VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

EXAMINATIONS – 2025
TRIMESTER 2
FRONT PAGE

COMP 103

INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS

21/10/2025

Time allowed: **120 MINUTES**

Permitted materials: **CLOSED BOOK**
You are allowed to use a (paper) language dictionary during the test. Electronic dictionaries, apps, and other digital resources are not permitted.

Instructions:
Attempt ALL **7** questions
The test will be marked out of a total of **120** marks.
You will be provided with a *concise Java documentation for Collections* and a *brief Java documentation*.
Record your answers in their designated spaces.
Assume that all necessary libraries for programming are already imported.

1. Properties of Collections [16 marks]

In questions (a) - (e), circle the *best one* answer:

a) Which Java collection type does allow duplicates? [3 marks]

i. TreeSet
ii. ArrayList
iii. HashSet
iv. HashMap

b) Which collection type would you use to record student data that makes it fastest to retrieve a student based on their ID number and to avoid duplicates when creating a new ID? [3 marks]

i. HashSet
ii. HashMap
iii. TreeMap
iv. ArrayDeque

c) Which Java Collections Framework class would you use to store a collection of elements where the order of insertion is preserved and fast addition and removal of elements from both ends is required (average-case O(1) time complexity)? [3 marks]

i. ArrayList
ii. PriorityQueue
iii. TreeSet
iv. ArrayDeque

d) Which of the following lines of code will result in a compilation or runtime error when executed? [3 marks]

i. `ArrayList<String> list = new ArrayList<>(); list.add("Hello");`
ii. `HashSet<String> set = new HashSet<>(); set.add("World"); set.add("World");`
iii. `HashMap<String, Integer> map = new HashMap<>(); map.put("Key", 42);`
iv. `TreeMap<Integer, String> map = new TreeMap<>(); map.add(1, "One");`

e) Consider the following piece of code that uses a `PriorityQueue` to store integers: [4 marks]

```
PriorityQueue<Integer> pq = new PriorityQueue<>();
pq.add(10);
pq.add(20);
```

Which of the following statements might result in a compilation or runtime error after executing the above lines?

i. `Integer top = pq.poll();`
ii. `Integer peek = pq.peek();`
iii. `pq.add(null);`
iv. `pq.clear();`

2. Using Collection [20 marks]

Consider a library management system in Java where books are represented by a `Book` class:

```
public class Book {
   private int id;
   private String title;
   private String author;
   private int year;
   public int getID() {return id;}
   public String getTitle() {return title;}
   public String getAuthor() {return author;}
   public int getYear() {return year;}
}
```

Each method below is partially implemented, and you must complete them.

a) The library uses a queue to manage book borrowing requests, where books are processed in the order they are requested, but urgent requests can be added to the front. The queue should support fast additions and removals at both ends ($O(1)$) and allow duplicate requests (e.g., multiple requests for the same book). Complete the `addBookToQueue` method to add the given book to the `borrowQueue`. **[6 marks]**

```
public class LibraryQueue {
   private ArrayDeque<Book> borrowQueue;

   public LibraryQueue() {
      borrowQueue = new ArrayDeque<>();
   }
   public boolean addBookToQueue(Book book, boolean urgent) {
      // TODO: Complete this method (need to check if book is null)
      //       Return true/false if added/not added









   }
}
```

Question 2 continues on next page

Question 2 continued from previous page

b) The library maintains a mapping from book IDs to Book objects for fast lookups ($O(1)$) and supports an undo feature for recent additions using a FIFO protocol, storing up to 3 books.

In the `LibraryDatabase` class on the facing page, you have a `HashMap<Integer, Book>` called `bookMap` for storing books by their IDs and an `ArrayDeque<Book>` called `undoDeque` for managing the undo feature.

- Complete the `addBookToMap` method to add the book to `bookMap` using its ID as the key and push it to `undoDeque`. If `undoDeque` has 3 books, remove the oldest before pushing. Return false if the book is `null` or its ID already exists in `bookMap`; otherwise, return `true`.
  [7 marks]

- Complete the `undoEarliestAddition()` method to remove the earliest added book from `undoDeque`, remove it from `bookMap` using its ID, and return the book. Return `null` if `undoDeque` is empty. **[7 marks]**
  Hint: *Undo here means removing the earliest added book (FIFO), not the most recent (LIFO). This uses `ArrayDeque` as a queue, not a stack.*

Question 2 continues on next page

Question 2 continued from previous page

```
public class LibraryDatabase {
   private HashMap<Integer, Book> bookMap;
   private ArrayDeque<Book> undoDeque;  // For FIFO protocol with 3-book limit

   public LibraryDatabase() {
      bookMap = new HashMap<>();
      undoDeque = new ArrayDeque<>();
   }

   public boolean addBookToMap(Book book) {
   // TODO: Complete this method (check if book is null or ID exists in bookMap)









   }
   public Book undoEarliestAddition() {
   // TODO: Complete this method (need to check if undoDeque is empty)









   }
}
```

3. Simulation [15 marks]

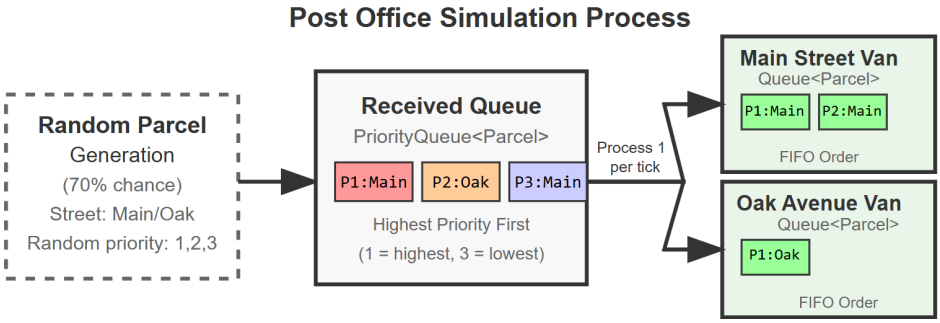
A local post office receives parcels and processes them based on priority. Incoming parcels are added to an initial priority queue, where lower numbers indicate higher priority (1 = highest, 3 = lowest). Parcels are then dispatched to delivery vans, each responsible for a specific street.
Each parcel is represented by the `Parcel` class:

```
public class Parcel {
   private String street;
   private int priority;    // 1 = highest priority, 3 = lowest priority
   public Parcel(String street, int priority) {
      this.street = street;
      this.priority = priority;
   }
   public String getStreet() { return street; }
   public int getPriority() { return priority; }
}
```

The post office operates two delivery vans: one for *Main Street* and one for *Oak Avenue*. Each van maintains its own queue of parcels to deliver. The simulation runs in discrete time ticks. At each tick:

- With 70% probability, a new parcel is created with a random street (`Main` or `Oak`) and a random priority (1–3). This parcel is added to the initial priority queue `receivedQueue`.
- If `receivedQueue` has more than 3 parcels, the highest-priority parcel is removed and added to the correct delivery van queue. If the van queue does not exist yet, it is created and added to the map `deliveryVans`.

Below is a graphical representation of the process:



The simulation uses `createRandomParcel()` helper method that generates a parcel with a random street (`"Main"` or `"Oak"`) and priority (1, 2 or 3):

```
   private Parcel createRandomParcel() {
      String[] streets = {"Main", "Oak"};
      String street = streets[random.nextInt(streets.length)];
      int priority = random.nextInt(3) + 1;
      return new Parcel(street, priority);
   }
```

Question 3 continues on next page

Question 3 continued from previous page

Your task is to complete the following class by filling in the missing lines marked with TODO: comments.

```
public class POSimulation {
   private Random random;
   private PriorityQueue<Parcel> receivedQueue; // to store received parcels, ordered
   ↪  by priority
   private Map<String, Queue<Parcel>> deliveryVans; // to store delivery van queues for
   ↪  each street.
   public POSimulation() { // Constructor: initialises fields
      random = new Random(); // Done for you
      // TODO: Initialise receivedQueue with comparator (lambda) for priority ordering
      // TODO: Initialise deliveryVans map









   }
   public void tick() { // One tick
      if (random.nextDouble() < 0.7) { // With 70% probability,
         // TODO: Add a random parcel to receivedQueue









      }
      // TODO: If receivedQueue has more than 3 parcels, process one parcel:
      // Add parcel to appropriate delivery van queue; if there is not a queue for
      // the street, create one first and put the pair (street, queue) in deliveryVans
















   }
}
```

4. Complexity: Big-O costs [15 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) What are the Big-O (worst-case) costs of the fragments of code below. Both `SetA` and `SetB` are `Set<Integer>` and contain n unique integers. [5 marks]

```
Set<Integer> result = new TreeSet<Integer>(); // cost = O(       ) times =
for(Integer n1: SetA){
   for(Integer n2: SetB){
      if (n1 != n2){                          // cost = O(       ) times =
         result.add(n1);                      // cost = O(       ) times =
      }
   }
}                                             // TOTAL cost = O(          )
```

b) What are the Big-O (worst-case) costs of the fragments of code below. Both `ListA` and `ListB` are `List<Integer>` and contain n integers. [6 marks]

```
Set<Integer> result = new TreeSet<Integer>(); // cost = O(     )    times =
for (Integer n : ListA) {
   result.add(n);                             // cost = O(     )    times =
}
for (int i = 1; i < ListB.size(); i = i * 2) {
   Integer n = ListB.get(i);                  // cost = O(     )    times =
   if (result.contains(n)) {                  // cost = O(     )    times =
      result.remove(n);                       // cost = O(     )    times =
   }
}
                                        // TOTAL cost = O(        )
```

Question 4 continues on next page

Question 4 continued from previous page

---

c) Anna builds a scheduling system for her research lab using a `PriorityQueue` in Java. The queue stores upcoming tasks, prioritised by earliest deadline. When the queue contains 1,024 tasks ($\approx 2^{10}$), inserting a new task takes 20 microseconds. If the queue grows to 32,768 tasks ($\approx 2^{15}$), how long would you expect an insertion of a new task to the queue to take? Explain your reasoning. **[4 marks]**

```
YOUR ANSWER:
```

5. Recursion [15 marks]

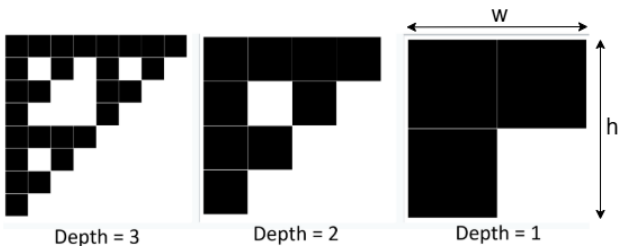
Complete the recursive method `drawTiling` that draws a pattern of filled rectangles on the graphics pane. It takes the following parameters::

- `double x`, `double y`: coordinates of the top-left corner of the current rectangle,
- `double w`, `double h`: width and height of the current rectangle,
- `int depth`: the level of recursion.

The method should do the following:

- Base case (`depth == 0`): Draw a filled rectangle at position `(x+1, y+1)` with width `w-1` and height `h-1`.
- Recursive case (`depth > 0`): Recursively draw three smaller rectangles, each with half the width and half the height of the current rectangle:
  1. Top-left quarter: `(x, y)`
  2. Bottom-left quarter: `(x, y + h/2)`
  3. Top-right quarter: `(x + w/2, y)`

The image shows examples of the tiling pattern for different depths.




```
public void drawTiling(double x, double y, double w, double h, int depth) {
   // TODO: Complete this recursive method




}
```

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working. Specify the question number for work that you do want marked.

6. General Trees [23 marks]

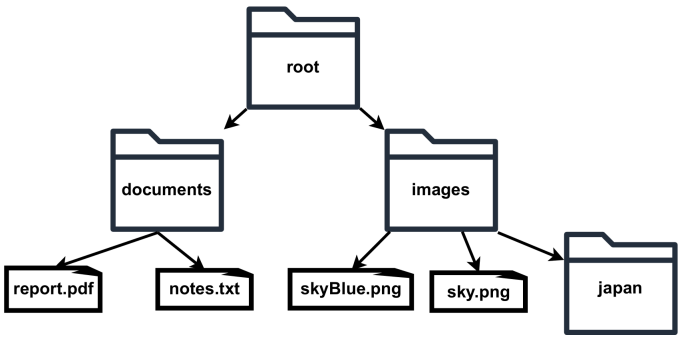
In this question we represent a simple File System as a general tree.

- Each node is either a *File* (with a name) or a *Folder* (with a name and children).
- A Folder has a name and may contain zero or more children (which can be either Files or other Folders).
- A File has a name but does not contain any children.

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

```
public FileSystemNode(String name, boolean isFile );
   // Node constructor: creates a new node with the given name
   // If isFile == True, this node is a File . Otherwise, it is a Folder
public String getName();
   // returns the name of the file or folder
public boolean isFile ();
   // true if this node is a File , false if Folder
public List<FileSystemNode> getChildren();
   // returns list of children (empty if File )
```

Note that `FileSystemNode` is not iterable.
The following figure shows an example of a general tree for a file system, in which `root`, `documents`, `images`, `japan` are folders, and `report.pdf`, `notes.txt`, `skyBlue.png`, `sky.png` are files.



a) Complete the `printFileSystem(FileSystemNode root)` method (on the facing page) which uses recursion to print all files and folders in the tree using dashes(-) to indicate depth. Both files and folders should print their names. **[10 marks]**

For example,for the above tree, `printFileSystem()` should print:

```
root
- documents
-- report.pdf
-- notes.txt
- images
-- skyBlue.png
-- sky.png
-- japan
```

```
public void printFileSystem(FileSystemNode root){
   // TODO: complete this method


}
```

Question 6 continues on next page

b) Complete the `insertFile` method, which inserts a new file into the directory specified by `String directoryPath`. **[13 marks]**

- The parameter `directoryPath` is a String such as `"root/documents/reports"`, which represents a sequence of nested folders separated by slashes
- You can assume that every `directoryPath` starts with `"root/"` where `root` is the name of the root folder
- If any folder in the path does not exist, create it using the constructor
- At the end of the path, insert a new file with the name `fileName`

```
public void insertFile(FileSystemNode root, String directoryPath, String fileName){
      if (root == null) {
      return;
   }
   // Split path into parts (e.g., ["root", "documents", "reports"])
   String[] parts = directoryPath.split("/");
   // TODO: complete this method









}
```

7. Graphs [16 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 an enhanced for 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) 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. **[8 marks]**

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

}
```

Question 7 continues on next page

b) Complete the following `isConnected` method which returns true if two `Servers` are connected, directly or indirectly, in the network **[8 marks]**

```
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. Specify the question number for work that you do want marked.

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working. Specify the question number for work that you do want marked.

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