University of Glasgow

Friday, 10 May 2013
9.30 am – 11.30 am
(Duration: 2 hours)

DEGREES OF MSc in Information Technology
AND MSc in Software Development

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)** Box 1 shows the array *quick-sort* algorithm. Illustrate its behaviour as it sorts the following array of numbers:

0	1	2	3	4	5	6	7	8	9
91	71	29	43	97	59	17	93	61	13

Your illustration must show the contents of the array, and the value of *p*, after step 1.1, after step 1.2, and after step 1.3.

Assume that step 1.1 takes *a*[*left*] as the pivot, and does not reorder the elements it puts into the left and right sub-arrays.

[4]

(b) Let *comps*(*n*) be the number of comparisons performed by the algorithm when sorting an array of *n* components. Assume that step 1.1 performs $n-1$ comparisons. Write down equations defining *comps*(*n*) when $n \leq 1$ and when $n > 1$: (i) in the best case, and (ii) in the worst case.

Write down the time complexity of the quick-sort algorithm: (i) in the best case, and (ii) in the worst case.

When does the best case arise? When does the worst case arise?

[5]

(c) Write down the array *merge-sort* algorithm. Assume that an array merging algorithm is already available.

[5]

(d) Compare the quick-sort and merge-sort algorithms in terms of their time and space complexity. Which is better in terms of time complexity? Which is better in terms of space complexity?

[6]

To sort *a*[*left*…*right*]:

1. If *left* < *right*:
   1.1. Partition *a*[*left*…*right*] such that
   a[*left*…*p*–1] are all less than or equal to *a*[*p*], and
   a[*p*+1…*right*] are all greater than or equal to *a*[*p*].
   1.2. Sort *a*[*left*…*p*–1].
   1.3. Sort *a*[*p*+1…*right*].
2. Terminate.

Box 1 The array quick-sort algorithm.

2. **(a)** You are given the following requirements for a *stack* abstract data type:

(1) It must be possible to make a stack empty.

(2) It must be possible to push a given element on to a stack.

(3) It must be possible to pop the topmost element from a stack.

(4) It must be possible to determine the depth of a stack.

Write a contract for a *homogeneous* stack abstract data type. Express your contract in the form of a Java generic interface, with a comment specifying the expected behaviour of each method.

[5]

(b) Briefly describe a possible representation for a stack.

[3]

(c) A *stack machine* has instructions that push integers on to a stack and pop integers off the stack. A typical stack machine has instructions such as those summarised in Box 2A.

A stack machine makes it easy to evaluate complicated expressions. Any integer expression can be translated to stack machine code. After the code is executed, the stack will contain a single integer, which is the result of evaluating the expression. For example:

Expression	Stack machine code	Expected result
4 + (5 × 6)	LOAD 4; LOAD 5; LOAD 6; MULT; ADD	+34
2 – (3 × 4) + 5	LOAD 2; LOAD 3; LOAD 4; MULT; SUB; LOAD 5; ADD	−5
(2 – 3) × 4 + 5	LOAD 2; LOAD 3; SUB; LOAD 4; MULT; LOAD 5; ADD	+1

Draw diagrams showing the contents of the stack after executing each instruction in the stack machine code "LOAD 4; LOAD 5; LOAD 6; MULT; ADD". Assume your stack representation of part (b).

[4]

(d) Assume that the stack-machine instructions are represented by the Java class of Box 2B.

In terms of your stack contract of part (a), implement the following method:

```
static int execute (Instruction[] instrs);
// Execute the stack-machine code instrs, and return the result.
```

[8]

Instruction	Effect
LOAD *i*	Push the integer *i* on to the stack.
ADD	Pop two integers off the stack, add them, and push the result back on to the stack.
SUB	Pop two integers off the stack, subtract the topmost integer from the second-topmost integer, and push the result back on to the stack.
MULT	Pop two integers off the stack, multiply them, and push the result back on to the stack.

Box 2A Summary of stack machine instructions.

```
public class Instruction {
    // Each Instruction object represents a stack machine instruction.

    public byte opcode;   // LOAD, ADD, SUB, or MULT
    public int operand;   // used only if opcode is LOAD

    public static final byte
        LOAD = 0, ADD = 1, SUB = 2, MULT = 3;
}
```

Box 2B Representation of stack machine instructions.

3. **(a)** Box 3 shows a contract for a `Map` abstract data type.

Explain carefully what is meant by a *map*.

[2]

(b) Using the contract of Box 3, write application code to do the following:

(i) Declare a map that will be used to record employees' names and their pay. Assume that all employees will have different names.

(ii) Add an employee named Homer with pay $20,000, and an employee named Monty with pay $500,000.

(iii) Remove Homer.

(iii) Decrease the pay of the employee named Carl by $1,000.

(v) Increase all employees' pay by 1%.

[6]

(c) Explain how a map could be represented by a binary-search-tree (BST). Briefly explain how each of the operations of Box 3 would be implemented.

(*Note:* If an operation is implemented by a standard BST algorithm, simply name the algorithm.)

[6]

(d) Suppose that we start with an empty map, represented by a BST. Then we add ("Homer", 20000), then add ("Monty", 500000), then add ("Carl", 100000), then add ("Lenny", 25000), then remove "Homer". Show the contents of the BST after each operation.

[6]

```
public class Map<K,V> {
    // A Map<K,V> object represents a homogeneous map with keys of type K and
    // values of type V.

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

    public void put (K k, V v);
    // Add an entry to this map with key k and value v, replacing any existing entry
    // with key k.

    public void remove (K k);
    // Remove the entry with key k from this map, if any.

    public V get (K k);
    // Return the value from the entry with key k in this map, or null if there is no
    // such entry.

    public Set<K> keyset ();
    // Return the set of all keys in this map.

}
```

Box 3 A `Map` contract.

4. **(a)** Define the terms *graph*, *undirected graph*, *directed graph*, and *edge attribute*.

Box 4 shows a computer network. Characterise this computer network using the terms you have defined.

Give two further examples of graph applications: (i) an undirected graph whose edge attributes are distances, and (ii) a directed graph whose edge attributes are flow rates.

[4+1+2]

(b) Suppose that a message is to be sent through the computer network of Box 4, from A to G. It is possible that some computers are down at the time. The message must be sent through as few computers as possible, but must not be sent through a computer that is down.

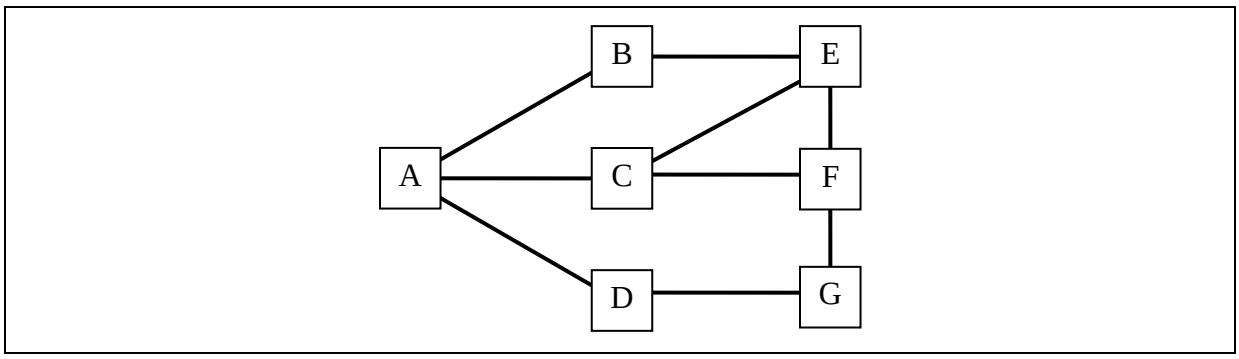
What is the shortest route for the message: (i) when no computers are down; (ii) when computers B and D are down; (iii) when computers B, D, and F are down?

[3]

(c) Name a general graph algorithm that can be adapted to solve this problem. Write down the adapted algorithm.

You may assume that any computer can be asked whether it is up or down. You may also assume that both A and G are up.

[10]



Box 4 A computer network.