1. **(a)** Table 1 shows the array *quick-sort* algorithm. Show that it correctly sorts the array in every case (on the assumption that step 2.1 correctly partitions the array).

(3)

(b) Illustrate the quick-sort algorithm's behaviour as it sorts the following array of country-codes alphabetically. Your illustration should show the contents of the array, and the value of *p*, after step 1.1, after step 1.2, and after step 1.3. You should state any assumptions you make about how step 1.1 works.

0	1	2	3	4	5	6	7	8	9	10	11
FR	DE	IT	BE	NL	LU	UK	IE	DK	PT	ES	GR

(3)

(c) Write down the array *merge-sort* algorithm. Show that it correctly sorts the array.

(5)

(d) Illustrate the merge-sort algorithm's behaviour as it sorts the above array of country-codes. Your illustration should show the contents of the array after each step.

(3)

(e) State the time complexities of the merge-sort and quick-sort algorithms (in terms of the number of comparisons performed). Justify your answers.

Note: Your justifications may be either mathematical or informal.

(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.
```

Table 1 Array quick-sort algorithm (Question 1).

continued overleaf/

2. **(a)** *Stack machine code* consists of the instructions summarised in Table 2. Each of these instructions acts on a value stack.

Any arithmetic expression can be translated to a sequence of stack machine instructions, e.g.:

$a + (b / c) - d \quad \Rightarrow$ `LOAD` a`; LOAD` b`; LOAD` c`; DIV; ADD; LOAD` d`; SUB`

$(a + b) / c - d \quad \Rightarrow$ `LOAD` a`; LOAD` b`; ADD; LOAD` c`; DIV; LOAD` d`; SUB`

In terms of the `Stack` contract of Table 3, implement the following Java method:

```
static void execute (byte opcode, int v,
                     Stack values);
// Predict the effect of executing a single stack-machine
// instruction, updating the stack values. Assume that
// opcode is either LOAD, ADD, SUB, MULT, or DIV.
// If opcode is LOAD, v contains the value to be loaded.

static final byte LOAD = 0,
     ADD = 1, SUB = 2, MULT = 3, DIV = 4;
```

(8)

(b) Outline how stacks can be represented (i) by arrays, and (ii) by linked lists. Draw diagrams showing each representation of a stack that contains the words ‘north’ (first in), ‘south’, ‘east’, and ‘west’ (last in).

(6)

(c) Suppose that a new requirement forces us to add the following operation to the `Stack` contract:

```
public Object get (int d);
// Return the element at depth d in this stack. (In particular, if
// d = 0, return the element at the top of this stack.)
```

Outline how this operation would be implemented when stacks are represented (i) by arrays, and (ii) by linked lists. State and justify the operation’s time complexity in each case.

(6)

continued overleaf/

Instruction	**Effect**
`LOAD v`	Push the value `v` on to the stack.
`ADD`	Pop two values off the stack, add them, and push the result back on to the stack.
`SUB`	Pop two values off the stack, subtract the second from the first, and push the result back on to the stack.
`MULT`	Pop two values off the stack, multiply them, and push the result back on to the stack.
`DIV`	Pop two values off the stack, divide the first by the second, and push the result back on to the stack.

Table 2 Stack-machine instructions (Question 2).

```
public interface Stack {

    // Each Stack object is a stack whose elements are objects.

    //////////// Accessors ////////////

    public boolean isEmpty ();
    // Return true if and only if this stack is empty.

    public Object getLast ();
    // Return the element at the top of this stack.

    //////////// Transformers ////////////

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

    public void addLast (Object elem);
    // Add elem as the top element of this stack.

    public Object removeLast ();
    // Remove and return the element at the top of this stack.

}
```

Table 3 Contract for a `Stack` ADT (Question 2).

continued overleaf/

3. **(a)** A *relation* (as in a relational database) may be viewed as a set of objects, where each object has several fields. The following illustrates a relation containing certain details about a company's employees:

employees

name	number	gender	pay
Lapping	1	M	45,000
Quintock	3	M	30,000
Piercemüller	7	M	40,000
Maureen	11	F	15,000

In terms of the `Set` contract of Table 4, implement the following Java methods:

```
static float female (Set employees);
// Return the proportion of female employees
// (0.25 in the example above).

static Set wellPaid (Set employees);
// Return the set of all employees whose pay is at least £50,000
// (none in the example above).
```

Assume in each case that `employees` is a set of `Employee` objects:

```
public class Employee {
   public String name;
   public int number;
   public char gender;
   public int pay;    // in £
}
```

(6)

(b) A set can be represented by a sorted array or by a binary search tree (BST). Summarise the advantages and disadvantages of the BST representation.

(4)

(c) Suppose that a set of words is represented by a BST. Starting with an empty set, show the effects of: (i) inserting 'to'; (ii) inserting 'be'; (iii) inserting 'or'; (iv) inserting 'not'; (v) inserting 'that'; (vi) inserting 'is'; (vii) inserting 'the'; (viii) inserting 'question'; (ix) deleting 'or'.

(10)

continued overleaf/

```
public interface Set {

   // Each Set object is a set whose members are objects.

   //////////// Accessors ////////////

   public boolean isEmpty ();
   // Return true if and only if this set is empty.

   public int size ();
   // Return the cardinality of this set.

   public boolean contains (Object obj);
   // Return true if and only if obj is a member of this set.

   public boolean equals (Set that);
   // Return true if and only if this set is equal to that.

   public boolean containsAll (Set that);
   // Return true if and only if this set subsumes that.

   //////////// Transformers ////////////

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

   public void add (Object obj);
   // Add obj as a member of this set.

   public void remove (Object obj);
   // Remove obj from this set.

   public void addAll (Set that);
   // Make this set the union of itself and that.

   public void removeAll (Set that);
   // Make this set the difference of itself and that.

   public void retainAll (Set that);
   // Make this set the intersection of itself and that.

   //////////// Iterator ////////////

   public Iterator iterator();
   // Return an iterator that will visit all members of this set, in no particular order.

}
```

Table 4 Contract for a `Set` ADT (Question 3).

continued overleaf/

4. **(a)** Compare and contrast the *closed-bucket hash table* (CBHT) and *open-bucket hash table* (OBHT) data structures.

(6)

(b) Consider a map whose entries are employee records, each of which contains an employee number (key) and other data of no concern here. Suppose that the map is to be represented by an OBHT. The number of buckets is 10, the hash function returns the employee number's rightmost digit, and the step length is 1.

Starting with an empty map, show the effects of adding records with the following employee numbers: 008; 011; 065; 029; 038; 108.

(6)

(c) What undesirable effect do you observe in your answer to part (b)?

(2)

Re-design the employee records OBHT to avoid such undesirable effects. Your answer must cover the number of buckets, the hash function, and the step length. Justify your design decisions. Assume that the number of employee records is about 100 at any given time.

(6)