VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

EXAMINATIONS – 2019

TRIMESTER 3

COMP 103
INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS

Time Allowed: TWO HOURS

CLOSED BOOK

Permitted materials: Silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination.

Printed foreign language–English dictionaries are permitted.

No other material is permitted.

Instructions: Attempt ALL Questions.
The examination will be marked out of 120 marks.
Brief Documentation is at the end of the examination script
Answer in the appropriate boxes if possible — if you write your answer elsewhere, make it clear where your answer can be found.
There are spare pages for your working and your answers in this examination, but you may ask for additional paper if you need it.

Questions:

1. Properties of Collections [16]
2. Lists, Maps, and Comparable [24]
3. Complexity: Big-O costs [13]
4. Simulation with Collections [27]
5. Traversing General Trees [30]
6. Traversing Graphs [10]

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.

Question 1. Properties of Collections **[16 marks]**

For these questions, circle "true" or "false" for each property. (Note some properties may be true for more than one type.)

(a) **[4 marks]** For a Stack, state whether each property is true or false.

true/false	The order of values in the collection is important
true/false	Items are added and removed from the same end
true/false	The first item added is the first item to be removed
true/false	The collection cannot contain any duplicate items

(b) **[4 marks]** For a Set, state whether each property is true or false.

true/false	Items can be added at a specified position
true/false	Items must have a natural ordering to be stored in the collection.
true/false	The time an item is added does not affect when it is removed
true/false	The collection cannot contain any duplicate items

(c) **[4 marks]** For a Map, state whether each property is true or false.

true/false	Items must be added with a key
true/false	The order of items in the collection can be changed
true/false	The time an item is added affects when it is removed
true/false	The order of items in the collection is unimportant

(d) **[4 marks]** For a PriorityQueue, state whether each property is true or false.

true/false	Items are stored in a sorted order
true/false	Items must always implement the Comparable interface
true/false	The first item added is the first item to be removed
true/false	Adding an item is less efficient than adding to a Queue

Question 2. Lists, Maps, and Comparable **[24 marks]**

The shape of molecules can be very important for understanding their function, especially for large biological molecules. Using X-ray crystallography and other techniques, chemists or biochemists can identify the structure of a molecule by identifying the (x, y, z) coordinates of each atom in the molecule.

The MoleculeRenderer program produces a simple visualisation that draws each atom in a molecule as a coloured circle. To ensure a correct 3D visualisation, the atoms that are further from the viewer are drawn before atoms that are closer to the viewer, so atoms in front will cover any atoms behind. The MoleculeRenderer program also lets the user step left or right around the molecule in 5 degree steps to view the molecule from multiple (horizontal) directions.

The program has a molecule field containing a List of Atoms.

```
private List<Atom> molecule;
```

It also has an elements field, containing a Map of String to Elements.

```
private Map<String, Element> elements;
```

Each Atom has fields storing the x, y, and z positions of the atom within the molecule, and methods:

- to render the atom onto the screen
- to compare atoms by their z position **(Subquestion a)**
- to calculate the distance between two atoms if the molecule is rotated

Documentation for the Atom class is below. You do not need to know (or use) any of the methods in the Element class.

```
public class Atom implements Comparable<Atom>{
    public Atom(double x, double y, double z, Element e); // constructor
    public double getX();          // returns this atom's x position
    public double getY();          // returns this atom's y position
    public double getZ();          // returns this atom's z position
    public void render(double rotation ); // draws the atom on the screen
    public int compareTo(Atom o); // compares atoms by z position
    // Returns the distance from this atom to the other atom, given
    // the specified rotation
    public double distanceInFront(Atom other, double rotation );
}
```

(Question 2 continued on next page)

(Question 2 continued)

(a) **[4 marks]** Complete the following compareTo() method for the Atom class.

It should compare the atoms based on their z positions. Larger z values are further away from the screen, and should be at the front of the list so they are drawn first.

```
    public int compareTo(Atom other){




    }
```

(b) **[4 marks]** Complete the following showFromFront() method for the MoleculeRenderer class.

The method should sort the atoms in the molecule field by their Z values, and then render each atom to the screen. Don't forget to clear the graphics pane first.

NOTE: The rotation is 0 when the molecule is being viewed from the front.

```
public void showFromFront(){




}
```

(Question 2 continued on next page)

(Question 2 continued)

(c) **[6 marks]** Complete the following showFromViewDirection() method in the MoleculeRenderer class.

The method should sort the atoms in the molecule field based on the rotation of the molecule and their distance in front of each other, and then render each atom to the screen. Don't forget to clear the graphics pane first.

Note: the current rotation of the molecule is passed in as a parameter.

```
public void showFromViewDirection(double rotation){

    // Sort the molecule
    Collections.sort(molecule, (                ,                ) -> {









    });
    // Now clear the graphics and draw the atoms









}
```

(Question 2 continued on next page)

(Question 2 continued)

(d) **[10 marks]** The MoleculeRenderer program needs to be able to load molecules from a text file.

The text file consists of one line per atom in the molecule, with each line containing the element of the atom (Carbon, Oxygen, Hydrogen, etc) followed by the position of the atom (x y z).

For example, a sample input file might have the first three lines:

```
O     0.1 -111.8  18.7
C     0.1  -37.8   5.7
C   -39.9   11.2  -3.3
```

Assume that the elements fields contains all of the elements, and that the symbol for each element is the key (O,C,H,etc).

Complete the method loadFromFile(...) that will open the specified file, read in each atom, and create a new List<Atom> to store in the molecule field.

```
public void loadFromFile(String filename) {

    molecule = new ArrayList<Atom>();
    try {
        Scanner in = new Scanner(new File(filename));
        while(in.hasNext()) {









    } catch (IOException e) {
        UI.printMessage("Error loading file.")
    }
}
```

Question 3. Complexity: Big-O costs **[13 marks]**

For each question below, work out the cost (in Big-O notation) by

- working out the cost of performing each line once.
- working out the number of times each line will be performed.
- computing the total cost.

(a) **[4 marks]** What are the Big-O costs of the fragments of code below. Assume the size of the list is n.

```
List<String> list = new ArrayList<String>();        // cost = O(      ) times=
Set<String> set = new TreeSet<String>();             // cost = O(      ) times=

for (String word : list ) {
    set.add(word);                                   // cost = O(      ) times=
}
for (String word : set) {
    UI.println(word);                                // cost = O(      ) times=
}
 // Total Cost = O(          )
```

(b) **[4 marks]** What are the Big-O costs of the fragments of code below? Assume the size of the list is n, and that the list is kept in order.

```
public int aMethod(List<String> list, String value) {
    int a = 0;
    int b = list.size();                             // cost: O(      ) times:
    while ( a < b ) {
        int c = (a + b) /2;                          // cost: O(      ) times:
        int d = value.compareTo(list.get(c));        // cost: O(      ) times:
        if (d == 0) return c;                        // cost: O(      ) times:
        if ( d < 0 ) b = c;                          // cost: O(      ) times:
        else a = c+1;                                // cost: O(      ) times:
    }
    return -1;
}
    // Total Cost = O(          )
```

(Question 3 continued on next page)

(Question 3 continued)

(c) **[5 marks]** A program has a TreeSet containing all the unique words in a document.

When the TreeSet has 1,000 unique words, the program takes 12 milliseconds to find the longest word in the document.

If the TreeSet had 10,000 words, how long would you expect the program to find the most common word? Explain why.

Question 4. Simulation with Collections **[27 marks]**

Suppose you are writing a program to simulate the distribution of rendering jobs for a local digital effects workshop.

At each timestep, the program

- decides whether to create a new render job, and if so adds the new job to the back of the queue of the correct rendering server.
- advances the rendering of the job at the front of each queue by one time "tick".
- removes any job that has now completed rendering from its queue and saves its output.

You are to write the addJob and advanceAllJobs methods.

The RenderSimulation class has the following field, constructor and run() method.

```
public class RenderSimulation{
    public static int NUM_SERVERS = 50;
    private List<Queue<Job>> allServers = new ArrayList<>();

    public RenderSimulation(){
        for(int i=0; i< NUM_SERVERS; i++) {
            allServers .add(new ArrayDeque<Job>());
        }
    }

    public void run (){
        int time = 0;
        while (true){
            time++;
            if (Math.random()<0.05) {
                addJob(new Job(time));   // subquestion (a)
            }
            advanceAllJobs();            // subquestion (b)
        }
    }
}
```

The Job class has the following constructor and methods:

```
Order class:
    public Job(int time);
    public void advanceJobByTick();
    public boolean completedRendering();
    public int getRemainingTime();
    public void saveJobOutput();
```

Hint: sketch a diagram of the content of allServers.

(Question 4 continued on next page)

(Question 4 continued)

(a) **[12 marks]** Complete the following addJob(...) method which should find out which server has the shortest remaining total processing time in its queue (the lowest total remaining time for all jobs in the queue) and add the job to that server's queue.

```
public void addJob(Job j){




}
```

(b) **[10 marks]** Complete the following advanceAllOrders() method which should

- advance all the orders that are at the head of their queue.
- remove any orders that have been completed

```
public void advanceAllOrders(){




}
```

(Question 4 continued on next page)

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.

(Question 4 continued)

(c) **[5 marks]** The digital effects company wants to introduce a better distribution system for their rendering jobs, where jobs are prioritised based on how long they will take to process (longer jobs have a higher priority) and how long the job has been waiting (the more time a job is in the queue, the higher its priority gets).

Describe below, using bullet points, what changes would be needed in the program for it to implement this system.

Question 5. Traversing General Trees **[30 marks]**

This question concerns a taxonomic classification program that displayes taxonomic trees. The program uses GTNodes containing Taxonomy objects.

The Taxonomy class implements a toString() method that returns a description of the taxonomic label and category (e.g. `"Species: Homo sapiens"` for humans, or `''Order: Primates''` for the primate order)

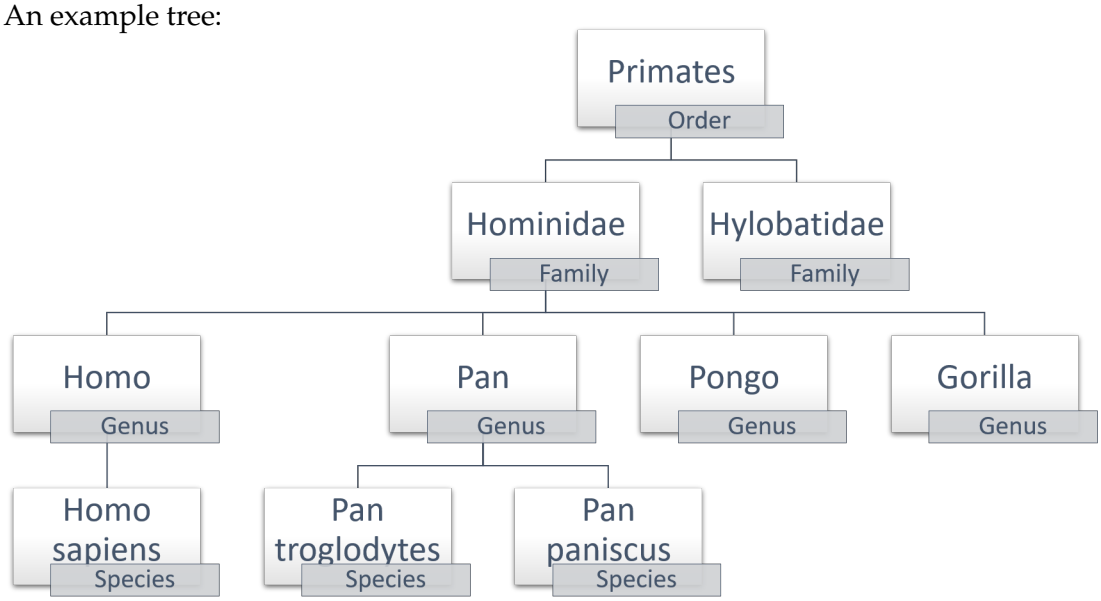
Note, this version of GTNode is **not** Iterable: to iterate through the children of a node, use:

```
for ( int i=0; i<node.numChildren(); i++){... node.getChild(i) ...}
```

```
class GTNode<E>:
    public GTNode(E item);                              // constructor
    public E getItem();                                 // return item in the node
    public int numChildren();                           // return number of children of the node
    public void addChild(GTNode<E> child);              // add a child
    public void addChild(int pos, GTNode<E> child);     // add a child at a specific position
    public GTNode<E> getChild(int i);                   // return i'th child
    public void removeChild(int i);                     // remove i'th child
```

```
class Taxonomy:
    public Taxonomy(String label, String category) // creates a new object
    public String getLabel()                       // returns the label
    public String getCategoty()                    // returns the category
    public String toString()                       // returns a String describing the taxonomy data,
                                                   // eg "Genus: Pongo" for the Orangutan genus
```

An example tree:



Note: The categories are "Order", "Family", "Genus", and "Species".

(Question 5 continued on next page)

(Question 5 continued)

(a) **[9 marks]** Complete the following printTaxonomicTree(...) method which is given the root node of the tree, and should print out all the cards in the tree, using indentation to show the structure.

For example, the tree on the previous page should be printed as:

```
Order: Primates
    Family: Hominidae
        Genus: Homo
            Species: Homo sapiens
        Genus: Pan
            Species: Pan troglodytes
            Species: Pan paniscus
        Genus: Pongo
        Genus: Gorilla
    Family: Hylobatidae
```

Hint: You can create a recursive function with more arguments.

```
public void printTaxonomicTree(GTNode<Taxonomy> node){




}
```

(b) **[2 marks]** what is the name for the tree traversal you used for printTaxonomicTree

(Question 5 continued on next page)

(Question 5 continued)

(c) **[9 marks]** The program needs to be able to add new nodes as new species are found.

Complete the addNewSpecies(...) method which is given the root of the tree, the *Genus* (eg: Homo, Pan) that the new species should be added under, and the name of the new species.

It should search the tree for a node of the Genus category that matches the given genus, and add a new node to it.

For the example tree on the previous page,

- addNewSpecies(root, "Homo", "Homo neanderthalensis") should add "Species: Homo neaderthanlesis" as a child of "Genus: Homo"
- addNewSpecies(root, "Canis", "Canis Lupus") should not add anything, as "Canis" is not a genus in the tree.

```
public void addNewSpecies(GTNode<Taxonomy> node, String genus, String newSpecies){




}
```

(Question 5 continued on next page)

(Question 5 continued)

(d) **[10 marks]** Sometimes, species are mis-classified, and need to be shifted to a new genus – or entire genuses to a new family.

Complete the following moveNode(...) method which is passed the root of a tree (node), the node that needs to be moved (child), and the node it needs to be moved to ((newParent).

- The method should move the given node from its current parent to the new parent.
- If any of the arguments are null, or if it can't find the current parent node, this method should return false.
- If the node is successfully moved, it should return true.

Note: GTNode does not have a parent field, so you will first need to find the parent of the node that needs to be moved.

```
public boolean moveNode(GTNode<Taxonomy> node, GTNode<Taxonomy> child,
                        GTNode<Taxonomy> newParent){









}
```

Question 6. Traversing Graphs **[10 marks]**

You are writing a program to investigate paths in a road map, where the map is defined as a graph of intersections connected by roads.

Your program stores information about all intersections in Set of Intersection objects.

```
private List<Intersection> allIntersections ;    // all intersections on the map
```

Each Intersection object contains a Set of other intersections it is connected to directly, and Intersection is Iterable so that you can use a foreach loop to iterate through the intersection's direct neighbours. You do not need to know any of the other fields or methods of the Intersection class.

(a) **[5 marks]** Complete the following numberOfBlocks method which returns the length of the path between two Intersections that are connected, directly or indirectly, on the map.

A value of -1 indicates that no path was found.

```
public int numberOfBlocks(Intersection i1, Intersection i2){
    Set<Intersection> visited = new HashSet<Intersection>();
    return numberOfBlocks(i1, i2, visited );
}

public int numberOfBlocks(Intersection p1, Intersection p2, Set<Server> visited){









}
```

(Question 6 continued on next page)

(Question 6 continued)

(b) **[5 marks]** Complete the following isConnected(...) method that checks that the road map is connected (ie: we can reach every intersection from every other intersection).

This method should be iterative.

```
public boolean isConnected( Intersection start ){




}
```

* * * * * * * * * * * * * * *

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.