TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Student ID: ....

EXAMINATIONS — 2011

END YEAR

COMP103
Introduction to
Data Structures and Algorithms

SOLUTIONS

Time Allowed: 3 Hours

Instructions:

1. Attempt **all** of the questions.
2. *Read each question carefully before attempting it.* (Suggestion: You do not have to answer the questions in the order shown. Do the questions you find easiest first.)
3. This examination will be marked out of **180** marks, so allocate approximately one minute per mark.
4. Write your answers in the boxes in this test paper and hand in all sheets.
5. Non-electronic translation dictionaries are permitted.
6. Calculators are allowed.
7. Documentation on some relevant Java classes, interfaces, and exceptions can be found at the end of the paper.

Questions	**Marks**
1. Basic Questions	[27]
2. Using Collections	[15]
3. Implementing Collections	[15]
4. Recursion, and Sorting	[16]
5. Linked Lists, and Trees	[36]
6. Binary Search Trees	[24]
7. Partially Ordered Trees and Heaps	[26]
8. Bitsets and Hashing	[21]

Question 1. Basic questions [27 marks]

(a) [2 marks] List two advantages of using ArrayList over Arrays.

ArrayList grows automatically while arrays have fixed size. ArrayList has get/set/size etc methods while arrays don't.

(b) [2 marks] Of the slower sorting sorting algorithms, which one has the best Big-O cost on almost sorted arrays? What is this cost?

Insertion Sort O(n)

(c) [2 marks] What is the depth of a binary tree whose maximum number of leaves is 16?

4

(d) [2 marks] Why is the contains() operation significantly faster to perform on a SortedArraySet than on a plain (unsorted) ArraySet?

Because we can use the Binary Search algorithm to find items (contains operation) in a SortedArraySet.

(e) [10 marks] For each example on the left, draw a line to the Collection type on the right that would be best for representing it.

The document files in an inbox tray.	Stack	SET
The keys on a keychain.	Set	QUEUE
Things to do, written in a dairy.	List	STACK
Passengers boarding a plane.	Queue	MAP
Cars and their registration numbers.	Map	LIST

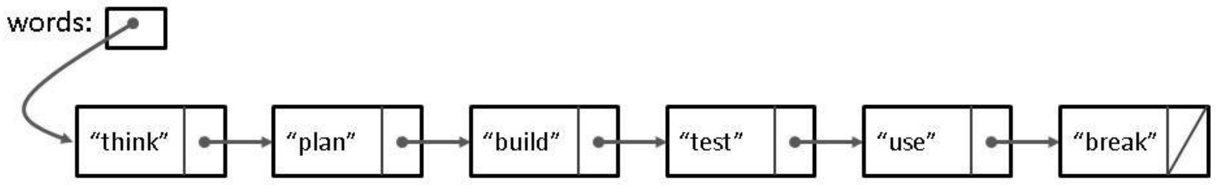
(f) [2 marks] Which implementation of a Set discussed in this course has the best average performance for add, remove, and find?

HashSet //(or BitSet, though this isn't strictly an implementation of Set)

(g) [3 marks] Write down the 3 main steps involved in the ensureCapacity() method of an ArrayList (which enables ArrayList to be "infinitely" stretchable).

Step1. create a new array of twice the size Step2. copy over all the items from old array into the new (bigger) array Step3. make the variable (e.g. data) point to the new array instead of the old one.

Consider the diagram below of a variable words containing the reference of a linked list. Assume that each node of the list has the fields value and next.



(h) [2 marks] What is the value of words.next.value?

plan

(i) [2 marks] What is the value of words.next.next.next.value?

test

Question 2. Using Collections [15 marks]

```
public class Customer{

    private String name;
    private Integer quantity;
    private String size;
    private String topping;
    :
    :
    :
    public Customer(String nName, Integer qQuantity, String sSize, String tTopping){
            name = nName;
            quantity = qQuantity;
            size = sSize;
            topping = tTopping;
    }

    public String toString(){
        return name+" has ordered "+quantity+" "+size+" "+topping+" pizza(s)";
    }

}
```

AngelsPizza is a popular Pizza joint in town. They need a simple system to simulate their customer orders. The program consists of a class AngelsPizza, which begins by initialising an ArrayQueue of Customer objects as follows:

```
public class AngelsPizza{
    private ArrayQueue<Customer> customers = new ArrayQueue<Customer>();
```

The class needs to have 2 methods, for loading customer details and for printing the details of the next customer in line to be served.

(a) [10 marks] Complete the loadCustomers method, which takes a String as the name of the file containing customer details, reads from that file, constructs new Customer objects (see Customer class on facing page), and adds them to the ArrayQueue. The file with the customer details has the following format:

```
Bob Smith 1 L Margherita
Jane Austen 2 S Chicago Style
P.G. Woodehouse 4 XL Seafood
Michael Clark 1 M Fruit
Susan Timon 2 L BBQ Chicken
```

```
  public void loadCustomers(String fname){
     try{
       Scanner f = new Scanner(new File(fname));
       //YOUR CODE HERE



          // while loop to read each line of the file
          while (f.hasNextLine()) {
               String name = f.next();
               while(!f.hasNextInt()){
                    name += "  " + f.next();
               }
               Integer quantity = f.nextInt ();
               String size = f.next();
               String topping = f.nextLine();
               Customer custom = new Customer(name,quantity,size,topping);
               customers.offer(custom);
          }
          f.close();



       }catch(IOException ex) { System.err.println(ex.getMessage());}
  }
```

(b) [5 marks] Complete the serveCustomer method, which prints out the details of the next Customer from the ArrayQueue, or prints that there are no customers to serve, if the ArrayQueue is empty.
Note: the Customer class is written for you on page 4 and defines a toString() method that returns the customer details to be printed.

Sample printouts:

```
Next customer : Bob Smith has ordered 1 L  Margherita pizza(s)
There are no customers to serve.
```

```
public void serveCustomer(){
    //YOUR CODE HERE



      if (customers.peek()!=null)
          UI. println ("Next customer :  " + customers.poll());
      else
          UI. println ("There are no customers to serve.");



}
```

Question 3. Implementing Collections [15 marks]

A local Rugby club maintains the Rugby World Cup (RWC) rankings for all top Rugby teams. They would like a simple system for maintaining rankings of the teams. The program consists of a class RWC, which begins by initialising a Map as follows:

```
public class RWC{
    private Map <Team, Integer> ratings = new HashMap<Team, Integer>();
```

The class needs to have 3 methods, for adding teams, removing teams, and printing all the teams along with their ratings.

(a) [5 marks] Complete the addTeam method, which adds the team along with their rating to the Map. It should either print out that the team has been added, or that the team already exists in the system. Assume that the Team class has been written for you and already defines a toString method that returns the name of the team (example: All Blacks, or Wallabies).

Sample printouts:

```
Team All Blacks have been added
Team Wallabies have been added
All Blacks already exist in the system!
```

```
public void addTeam(Team myTeam, Integer rating){
//YOUR CODE HERE

        if (ratings.containsKey(myTeam)) {
            UI.printf ("%s already exist in the system!\n", myTeam);
        }else {
            ratings.put(myTeam, rating);
            UI.printf ("Team %s have been added\n", myTeam);
        }

}
```

(b) [5 marks] Complete the removeTeam method, which removes the given team and prints out that it has been removed, or prints out that the team is not in the system if that team does not exist. Sample printouts:

```
Springboks have been removed!
Powerpuff Boys are not in the system.
```

```
 public void removeTeam(Team myTeam){
//YOUR CODE HERE

        if (ratings.containsKey(myTeam)){
            ratings.remove(myTeam);
            UI.printf("%s have been removed!\n", myTeam);
        }else{
            UI.printf("%s are not in the system\n", myTeam);
}       }
```

(c) [5 marks] Complete the printAllTeams method, which prints out all the teams and their ratings. Assume the Team class is written for you and already defines a toString method that returns the name of the team (example: All Blacks or Wallabies). Sample printout:

```
RWC ratings
All Blacks: 1
Wallabies: 2
Brave Hearts: 6
Springboks: 5
----------------------
```

```
public void printAllTeams(){

        // YOUR CODE HERE

    UI.println("RWC ratings");
    for(Map.Entry<Team,Integer> rat: ratings.entrySet())
        UI.println(rat.getKey() + ":  " + rat.getValue());
    UI.print("----------------------\n");

}
```

Student ID: ....

Question 4. Recursion and Sorting [16 marks]

(a) [10 marks] A pebble is drawn as a black oval with a white outline, whose the height is $\frac{1}{4}$ its width. Write a **recursive** drawPebbles method that draws a sequence of pebbles, starting at (x,y), with the first pebble of the given width, and each successive pebble 80% the width of the previous one. The method should not draw any pebbles smaller than 15 in width.

[Note: remember to give a *recursive* solution]

```
public void drawPebbles(double x, double y, double width) {
        // YOUR CODE HERE



    if (width >= 15) {
            double depth = width/4;
            UI.setColor(Color.white);
            UI. fillOval (x, y, width, depth);
            UI.setColor(Color.black);
            UI.drawOval(x, y, width, depth);
            this.drawPebbles(x+depth, y−depth, width * 4 /5);
        }
    }



}
```

(b) [3 marks] Given the initial array shown (1st row), write down the array as it would be after **3 passes** through the outer loop of Selection Sort (*assume the usual A-to-Z sort is required*).

Start:	S	B	R	O	Q	D	F

After 3 passes of Selection sort: B D F O Q S R

(c) [3 marks] Given the initial array shown below (1st row), write down the array as it would be after **3 passes** through the outer loop of Insertion Sort (*assume the usual A-to-Z sort is required*).

Start:	M	P	L	Z	A	D	Q

After 3 passes of Insertion sort: L M P Z A D Q

Question 5. Linked Lists and Trees [36 marks]

Consider the following declaration of LinkedNode, which can be thought of as the first node in a linked list.

```
public class LinkedNode<E> {
  // fields and constructor
  private E value;
  private LinkedNode<E> next;
  public LinkedNode(E val, LinkedNode<E> nd) {
     value = val;
     next = nd;
  }
  public void setValue(E item) { value = item;}     // the setters and getters
  public void setNext(LinkedNode<E> nd) { next = nd;}
  public E getValue() {return value;}
  public LinkedNode<E> getNext() { return next;}
}
```

(a) [4 marks] Complete the size method for the above LinkedNode class, that returns the number of items in the linked list which starts at the current node. You must give a *recursive* version.

```
public int size() {
     if (next == null) return 1; // this is the last node

     return next.size() + 1;
}
```

(b) [8 marks] Suppose you are writing code to generate a chain of LinkedNodes, each containing an integer. The first node should contain the integer 5000, and each successive node should contain a value that is half (via integer division) the value stored in the preceding node. The chain should end when the value reaches 1 (*ie.* the final node will contain "1"). Write code to accomplish this.

```
    int n = 5000;


    LinkedNode<Integer> halvings = new LinkedNode<Integer>(n, null);
    LinkedNode<Integer> lastNode = halvings; // since we don't want to overwrite halvings!
    while (n > 1) {
       n = n/2;
       lastNode.setNext( new LinkedNode<Integer> (n, null);
       lastNode = lastNode.getNext();
    }
.
```

(c) [1 mark] What value would be returned by calling the size() method on the first LinkedNode from the previous question?

12

(d) [2 marks] What is the main problem with using LinkedNode on its own, as an implementation of the *List* interface?

A linked list defined by LinkedNode can't be empty (unless we take care to make a special first node that's not a "real" node). An empty list would have no nodes, but then you wouldn't be able to call any methods on it!

(e) [5 marks] In lectures you met the idea of LinkedList as a way to implement the *List* interface. A LinkedList uses objects of the LinkedNode class to store the items.

Consider instead using a class which we shall call TreeList to implement the *List* interface by storing the items in a binary tree instead. **Discuss** any advantages (and / or disadvantages) you can see for this idea.

For a generic binary tree, there is no advantage (and the only disadvantage is obscurity!). But IF we make the tree a BST, we can then have contains() that is O(log n) fast provided the tree is reasonably balanced.
remove() could also be fast, since we can get to node i in O(log n) steps.
add(): also fast, if balanced.
Disadvantage: worst thing is the tree could be very unbalanced, but EVEN THEN it's no worse than a linked list.

(f) [2 marks] A full binary tree with a depth of 10 has $2^{10} = 1024$ leaves on its bottom layer. How many *other* nodes are there in the tree?

1023

(g) [2 marks] The term "Ternary" refers to 3 in the way that "Binary" refers to 2. What is the depth of a complete, balanced *ternary* tree having 41 nodes?

4

(h) [8 marks] Consider the following TernaryTreeNode class:

```
public class TernaryTreeNode <E> {
    private String word;
    private TernaryTreeNode <E> firstChild;
    private TernaryTreeNode <E> secondChild;
    private TernaryTreeNode <E> thirdChild;
     :
     :
}
```

In the box below, complete the find method in a TernaryTreeNode class. This should use recursion to search the tree below node for a node having a field word matching the argument text. It should return a reference to this node.

```
private TernaryTreeNode <E> find(TernaryTreeNode node, String text) {


    if (node.word.equals(text))
        return node;

    TernaryTreeNode ans = find(firstChild, text );
    if (ans != null) return ans;

    TernaryTreeNode ans = find(secondChild, text);
    if (ans != null) return ans;

    TernaryTreeNode ans = find(thirdChild, text );
    if (ans != null) return ans;

    return null;

}
```

The following is pseudocode for a particular tree traversal:

```
make a queue,
put root on the queue,
while queue not empty:
    poll a node off the queue
    process that node
    put its children on the queue
```

(i) [1 mark] Is this a recursive, or iterative, way of carrying out a traversal?

Iterative

(j) [3 marks] What type of traversal would be performed by the above algorithm?

Breadth-first

Question 6. Binary Search Trees [24 marks]

(a) [4 marks] Draw the *Binary Search Tree* that results from adding the following in the order given:

F, I, D, J, K, C, A, E, B

```
      F
     / \
    D   I
   / \   \
  C   E   J
 /         \
A           K
 \
  B
```

(b) [2 marks] What is special about an in-order traversal of a Binary Search Tree (BST)?

It processes the nodes in sorted order.

(c) [8 marks] As pseudocode, give an algorithm for conducting an **in-order** traversal of a binary tree, by making use of a Stack.

```
make a stack,
put root on the stack,
while stack not empty:
   pop a node off the stack
   process that node
   if it has a right child:
      put its right child, and that node's left subtree, on the queue
```

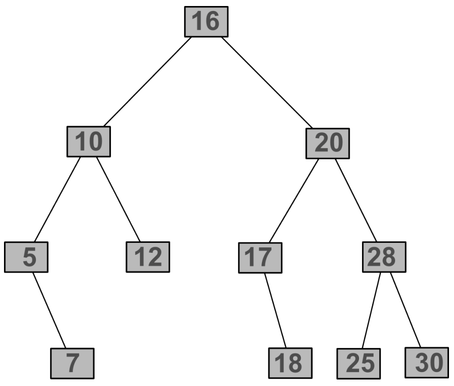
(d) [2 marks] What are “rotations”, in the context of Binary Search Trees?

“Rotations” are algorithms for balancing the tree. Adding to a BST can leave it poorly balanced, leading to slow search.

(e) [4 marks] Suppose that items consisting of the 7 letters `A` to `G` are to be added to a (BST) that is initially empty. Give an ordering of the letters such that, if the items are added in this order, the resulting BST will be perfectly balanced. (*NB.* You don’t need to show the actual tree).

For example: D,B,A,C,F,E,G - but note that other solutions are possible.

(f) [4 marks] In the box, draw the tree that results if the values 30, 5, 10, and 20 are removed (in that order) from the following BST.
Note there is spare working paper included with this exam.



```
          16
         /  \
        /    \
      12      25
      /      / \
     /      /   \
    7      17    28
             \
             18
```

Question 7. Partially Ordered Trees and Heaps [26 marks]

(a) [6 marks] Two efficient implementations of a priority queue are:

- an array of Queues, indexed by the priority, one queue for each priority value
- a HeapQueue, using a partially ordered tree in an array

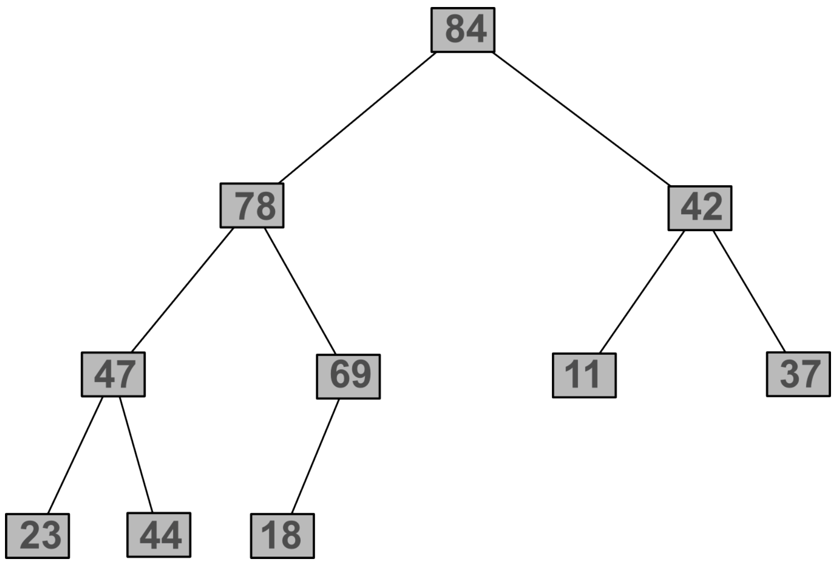
Give one *advantage* and one *disadvantage* of using the first, compared to the second, implementation.

```
Advantage of using an array of Queues:
```

Very fast O(1) dequeue and enqueue, cf: O(log n) for HeapQueue

```
Disadvantage:
```

Only applicable if we have finite, discrete priorities.



(b) [4 marks] Draw the *heap* (as an array) that corresponds to the above *POT*.

84, 78, 42, 47, 69, 11, 37, 23, 44, 18

(c) [6 marks] Show the *POT* that results when the following values are added, in order, to the same tree (*ie.* to the original tree, not your answer to a previous question):

81, 25, 22, 86

```
                 86
                /  \
               /    \
              /      \
           81          84
          / \          / \
         /   \        /   \
       47     78    25    42
      /  \   /  \  /  \   /
    23  44  18  69 11  22 37
```

(d) [4 marks] *As an array*, show the Heap that results from the following values being added, in order, to a *NEW* heap. Assume it is a "max" heap, with the largest value at the root.

3, 9, 11, 7, 5

11, 7, 9, 3, 5

(e) [6 marks] In words or pseudocode, give the algorithm for carrying out poll() on a Priority Queue that is implemented as a *Heap*.

```
poll() removes the root of the POT, which is in site 0 of the array.
Replace data[0] (ie. the "root") with the last item in the array.
Then pushdown from the root.

To pushdown from site i:
    look up the children, if they exist, at sites 2i+1 and 2i+2
    if site i elt's priority < that of child with largest priority:
         swap them in the array, and
         pushdown from the child.
```

Question 8. Bitsets and Hashing [21 marks]

(a) [2 marks] What does the term "load factor" mean, in the context of hashing?

Load factor is the max percentage occupancy allowed in a hash table before it is resized.

(b) [4 marks] Lectures covered two different implementations for Hash Tables, namely "Chaining" (or buckets), and "Open Addressing". Deletions need to be handled differently in the two approaches. In words, describe how each implementation carries out deletions.

```
Chained implementation:
```

For the chained case, we can just ask the corresponding Set or List to carry out the deletion. With Open Addressing however there is the issue that we need to replace the item by a "tombstone" in order for Open Addressing to continue to work - otherwise subsequent items may appear to be missing when they are not.

```
Open Addresssing implementation:
```

(c) [5 marks] Draw the Hash Table that results from adding the following hash codes to a hash table of size 7, using the division method and linked chaining: **11, 7, 4, 22, 74, 6**

```
You have to use your imagination a bit to interpret this pic!:
index:  0   1   2   3   4   5   6

array:  #   #   #   #   #   #   #
        |   |           |       |
linked  7  22          11       6
nodes:                  |
                        4
                        |
                       74
```

(d) [5 marks] Draw the Hash Table from the previous question using a table size of 7, and open addressing using *linear probing*.

```
index:  0   1   2   3   4   5   6
array:  7  22   6      11   4  74
```

(e) [5 marks] Draw the Hash Table from the previous questions, using a table size of 7, and open addressing using *quadratic probing*.

```
index:  0   1   2   3   4   5   6
array:  7  22       6  11   4  74
```

****

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Appendix (may be removed)

Brief (and simplified) specifications of some relevant interfaces and classes.

```
public class UI extends java.lang.Object
   public static void clearGraphics();
   public static void setColor(java.awt.Color col);
   public static void drawOval(double x, double y, double width, double height);
   public static void fillOval (double x, double y, double width, double height);

public class Scanner
   public boolean hasNext(); // there is more to read
   public String next(); // return the next token (word)
   public String nextLine(); // return the next line
   public int nextInt(); // return the next integer

public class Random
   public int nextInt(int n);  // return a random integer between 0 and n−1
   public double nextDouble(); // return a random double between 0.0 and 1.0

public interface Iterator <E>
   public boolean hasNext();
   public E next();
   public void remove();

public interface Iterable <E>            // Can use in the "for each" loop
   public Iterator <E> iterator();

public interface Comparable<E>           // Can compare this to another E
   public int compareTo(E o);

public interface Comparator<E>           // Can use this to compare two E's
   public int compare(E o1, E o2);
public interface Collection<E>
   public boolean isEmpty();
   public int size ();
   public boolean add();
   public Iterator <E> iterator();

public interface List<E> extends Collection<E>
   // Implementations: ArrayList
   public E get(int index);
   public void set(int index, E element);
   public void add(E element);
   public void add(int index, E element);
   public void remove(int index);
   public void remove(Object element);

public interface Set extends Collection<E>
   // Implementations: ArraySet, SortedArraySet, HashSet
   public boolean contains(Object element);
   public boolean add(E element);
   public boolean remove(Object element);

public interface Queue<E> extends Collection<E>
   // Implementations: ArrayQueue, LinkedList
   public E peek ();                        // returns null if queue is empty
   public E poll  ();                       // returns null if queue is empty
   public boolean offer (E element);

public class Stack<E> implements Collection<E>
   public E peek ();                        // returns null if stack is empty
   public E pop ();                         // returns null if stack is empty
   public E push (E element);

public interface Map<K, V>
   // Implementations: HashMap, TreeMap, ArrayMap
   public V get(K key);                     // returns null if no such key
   public void put(K key, V value);
   public void remove(K key);
   public Set<Map.Entry<K, V>> entrySet();
```