CS211FZ (Algorithms & Data Structures 2)

CA2 - this lab is worth 5% of the module marks and is marked

out of 100 Due: Friday 18th April. 1pm

This is an open-book, graded assignment. Please cite all references as comments in your submissions. You cannot directly reuse a solution from online sources. You must not engage with another student, in person or electronically (via phone, social media, etc.), to secure assistance with this Assignment. If you do so (even for only one of the questions), you will receive an automatic failure (0%), and it will also be reported to the Executive Vice-Dean of MIEC and/or Maynooth University Plagiarism board. We will perform similarity checks on submitted assignments to check for collaborative efforts. The lecturer reserves the right to interview you about your submission in special cases.
IMPORTANT: You must demonstrate your software to a TA and answer any questions before the end of the lab session to get the Programming Marks. So it is suggested you do the final 2 programming questions first. Software templates are available on Moodle but you don' t have to use them. You must also submit your programs in the pdf submission, submitted as one Class.

Question 2-1

Why does quicksort have an average-case time complexity of O(n log n), but a worst-case of $O(n^2)$?
[5 Marks]
The average time complexity of fast sort is O(n log n) because:
Each recursion divides the array into two approximately equal parts (i.e., divide and conquer).
Dividing once requires traversing the entire array (O(n)), and the depth of the recursion is log n, so the total complexity is O(n log n).

But in the worst case, the complexity is $O(n^2)$, and the reason is:

If the pivot (baseline value) is chosen to be the smallest or the largest value each time,

this results in an extremely uneven split (one subarray is empty and the other is of size n-1).
The recursion depth in this case becomes n and the overall complexity becomes O(n^2).

Under what conditions does the worst case occur, and how can it be mitigated? [10 Marks]

Worst case conditions:
The input is already ordered or nearly ordered (ascending or descending);
Each selected pivot is either the maximum or minimum value;
Too many duplicate elements can also lead to unbalanced splits.
How to mitigate:
Randomized Quicksort:
Before each split, randomly select an element as pivot to avoid always selecting the worst position.
This reduces the probability of the worst case scenario to a very low level.
Median-of-three:
Selects the median of the first, middle, and last elements as the pivot.
Increases the chance of picking a “good pivot”.
Switch to Hybrid Sort:
When the size of the subarray is less than a certain threshold (e.g. 10), insertion sort is used instead of recursion.
This improves performance in real-world applications (e.g. Java's TimSort).

Question 2-2
How would you adapt mergesort for a multi-core processor? What challenges arise, and how do you address them? [10 Marks]

Adaptation Ideas:
Parallel sort is naturally suited for parallelization because it is a partitioning algorithm that allows data to be continuously split up while processing each part in parallel on multiple processors.

Parallel Split Phase:
Splits the data into multiple parts and recursively sorts them in multiple threads simultaneously.
Parallel merge phase:
Merges sorted subarrays into larger ordered arrays, which can also be executed in parallel.

Question 2-3
Bubble sort compares adjacent elements, swapping them if out of order, repeating until

no swaps occur (O(n^2)). What are 2 optimisations for bubble sort, explain them, and provide pseudocode. [15 Marks]

Optimization 1: Early Termination

If no swaps occur during a full pass through the array, the array is already sorted. We can terminate early to avoid unnecessary comparisons.

Optimization 2: Last Swapped Index

Record the position of the last swap during each pass. Elements beyond that position are already in the correct order, so the next pass only needs to go up to that index.

```
function OptimizedBubbleSort(arr):
    n = length(arr)
    repeat
        new_n = 0
        for i from 1 to n - 1:
            if arr[i - 1] > arr[i]:
                swap arr[i - 1] and arr[i]
                new_n = i
        n = new_n
    until n == 0
```

Question 1-4

Design a sorting algorithm using pseudocode, based on Mergesort, for objects with priority (descending) and timestamp (ascending) tiebreaker? Also, tell us why it is based on Mergesort [20 Marks]

```
function mergeSort(arr):
    if length(arr) <= 1:
        return arr

    mid = length(arr) / 2
    left = mergeSort(arr[0:mid])
    right = mergeSort(arr[mid:end])
    return merge(left, right)

function merge(left, right):
    result = []
    while left is not empty and right is not empty:
        if left[0].priority > right[0].priority:
            result.append(left.pop(0))
        else if left[0].priority < right[0].priority:
            result.append(right.pop(0))
        else:
            # Priority is equal, compare timestamp (ascending)
            if left[0].timestamp <= right[0].timestamp:
                result.append(left.pop(0))
            else:
                result.append(right.pop(0))

    # Append any remaining elements
    result += left
```

```
        result += right
    return result
```

Why use Mergesort?

- **Stability:** Mergesort is a stable sorting algorithm, meaning it preserves the relative order of elements with equal priority. This is especially useful when using timestamp as a tiebreaker.
- **Time Complexity:** Mergesort has a time complexity of **O(n log n)** in all cases, including the worst case, making its performance consistently reliable.
- **Suitable for large datasets:** Since Mergesort does not rely on random access, it works well with linked lists or large files.
- **Easy to implement recursively:** Mergesort follows a clear divide-and-conquer strategy, making it conceptually simple and elegant to implement.

Programming Questions

You may find it helpful to use the Java template supplied on Moodle

Question 1-5

Revisit question 1-4 and write a java program the executes your pseudocode on the file numbers.csv which is supplied on Moodle. **[40 marks]**

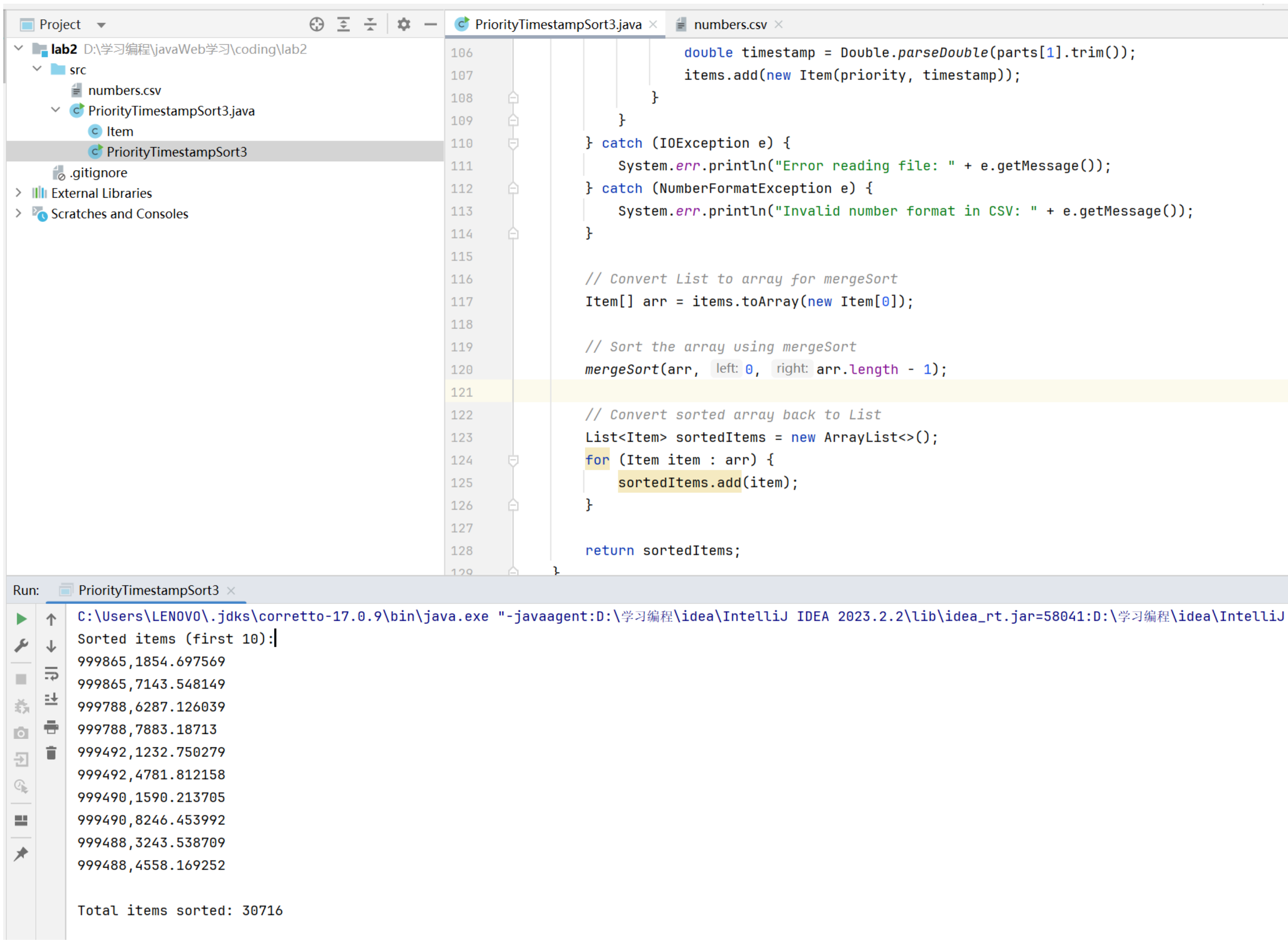
Important submission details

Please indicate the Operating System (Linux/Windows/MacOS/Online), IDE (e.g. Eclipse, Visual Studio Code), and Java SDK version used for testing in your submission. If you use an online IDE, please specify the IDE (http://repl.it) and provide a link where possible.

All work must be submitted via Moodle (see "CA" section for submission). Work submitted via other means will not be accepted unless you have prior arrangements with the lecturer. All work MUST be submitted by the due date deadline. Late submissions will not be accepted.

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

// Class to represent an item with priority and timestamp
class Item {
    int priority;
    double timestamp;

    // Constructor
    Item(int priority, double timestamp) {
        this.priority = priority;
        this.timestamp = timestamp;
    }

    // For printing the item
    @Override
    public String toString() {
        return priority + "," + timestamp;
    }
}

public class PriorityTimestampSort3 {
    // Compare function adapted to return -1, 0, or 1 for sorting
    private static int compare(Item a, Item b) {
        if (a.priority > b.priority) return -1; // Higher priority comes first
        else if (a.priority < b.priority) return 1;
        else {
            // If priority is equal, compare timestamp ascending
            return Double.compare(a.timestamp, b.timestamp);
        }
    }

    // Merge function to combine two sorted subarrays
    private static void merge(Item[] arr, int left, int mid, int right) {
        // Calculate sizes of two subarrays to merge
        int n1 = mid - left + 1;
        int n2 = right - mid;

        // Create temporary arrays for left and right subarrays
        Item[] leftArr = new Item[n1];
        Item[] rightArr = new Item[n2];

        // Copy data to temporary arrays
        for (int i = 0; i < n1; i++) {
            leftArr[i] = arr[left + i];
        }
        for (int j = 0; j < n2; j++) {
            rightArr[j] = arr[mid + 1 + j];
        }

        // Merge the temporary arrays back into arr[left..right]
        int i = 0, j = 0, k = left;

        while (i < n1 && j < n2) {
            if (compare(leftArr[i], rightArr[j]) <= 0) {
                arr[k] = leftArr[i];
                i++;
            } else {
                arr[k] = rightArr[j];
                j++;
            }
            k++;
        }

        // Copy remaining elements of leftArr, if any
        while (i < n1) {
            arr[k++] = leftArr[i++];
        }

        // Copy remaining elements of rightArr, if any
        while (j < n2) {
            arr[k++] = rightArr[j++];
        }
    }

    // MergeSort function to sort the array
    private static void mergeSort(Item[] arr, int left, int right) {
        if (left < right) {
            // Find the middle point
            int mid = left + (right - left) / 2;

            // Recursively sort the left and right halves
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);

            // Merge the sorted halves
            merge(arr, left, mid, right);
        }
    }

    // Method to read items from numbers.csv and sort them
    public static List<Item> readAndSortCSV(String fileName) {
        List<Item> items = new ArrayList<>();

        // Read the file
```

```java
        try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {
            String line;
            while ((line = br.readLine()) != null) {
                // Split each line into priority and timestamp
                String[] parts = line.split(",");
                if (parts.length == 2) {
                    int priority = Integer.parseInt(parts[0].trim());
                    double timestamp = Double.parseDouble(parts[1].trim());
                    items.add(new Item(priority, timestamp));
                }
            }
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.err.println("Invalid number format in CSV: " + e.getMessage());
        }

        // Convert List to array for mergeSort
        Item[] arr = items.toArray(new Item[0]);

        // Sort the array using mergeSort
        mergeSort(arr, 0, arr.length - 1);

        // Convert sorted array back to List
        List<Item> sortedItems = new ArrayList<>();
        for (Item item : arr) {
            sortedItems.add(item);
        }

        return sortedItems;
    }

    // Main method to run the program
    public static void main(String[] args) {
        String fileName = "D:\\学习编程\\javaWeb学习\\coding\\lab2\\src\\numbers.csv";

        // Read and sort items
        List<Item> sortedItems = readAndSortCSV(fileName);

        // Print results
        if (sortedItems.isEmpty()) {
            System.out.println("No items to sort.");
            return;
        }

        System.out.println("Sorted items (first 10):");
        for (int i = 0; i < Math.min(10, sortedItems.size()); i++) {
            System.out.println(sortedItems.get(i));
        }

        System.out.println("\nTotal items sorted: " + sortedItems.size());
    }
}
```



```java
                    double timestamp = Double.parseDouble(parts[1].trim());
                    items.add(new Item(priority, timestamp));
                }
            }
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
        } catch (NumberFormatException e) {
            System.err.println("Invalid number format in CSV: " + e.getMessage());
        }

        // Convert List to array for mergeSort
        Item[] arr = items.toArray(new Item[0]);

        // Sort the array using mergeSort
        mergeSort(arr, left: 0, right: arr.length - 1);

        // Convert sorted array back to List
        List<Item> sortedItems = new ArrayList<>();
        for (Item item : arr) {
            sortedItems.add(item);
        }

        return sortedItems;
    }
```

```
C:\Users\LENOVO\.jdks\corretto-17.0.9\bin\java.exe "-javaagent:D:\学习编程\idea\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=58041:D:\学习编程\idea\IntelliJ
Sorted items (first 10):
999865,1854.697569
999865,7143.548149
999788,6287.126039
999788,7883.18713
999492,1232.750279
999492,4781.812158
999490,1590.213705
999490,8246.453992
999488,3243.538709
999488,4558.169252

Total items sorted: 30716
```

Your submission should be one single PDF file, including also your Java codes. Submitting in any other format cannot be accepted.