Aston University

School of Engineering and Applied Science

CS2310 Data Structures and Algorithms with Java

Stage Two Examination

CLOSED BOOK

Date: **19th January 2017**
Time: **09:30 – 11:00**
Duration: **1.5 hours**

Instructions to Candidates

1. **Answer ALL questions from Section A (50 marks).**
2. **Section A contains FIVE questions.**
3. **Answer TWO questions from Section B (50 marks).**
4. **Section B contains THREE questions, each question is worth 25 marks.**
5. **Use of calculators is NOT allowed.**
6. **In all questions that involve the writing of Java program code, a few MINOR syntactic errors will not be penalised. Program code that contains more substantial errors may still gain a substantial proportion of the available marks provided the intended meaning of the code is clear. Thus candidates are advised to include appropriate comments that briefly explain the intended meaning of their code.**

Materials provided

1. **Answer booklets**

This exam paper cannot be removed from the exam room.

This examination is subject to the Examination Regulations for Candidates REG/08/415(2)

Section A — Answer ALL questions in this section

1. a) Consider the following phrases:

   - executes in quadratic time
   - executes in linear time
   - executes in log-linear time
   - executes in constant time
   - executes in logarithmic time
   - executes in exponential time

   Making use of the above phrases, state the meaning of EACH of the following expressions in the **Big-O** notation:

   i) $\mathrm{O}(1)$

   ii) $\mathrm{O}(3^{\mathrm{n}})$

   iii) $\mathrm{O}(\log\ \mathrm{n})$

   (3 marks)

   b) Making use of the above phrases, state the time complexity of EACH of the following **growth functions**:

   i) $T(n) = 13n\ log_2\ n + 55n$

   ii) $T(n) = 3n^2 + \frac{2^n}{4500} + 5$

   iii) $T(n) = 2.5n^2 + 46$

   (3 marks)

2. Consider the following Java code for computing the sum from `1` to `n`:

Beginning of code

```
1 public int sum(int n)
2 {
3     return n*(1+n)/2;
4 }
```

End of code

Making use of a suitable **Big-O** notation, explain in some detail the time complexity of method `sum`.

(5 marks)

3. a) State the purpose of a **hash function** in relation to a **hash table**.

(2 marks)

b) Briefly describe how a hash function which uses **extraction** may work.

(3 marks)

c) Define what is meant by **dynamic resizing**. Explain why **dynamic resizing** should be avoided when using a **hash table**.

(5 marks)

4. a) Illustrate how the **insertion sort** algorithm works when used to sort the following names in alphabetical order:

Ali Betty Katy Felix Ben

(8 marks)

b) The **binary search** algorithm can only be applied to a certain type of search pools. State the crucial characteristic that such search pools must possess.

(2 marks)

5. a) In data structures and algorithms, what does the abbreviation **ADT** stand for?

(1 mark)

b) With the aid of a suitable diagram, briefly describe what is meant by the ADT **stack**.

(5 marks)

c) Write down the complete code, including appropriate comments, for a generic Java interface to model a **stack ADT**.

(9 marks)

d) State ONE difference and ONE similarity between the data structures **map** and **set** in terms of their structures.

(4 marks)

END OF SECTION A

Section B — Answer TWO questions in this section

Each question in this section carries 25 marks.

6. Consider the following partial implementation of a **priority queue** and the interface `Prioritised`.

   Package `dsaj` has been described in the lecture notes. Class `LinkedQueue` models a normal **queue** data structure implemented using a linear linked structure. It is assumed that class `LinkedQueue` has been fully implemented and package `dsaj` contains class `LinkedQueue`.

Listing 1: Class `PriorityQueue`

```
1  package dsaj;
2  public class PriorityQueue<T extends Prioritised> implements QueueADT<T> {
3          private QueueADT<T>[] queueList;
4          // priorities run from 0 .. maxPriority inclusive
5          private final int maxPriority;
6          private int count;    // number of elements in this priority queue
7
8          /* constructs a new empty priority queue */
9          public PriorityQueue(int maxPriority) {
10                 this.maxPriority = maxPriority;
11                 queueList = (QueueADT<T>[]) new Object[maxPriority + 1];
12                 for (int i = 0; i <= maxPriority; i++) {
13                         queueList[i] = new LinkedQueue<T>();
14                 }
15                 count = 0;
16         }
17
18         /* removes and returns an element from the priority queue */
19         public T dequeue() {
20                 // detailed implementation omitted
21         }
22
23         /* adds the specified element to the queue */
24         public void enqueue(T element) {
25                 // detailed implementation omitted
26         }
27
28         // Other implementation details omitted
29 }
```

(question continues on next page. . .)

(Question 6 continued. . .)

Listing 2: Interface `Prioritised`

```
1 package dsaj;
2 public interface Prioritised {
3   int getPriority();
4   void setPriority(int level);
5 }
```

a) Making use of a suitable diagram, describe what is meant by a **queue** data structure.

(4 marks)

b) Describe ONE similarity and ONE difference between a normal **queue** and a **priority queue**.

(4 marks)

c) Class `PriorityQueue` implements a **priority queue** using an array of linked lists (see Line 13 in Listing 1). Explain why such an implementation is better, in terms of time efficiency, than the implementations using a single linked list or a single array.

(5 marks)

d) Complete the implementation of the method `enqueue` in class `PriorityQueue`.

(4 marks)

e) Complete the implementation of the method `dequeue` in class `PriorityQueue` including Java code for any subsidary helper method(s) that may be necessary.

(8 marks)

7. a) Write Java code to define a generic Java interface `IndexedListADT`. This interface should specify at least SIX typical operations of an **indexed list**. Instances of classes which implement `IndexedListADT` must also support the operation of an **enhanced** `for` statement.

(9 marks)

b) Briefly describe TWO distinct ways to implement an indexed list.

(4 marks)

c) Explain what is meant by an **ordered list**, highlighting the differences from an **indexed list**, in particular in terms of their typical operations.

(6 marks)

d) Consider the following definition for class `ActorRole`. Class `ActorRole` models a character in a film, which an actor plays.

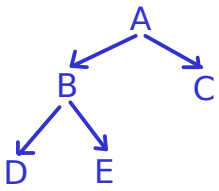
```
1  public class ActorRole
2  {
3      private String name;  // the name of an actor
4      private String role;  // the role played the actor
5      private int billingOrder;
6
7      public String getName() {
8        return name;
9      }
10
11     public String getRole() {
12       return role;
13     }
14
15     public int getBillingOrder() {
16       return billingOrder;
17     }
18
19     // Details of constructor, other method and fields omitted
20 }
```

Suppose a film database application is to use an **ordered list** to store the collection of `ActorRole` objects for each film. A brochure for each film is to be printed with the actors listed in their billing order. If two actors have the same billing order, they are listed alphabetically by their role in the film. Write out the necessary modifications for class `ActorRole` to enable such a film brochure to be prepared.

You should use the given line numbers to clearly indicate the location(s) of your modifications.

(6 marks)

8. a) Consider the following **tree**:



At least one node in the above tree has the maximum possible number of child nodes for this type of tree.

i) What is the **size** of the tree? (1 mark)

ii) What is the **height** of the tree? (1 mark)

iii) What is the **order** of the tree? (1 mark)

iv) How many **leaf** nodes are there on the tree? (1 mark)

b) Briefly describe what is meant by a **binary search tree**. (3 marks)

c) Consider the following partial implementation of a generic **binary search tree** class:

```
1  public class BinarySearchTree<T> {
2      private BinaryTreeNode<T> root; // the root node of this tree
3      private int size;
4
5      // Constructor: Creates an empty tree.
6      public BinaryTreeNode() {
7          root = null;
8      }
9
10     // inner class: details omitted
11 }
```

i) Write a `protected` inner class named `BinaryTreeNode` for modelling a node in a binary search tree. Your definition of `BinaryTreeNode` must contain a constructor for creating a leaf node storing a specified element.

You are expected to declare the instance variables required in `BinaryTreeNode` *as* `public` *so as to avoid having to define a complementary set of accessor and mutator methods.* (7 marks)

(question continues on next page. . .)

(Question 8 continued. . .)

ii) All `BinarySearchTree` objects are expected to support the operation of an enhanced `for` statement.

Correct the header line of the given binary search tree class `BinarySearchTree`.

(3 marks)

iii) Making use of **recursion**, write a `public` method (plus any necessary helper methods) that returns the **size** of a **binary search tree**.

(8 marks)

END OF EXAMINATION PAPER