University of Glasgow

XXXday May XX, 2014
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 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 int   item;
    private BNode left;
    private BNode right;

    public BNode(int e, BNode left,BNode right){
       this.item  = e;
       this.left  = left;
       this.right = right;
    }

    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;

    public BSTree(){root = null;}

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

    public void insert(int e){
       if (isEmpty()) {root = new BNode(e,null,null);}
       else insert(e,root);
    }

    private static void insert(int e,BNode nd){...}

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

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

}
```

(a) Write java code for the method `insert(int e,BNode nd)` in class `BSTree`, where the method inserts the integer `e` into the tree if and only if `e` is not already present in the tree. **[5]**

(b) Write java code for the method `isPresent(int e,Bnode nd)`, where the method delivers true if and only if `e` is in the tree. **[5]**

(c) Assume that the following items are inserted into an empty `BSTree` in the following order: 30, 40, 24, 58, 48, 26, 11, 13, 36.

- Draw the tree.
- What is the height of the tree?
- Write out the preorder, inorder and postorder traversals of the tree. **[5]**

(d) Draw the tree after the node with item 30 has been deleted and outline the algorithm you used for the deletion (you do not need to write Java code). **[3]**

2. An organisation has a data set of 1 million customers. The information the organisation holds on customers includes their name and height in centimetres (cm). The data set has been sorted into order using name as the primary key. The organisation now wants to sort that data using height as a key, where height is an integer in the range 100cm to 220cm. This might be done using a pigeonhole sort or a radix sort.

   (a) What is a pigeonhole sort? What is its complexity? Why might it be suitable for this task? Assuming Java was being used, what data structures might you use? **[8]**

   (b) What is radix sort? What is its complexity? Why might it be suitable for this task? **[6]**

   (c) A sorting algorithm is stable if it preserves the original order of records with equal keys. Is your proposed implementation of pigeonhole sort stable? Explain your answer. **[2]**

3. Given a list of people and an individual person x we might want to produce a new list of people that x knows, i.e. a list of friends of x. Below are three code snippets that perform such a function, where `x.knows(y)` delivers true if `x` is a friend of `y`, false otherwise.

```
ArrayList<Person> filter1(Person x, ArrayList<Person> people){
    ArrayList<Person> friends = new ArrayList<Person>();
    for (Person y : people) friends.add(y);
    for (Person y : people) if (!x.knows(y)) friends.remove(y);
    return friends;
}

ArrayList<Person> filter2(Person x, ArrayList<Person> people){
    ArrayList<Person> friends = new ArrayList<Person>();
    for (Person y : people) if (x.knows(y)) friends.add(y);
    return friends;
}

ArrayList<Person> filter3(Person x, ArrayList<Person> people){
    ArrayList<Person> friends = new ArrayList<Person>();
    for (int i=0;i<people.size();i++){
        Person y = people.get(i);
        if (x.knows(y)) friends.add(y);
    }
    return friends;
}
```

   (a) What is the complexity of `filter1` and `filter2`? Explain your answer. **[4]**

   (b) We might expect `filter3` to run faster than `filter2`. Why is that? **[2]**

4. Suppose we are hashing integers into a 7-bucket hash table using the hash function

```
int hash(int i){return i % 7;}
```

The call `put(i)` will put the integer `i` into the hash table, `get(i)` delivers true if the integer `i` is in the hash table (false otherwise) and `remove(i)` removes the integer `i` if it is in the hash table. The functions `put`, `get` and `remove` use the above `hash` function.

(a) Show the resulting open hash table, using linear probing, if the sequence of calls `put(1)`, `put(8)`, `put(27)`, `put(64)`, `put(125)`, `put(216)` are made on an initially empty hash table. **[3]**

(b) **S**how what steps would be performed, and the resultant hash table, due to a call to `remove(8)` followed by a call to `get(64)` and then `get(15)` **[4]**

(c) Show the resulting hash table, using separate chaining, if the sequence of calls `put(10)`, `put(21)`, `put(17)`, `put(19)`, `put(8)`, `put(5)`, `put(22)`, `put(11)` are made on an initially empty hash table **[3]**