University of Glasgow

XXXday May XX, 2012
XX.XX am/pm – XX.XX am/pm
(Duration: X hour XX minutes)

DEGREES OF MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

Algorithms and Data Structures 2

(Answer all 4 questions.)

This examination paper is worth a total of 50 marks

You must not leave the examination room within the first hour or the last half-hour of the examination. (for exams of 2 hours duration)

or

You must not leave the examination room within the first half hour or the last fifteen minutes of the examination. (for exams of less than 2 hours duration)

1. A binary search tree is a binary tree T such that each node of T stores an item e. Items stored in the left subtree rooted at a node v are less than the item in node v, and items stored in the right subtree rooted at a node v are greater than the item in node v. Below is Java code for the **BNode** class, and below that code for the **BSTree** class.

```
public class BNode {
    private BNode left;
    private int   item;
    private BNode right;

    public BNode(int e){left = null; item = e; right = null;}

    public int   getItem(){return item;}
    public BNode getLeft(){return left;}
    public BNode getRight(){return right;}
    public void  setLeft(BNode nd){left = nd;}
    public void  setRight(BNode nd){right = nd;}

}

public class BSTree {
    private BNode root;
    private int size;

    public BSTree(){root = null;}

    public boolean isEmpty(){return root == null;}
    public int size(){return size;}

    public void insert(int e){...}

    public boolean isPresent(int e){return root != null && isPresent(e,root);}

    private boolean isPresent(int e,BNode nd){...}

    public boolean subsumed(BSTree t){return contains(root,t);}

    private boolean subsumed(BNode nd,BSTree t){...}

}
```

(a) Write java code for the method **isPresent(int e,Bnode nd)**, where the method delivers true if and only if **e** is in the subtree rooted at **nd**. **[5]**

(b) Write java code for the method **subsumed(Bnode nd, BSTree t))**, where the method delivers true if and only if every item in the subtree rooted at **nd** is present in **t**.

[5]

(c) What is the worst case and best case complexity for the **subsumed** method you have described? And what property should be established to ensure optimum performance? Explain your answers. **[4]**

(d) With regard to the code segment below (**BSTreeTest**)

- Draw the trees **t1** and **t2**
- What are the height of the trees?
- Write out the preorder and postorder traversals of the trees. **[6]**

```
public class BSTreeTest {

    public static void main(String[] args) {

        BSTree t1 = new BSTree();
        BSTree t2 = new BSTree();

        int a[] = {4,2,6,1,3,5,1,7};
        int b[] = {1,2,7,6,3,4,5};

        for (int i : a) t1.insert(i);
        for (int i : b) t2.insert(i);

    }
}
```

2. A queue is a container of objects that are inserted and removed according to the first-in first-out (FIFO) principle. In the class definition below for Queue, the queue is implemented as a dynamic data structure of linked nodes, with a front node (the front of the queue) and a rear node (the rear of the queue).

```
public class Node<E> {

    private E element;
    private Node<E> next;

    public Node(){this(null,null);}

    public Node(E element, Node<E> next){
        this.element = element;
        this.next = next;
    }

    public E getElement(){return element;}
    public void setElement(E element){this.element = element;}
    public Node<E> getNext(){return next;}
    public void setNext(Node<E> next){this.next = next;}

}


public class Queue<E> {

    private Node<E> front;
    private Node<E> rear;
    private int size;

    public Queue(){front = rear = null; size = 0;}

    public int size(){return size;}
    public boolean isEmpty(){return front == null;}

    public E dequeue() throws QueueException {...}

    public void enqueue(E s){...}

}
```

(a) Give a Java implementation for the undefined methods above (**dequeue** and **enqueue**). In your implementation you should address the possibility of underflow. **[8]**

(b) What is the complexity of the operations **enqueue** and **dequeue**? Explain your answer. **[2]**

3. Given an array of integers we might sort it using a radix sort. Assuming we have the following data set **int S[] = {89,28,81,69,14,31,29,18,39,17}** demonstrate the working of radix sort. Also, what is the complexity of radix sort? Explain your answer. **[10]**

4. A map abstract data structure allows us to store key-value pairs (k,v) , but is different from the dictionary abstract data type because each key is unique. Consequently the association of keys to values defines a mapping. We might implement the map abstract data type using a hash table.

   (a) What is a hash table and why would we want to use one? **[2]**

   (b) What is a hash function and what properties would we like it to have? **[2]**

   (c) What is a "collision"? **[1]**

   (d) Describe the "separate chaining" method for hash collision resolution. **[2]**

   (e) Describe the "linear probing" method for hash collision resolution, taking into consideration the methods insert, find, and remove **[3]**