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

EXAMINATIONS – 2019

TRIMESTER 2

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:

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 Queue, 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 TreeSet, 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	:	Items must implement consistent hashCode and equals methods
true/false	:	Adding an item is more efficient than adding to a HashSet

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

Suppose you are writing part of a program for a company that organises events. The program keeps track of people known to the company, people who have been invited to an event, and the guests, that is, people who have accepted their invitations.

The program has an allPeople field that contains a Map containing information about each Person known to the system. The key of the allPeople map is the personID (eg "P6243". Each Person object contains a name and a set of friends.

```
private Map<String, Person> allPeople;
```

The program also has an allEvents field containing a List of Events.

```
private List<Event> allEvents;
```

Each Event has fields storing

- a List of Invitees *i.e.*, people who were invited to the event;
- the number of guests attending *i.e.*, people who have accepted their invitation;

Documentation of the methods of the Event and Person classes:

```
public class Event {
    public List<Invitee> getInvitees()          // returns the List of Invitees in the event
    public void updateNumbersAttending(int num) // sets the number of invitees who have accepted
    public int getNumbersAttending()            // returns the number of invitees who have accepted
}
```

```
public class Person {
    public String getName()                 // returns the name of the person
    public Set<String> getFriends()         // returns a set of personIDs of friends
}
```

Implementation of the Invitee class:

```
public class Invitee {
    private final String personID; // unique person identifier (key for the allPeople Map)
    private boolean accepted = false; // whether the person has accepted the invitation or not

    public Invitee(String personID) {
        this.personID = personID;
    }

    public String getPersonID() {return personID;}

    public boolean hasAccepted() {return accepted;}

    public void recordAccepted() {accepted = true;}

}
```

(a) **[4 marks]** Complete the following updateAllNumbersAttending() method. For each Event in the allEvents field, it should compute and update the number of guests attending each event, based on how many Invitees in the event have accepted their invitation.

```
public void updateAllNumbersAttending(){




}
```

(Question 2 continued on next page)

(Question 2 continued)

(b) **[10 marks]** Complete the following friendsOfEvent(...) method which should return a collection of the personIDs of everyone who is a friend of a guest who has accepted their invitation for the given event. Friends who were invited to the event themselves should not be included.

```
public Set<String> friendsOfEvent(Event event){




}
```

(Question 2 continued on next page)

(Question 2 continued)

(c) **[10 marks]** It has been decided that the Invitee class needs to implement the Comparable interface (see documentation sheet). The natural order of the class should be by personID using normal alphabetical order. Provide the code that does this.

The revised Invitee class should work correctly with both TreeSets and HashSets: *i.e.,* hash code and comparisions should be consistent.

```
public class Invitee implements Comparable {

    private final String personID;      // personID cannot change once set
    private boolean accepted = false; // this initial value can be changed later

    public Invitee (String personID) {
        this.personID = personID;
    }

    public String getPersonID() {return personID;}

    public boolean hasAccepted() {return accepted;}

    public void recordAccepted() {accepted = true;}









}
```

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.

```
for (int k=list.size()-1; k>0; k--) {
    Float f = list.remove(rand.nextInt(k+1))      // cost = O(     ) times=
    list.add(k, f);                               // 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.

```
List<Integer> r = new ArrayList<Integer>();       // cost = O(     ) times=

for (Integer num : list) {
    for (int i = 0; i <= r.size(); i++) {
        if(i== r.size() || r.get(i) > num) {      // cost = O(     ) times=
            r.add(i, num);                        // cost = O(     ) times=
        }
    }
}
 // Total Cost = O(     )
```

(Question 3 continued on next page)

(Question 3 continued)

(c) **[5 marks]** A program has a HashMap. The keys are all the words that are in a document. The value associated with a key is an integer that stores the number of times the word is in the document.

When the HashMap has 1,000,000 unique words, the program takes 120 milliseconds to find the most common word in the document.

If the HashMap had 10,000,000 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 customers placing their orders on an online fastfood delivery service that takes orders for multiple restaurants.

At each timestep, the program

- decides whether to create a new order, and if so adds the new order to the back of the queue of the correct restaurant.
- advances the preparation of the order at the front of each queue by one time "tick".
- removes any order that has now completed preparation from its queue and delivers the order.

You are to write the addOrder and advanceAllOrders methods.

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

```
public class DeliverySimulation {
    private Map<String, Queue<Order>> allRestaurants;

    public DeliverySimulation (){
        allRestaurants = new HashMap<String, Queue<Order>>();
    }

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

The Order class has the following constructor and methods:

```
Order class:
    public Order(int time);
    public void advanceOrderByTick();
    public boolean completedPreparation();
    public String getRestaurant();
    public void deliverOrder ();
```

Hint: sketch a diagram of the content of allRestaurants.

(Question 4 continued on next page)

(Question 4 continued)

(a) **[10 marks]** Complete the following addOrder(...) method which should find out which restaurant the order is for, and then add the order to the queue for that restaurant, creating the queue if there isn't one already.

```
public void addOrder(Order o){




}
```

(b) **[12 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 food delivery service wants to introduce a "preferred customer" system, where a customer can pay more money to get higher priority in the order queue and therefore get their food faster.

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 solitaire card game with cards laid out in a tree structure. The program uses GTNodes containing Card objects.

The Card class implements a toString() method that returns a description of the card (e.g. `"S-1"` for Ace of Spades, or `H-8` for the 8 of Hearts)

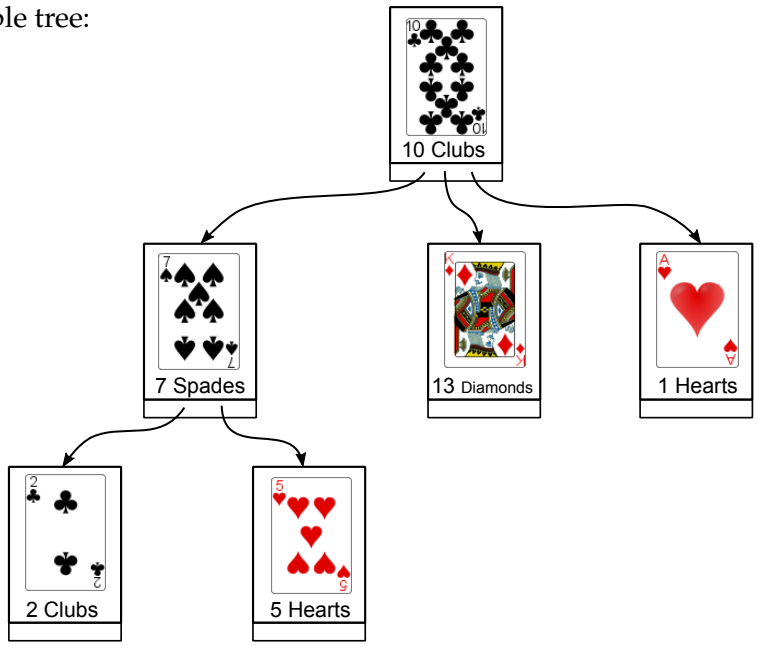
Note, this version of GTNode is **not** Iterable: to iterate through the children of a node, use:

```
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 Card:
    public Card(String suit, int value) // returns a card with the specified suit and value
    public int getValue()               // returns the numeric value of the card
    public String getSuit()             // returns the suit of the card
    public String toString()            // returns a String describing the card,
                                        // eg "D-13" for King (13) of Diamonds
```

An example tree:



Note: The suits are "Hearts", "Diamonds", "Clubs", "Spades". Values go from 1 to 13.

(Question 5 continued on next page)

(Question 5 continued)

(a) **[9 marks]** Complete the following printCardTree(...) 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:

```
C-10
   S-7
      C-2
      H-5
   D-13
   H-1
```

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

```
public void printCardTree(GTNode<Card> node){




}
```

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

(Question 5 continued on next page)

(Question 5 continued)

(c) **[9 marks]** One action in the game is to remove certain cards from the table. Complete the removeUnderSuit(...) method which is given the root of a tree and a suit. It should remove from the tree all the cards in any subtree under a card of the specified suit.

For the example tree on the previous page,

- removeUnderSuit(root, "Spades") should remove the 2 of Clubs and the 5 of Hearts because they are under the 7 of Spades.
- removeUnderSuit(root, "Clubs") should remove all the cards except the 10 of Clubs at the root.

```
public void removeUnderSuit(GTNode<Card> node, String suit){




}
```

(Question 5 continued on next page)

(Question 5 continued)

(d) **[10 marks]** [Harder!]In a correct layout of this card game, a child card must have the same or lower value than its parent. Therefore the layout above (in part (a)) is incorrect because H-13 (the king) is a child of S-10.

Complete the following fixLayout(...) method which is passed the root of a tree and fixes it to ensure that the layout is correct.

For every node in the tree with a Card that has a higher value than the Card in its parent, the node should be removed from the tree, and replaced by a new node containing a Card of the same suit, but a valid value. The children of the new node should be the children of the old node.

Note: The same card is allowed in the tree multiple times

```
public void fixLayout(GTNode<Card> node){




}
```

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

You are writing a program to investigate servers on a computer network.

Your program stores information about all servers in List of Server objects.

```
private List<Server> allServers;    // all servers in the network
```

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

(a) **[5 marks]** Complete the following secondDegreeServers method. It should return a Set of the Servers that are two steps from p: *i.e.*, servers that p can access via a path of two direct connections, but not directly.

```
public Set<Server> secondDegreeServers(Server p){




}
```

(Question 6 continued on next page)

(Question 6 continued)

(b) **[5 marks]** Complete the following isConnected method which returns true if two Servers are connected, directly or indirectly, in the network

```
public boolean isConnected(Server p1, Server p2){
    Set<Server> visited = new HashSet<Server>();
    return isConnected(p1, p2, visited );
}

public boolean isConnected(Server p1, Server p2, Set<Server> visited){









}
```

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

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.