University of Glasgow

XXXday May XX, 2015
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. The method `present1D`, below, searches a one dimensional integer array `A` and returns `true` if `x` is present in `A`, `false` otherwise. Function `present2D` then uses `present1D` to determine if `x` is present in an n by n array `B`

```
boolean present1D(int x,int[] A){
     for (int i=0;i<A.length;i++)
        if (A[i] == x) return true;
     return false;
 }


boolean present2D(int x,int[][] B){
    for (int i=0;i<B.length;i++)
       if (present1D(x,B[i])) return true;
    return false;
 }
```

   (a) Assuming the length of the array `A` is n what is the worst case running time of `present1D` in terms of n? Explain your answer. **[2]**

   (b) If N is the total size of `B` (i.e. number of array elements in `B`) could we claim that `present2D` is of linear complexity? Explain your answer. **[2]**

   (c) Two programs, X and Y are analysed and found to have average-case running times no greater than $100.n.\log_2 n$ and n^2 respectively. Should we always prefer program X to program Y? Explain your answer, giving evidence using values of n. **[3]**

2. Suppose we are hashing integers into an 8-bucket hash table using the hash function

```
int hash(int i){return i % 8;}
```

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

   (a) Show the resulting open hash table, using linear probing, if the sequence of calls `put(22)`, `put(8)`, `put(13)`, `put(21)`, `put(23)`, `put(12)`, `put(2)` are made on an initially empty hash table. **[3]**

   (b) **S**how what steps would be performed, and the resultant hash table, following a call to `contains(9)` followed by a call to `remove(13)` and `contains(20)`. **[3]**

(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]**

(d) What happens to the performance of the separate chaining method as the size of the hash table decreases? Explain your answer. **[2]**

3. A linked list is a data structure consisting of a group of nodes which together represent a sequence. In the class definition below for `List`, the list is implemented as a dynamic data structure of linked nodes, with a `head` node (the head of the list) and an integer `size` (keeping count of the number of nodes in the list).

```
public class Node<E extends Comparable<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 boolean equals(Node<E> node){return element.equals(node.getElement());}

}
```

```
public class List<E extends Comparable<E>> {

    private Node<E> head;
    private long size;

    public List(){head = null; size = 0;}

    public Node<E> getHead(){return head;}

    public void setHead(Node<E> node){head = node;}

    public boolean isEmpty(){return head == null;}

    public long size(){return size;}

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

    public boolean equals(List<E> L){...}

    public boolean isPresent(E s){...}

    public List<E> intersection(List<E> L){...}

}
```

(a) Give a Java implementation for the undefined methods `addFront`, `equals`, `isPresent` and `intersection`.

- Method `addFront` creates a new `Node`, adds it to the front of the list, updates the head pointer and increments the size of the list. **[2]**
- Method `equals` delivers `true` if the current list (`this`) and the argument `L` contain the same elements in the same order. **[4]**
- Method `isPresent` delivers `true` if there is a node in the current list (`this`) that has an element equal to `s`, `false` otherwise. **[2]**
- Method `intersection` delivers a new list that is the intersection of the elements in the current list (`this`) and the argument `L`. You can assume that neither list contains any duplicate elements. **[3]**

(b) Using big-O notation, what is the complexity of `addFront`, `equals`, `isPresent` and `intersection`? **[4]**

4. A binary search tree is a binary tree T such that each internal node v of T stores an item e. Items stored at nodes in the left subtree of v are less than or equal to e, and items stored in the right subtree of T are greater than e.

(a) Insert into an initially empty binary search tree the following items in the order shown: 30, 40, 24, 58, 26, 11, 13, 36. Draw the tree after the insertions have been completed. **[2]**

(b) Give the preorder, inorder and postorder traversals of the tree. **[3]**

(c) Draw the tree after the node with item 30 is deleted and describe the algorithm you have used for the deletion. **[3]**

(d) If we had to insert 1023 items into a binary search tree, what could be the minimum and the maximum height of the binary search tree? Suggest what property a data set might have to create a tree of maximum height? In your answer explain what we mean by the height of a tree. **[4]**

(e) Briefly describe what is meant by an AVL tree and how an AVL tree avoids the worst case described in part (d) above. **[3]**

(f) Which of the following (if any) are AVL trees? Justify your answer. **[2]**