VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

EXAMINATIONS – 2026

TRIMESTER 1

XMUT 103 MOCK TEST

INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS

22/06/2026

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– there are 6 questions.
The examination will be marked out of 79 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	**Marks**	
1. Trees	[8]	
2. General Tree	[20]	
3. Traversing General Trees	[20]	
4. Traversing Graph	[20]	
5. Priority Queue	[10]	
6. Binary Search	[10]	
	TOTAL:	

Question 1. Multiple choice ☑ **[10 marks]**

(a) **[1 mark]** Which of the following are types of tree traversal?

- ☐ Inorder Traversal
- ☐ Preorder Traversal
- ☐ Postorder Traversal
- ☐ Proorder Traversal

(b) **[1 mark]** In a Binary Search Tree (BST), inorder traversal produces?

- ☐ Sorted order of elements
- ☐ Reverse sorted order always
- ☐ Random order
- ☐ Ascending order (if BST property is maintained)

(c) **[1 mark]** Which traversals belong to Depth-First Search (DFS)?

- ☐ Inorder Traversal
- ☐ Preorder Traversal
- ☐ Postorder Traversal
- ☐ Level Order

(d) **[1 mark]** Which statements about Preorder Traversal are correct?

- ☐ Root is visited last
- ☐ Root is visited first
- ☐ Traversal order is Root $\rightarrow$ Left $\rightarrow$ Right
- ☐ Useful for copying a tree

(e) **[1 mark]** Which statements about Postorder Traversal are true?

- ☐ Useful for deleting a tree
- ☐ Order is Left $\rightarrow$ Right $\rightarrow$ Root
- ☐ Traversal order is Root $\rightarrow$ Left $\rightarrow$ Right
- ☐ Useful for copying a tree

(f) **[1 mark]** Which data structures are commonly used for tree traversal?

- ☐ Array
- ☐ Stack
- ☐ Linked List
- ☐ Queue

(g) **[1 mark]** Level Order Traversal uses:

- ☐ Breadth-First Search (BFS)
- ☐ Stack
- ☐ Depth-First Search (DFS)
- ☐ Queue

(h) **[1 mark]** Which traversals visit every node exactly once?

- ☐ Inorder
- ☐ Preorder
- ☐ Postorder
- ☐ Level order

Question 2. General Tree **[20 marks]**

- University
  - Computing
    - Java
    - Data Structures
    - AI
  - Engineering
    - Mechanics
    - Electronics
  - Business
    - Accounting
    - Marketing

(a) **[5 marks]** Create the node class for the tree with the data: name of department/course the methods:

- To add a child node
- To remove a child node
- To get the name of the node

```
class TreeNode {
    // Fields



    // Constructor




    // Add child node



    // Remove child by name



    // Get the node's name


}
```

(b) **[10 marks]** Create the tree in the main method of your program. The name of the root is "university".

```
public class UniversityTree {
    public static void main(String[] args) {

        // Departments



        // Add departments to university



        // courses

















        // Print tree
        printTree( university , 0);
    }
}
```

(c) **[5 marks]** Create the printTree() method using Recursive

```
public static void printTree(TreeNode node, int level ) {









}
```

Question 3. Traversing General Trees **[20 marks]**

This question concerns a Tic Tac toe game with possible moves laid out in a tree structure. The program uses containing GameState objects.

Note, this version of Gamestate is **not** Iterable.

```
class GameState {
    private String board;
    private List<GameState> children;
    public String getBoard() {} // return board
    public List<GameState> getChildren() {} //return children
    public boolean isWinningState() {} // returns true if X has won
}
```

(a) **[5 marks]** Complete the following printTree(...) method which is Write a recursive method that prints all board states in the game tree using a depth-first traversal.

```
public static void printTree(GameState node) {




}
```

(b) **[8 marks]** Write a recursive method that counts the total number of nodes (game states) in the tree.

```
public static int countNodes(GameState node) {




}
```

(Question 3 continued on next page)

(Question 3 continued)

(c) **[8 marks]** Write a recursive method that searches the Tic-Tac-Toe game tree for a specific board state.The method should return true if the target board is found anywhere in the tree and false otherwise. Assume the target board is provided as a String.
String targetBoard

```
public static boolean findBoard(GameState node, String targetBoard) {




}
```

Question 4. Traversing Graphs **[15 marks]**

You are writing a program to create the post office's route using graph.

Your program stores information about all nodes in a list of Buildings objects:

```
private Set<Building> neighbours;; // List of all nodes in delivery route
```

(a) **[10 marks]** Create the Building Class that has getName() and addNeighbour() methods

```
class Building implements
{









    // Make Building iterable
    @Override
    public Iterator <Building> iterator() {
        return neighbours. iterator ();
    }
}
```

(Question 4 continued on next page)

(Question 4 continued)

(b) **[10 marks]** Complete the bfs(...) method toperforms a Breadth-First Search (BFS) traversal on a graph of buildings. Its role is to visit all reachable buildings level by level, starting from a given building.

```
public static void bfs(Building start) {




}
```

Question 5. Priority Queue **[30 marks]**

An airline wants to manage boarding passengers using a Priority Queue implemented with a Heap.

1. First Class - Priority = 3
2. Business Class - Priority = 2
3. Economy Class - Priority = 1

```
public class Passenger {
  private String name;
  private int priority ;
  public Passenger(String name, int priority ) {}// constructor
  public String getName() {} //return caseName
  public int getPriority () {} // return priority ;

}

public class HeapQueue {
    private Passenger[] heap;
    private int size ;
    public HeapQueue() {
```

```
        heap = new Passenger[10]; // Initial capacity
        size = 0;
}
```

(a) **[10 marks]** Write the heapifyUp method used in a priority queue implemented with a binary heap. Its role is to restore the heap property after a new element is inserted, based on max-heap. Every parent node must have a priority greater than or equal to its children. If the newly inserted element has a higher priority than its parent, it must move upward.

```
private void heapifyUp(int index) {




}
```

(b) **[5 marks]** Write the peek() method for the HeapQueue class. The method should return the highest priority without removing any node.

```
public Passenger peek() {




}
```

Question 6. Binary Search **[10 marks]**

Write a binary search method to search for a word in a dictionary. The list of contacts is using Arraylist and is in the main program. Assume the program runs.

```
public static void main(String[] args) {
    String[ ] dict = {};

    public void searchWord(String word){}
    public static int binarySearch( String [] dict , String key) {}
}
```

Write the binarySearch method (Recursive) to get the index of word given.

```
public static int binarySearch( String [] dict , String key) {




}
```

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

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()
pubic 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
// Note: The String class is Comparable, and has this method
```

```
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;