

EXAMINATIONS – 2018

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:

1. Collection Types [16]
2. Lists, Maps, and Sorting [28]
3. Complexity, Big-O costs [24]
4. Simulation with Collections [22]
5. Traversing General Trees [20]
6. Traversing Graphs [10]

Question 1. Collection Types**[16 marks]**

(a) **[4 marks]** What is the key property of the Java Set type that distinguishes it from the more general Collection type?

(b) **[4 marks]** Suppose you are writing a program to plan the procession of floats in the city's annual Christmas parade.

- What collection type would you use to record the floats and their order in the parade?
- Justify your choice.

(c) **[4 marks]** Suppose you are writing a program to manage email requests for an on-line help-desk.

- Why would it be a bad idea to store the requests in a Stack?
- What would be a better Collection type?

(Question 1 continued on next page)

(Question 1 continued)

(d) [4 marks] A PriorityQueue has the same operations (offer, poll, and peek) as an ordinary Queue (eg ArrayDeque) but it behaves differently.

- How does PriorityQueue behave differently from ordinary Queues?
- Give an example of when a PriorityQueue would be a better choice than an ordinary queue.

Difference:

Example:

Question 2. Lists, Maps, and Sorting**[28 marks]**

Suppose you are writing part of a program that processes orders from an online store.

The program has an allOrders field containing a List of Orders.

- Each Order has a list of items in the order;
- Each Item has a productID and a count of how many of the product are wanted.

The program also has an allProducts field that contains a Map containing information about each of the Products. The key of the allProducts map is the productID (a String). Each Product contains a productID, a price, and a set of products that complement the product.

```
private List<Order> allOrders;
private Map<String, Product> allProducts;
```

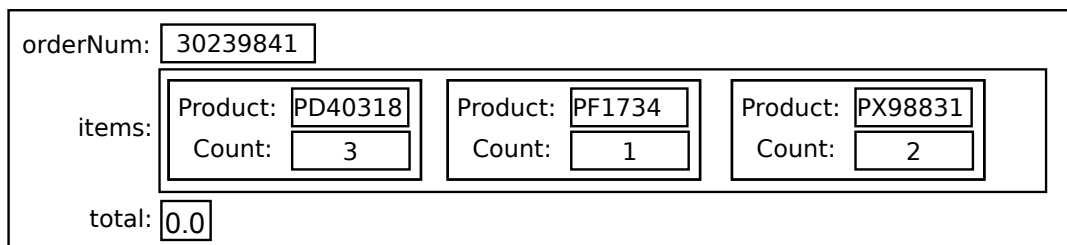
Here are descriptions of the methods of the Order, Item, and Product classes:

```
public class Order {
    public String getOrderNumber() // returns the invoice number of the order
    public List<Item> getItems() // returns the List of Items in the order
    public void setTotal(double amt) // records the total price of an order
}
```

```
public class Item {
    public String getProductID() // returns the productID being ordered
    public int getCount() // returns how many of the product are being ordered
}
```

```
public class Product {
    public String getID() // returns the productID of the product
    public double getPrice() // returns the price of the produce
    public Set<String> getComplements() // returns a set of productIDs of other products
                                        // that complement this product
}
```

Here is a sketch of one possible Order in allOrders. The Order contains three Items: 3 of product PD40318, 1 of product PF1734, and 2 of product PX98831. The total of the Order has not yet been computed.



(Question 2 continued on next page)

(Question 2 continued)

(a) [9 marks] Complete the following computeAllTotals method. For each Order in the allOrders field, it should compute the total price of the order, based on the price of each product in the order and the count of that product and then store the total in the Order.

```
public void computeAllTotals(){
```

```
}
```

(b) [11 marks] Complete the following makeRecommendation method which should return a Set of Products to recommend to the customer based on the products in the order. It should recommend a product if it is not in the order already and is a complement to any product in the order.

```
public Set<Product> makeRecommendation(Order order){
```

```
}
```

(Question 2 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 2 continued)

(c) [8 marks] Complete the following `sortItemList` method that should sort the List of Items in an Order. Items with higher counts should come before Items with lower counts. Items with the same count should be sorted by their `productID` in standard String order. Note that Items are not comparable. You may use a lambda or define a `compareItems` method.

```
public void sortItemList (Order order){
```

```
}
```

Question 3. Complexity: Big-O costs**[24 marks]**

(a) **[8 marks]** The two fragments of code below are very similar, and print out the same elements of a list. Assume the size of the list is n .

For each fragment, 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.

Note the two highlighted differences in the second fragment.

```

for ( int k = 0; k < list . size (); k++) {
    for ( int j = 0; j < list . size (); j++) {
        if (k==j){ // cost = O( ) times =
            UI . println ( list . get(k)); // cost = O( ) times =
        }
    }
}

// Total Cost = O( )

```

```

for ( int k = 0; k < list . size (); k++) {
    for ( int j = k; j < list . size (); j++) {
        if (k==j){ // cost = O( ) times =
            UI . println ( list . get(k)); // cost = O( ) times =
            break ;
        }
    }
}

// Total Cost = O( )

```

(b) **[8 marks]** Assume that allWords is a Collection containing n Strings. The following code fragment prints the Strings out in alphabetical order. Work out the cost in Big-O notation.

```

Queue<String> pq = new PriorityQueue<String>(); // cost = O( ) times =
for ( int i = allWords . size () - 1; i >= 0; i -- ) {
    pq . offer ( allWords . get ( i )); // cost = O( ) times =
}
while ( ! pq . isEmpty ()) { // cost = O( ) times =
    String item = pq . poll (); // cost = O( ) times =
    outFile . println ( item ); // cost = O( ) times =
}

// Total Cost = O( )

```

(Question 3 continued on next page)

(Question 3 continued)

(c) [4 marks] A program uses an ArrayList to store all the words in a document.

When the List has 100,000 words, the program takes 12 microseconds to remove the first word from the list.

If the List had 10,000,000 words, how long would you expect the program to take to remove the first word? Explain why.

(d) [4 marks] A program uses a HashSet to store all the *distinct* words in a document.

When the Set has 10,000 words, the program takes 120 nanoseconds to check whether the set contains a given word.

If the Set had 1,000,000 words, how long would you expect the program to take to check if the set contains a given word? Explain why.

Question 4. Simulation with Collections**[22 marks]**

Suppose you are writing a program to simulate passengers going through an airport security screening area. There are a set of queues, one for each X-ray machine. When a passenger arrives at the area, they join the shortest queue.

At each timestep, the program

- decides whether a passenger arrives, and adds them to the back of the shortest queue.
- advances the screening of the passenger at the front of each queue by one “tick”.
- any passenger who has now completed their screening is removed from their queue.

You are to write the `addPassenger` and `advanceAllScreening` methods.

The `Passenger` class has the following constructor and methods:

`Passenger` class:

```

public Passenger(int time);           // make a new passenger, recording arrival time
public void advanceScreeningByTick(); // removes 1 tick from remaining screening time
public boolean completedScreening(); // true if passenger has completed screening
public int getArrivalTime();          // returns time tick when passenger arrived

```

The `SecuritySimulation` class has the following field, constructor and run method.

```

public class SecuritySimulation {
    private Set<Queue<Passenger>> allLanes;

    public SecuritySimulation () {
        allLanes = new HashSet<Queue<Passenger>>();
        for (int i = 0; i < 5; i++) {           // initialise queues
            allLanes.add(new ArrayDeque<Passenger>());
        }
    }

    public void run () {
        int time = 0;
        while (true) {
            time++;
            if (Math.random() < 0.05) {
                addPassenger(new Passenger(time)); // subquestion (a)
            }
            advanceAllScreening ();                // subquestion (b)
        }
    }
}

```

Hint: sketch a diagram of the content of `allLanes`.

(Question 4 continued on next page)

(Question 4 continued)

(a) [11 marks] Complete the following addPassenger method which should add the passenger to the shortest queue (fewest number of passengers) in allLanes:

```
public void addPassenger(Passenger p){
```

```
}
```

(b) [11 marks] Complete the following advanceAllScreening method which should

- advance the screening of each passenger at the head of a queue in allLanes by one tick
- dequeue any passenger who has completed their screening

```
public void advanceAllScreening(){
```

```
}
```

Question 5. Traversing General Trees**[20 marks]**

In the lectures and assignment 6, we used a general tree implemented using GTNode to represent expressions for a calculator. The item in a GTNode was an operator or a number.

Note, this version of GTNode is **not** Iterable: You must use

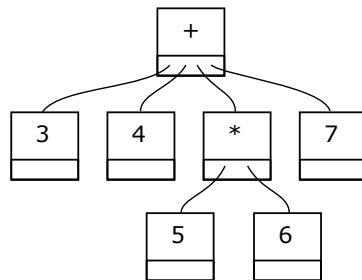
```
for ( int i=0; i<node.numChildren(); i++){... node.getChild(i) ...}|
```

to iterate through the children of a node.

```
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 GTNode<E> getChild(int i); // return i'th child
public void removeChild(int i); // remove i'th child
```

(a) [9 marks] Complete the following printBRPN method which should print out an expression in "Bracketed Reverse Polish Notation", where every subtree is printed out with brackets around it, and each operator is printed after its arguments.

For example, the expression tree



should be printed as: ((3)(4)((5)(6)*) (7)+)

```
public void printBRPN(GTNode<String> node){
```

```
}
```

(Question 5 continued on next page)

(Question 5 continued)

(b) [11 marks] An expression which has a + expression inside another + expression is a bit redundant. For example, the expression

$$(*\ 1\ (+\ 2\ (+\ 3\ 4)\ (+\ 5\ (+\ 6\ 7)\ 8)\ 9)\ 10)$$

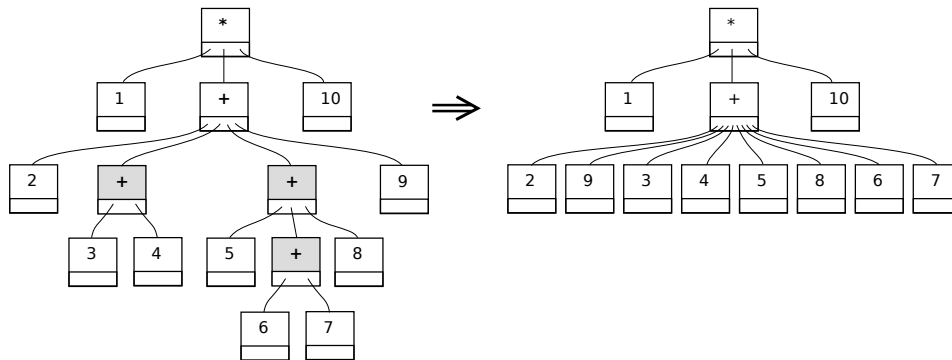
could be written more simply as

$$(*\ 1\ (+\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)\ 10)$$

or equivalently

$$(*\ 1\ (+\ 2\ 9\ 3\ 4\ 5\ 8\ 6\ 7)\ 10)$$

where the arguments of the lower three “+” nodes (shaded) have been moved up to the highest “+” node.



Complete the following fixNestedPlus method which should search an expression tree for nodes with a “+” operator which have a child node that also has a “+” operator. The method should remove the child node, and add all the children of the child node to the parent node. **Hint:** use a post-order traversal.

```
public void FixNestedPlus(GTNode<String> expr){
```

```
}
```

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

You are writing a program to set up temporary cellphone networks in disaster areas. A network consists of many small base stations, each of which is connected to several nearby base stations by cables or line-of-sight wireless links.

A critical property of such a network is that there should be a path through the network from every base station to every other base station. Your program needs a method for checking that.

Your program stores information about all the base stations in a List of BaseStation objects.

```
private List<BaseStation> allStations;    // List of all BaseStations in the network
```

Each BaseStation object contains a Set of its neighbours, and BaseStation is Iterable so that you can use a foreach loop to iterate through the neighbours of a BaseStation. You do not need to know any of the other fields or methods of the BaseStation class.

(a) **[5 marks]** Complete the following getConnected method which should return a Set of all the BaseStations to which a base is connected over the network (including itself).

Note: it uses a visited Set and returns the set at the end.

```
public Set<BaseStation> getConnected(BaseStation base){
    Set<BaseStation> visited = new HashSet<BaseStation>();
    getConnected(base, visited );
    return visited ;
}

public void getConnected(BaseStation base, Set<BaseStation> visited){

}

}
```

(Question 6 continued on next page)

(Question 6 continued)

(b) [5 marks] Complete the following `checkNetwork()` method which should use the `allStations` list and the `getConnected` method to check that every base station is connected to every other base station. For full marks, your method should be reasonably efficient and not do unnecessary work.

```
public boolean checkNetwork(){
```

```
}
```

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.

Documentation for COMP 103 Exam

Brief, simplified specifications of some relevant Java collection types and classes.

Note: E stands for the type of the item in the collection.

```
interface Collection< $E$ >
    public boolean isEmpty()           // cost:  $O(1)$  for standard collection classes
    public int size()                 // cost:  $O(1)$  for standard collection classes
    public void clear()
    public boolean add( $E$  item)
    public boolean contains(Object item)
    public boolean remove(Object element)
```

```
interface List< $E$ > extends Collection< $E$ >
    // Implementations: ArrayList
    public boolean isEmpty()
    public int size()
    public void clear()
    public  $E$  get(int index)           // cost:  $O(1)$ 
    public  $E$  set(int index,  $E$  element) // cost:  $O(1)$ 
    public boolean contains(Object item) // cost:  $O(n)$ 
    public void add(int index,  $E$  element) // cost:  $O(n)$  (unless index close to end.)
    public  $E$  remove(int index)         // cost:  $O(n)$  (unless index close to end.)
    public boolean remove(Object element) // cost:  $O(n)$ 
```

```
interface Set extends Collection< $E$ >
    // Implementations: HashSet, TreeSet
    public boolean isEmpty()
    public int size()
    public void clear()
    public boolean add( $E$  item)         // cost:  $O(1)$  for HashSet
                                         //  $O(\log(n))$  for TreeSet
    public boolean contains(Object item) // cost:  $O(1)$  for HashSet
                                         //  $O(\log(n))$  for TreeSet
    public boolean remove(Object element) // cost:  $O(1)$  for HashSet
                                         //  $O(\log(n))$  for TreeSet
```

```
class Stack< $E$ > implements Collection< $E$ >
    public boolean isEmpty()
    public int size()
    public void clear()
    public  $E$  peek()                   // cost:  $O(1)$ 
    public  $E$  pop()                     // cost:  $O(1)$ 
    public  $E$  push( $E$  element)         // cost:  $O(1)$ 
    // (peek and pop return null if the queue is empty)
```

```

interface Queue<E> extends Collection<E>
    // Implementations: ArrayDeque, LinkedList, PriorityQueue
    public boolean isEmpty()
    public int size()
    public void clear()
    public E peek () // cost : O(1) for ArrayDeque, LinkedList
                    // O(1) for PriorityQueue
    public E poll () // cost : O(1) for ArrayDeque, LinkedList
                    // O(log(n)) for PriorityQueue
    public boolean offer (E element) // cost : O(1) for ArrayDeque, LinkedList
                                    // O(log(n)) for PriorityQueue
    // (peek and poll return null if the queue is empty)

```

```

interface Map<K, V>
    // Implementations: HashMap, TreeMap
    public V get(K key) // cost : O(1) for HashMap
                       // O(log(n)) for TreeMap
    public V put(K key, V value) // cost : O(1) for HashMap
                                // O(log(n)) for TreeMap
    public V remove(K key) // cost : O(1) for HashMap
                           // O(log(n)) for TreeMap
    public boolean containsKey(K key) // cost : O(1) for HashMap
                                      // O(log(n)) for TreeMap
    public Set<K> keySet() // cost : O(1)
    public Collection<V> values() // cost : O(1)
    // get returns null if key not present ; put & remove return the old value, if any)

```

```

class Collections
    public void sort ( List<E> list); // cost = O(n log(n)) in general
                                    // O(n) almost sorted
    public void sort ( List<E> list, (E e1, E e2)->{..}); // cost = O(n log(n)) in general
                                                         // O(n) almost sorted
    public void swap(List<E> list, int i, int j); // cost = O(1)
    public void reverse ( List<E> list); // cost = O(n)
    public void shuffle ( List<E> list); // cost = O(n)

```

```

interface Comparable<E> // Items can be compared for sorting or a priority queue.
    public int compareTo(E other); // Comparable objects must have a compareTo method:
    // returns -ve if this comes before other ;
    // +ve if this comes after other ,
    // 0 if this and other are the same

```

```

interface Iterable <E> // Can use a foreach loop on these items
    public Iterator <E> iterator(); // Iterable objects must have an iterator method:

```

```

Integer and Double constants:
    Integer.MAX_VALUE; Integer.MIN_VALUE;
    Double.MAX_VALUE; Double.NaN; Double.POSITIVE_INFINITY; Double.NEGATIVE_INFINITY;

```
