University of Glasgow

Saturday, 1st May 2010
2.00 pm – 4.00 pm
(Duration: 2 hours)

DEGREE OF MSc in Information Technology

ALGORITHMS & DATA STRUCTURES (M)

Answer any 3 out of 4 questions.

This examination paper is worth a total of 60 marks.

You must not leave the examination room within the first hour or the last half-hour of the examination.

1. **(a)** What is meant by the *time complexity* of an algorithm?

[2]

(b) Suppose that you have to choose between two alternative algorithms to solve the same problem. Given n items of data, algorithm A take $2n$ seconds, whilst algorithm B take $(10 \log_2 n)$ seconds.

(i) What is each algorithm's time complexity?

(ii) *Without doing any calculations*, state which algorithm you would choose. Explain your answer.

(iii) Now calculate and tabulate both algorithms' running times for n = 10, 20, 30, 40. Do your calculations confirm your answer to (b)(ii)?

(*Note:* $\log_2 10 \approx 3.3$; $\log_2 20 \approx 4.3$; $\log_2 30 \approx 4.9$; $\log_2 40 \approx 5.3$.)

[2+2+2]

(c) Assume that you are given the following standard algorithms:

(i) a merging algorithm, which merges the elements of two sorted arrays *a1* and *a2* into a third sorted array *a3*;

(ii) a sorting algorithm, which takes an unsorted array *a* and rearranges its elements into ascending order.

Write down an algorithm to create a sorted array containing all the elements of two unsorted arrays *a1* and *a2*. Your algorithm should call algorithm (i) and/or (ii) where required. (It should not reproduce the steps of algorithm (i) or (ii).)

Your algorithm should be as efficient as possible.

[6]

(d) Analyse the efficiency of your algorithm, in terms of the number of comparisons performed. For simplicity, assume that *a1* and *a2* each contains n elements.

Assume that the sorting algorithm (ii) performs approximately $n^2/2$ comparisons.

What is your algorithm's time complexity?

[3]

(e) Repeat (d), now assuming that the sorting algorithm (ii) performs approximately $n \log_2 n$ comparisons.

[3]

[total 20]

2. **(a)** Define what is meant by a *list*.

Explain clearly how lists differ from stacks and queues.

[2]

(b) Write a contract for a list abstract data type to meet the following requirements:

(1) The values must be lists, of any length, whose elements are of a particular type `E`.

(2) It must be possible to determine the length of a list.

(3) It must be possible to inspect the element at a given position in a list.

(4) It must be possible to add an element at the end of a list.

(5) It must be possible to add an element at any given position in a list.

(6) It must be possible to iterate from left to right over all the elements of a list.

Your contract must be expressed as a Java interface `List<E>`, including comments specifying the behaviour of each method.

[5]

(c) Suppose that your contract is to be implemented by a Java class `LinkedList<E>`, in which the data structure is a *doubly-linked-list*. Use a diagram to show this representation. Also explain briefly how each method in the class would work.

[5]

(d) Using the interface `List<E>` and the class `LinkedList<E>`, write application code that performs the following steps:

(i) create an empty list of strings;

(ii) add “the” at the end of the list;

(iii) add “spring” at the end of the list;

(iv) add “Paris” before the first element of the list;

(v) add “in” after the first element of the list;

(vi) add “the” after the second element the list;

(vii) print all the strings in the list in order from left to right.

[4]

(e) Draw diagrams to show the doubly-linked-list data structure after each of the steps (i) – (vi) of the application code of part (d).

[4]

[total 20]

3. **(a)** Define what is meant by a *set*.

Explain clearly how sets differ from lists.

[2]

Box 1 (next page) shows a contract for a homogeneous `Set` abstract data type, expressed in the form of a Java interface `Set<E>`.

In parts (b) – (d) of this question, assume that a set is to be represented by a *binary-search-tree* (BST).

(b) Draw diagrams showing the contents of the BST after adding each of the following strings to an empty set:

"to", "be", "or", "not", "to", "be".

[5]

(c) Name and write down an algorithm that would be used to implement the method `contains`.

[4]

(d) Tabulate the best-case and worst-case time complexities of the methods `contains`, `add`, and `remove`.

Explain any differences between the best and worst cases. What could be done to improve the worst case?

[5]

(e) How would your answer to (d) be affected if a set were represented by a *closed-bucket hash-table*?

[4]

[total 20]

```
interface Set<E> {

      // Each Set<E> object is a homogeneous set whose elements are of type E.

      public void clear ();
      // Make this set empty.

      public boolean contains (E x);
      // Return true iff x is an element of this set.

      public void add (E x);
      // Add the element x to this set.

      public void remove (E x);
      // Remove the element x (if any) from this set.

      public void addAll (Set<E> that);
      // Add all elements of that to this set.

      public boolean equals (Set<E> that);
      // Return true if this set is equal to that.

}
```

Box 1 A contract for homogeneous sets (Question 3).

4. This question is about the tree abstract data type (*not* about search-tree data structures).

(a) What is meant by a *tree*?

[2]

(b) Suppose that a fictional university comprises Colleges of Humanities, Biomedicine, and Science. The College of Humanities comprises Schools of Languages, History, and Economics. The College of Biomedicine comprises Schools of Biology and Medicine. The College of Science comprises Schools of Mathematics, Physics, and Chemistry. Draw a tree that captures the structure of this university.

[2]

(c) Explain what are meant by *breadth-first traversal* and *depth-first traversal* of a tree.

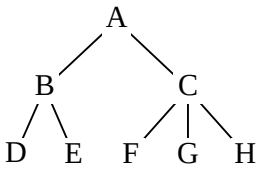
[2]

(d) Write down a recursive algorithm that performs a depth-first traversal of a given tree *t*.

[6]

(e) Box 2 (next page) shows a contract for a heterogeneous `Tree` abstract data type, expressed in the form of a Java interface `Tree` with an inner interface `Tree.Vertex`.

Assuming this contract, develop a Java method that traverses a given tree (with string elements) and prints out the tree elements indented to show their relationship. For example, the tree below left should be printed as shown below right:




```
A
   B
      D
      E
   C
      F
      G
      H
```

[8]
[total 20]

```
public interface Tree {

    // Each Tree object is a heterogeneous tree whose elements are arbitrary objects.

    public Tree.Vertex root ();
    // Return the root vertex of this tree, or null if this tree is empty.

    public Tree.Vertex parent (Tree.Vertex v);
    // Return the parent of vertex v in this tree, or null if v is the root vertex.

    public void makeRoot (Object elem);
    // Make this tree consist of just a root vertex containing elem.

    public Tree.Vertex addChild (Tree.Vertex v,
                                 Object elem);
    // Add a new vertex containing elem as a child of vertex v in this tree, and return
    // the new vertex. The new vertex has no children of its own.

    public void remove (Tree.Vertex v);
    // Remove vertex v from this tree, together with all its descendants.

    public Iterator<Tree.Vertex> children (Tree.Vertex v);
    // Return an iterator that will visit all the children of vertex v in this tree.

    /////////////  Inner interface for tree vertices  /////////////

    public interface Vertex {

        // Each Tree.Vertex object is a vertex of a tree, and contains a single
        // element.

        public Object getElement ();
        // Return the element in this vertex.

        public void setElement (Object elem);
        // Change the element in this vertex to be elem.

    }
}
```

Box 2 A contract for heterogeneous trees (Question 4).