



EXAMINATIONS — 2004

END-YEAR

COMP 103

Data Structures and Algorithms

Time Allowed: 3 Hours

Instructions: There are a total of 180 marks on this exam.
Attempt **all** questions.
Calculators may be used.
Non-electronic foreign language translation dictionaries may be used.
Write your answers in the boxes in this test paper and hand in all sheets.
There is documentation on java and jds interfaces at the end of the exam paper.

Questions	Marks
1. Basic Questions	[28]
2. Programming with Collections	[16]
3. Implementing Linked Lists	[20]
4. Using Stacks	[8]
5. Implementing Queues	[22]
6. Trees	[32]
7. Priority Queues	[28]
8. Graphs	[26]

Student ID:

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.

Student ID:

Question 1. Basic Questions

[28 marks]

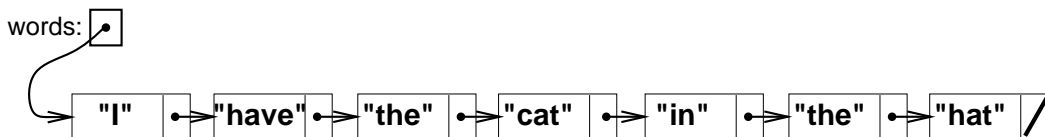
(a) [2 marks] Which collection type keeps items in order, but only allows you to add and remove elements from one end?

(b) [2 marks] What is the asymptotic cost of searching for an item in a Bag implemented using an unsorted array?

(c) [2 marks] Name a sorting algorithm with an asymptotic cost of $O(n \log(n))$.

(d) [2 marks] What is the asymptotic cost of adding an item to a set implemented using a bucket hash table, where the buckets are unsorted ArraySets.

Consider the diagram below of a variable `words` containing the reference of a linked list. Assume that each node of the list has the fields `value` and `next`.



(e) [2 marks] What is the value of `words.value`?

(f) [2 marks] What is the value of `words.next.next.value`?

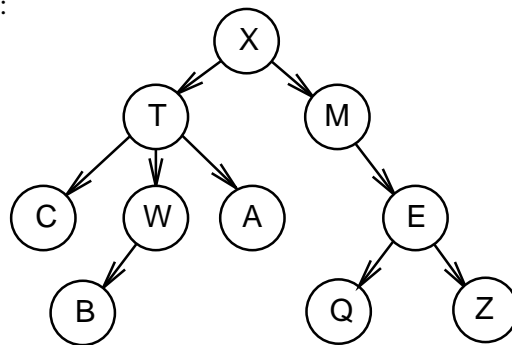
Student ID:

(g) [2 marks] Suppose we add the item “A” to an empty queue, and then add “B”, and then “C”. If we now remove an item from the queue, which items will still be on the queue?

(h) [2 marks] If we are adding a value to a Hash Table using linear probing (“open hashing”), and the first cell we look at is already full, where do we look next?

(i) [2 marks] What is the maximum number of nodes that could be in a binary tree of depth 2 (three levels of nodes)

Consider the following general tree:



(j) [2 marks] What value is in the root node of the tree above?

(k) [2 marks] What value is in the parent of the node with the value “E”?

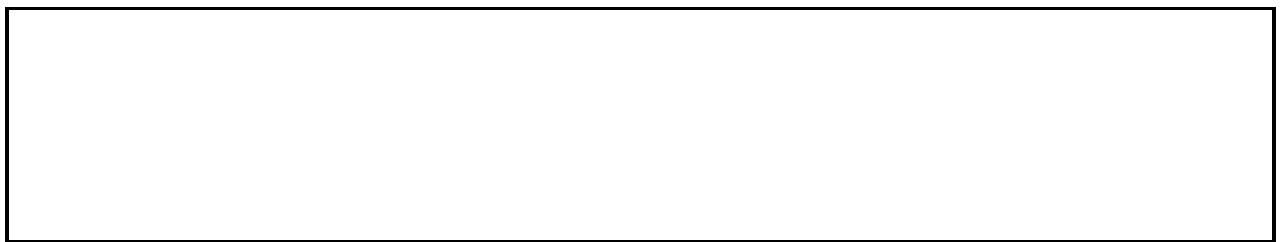
(l) [2 marks] How many leaf nodes does the tree have?

Student ID:

(m) [2 marks] Draw a graph containing four nodes and five edges.



(n) [2 marks] What is the difference between a directed graph and an undirected graph?



Student ID:

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.

Student ID:

Question 3. Implementing Linked Lists

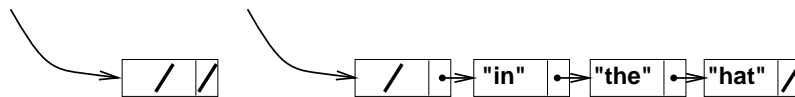
[20 marks]

Suppose you want to implement a Collection using a linked list.

(a) [3 marks] State a problem that arises if we implement the Collection using a “lisp-style” linked list with no header and where the empty collection is represented by **null**.

Suppose you decide to use lists with no header but with a dummy node – where every list has a node at the front that contains the **null** value, which is not part of the collection.

For example, here is an empty Collection, and a Collection with three items.



The DLList class contains the following fields and constructors:

```
public class DLList implements Collection{  
    private Object value;  
    private DLList next;  
  
    /* Create an empty DLList collection */  
    public DLList(){  
        value = null;  
        next = null;  
    }  
  
    /* Create a new node. */  
    private DLList(Object val, DLList nx){  
        value = val;  
        next = nx;  
    }  
}
```

(b) [2 marks] Draw two diagrams below to show the structure of the collections if the element “cat” were added as the first element to each of the DLLists in the diagrams above.

Student ID:

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.

Student ID:

Question 5. Implementing Queues

[22 marks]

The `ArrayQueue` class is a queue that holds the data items in an array. An `ArrayQueue` object has two indexes: the index of the cell containing the item at the front of the queue and the index of the first cell after the item at the back of the queue.

The code for `ArrayQueue` is shown below:

```
public class ArrayQueue {  
    private Object[] data;  
    private int front;  
    private int back;  
  
    public ArrayQueue() {  
        this.data = new Object[13];  
        this.front = 0;  
        this.back = 0;  
    }  
  
    public void enqueue(Object val) {  
        if (this.isFull())  
            throw new RuntimeException("Queue Full.");  
        this.data[this.back++] = val;  
    }  
  
    public Object dequeue() {  
        if (this.isEmpty())  
            throws new RuntimeException("Queue Empty.");  
        return this.data[this.front++];  
    }  
  
    public boolean isFull() {  
        return (this.back == this.data.length);  
    }  
  
    public boolean isEmpty() {  
        return (this.front == this.back);  
    }  
}
```

(a) [6 marks] Suppose a new queue is created, and then the following seven operations are performed:

enqueue("A"), enqueue("B"), dequeue(),
enqueue("C"), enqueue("D"), dequeue(), dequeue().

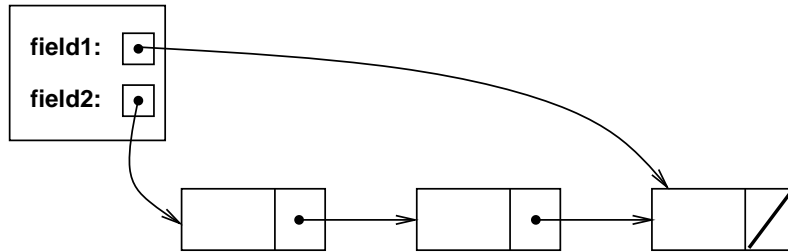
Show the resulting state of the queue:

Front:	<input type="text"/>																										
Back:	<input type="text"/>																										
Data:	<table border="1"><tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td></tr></table>														0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	2	3	4	5	6	7	8	9	10	11	12															

Student ID:

(Question 5 continued)

(d) [4 marks] Suppose you decided to implement a queue using a linked-list data structure with a header containing two fields with pointers to the nodes at each end of the linked-list, as shown below:



Which field would you use for the back of the queue (where items are enqueued) and which field would you use for the front of the queue (where items are dequeued from)? Briefly explain your choice.

Front:
Back:

Student ID:

Question 6. Trees

[32 marks]

(a) [3 marks] What property must a binary tree have to be a binary search tree?

(b) [3 marks] What property must be true of a binary search tree (BST) for insertion/access/deletion to have $O(\log(n))$ complexity in all cases?

(c) [2 marks] Show the resulting tree when the following items are inserted (in this order) into an empty binary search tree:

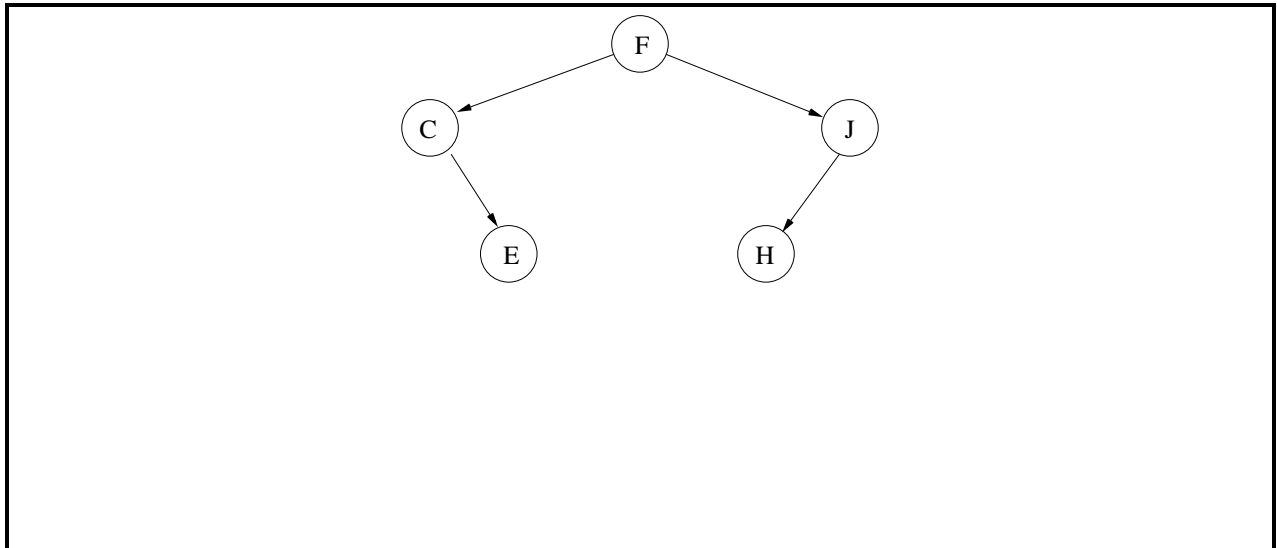
“A”, “B”, “C”, “D”, “E”, “F”, “G”, “H”

Student ID:

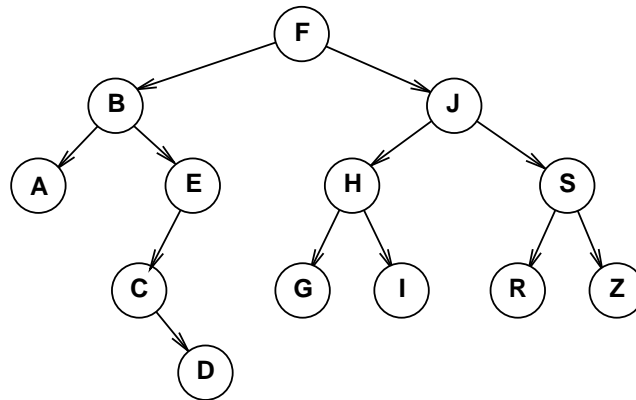
(Question 6 continued)

(d) [4 marks] Show the resulting tree when the following items are inserted (in this order) into the binary search tree below:

“P”, “L”, “G”, “I”, “D”, “B”



(e) [4 marks] Consider the following binary search tree:



State the order that the values of this tree would be visited during a post-order, depth-first traversal of the tree.

(f) [5 marks] The breadth-first traversal algorithm uses a collection to store the nodes that are to be visited. This collection keeps the nodes in the order that they are to be visited. Draw the state of this collection during a breadth-first traversal of the binary tree above at the point when node “C” has just been added to the collection.

Student ID:

Question 7. Priority Queues

[28 marks]

Consider the following two implementations of a priority queue.

Implementation 1: An array of queues, with the array indexed by the priority level, and the queues having $O(1)$ complexity for both enqueue and dequeue operations.

Implementation 2: A heap, with $O(\log(n))$ complexity for both enqueue and dequeue operations.

(a) [4 marks] Which implementation is better when the number of possible priority levels is small? Explain why.

(b) [4 marks] Which implementation is better when the number of possible priority levels is large? Explain why.

(c) [3 marks] Suppose a priority queue is implemented as an array of queues. Draw the priority queue after the following data has been inserted into an empty priority queue. (The letter is the data, and the number in brackets is the priority.)

Data Items: A (1), B (5), C (3), T (4), Y (2), R (5), H (2), S (3)

1	2	3	4	5	6	7	8

(d) [3 marks] List the values in the order they would be dequeued from the priority queue in question (c).

Student ID:

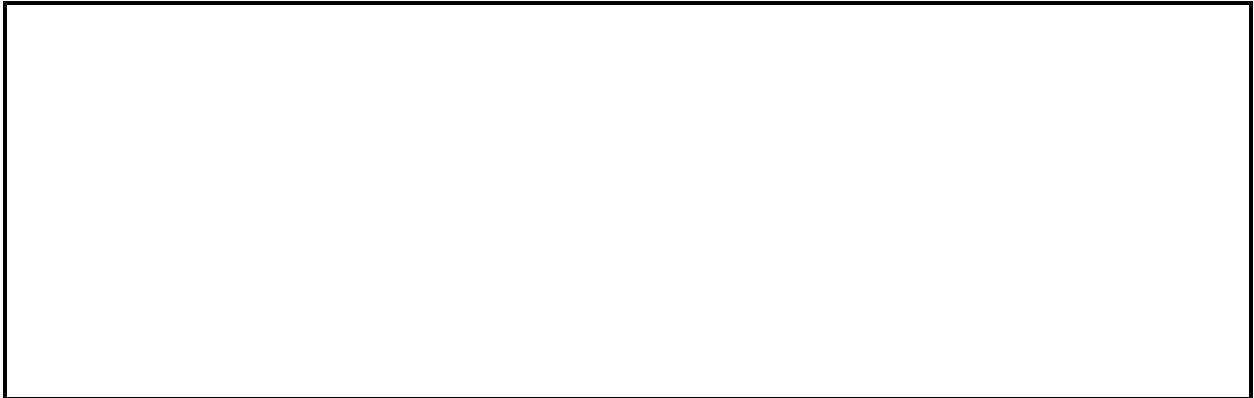
(Question 7 continued)

(e) [4 marks]

Suppose a priority queue is implemented as a heap. Draw the heap after the following data items have been inserted. The letter is the value, and the number in brackets is the priority.

Data Items: A (1), B (5), C (3), T (4)

Hint: draw this heap as a tree rather than as an array.



(f) [3 marks] Draw the new heap after the following data items have been inserted into the heap you created in question (e).

Data Items: Y (2), R (5), H (2), S (3)



Student ID:

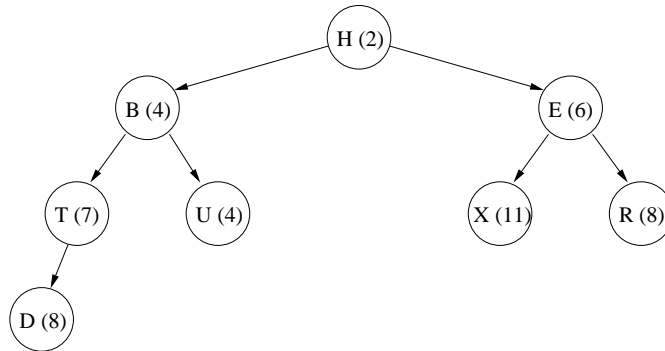
SPARE PAGE FOR EXTRA ANSWERS

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

Student ID:

(Question 7 continued)

Suppose a heap contains the following items:



(g) [4 marks]

Draw the new heap after *dequeue()* has been called two times on the heap above.

(h) [3 marks]

Draw the heap after *dequeue()* has been called another two times on the heap created in question (g).

Student ID:

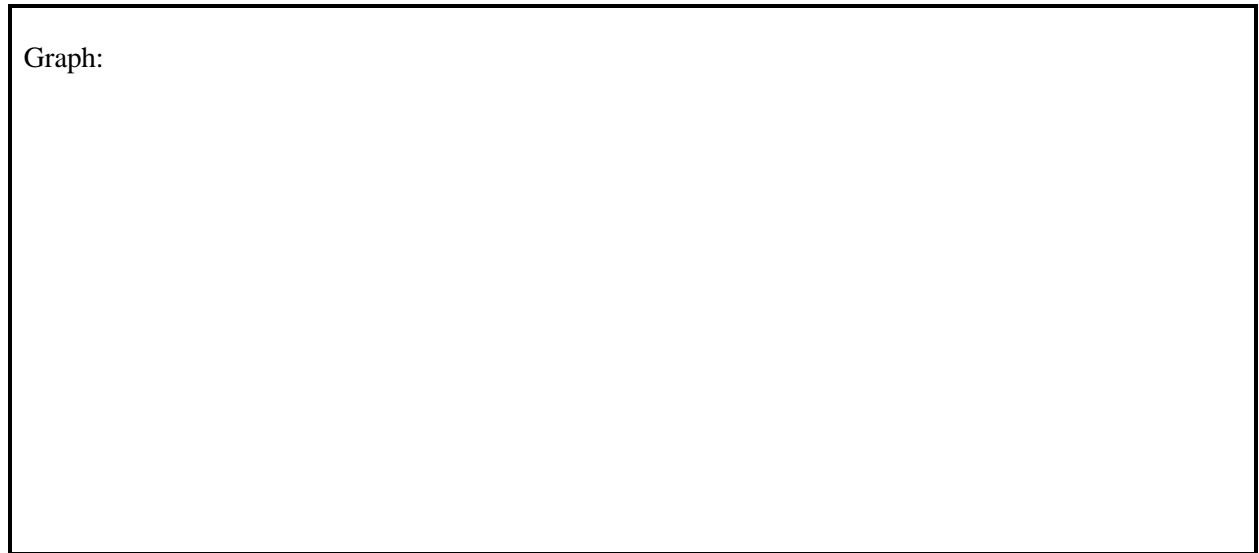
Question 8. Graphs

[26 marks]

(a) [6 marks] Draw the graph that is represented by the following undirected, weighted adjacency matrix. Assume that a blank square represents no edge.

	A	B	C	D	E	F	G
A			4				
B			1			3	
C	4	1		6		2	4
D			6				
E							
F		3	2				
G			4				2

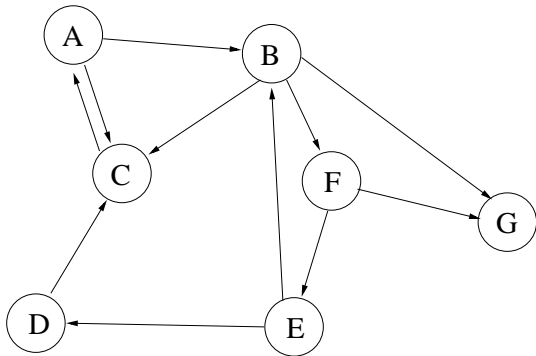
Graph:



Student ID:

(Question 8 continued)

(b) [6 marks] Draw an adjacency list representation of the following directed, unweighted graph.



(c) [5 marks] State one reason why a tree traversal algorithm might not work on a directed graph.

(d) [5 marks] To turn a tree traversal algorithm into a graph traversal algorithm, one change you would usually make is to use an additional Set data structure. What would this Set contain at any given point during a graph traversal?

(e) [4 marks] Give an example of a directed, weighted graph where a breadth-first traversal would not find the least-cost path.

Appendix

Brief specifications of the `jdk` interfaces that you may need to use in this test.

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

```
public interface Comparable {  
    public int compareTo(Object o);  
}
```

```
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 Stack extends Collection {  
    // Implementations: Vector, .....  
    public void addLast (Object value);  
    public Object getLast ();  
    public void removeLast ();  
}
```

```
public interface Queue extends Collection {  
    // Implementations:  
    public void addLast (Object value);  
    public Object getFirst ();  
    public void removeFirst ();  
}
```

```
public interface FindMin extends Collection {  
    // Implementations: HeapPQueue  
    public void addElement (Object value);  
    public Object getFirst ();  
    public void removeFirst ();  
}
```

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