# Cardiff School of Computer Science and Informatics

## Coursework Assessment Pro-forma

**Module Code**: CMT219

**Module Title**: Algorithms, Data Structures & Programming

**Lecturer**: Matthew Morgan, Neetesh Saxena, Crispin Cooper

**Assessment Title**: Coursework

**Assessment Number**: 2

**Date Set**: 18th March 2024

**Submission Date and Time**: by 9th May 2024 at 9:30am

**Feedback return date**: 6th June 2024

**If you have been granted an extension for Extenuating Circumstances, then the submission deadline and return date will be later than that stated above. You will be advised of your revised submission deadline when/if your extension is approved.**

**If you defer an Autumn or Spring semester assessment, you may fail a module and have to resit the failed or deferred components.**

**If you have been granted a deferral for Extenuating Circumstances, then you will be assessed in the next scheduled assessment period in which assessment for this module is carried out.**

**If you have deferred an Autumn or Spring assessment and are eligible to undertake summer resits, you will complete the deferred assessment in the summer resit period.**

**If you are required to repeat the year or have deferred an assessment in the resit period, you will complete the assessment in the next academic year.**

**As a general rule, students can only resit 60 failed credits in the summer assessment period (see section 3.4 of the** [academic regulations](https://intranet.cardiff.ac.uk/students/study/your-rights-and-responsibilities/academic-regulations)**). Those with more than 60 failed credits (and no more than 100 credits for undergraduate programmes and 105 credits for postgraduate programmes) will be required to repeat the year. There are some exceptions to this rule and they are applied on a case-by-case basis at the exam board.**

**If you are an MSc student, please note that deferring assessments may impact the start date of your dissertation. This is because you must pass all taught modules before you can begin your dissertation. If you are an overseas student, any delay may have consequences for your visa, especially if it is your intention to apply for a post study work visa after the completion of your programme.**

**NOTE: The summer resit period is short and support from staff will be minimal. Therefore, if the number of assessments is high, this can be an intense period of work.**

This assignment is worth 50% of the total marks available for this module. If coursework is submitted late (and where there are no extenuating circumstances):

1 If the assessment is submitted no later than 24 hours after the deadline, the mark for the assessment will be capped at the minimum pass mark;

2 If the assessment is submitted more than 24 hours after the deadline, a mark of 0 will be given for the assessment.

Extensions to the coursework submission date can ***only*** be requested using the [Extenuating Circumstances procedure](https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/extenuating-circumstances). Only students with ***approved*** extenuating circumstances may use the extenuating circumstances submission deadline. Any coursework submitted after the initial submission deadline without \*approved\* extenuating circumstances will be treated as late.

More information on the extenuating circumstances procedure and academic regulations can be found on the Student Intranet:

<https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/extenuating-circumstances>

<https://intranet.cardiff.ac.uk/students/study/your-rights-and-responsibilities/academic-regulations>

By submitting this assignment you are accepting the terms of the following declaration:

**I hereby declare that my submission (or my contribution to it in the case of group submissions) is all my own work, that it has not previously been submitted for assessment and that I have not knowingly allowed it to be copied by another student. I declare that I have not made use of AI chatbots or tools to complete this work. I understand that deceiving or attempting to deceive examiners by passing off the work of another writer, as one’s own is plagiarism. I also understand that plagiarising another’s work or knowingly allowing another student to plagiarise from my work is against the University regulations and that doing so will result in loss of marks and possible disciplinary proceedings***[[1]](#footnote-2).*

## Assignment

**PART 1: ALGORITHMS**

In this *individual* work, you will implement sorting algorithms and then test their performance. This part of the assignment consists of not just coding, but also testing your code and providing evidence. Note that when your code is complete, you still have a reasonable amount of work to do -- so start early!

## *Storing and Sorting Items*

Consider a manufacturing company, XYZ that runs 24 hours and 7 days operational. The company produces a very large number of items per week. In a week from Monday to Sunday, it produces 1000, 5000, 10000, 50000, 75000, 100000 and 500000 items, respectively. Each item has a random integer identifier.

You are expected to create an Integer ArrayList and store the item IDs for each day of the week – so you will have 7 different ArrayList storing items accordingly. Now, you are required to randomly generate the appropriate number of item IDs for each day as stated above. (These IDs do not need to be unique – it does not matter if the ArrayLists contain a few duplicate IDs).

Now think about the logic of applying the Quicksort algorithm and first write its pseudocode. Once you’re done, convert your pseudocode into a Java code and run your code successfully.

## *Efficiency Testing*

Once you have implemented your algorithm, you need to test it to see how long your sorting algorithm takes to run. You should test the speed of the algorithm on all three: random, sorted and reverse-sorted lists. You are expected to use System.currentTimeMillis() to estimate the execution time. Using currentTimeMillis only gives an accurate time estimate if the function takes a long time to run (at least a couple of seconds). Run your code a number of times and compute the average time, print the output screenshot and discuss in the report -- reflect how many iterations are sufficient to provide an accurate time of the algorithm.

## *Developing a Better Sorting Algorithm*

Quicksort is faster when the number of items to be sorted in a list is large. But when lists are small enough, quicksort runs slower than some of the Θ(n2) algorithms.

If you carefully notice, each time the quicksort method is recursively called, the sublist to be sorted could be either small or large. The time to sort one small sublist with a faster algorithm can be negligible. However, sorting such hundreds of small sublists can make a difference in the overall efficiency of the sort.

Here, you will combine Quicksort with another sorting algorithm to create the fastest possible sorting algorithm. We call this “hybrid Quicksort”. You have several options --

* Use Quicksort until the sublist gets “small enough”, and then use selection sort to sort the small lists.
* Use Quicksort until the sublist gets “small enough”, and then use insertion sort to sort the small lists.
* Use Quicksort to "mostly" sort the list, i.e., use the Quicksort to sort any sublists larger than a cut-off size, and then stop. The list will now be mostly sorted, and you can use selection sort on the entire list to quickly complete the sorting.
* Use Quicksort to "mostly" sort the list, i.e., use the Quicksort to sort any sublists larger than a cut-off size, and then stop. The list will now be mostly sorted, and you can use insertion sort on the entire list to quickly complete the sorting.
* Use some other method of your own devising.

What does “small enough” mean?  You can try a percentage of the list (say, 5% or 10%), or an absolute number (8, 10, 15, 100 elements, etc.), or something else of your choice.

What does “mostly” sort mean? A cut-off size of the items to be sorted in the list.

You should test your choices to ensure that you have the most efficient algorithm possible. You should also be sure that your hybrid Quicksort has reasonable performance on all lists: sorted and inverse sorted lists as well as random lists. Try various methods for choosing the pivot element, to try to get the best possible behaviour.

You need to complete the following tasks and submit electronically:

**Q1-A.** Source code for sorting all items in the ArrayLists for the week using Quicksort. [10 marks]

**Q1-B.** Source code for efficiency testing. [10 marks]

**Q1-C.** Source code for your hybrid sorting algorithm. [10 marks]

**Q1-D.** Write a 500-word (maximum) explanation of how you developed your hybrid sorting algorithm. What combinations of the algorithms/approaches you tried, what different parameters you chose, and how much of a difference these changes made to the efficiency of your algorithm, including the run time complexity. [20 marks]

Create a **single** report .pdf file containing the following:

**(i)** Screenshot of the output from Q1-A for all list sizes. Also, provide a pseudocode of your Quicksort algorithm.

**(ii)** Screenshot of the output from Q1-B. Make sure to include sorted and inverse sorted lists as well as random lists for each list size in these tests.

**(iii)** Your explanation for Q1-D above.

**(iv)** Screenshot of the final output from Q1-C. Make sure your hybrid quicksort has reasonable performance on all lists, sorted and inverse sorted lists as well as random lists for each list size.

**Important Note:** All source code files must be submitted as a single .zip file.

It is okay for you to discuss solutions to Part-1 with your classmates. However, no collaboration should ever involve looking at one of your classmate's source codes. It is usually fairly easy to determine that someone has copied a code, even when the individual copying the code has changed identifier names and comments.

**PART 2: DESIGN PATTERNS & MULTITHREADING**

*Parts a-c of this question can be started after you are taught the Observer pattern, in the Design Patterns sessions. Based on content you are taught in Design Patterns, your solution to part (a) is likely to change the colour of some, but not all balls. The Multithreading session will cover additional material needed to achieve a fully working code for part a, and to answer part d.*

Download and unzip BouncingBalls.zip, which contains three java source files for a small app:

BouncingBalls.java

ColorPublisher.java

ColorSubscriber.java

Build and run the application, which launches a window containing a large number of bouncing balls. Two buttons are also present which are intended to change the colour and size of the balls, however, **the buttons intentionally do not work in the app you have just downloaded:** it will be your task to implement this functionality**.**

Note also that BouncingBalls.java contains Graphical User Interface (GUI) code which has not been covered in this course. Its purpose is explained in the comments, however you are not expected to understand the GUI code in any detail. The comments alone give sufficient detail on how the GUI works for you to complete your part of the app.

BouncingBalls.java makes use of an Observer pattern to communicate colour changes to each of the balls, when the buttons are clicked. This pattern is currently incomplete. **Add code to ColorPublisher.java to complete the pattern**, (i) allowing each ball to subscribe to the ColorPublisher when it calls the addSubscriber() method, (ii) allowing buttons to publish colour changes to all subscribed balls, by calling ColorPublisher’s publish() method.

* **DO NOT** change the code in BouncingBalls.java or ColorSubscriber.java. You should, however, read the code of BouncingBalls.java to understand how the ColorPublisher class is used there, and also ColorSubscriber.java which is used in both of the other files.
* **DO** test your changes by running the app.

Now write a report answering the following questions:

1. Paste the contents of your completed ColorPublisher.java file into the report [12 marks for correct code].
2. Which classes make use of the ColorSubscriber interface? What do they use it for? [12 marks, max 80 words].
3. The Observer pattern is actually a bad choice for communicating colour changes between threads in this particular application. Explain why this is so, and how the same functionality could be implemented more simply [14 marks, max 300 words].
4. The app uses a separate thread to process the movement of each ball. Is this a good choice of approach? Why? [12 marks, max 150 words].

## Learning Outcomes Assessed

This assignment particularly addresses the following module learning outcomes:

* Write code in a given programming paradigm (for example Object-Oriented) using a high-level programming language
* Reflect upon the relationship between system and code design and implementation in the given programming paradigm
* Critically evaluate design patterns and their applicability to different scenarios
* Select and use appropriate algorithms and data structures to provide best performance in a given scenario

## Criteria for assessment

Credit will be awarded against the following criteria.

**PART 1:**

***Distinction (70-100%)*** –[Q1-A, B, C] Fully working codes that demonstrate excellent understanding of the assignment problem and scenarios using a relevant Java approach. Excellent formatting of input/output, catch exception handling, the application deals well with invalid user input and doesn’t crash for any data. Excellent and consistent code layout. Appropriate use of comments. [Q1-D] All required outputs. Clear and appropriate write-up with all justified choices.

***Merit (60-69%)*** – [Q1-A, B, C] All required functionalities of the codes are met (with no runtime error) demonstrating good understanding of the assignment problem and scenarios using a relevant Java approach. Good formatting of input/output, catch exception handling, the application may have a few errors with invalid user input. Good and consistent code layout. Good use of comments. [Q1-D] Most of the required outputs are presented. Clear and suitable write-up with mostly justified choices.

***Pass (50-59%)*** –[Q1-A, B, C] Some required functionalities of the codes are met (with only minor errors) demonstrating some understanding of the assignment problem and scenarios using a relevant Java approach. Some formatting of input/output, catch exception handling, the application may have some errors with invalid user input. Some and partially consistent code layout. Some use of comments. [Q1-D] Only some required outputs. Some write-up with partially justified choices.

***Fail (0-49%)*** - [Q1-A, B, C] Faulty codes with wrong implementation, poorly demonstrating the understanding of the assignment problem and scenarios using a relevant Java approach. Bad formatting of input/output, catch exception handling, the application has major errors with invalid user input, and crashes with most data. Poor and inconsistent code layout. No/poor use of comments. [Q1-D] No or only a few required outputs. Unclear and poor write-up with no/bad justified choices.

**PART 2:**

2a: 100% for fully correct, thread safe code. 50% for mostly correct code with some bugs (e.g. some, but not all balls change colour correctly). 0% for non working code.

2b-2d: 80% of the marks are allocated for correct answers covering all key points, with lower marks awarded if e.g. some points are missed. 20% of marks remain for quality of writing/communication.

Indicative levels of attainment for the MSc programme are as follows:

Distinction (70-100%)

Merit (60-69%)

Pass (50-59%)

Fail (0-50)

## Feedback and suggestion for future learning

Feedback on your coursework will address the above criteria. Feedback and marks will be returned on 29th May 2024 via Learning Central.

Feedback from this assignment will be useful for other programming and data structure and algorithms related modules.

## Submission Instructions

|  |  |  |  |
| --- | --- | --- | --- |
| Description | | Type | Name |
| Q1 | **Compulsory** | One PDF (.pdf) file for Q1-D | Q1\_[student number].pdf |
|  | **Compulsory** | One zip file containing the source code (.java files) for all parts of question 1 | Q1\_[student number].zip |
| Q2 | **Compulsory** | One PDF (.pdf) file containing your report | Q2\_[student number].pdf |
|  | **Compulsory** | One zip (.zip) file containing the source code for your app (BouncingBalls.java, ColorPublisher.java, ColorSubscriber.java) | Q2\_[student number].zip |

Any code submitted will be run on a system equivalent to the university provided Windows/Linux laptops and must be submitted as stipulated in the instructions above.

Any deviation from the submission instructions above (including the number and types of files submitted) may result in a mark of zero for the assessment or question part.

Staff reserve the right to invite students to a meeting to discuss coursework submissions

## Support for assessment

Questions about the assessment can be asked during weekly lab classes, and on the departmental StackOverflow (StackOverflow questions must be tagged cmt-219).

1. https://intranet.cardiff.ac.uk/students/study/exams-and-assessment/academic-integrity/cheating-and-academic-misconduct [↑](#footnote-ref-2)