Surname: . . . . . **First Name:** . . . . . **Student ID:** . . . . . .

VICTORIA UNIVERSITY OF WELLINGTON
TE HERENGA WAKA

EXAMINATIONS – 2025
TRIMESTER 2 (MID-TRIMESTER TEST)
FRONT PAGE

COMP 103
SOLUTIONS
SEP 8, 2025

Time allowed: **60 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 **5** questions
The test will be marked out of a total of **50** marks.
You will be provided with a *concise Java documentation for Collections* and a *brief Java documentation*.
Record your answers in their designated spaces.
If you find any question unclear, request clarification from the invigilator.
Assume that all necessary libraries for programming are already imported.

For markers use only:

Question	#1	#2	#3	#4	#5	**Total**
Max/Quest.	8	8	12	10	12	**50**
Mark						

1. Properties of Collections [8 marks]

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

a. Which Java collection interface is best suited to store distinct words (i.e., vocabulary) found in a text? [2 marks]

i. Set ✓
ii. Map
iii. Queue
iv. List

b. Which collection class automatically sorts elements according to their natural order or a specified comparator? [2 marks]

i. HashSet
ii. HashMap
iii. TreeSet ✓
iv. Stack

c. Which Java collection class supports both LIFO (Last-In-First-Out) and FIFO (First-In-First-Out) behaviour, depending on method usage? [2 marks]

i. ArrayDeque ✓
ii. Stack
iii. PriorityQueue
iv. HashSet

d. Consider the piece of code given below: [2 marks]

```
Deque<Integer> dq = new ArrayDeque<>();
dq.addFirst(10);
dq.addLast(20);
```

Which of the following statements might result in an error after executing the above lines?

i. `Integer first = dq.removeFirst();`
ii. `Integer last = dq.getLast();`
iii. `dq.clear();`
iv. `None of the above.` ✓

2. Using a Stack [8 marks]

Complete the following method called `reverse` that takes a list of Strings type and returns a list with reversed order of elements. The method should:

- Use a stack to temporarily hold the elements.
- Pop elements from the stack and add them back to the list in reverse order.
- You must not use built-in reversal methods such as `Collections.reverse` or insert-at-front techniques like `List.add(0, ...)`.
- Avoid any `NullPointerException`.

```
public List<String> reverse(List<String> list) {
   // TODO: your code here
   // NOTE: you should avoid exceptions!



   if (list == null) {
      return new ArrayList<>(); //or just return null
   }

   Stack<String> stack = new Stack<>();
   for (String str : list) {
      stack.push(str);
   }

   List<String> reversed = new ArrayList<>();
   while (!stack.isEmpty()) {
      reversed.add(stack.pop());
   }

   return reversed;









}
```

3. More on Using Collections [12 marks]

For a given number of books, we want to analyse and compare their vocabulary content. Assume that the words from each book have been extracted into an `ArrayList<String>` — one list per book. Complete the following methods.

a. *Vocabulary:* [6 marks]

Complete the `distinctWords` method below so that it returns a list of all the **distinct words** from the input.

You **must** use a set to eliminate duplicates before returning the result.

```
public List<String> distinctWords(List<String> words) {
   // TODO: your code here


   if (words == null) {
      return new ArrayList<>(); // Just returning null is also acceptable but not
      ↪  preferred
   }


   TreeSet<String> set = new TreeSet<>(); // Or, HashSet instead of TreeSet which
   ↪  is faster

   for (String word : words) {
      set.add(word);
   }


   return new ArrayList<>(set);









}
```

Question 3 continues on next page

Question 3 continued from previous page

---

b. *Sort Books by Vocabulary Size:* **[6 marks]**

Complete the method `sortByVocabSize`, which takes a map from book titles to lists of all words (including duplicates) found in each book. Your task is to return a list of book titles sorted in ascending order of their vocabulary sizes—that is, by the number of distinct words in each book.

Hint: You may find the method `distinctWords` from part **a** useful, even if you haven't completed part **a** yet.

Example: The following table represents the key-value pairs of an input `Map<String, List<String>>` representing books and the words they contain:

Book Title	List of words
`"Book A"`	`"server", "cloud", "router"`
`"Book B"`	`"cloud", "server", "server", "server"`
`"Book C"`	`"cloud", "server", "router", "firewall", "cloud"`

The expected output is a list of book titles sorted by the number of distinct words (vocabulary size), in ascending order: `["Book B", "Book A", "Book C"]`

```
public List<String> sortByVocabSize(Map<String, List<String>> bookWords) {
   if (bookWords == null) {
      return new ArrayList<>();
   }
   // TODO: your code here









   List<String> titles = new ArrayList<>(bookWords.keySet());

   titles.sort((t1, t2) -> { // Can use Collections.sort()

      List<String> wordsList1 = bookWords.get(t1);
      List<String> wordsList2 = bookWords.get(t2);
      int vocabSize1 = distinctWords(wordsList1).size();
      int vocabSize2 = distinctWords(wordsList2).size();

      return vocabSize1 - vocabSize2; // ascending order

   });

   return titles;




}
```

Or, better

```
public List<String> sortByVocabSize(Map<String, List<String>> bookWords) {
   if (bookWords == null) {
      return new ArrayList<>();
   }
   // TODO: your code here
   List<String> titles = new ArrayList<>(bookWords.keySet());

   // Pre-calculate vocabSize to avoid repeated computation
   Map<String, Integer> vocabSizeMap = new HashMap<>();
   for (String title : titles) {
      vocabSizeMap.put(title, distinctWords(bookWords.get(title)).size());
   }

   titles.sort((t1, t2) -> vocabSizeMap.get(t2) - vocabSizeMap.get(t1)); // Can
↪  use Collections.sort()

   return titles;

}
```

SPARE SPACE FOR EXTRA ANSWERS

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

4. Complexity analysis **[10 marks]**

Analyse the time complexity of the `analyseNumbers` method shown below. For each line, indicate its cost and the number of times it executes. Then, determine the overall time complexity of the entire method. Assume the input list has size n. Some lines have already been completed; fill in the remaining lines.

```
public int[] analyseNumbers(List<Integer> numbers) {

   List<Integer> sorted = new ArrayList<>(numbers);       // cost=O(n), times=1

   Collections.sort(sorted);                              // cost=O(nlog(n)),times=1

   Set<Integer> uniqueNumbers = new HashSet<>(numbers);   // cost=O(n), times=1

   int distinctPairs = 0;                                 // cost=O(1), times=1

   int n = numbers.size();                                // cost=O(1), times=1

   for (int i = 0; i < n; i++) {

      for (int j = i + 1; j < n; j++) {

         if (!numbers.get(i).equals(numbers.get(j))) {    // cost=O(1), times=n²

               distinctPairs++;                           // cost=O(1), times=n²
         }
      }
   }

   return new int[]{distinctPairs, uniqueNumbers.size()}; // cost=O(1), times=1
}
                                                          // TOTAL COST = O(n²)
// Testing
System.out.println(Arrays.toString(analyseNumbers(Arrays.asList(4,2,5,1,3,2))));
```

Bonus question: **[Extra 2 marks]**

What is the cost of executing `Set<Integer> sortedNumbers = new TreeSet<>(numbers);` assuming that `numbers` is a list of size n?

Hint: Consider the cost of inserting each element from the list into the TreeSet, and how that scales with the number of elements.

```
YOUR ANSWER: O(n log n)

Explanation: The TreeSet constructor iterates through all n elements
from the input list. For each element, it performs an insertion into
the TreeSet, which takes O(log k) time where k is the current size
of the tree. In the worst case, we insert n elements, so the total
cost is O(log 1 + log 2 + ... + log n) = O(log n!) = O(n log n).
```

5. Recursion [12 marks]

a. Complete the following recursive method to count the number of vowels in a string. For example, `countVowels("Recursion Is Cool!")` should return `7`. [6 marks]

- You must use a first-and-rest recursion: `countVowels` should check the first character and then call itself recursively on the rest of the string.
- The method should treat both uppercase and lowercase vowels (`a, e, i, o, u`) as valid.
- You may need the following methods (from the `String` class) along with those in the documentation provided:

Method	Description
`text.length()`	Returns the number of characters in the string
`text.charAt(i)`	Returns the character at position `i` (starting from 0)
`text.substring(i)`	Returns the part of the string starting from position `i`

```
public int countVowels(String text) {
   // TODO: your code here


   if (text == null) return 0;

   text = text.toLowerCase(); // Normalize the entire string

   if (text.length() == 0) {
      return 0; // Base case: no characters left
   }

   char first = text.charAt(0);
   int count = (first == 'a' || first == 'e' || first == 'i' || first == 'o' ||
   ↪  first == 'u') ? 1 : 0;

   return count + countVowels(text.substring(1));









}
```

Question 5 continues on next page

Question 5 continued from previous page

---

b. Write a recursive method `drawNestedSquares` that draws a series of nested squares on the graphics pane. The method should take the following parameters: **[6 marks]**

- `double x, double y`: the coordinates of the top-left corner of the square,
- `double size`: the length of the sides of the square,
- `int depth`: the number of nested squares to draw.

The method should:

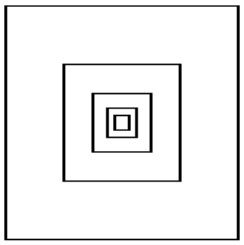
- Draw a square at `(x, y)` with side length `size`.
- If `depth > 1`, recursively call itself to draw a smaller square inside the current one, centred within the current square, with side length half of `size` and `depth - 1`.

You may use the following UI method to draw a rectangle, where `x` and `y` specify the top-left corner, and `width` and `height` specify its size:

```
UI.drawRect(double x, double y, double width, double height);
```

Example: `drawNestedSquares(100,100,120,5)` should draw 5 nested squares, each centred inside the previous one.

The figure on the right illustrates the expected output.



```
public void drawNestedSquares(double x, double y, double size, int depth) {
   // TODO: your code here


   if (depth == 0) {
      return;
   }


   UI.drawRect(x, y, size, size);


   // Calculate new top-left for the next (smaller) square
   double newX = x + size / 4;
   double newY = y + size / 4;
   double newSize = size / 2;

   drawNestedSquares(newX, newY, newSize, depth - 1);




}
drawNestedSquares(100, 100, 120, 5); // Testing
```

SPARE PAGE FOR EXTRA ANSWERS

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

---

* * * * * * * * * *