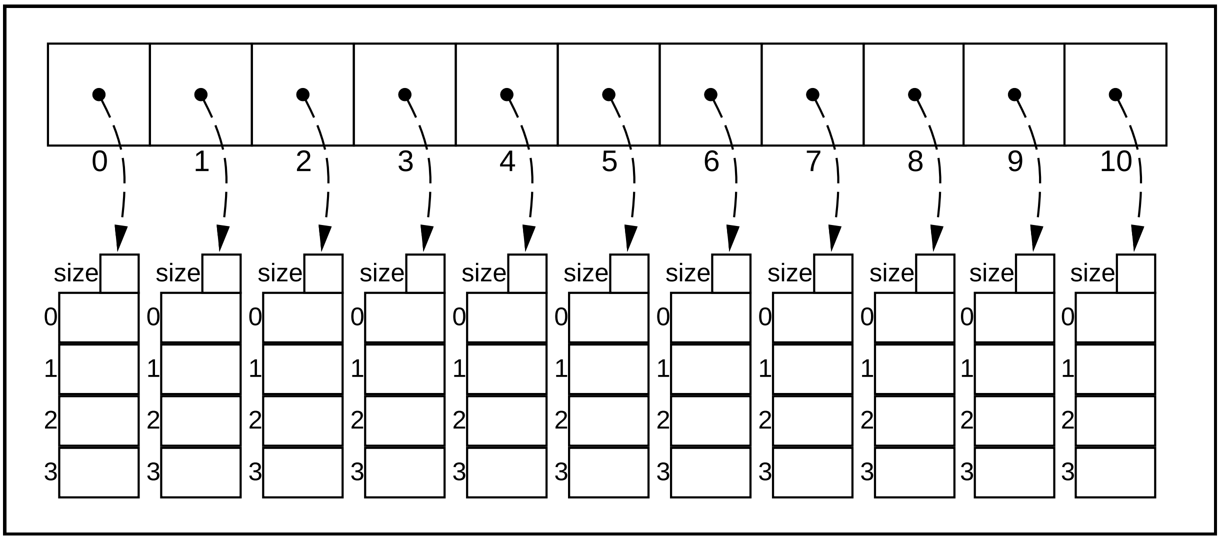
Types	Descriptions
	Collection of values Order of values is not meaningful Duplicate values not allowed.
Set	
	Collection of key – value pairs Order of values is the order in which their keys were added. Duplicate values allowed
Indexed	Collection of values Order of values is the order in which the values were added. Duplicate values not allowed
	Collection of key – value pairs Order of values and keys is not meaningful Only one value may be associated with any key
Map	
	Collection of values Order of values is the "natural" sorting order of the values Duplicate values not allowed
Bag	Collection of values Order of values is not meaningful Duplicate values allowed
	Collection of values Order of values is the order in which 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 2. Hash Tables [10 marks]

(a) [5 marks] The diagram below shows a Set implemented using a BucketHashSet with 11 buckets. Each bucket is implemented as an ArraySet.

Show (on the diagram) the contents of the Set when the following seven items are added to the set in the order "Hen", "Owl", "Cat", etc.

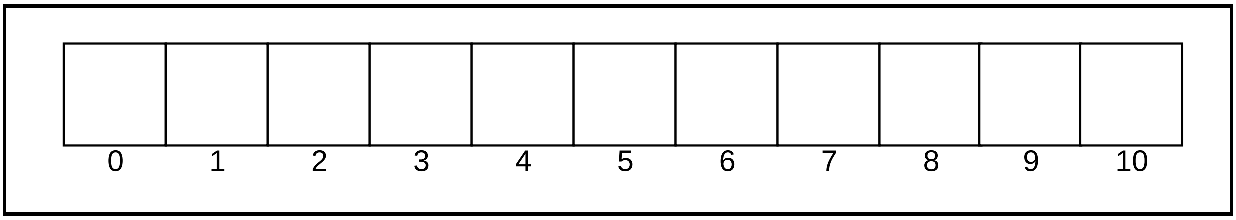
item:	Hen	Owl	Cat	Cow	Bee	Emu	Yak
hash value:	9	1	10	4	9	4	0



(b) [5 marks] The diagram below shows a Set implemented using an OpenHashSet that uses linear probing.

Show (on the diagram) the contents of the Set when the following seven items are added to the Set in the order "Hen", "Owl", "Cat", etc.

item:	Hen	Owl	Cat	Cow	Bee	Emu	Yak
hash value:	9	1	10	4	9	4	0



Question 3. Asymptotic or "Big O" analysis [18 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( )
- SortedArraySet (ordered, array) — O( )
- BucketHashSet (Hashtable with k buckets where each bucket is an ArrayBag) — O( )
- OpenHashSet (Hashtable with open addressing, linear probing, and guaranteed less than 90% full). — O( )

(b) [5 marks] What are the average case asymptotic costs ("Big O") of the following sorting algorithms?

- Bubble Sort — O( )
- Insertion Sort — O( )
- Merge Sort — O( )
- Quicksort — O( )
- Selection Sort — O( )

(Question 3 continued on next page)

(Question 3 continued)

(c) [3 marks] Suppose a program uses an algorithm with an average case asymptotic cost of $O(n)$. When the program is run on a case where $n = 1,000$, the measured running time is 4.5 seconds. When $n = 2,000$, the measured running time is 9.0 seconds.
Give a reasonable estimate of the running time of the program on a case where $n = 4,000$.
Justify your answer.

Estimated running time: Justification:

(d) [3 marks] Suppose a program uses an algorithm with an average case asymptotic cost of $O(n^2)$. When the program is run on a case where $n = 1,000,000$, the measured running time is 3.5 minutes.
Give a reasonable estimate of the running time of the program on a case where $n = 4,000,000$.
Justify your answer.

Estimated running time: Justification:

(e) [3 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 0.7 seconds. When $n = 2,000$, the measured running time is 0.9 seconds.
Give a reasonable estimate of the running time of the program on a case where $n = 8,000$.
Justify your answer.

Estimated running time: Justification:

Question 4. Programming with Collections [21 marks]

For this question, you are to complete three methods of a program that draws a collection of balloons on the screen. Each balloon is specified by a colour, a size, and a position (x, y). The program reads the specifications of the balloons from a file, and then draws them on the screen. The balloons should be drawn in order of size, so the largest balloons are at the back and the smallest balloons at the front.

The main part of the progam is the DrawBalloons class which has two fields:

```
private DrawingCanvas canvas;
private Indexed balloons;     // Indexed collection of Balloon objects
```

The program also include a Balloon class; its constructor and methods are shown on the facing page.

(a) [6 marks] Complete the renderBalloons method that renders each Balloon in balloons on the canvas:

```
public void renderBalloons(){
    canvas.clear(false);









    canvas.display();
}
```

(b) [7 marks] Complete the readBalloonFile method that reads the specifications of balloons from a file then constructs and stores the balloons in the balloons field. fname is the name of the file to read from. Assume that balloons may be **null** when the method is called.

```
public void readBalloonFile(String fname){
    try {
        BufferedReader f = new BufferedReader(new FileReader(fname));













    }
    catch(IOException ex) { }
}
```

(Question 4 continued on next page)

(Question 4 continued)

(c) [8 marks] Complete the following sortBalloons and compare methods. sortBalloons sorts the collection of balloons using a SortAlgorithm (see the documentation on page 17). compare compares two Balloons based on their size.

```
public void sortBalloons(){

}

public class BalloonComparator implements Comparator{
    /** o1 should come before o2 if o1 is larger than o2 */
    public int compare(Object o1, Object o2){

    }
}
```

The Balloon Class:

```
public class Balloon{
    ⋮
    /** Constructs a Balloon object, given a String specifying the color, size, and position.*/
    public Balloon (String params){...}

    /** Returns the size of the balloon */
    public int getSize(){...}

    /** Renders the balloon on a canvas. Does not redisplay the canvas. */
    public void render (DrawingCanvas canvas){...}
    ⋮
}
```

Question 5. Cost of programs [13 marks]

In each of the following, express your answer as a function of n. Do NOT use Big-O notation.

(a) [3 marks] Consider the following program fragment:

```
for (int i = 0; i < n; i++)
    System.out.println(i*i);
```

How many lines will be printed out by this piece of code?

(b) [3 marks] Consider the following program fragment:

```
for (int i = 1; i <= n; i++)
    for (int j = i; j <= n; j++)
        System.out.println(i + j);
```

How many lines will be printed out by this piece of code?

(c) [4 marks] Consider the following program fragment, assuming that data is an array of n Strings.

```
for (int i = 1; i < data.length; i++)
    if (data[ i ].compareTo(data[ i–1]) < 0)
        System.out.println(data[ i ]+ " is out of order");
```

In the worst case, how many lines will be printed out by this piece of code?

In the best case, how many lines will be printed out by this piece of code?

(d) [3 marks] Consider the following program fragment, assuming that data is an array of Strings of length n, and that n is a power of 2.

```
int i = data.length;
while ( i > 0){
    System.out.println(data[ i–1]);
    i = i/2;
}
```

How many lines will be printed out by this piece of code?

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. Timing Performance of Bags [16 marks]

The tables below give the results of a timing test on four different implementations of Bag (all run on the same computer):

- ArrayBag: an array of items, not kept in order
- SortedArrayBag: an array of items, kept in sorted order
- BucketHashBag: a hash table of 1000 buckets, where each bucket was a SortedArrayBag
- OpenHashBag: a hash table using open addressing, guaranteed to never be more than 80% full.

For each implementation of Bag, the test measured the average time (in microseconds) it took to search for an item (both for items in the Bag and for items not in the Bag) and the average time it took to add a new item to the bag when the bag contained n items. For each implementation, the test was run with four different values of n.

For each table, say which implementation of Bag it is, give the asymptotic (Big-O) cost of searching and adding in that implementation, and a brief justification of why the table must be from that implementation of Bag.

(a) [4 marks] Table 1:

n	μS per Search	μS per Add
5000	0.261	0.23
10000	0.371	0.23
20000	0.521	0.23
40000	0.631	0.31

Implementation:
Big-O costs:
Justification:

(b) [4 marks] Table 2:

n	μS per Search	μS per Add
5000	1.8	41
10000	2.5	100
20000	3.6	481
40000	4.8	1738

Implementation:
Costs:
Justification:

(Question 6 continued on next page)

(Question 6 continued)

(c) [4 marks] Table 3:

n	μS per Search	μS per Add
5000	0.41	0.51
10000	0.71	0.81
20000	1.06	1.07
40000	1.73	1.51

Implementation:

Costs:

Justification:

(d) [4 marks] Table 4:

n	μS per Search	μS per Add
5000	133	0.1
10000	363	0.1
20000	1159	0.1
40000	2520	0.1

Implementation:

Costs:

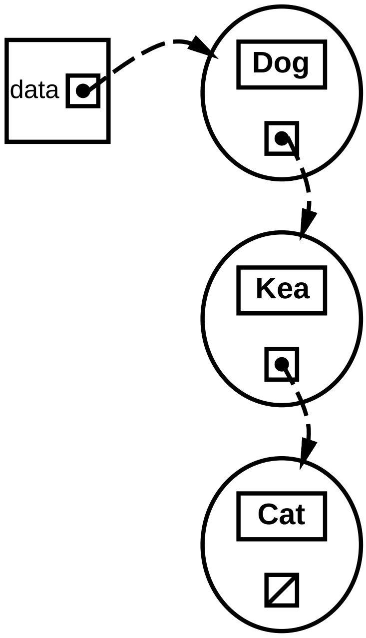
Justification:

Question 7. Linked Lists [16 marks]

This question concerns the SingleList class that implements a collection using a linked list data structure to store the items. Like the Indexed type, the order of items in a SingleList is significant.

The SingleList class uses a singly linked list with no tail pointer. Part of the implementation is given below, along with a diagram of a SingleList object containing three items.

```
public class SingleList implements Collection {
    private Node data;
    ⋮
    private class Node{
        public Object value;
        public Node next;
        public Node(Object val, Node n){
            value = val;
            next = n;
        }
    }
}
```



You are to complete several of the method definitions for SingleList class.

(a) [4 marks] Complete the following definition of the removeFirst method that removes the first item of the collection. It throws a NoSuchElementException exception if there are no items in the collection.

```
public void removeFirst(){

}
```

(Question 7 continued on next page)

(Question 7 continued)

(b) [4 marks] Complete the following definition of the elementAt method that returns the *i*'th value in the list (counting from 0). It throws a NoSuchElementException if i is greater than or equal to the number of values in the list.

```
public Object elementAt (int i) {



}
```

(c) [4 marks] Complete the following definition of the addLast method that adds an item at the end of the collection. Note that the first element in the collection is at position 0.

```
public void addLast (Object val) {



}
```

(Question 7 continued on next page)

(Question 7 continued)

(d) [4 marks] Complete the following definition of the removeLast method that removes the item at the end of the collection, making the collection one item smaller.

```
public void removeLast(){



}
```

Question 8. Union for Sets [14 marks]

The union of two Sets is the the set of items that are in one or other or both of the sets. Computing the union of two sets can be done efficiently by the following algorithm:

Copy each Set to an Indexed collection,
Sort each Indexed collection,
Merge the two sorted collections, keeping only one copy of
items that are in both collections.

Complete the following union method that performs the last step of this algorithm: given two (sorted) Indexed collections, it constructs a Set of all the items that are in either collection, using a Merge-like algorithm. The cost of your method should be ($O(n)$), where n is the number of items in the two collections. Note, adding an item to the end of a SortedArraySet is $O(1)$.

```
public static Set union (Indexed a, Indexed b, Comparator test){
    //Assumes that a and b are sorted
    SortedArraySet union = new SortedArraySet(test);

    return union;
}
```

****

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.

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 v, int index);
    public void addElementAt (Object val, int index);
    public void removeElementAt (int index);
}
public interface Bag extends Collection {
    // Implementations: ArrayBag, SortedArrayBag, BucketHashBag, OpenHashBag
    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
    // operations of 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
    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);
}
```