

**EXAMINATIONS – 2023**  
**TRIMESTER 2**  
**FRONT PAGE**

**COMP 103**  
**INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS**  
**1/11/2023**

**Time allowed:** TWO HOURS

**Instructions:**

- This test is a **Closed-Book** test
- No calculators permitted
- Answer the questions in the provided spaces 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 test, but you may ask for additional paper if you need it.
- If you encounter a question that appears unclear, feel free to request **clarification** from the invigilator.
- A brief **Java Documentation** and a summary of **Collections and Methods** are made available with the test.
- You can assume that all libraries required for programs are imported and are available for you to use.
- The number of questions and the distribution of marks are as in the following table:

<b>Question</b>	<b>Max. mark</b>	<b>Earned mark</b>
1. Properties of Collections	20	
2. Using Collections	22	
3. Cost of Algorithms	20	
4. General Trees	30	
5. Traversing Graphs	12	
6. Binary Search	8	
7. Heaps	8	
<b>TOTAL</b>	<b>120</b>	

## Question 1. Properties of Collections [20 marks]

---

For questions 1.a., 1.b, 1.c, 1.d. and 1.e., circle the right answer(s) from the list.

### 1.a. [4 marks]

Which implementation of the 'Set' interface in Java's collections framework maintains elements in a sorted order?

1. ArrayList                      2. LinkedHashSet                      3. HashSet                      4. TreeSet

### 1.b. [4 marks]

Which data structure in Java's collections framework provides fast random access to elements but may be less efficient for insertions and removals in the middle?

1. HashSet                      2. PriorityQueue                      3. TreeMap                      4. ArrayList

### 1.c. [4 marks]

When implementing the `equals()` method for a class, which of the following statements is true regarding the `compareTo()` method if we want to *maintain consistency*, assuming the class has both methods implemented?

1. The `compareTo()` method should return -1 for any two objects that are considered equal by `equals()`.
2. The `compareTo()` method should return 0 for any two objects that are considered equal by `equals()`.
3. The `compareTo()` method should return 1 for any two objects that are considered equal by `equals()`.
4. None of the above is necessary.

### 1.d. [4 marks]

Consider the following lines of Java code. Which of these examples demonstrates the practice of 'programming to interface'? Select the line(s) that represent(s) the 'programming to interface' concept.

1. `ArrayList<String> list = new ArrayList<>();`
2. `Map<String, Integer> map = new HashMap<>();`
3. `Set<Double> set = new HashSet<>();`
4. `LinkedList<Character> linkedList = new LinkedList<>();`

### 1.e. [4 marks]

Which one of the following data structures in Java's Collection Framework cannot have a `null` element?

1. ArrayList                      2. HashSet                      3. TreeSet                      4. LinkedList

## 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. Using Collections [22 marks]

---

Given `WordData` class as bellow,

---

```
public class WordData {
    public String word; // a word encountered in the text
    public int count; // frequency count
    WordData(String w) {
        word = w;
        count = 1; // count is set to be 1 when a WordData object is created
    }
}
```

---

complete the following class called `WordFrequencyAnalyser`. The goal of this program is to count the frequency of words encountered in a text and save them along with their corresponding `WordData` objects in an appropriate data structure in *ascending* alphabetical order and ignoring the case of letters. You'll need to complete the following tasks:

### 2.a. [3 marks]

Declare and initialise a `TreeMap` to store the words and their corresponding `WordData` objects.

---

```
public class WordFrequencyAnalyser {
    // Answer to 2.a.=====
```

---

### 2.b. [8 marks]

Implement the `processWord` method to process a word and update the `WordData` objects in your data structure. If the word is encountered for the first time, create a new `WordData` object. If it has been encountered before, update the count. Ignore the case of the letters: e.g., `Book` and `book` are considered the same.

---

```
public void processWord(String word) {
    //Answer to 2.b.=====
```

---

```
}
```

---

### 2.c. [6 marks]

Implement the `getFrequencySorted` method to return a `List` of `WordData` objects from `words` that is sorted by word frequency in *descending order*. Use a lambda expression for sorting.

---

```
public List<WordData> getFrequencySorted() {  
    // Answer to 2.c:=====
```

```
}
```

---

### 2.d. [5 marks]

Implement the `print()` method to

1. Using the `map`, print all the words along with their frequencies where the words are sorted **alphabetically**, and then
2. Using the `list`, print all the words along with their frequencies where the words are sorted by **frequency**. Make use of `getFrequencySorted()` method.

---

```
public void print() {  
    // Answer to 2.d:=====
```

```
} // End of print() method  
} // End of WordFrequencyAnalyser class
```

---

### Question 3. Cost of Algorithms [20 marks]

#### 3.a. [5 marks]

Consider the Big-O (worst-case) costs of the fragment of code below, where `mylist` is an `ArrayList<String>`, of size `n`.

---

```
List<String> words = new ArrayList<String>(); //cost of line = O( ), times=
for (int i = 0; i < mylist.size(); i++) {
    for (int j = 0; j < mylist.size(); j++) {
        if (mylist.get(i).charAt(0) ==
            mylist.get(j).charAt(0)){ //cost of line = O( ), times=
            words.add(mylist.get(i)); //cost of line = O( ), times=
        }
    }
}

// Total Cost=O( )
```

---

#### 3.b. [5 marks]

Consider the Big-O (worst-case) costs of the fragment of code below, where `mylist` is an `ArrayList<Integer>`, of size `n`.

---

```
Set<Integer> numbers = new TreeSet<Integer>();// cost of line = O( ), times=
for (int i = 0; i < mylist.size(); i+=3) {
    if (mylist.get(i) % 2 == 0) { // cost of line = O( ), times=
        numbers.add(mylist.get(i)); // cost of line = O( ), times=
    }
}

// Total Cost=O( )
```

---

### 3.c. [5 marks]

A clinic uses a HashMap to store its patients, where the Patient ID is the key, and the patient's medical history is the value. If it takes 80 nanoseconds to retrieve a patient's medical history when the clinic has 10,000 patients, how long would you expect it to take when the clinic has 100,000 patients? Explain your reasoning.

---

**Answer to 3.c.**

---

### 3.d. [5 marks]

A hospital uses a *sorted* ArrayList to maintain its list of appointments in chronological (i.e., sorted by appointment time) order. When the hospital has 1,000 appointments scheduled, it takes 5 milliseconds to insert a new appointment into the correct position in the list. If the hospital had 10,000 scheduled appointments, how long would you expect it to take to insert a new appointment in the correct position? Explain your reasoning.

---

**Answer to 3.d.**

---

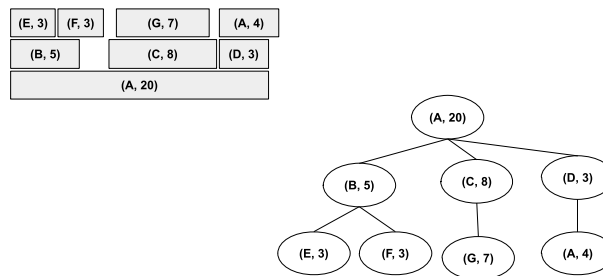
## Question 4. General Trees [30 marks]

In this question we use general trees to implement a simple Box Stacking game. Each box has a width, and contains a letter ("A", "B",...). The program represents each box as a Box object, which stores data about its width, the letter, and a list of any other boxes that are stacked directly on top. The Box class has the following methods:

Box **class**:

```
public String getLetter (); // get the letter of the box
public double getWidth(); // get the width of the box
public List<Box> getTopBoxes(); // get the list of the boxes
                                // directly on top of the selected box
public String toString (); // return a String with format "(letter, width)"
```

The following figure shows an example of a set of stacked boxes (left-hand side) and its representation as a general tree (right-hand side). (A, 20) represents a box with letter A and width 20. If `getTopBoxes()` is called on (A,20) it will return a list consisting of (B,5), (C,8) and (D,3). And, if it is called on (B,5) it will return a list consisting of (E,3) and (F,3)



### 4.a. [10 marks]

Complete `printBoxes()` method on the following page (page 9) which is given the root node, and prints out all the boxes in the tree, using indentation to show the structure. For example, the tree on the previous page should be printed as:

```
(A, 20)
  (B, 5)
    (E, 3)
    (F, 3)
  (C, 8)
    (G, 7)
  (D, 3)
    (A, 4)
```



---

```
public void printBoxes(Box root){  
    // Answer to 4.a.:=====
```

```
}
```

#### 4.b. [10 marks]

Complete the following `numOccurrences()` method which is given the root node and a letter. The method should return the number of boxes containing the given letter (in lowercase or uppercase).

For example, calling `numOccurrences(root, "A")` returns 2.

---

```
public int numOccurrences(Box root, String letter ){  
    // Answer to 4.b.: =====
```

```
}
```

#### 4.c. [10 marks]

In the game, a box is unstable if the total width of boxes directly on top is greater than its own width.

Complete the following `findUnstableBoxes()` method which is given the root node, and returns a list of unstable boxes.

For example, calling `findUnstableBoxes(root)` returns a list containing two boxes (B, 5) and (D, 3).

---

```
public List <Box> findUnstableBoxes(Box root){  
    // Answer to 4.c.:=====
```

```
}
```

---

## 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 5. Traversing Graphs [12 marks]

---

You are writing a travel app that helps customers to navigate in a train network containing multiple train stations. You can use a graph to represent the train network where each node represents a train station. Two stations are considered 'neighbours' if they are connected directly in the graph (without any other station/nodes in between). The Station class has the following methods:

---

```
Station class :
public String getName(); // get the name of the Station
public Set<Station> getNeighbours(); // get the set of neighbouring stations
```

---

### 5.a. [6 marks]

Complete the following `isConnected()` method which returns `true` if two Stations are connected, directly or indirectly, in the network and returns `false` otherwise.

---

```
public boolean isConnected(Station s1, Station s2){
    // Answer to 5.a.:=====
```

```
}
```

---

### 5.b. [6 marks]

Complete the following `withinDistance()` method, which should return a set of train stations reachable from the starting station (`start`) while considering a maximum number (`maxDist`) of intermediate stations that can be traversed.

---

```
public Set<Station> withinDistance(Station start, int maxDist) {  
    // Answer 5.b.:=====
```

```
}
```

---

## Question 6. Binary Search [8 marks]

---

### 6.a. [4 marks]

(A music store has a collection of albums sorted by release date. If the store has 1,024 albums and uses binary search to find an album, what is the maximum number of comparisons the store would need to make?)

---

**Answer to 6.a.:**

---

### 6.b. [4 marks]

Consider the sequence [10, 20, 30, 40, 50, 60, 70, 80, 90]. If you're searching for the number 35 using binary search, how many comparisons will you need before determining that 35 is not in the list?

---

**Answer to 6.b.**

---

## Question 7. Heaps [8 marks]

---

Background: In this course, we have studied the "heap": a complete binary tree, implemented using an array, that maintains the heap property. For a max heap, every parent node has a value greater than or equal to any of its children.

Suppose you have a max heap that was created as follows:

```
private List<Integer> heap = new ArrayList<>();
```

And which has since then been populated with several integer values. Below is a method that inserts a value into the Max Heap. It uses a helper method `pushUp(index)` which ensures the heap property is maintained after the insertion. Complete the `pushUp(index)` method.

```
public void insert(int value) {
    heap.add(value);           // add to the end of the heap
    pushUp(heap.size() - 1); // start pushing up from the last
                             // position (where the new value was added)
}
```

---

```
private void pushUp(int index) {
    // Answer to 7:=====
```

```
}
```

---

\*\*\*\*\* THE END \*\*\*\*\*