Assessment Proforma 2024-25

Key Information

Module Code	CM1210
Module Title	Object Oriented Java Programming
Module Leader	Dr Matthew Morgan
Module Moderator	Ms Ramalakshmi Vaidhiyanathan
Assessment Title	Data Structures and Algorithms in Java
Assessment Number	2
Assessment Weighting	50%
Assessment Limits	2 reports submitted as separate file with a word limit of 500 maximum on each report

The Assessment Calendar can be found under ‘Assessment & Feedback’ in the COMSC-ORG-SCHOOL organisation on Learning Central. This is the single point of truth for (a) the hand out date and time, (b) the hand in date and time, and (c) the feedback return date for all assessments.

Learning Outcomes

The learning outcomes for this assessment are as follows:

LO1 - Develop a Object-Oriented program that has input and output functionality and that is event driven

LO3 - Implement basic data structures and algorithms

LO4 - Analyse and describe the performance of data-structures and algorithms

Submission Instructions

The coversheet can be found under ‘Assessment & Feedback’ in the COMSC-ORG-SCHOOL organisation on Learning Central.

All files should be submitted via Learning Central. The submission page can be found under ‘Assessment & Feedback’ in the 24/25 CM1210 module on Learning Central. Your submission should consist of multiple files:

Description		Type	Name
Coversheet	**Compulsory**	One PDF (.pdf) file	Coversheet.pdf
Q1	**Compulsory**	One PDF (.pdf) file for the report covering (i)–(iv) including Q1-D	Q1_[student number].pdf
Q2	**Compulsory**	One PDF (.pdf) file containing your report	Q2_[student number].pdf
	Compulsory	One zip (.zip) file containing your completed Source code files for Q1-A, B, C and Q2 - MyArrayQueue.java file.	[student number].zip

If you are unable to submit your work due to technical difficulties, please submit your work via e-mail to comsc-submissions@cardiff.ac.uk and notify the module leader.

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

Assessment Description

Assignment [Total marks: 80]

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.

You are expected to create an Integer ArrayList and store the items (randomly generated) for the week – so you will have 7 different ArrayList storing items accordingly. Now, you are required to randomly generate items from 1000 to 500000 for each day as stated above.

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 items to be sorted in a list are large. But when the lists are small enough, quicksort runs slower than some of the $\Theta(n^2)$ algorithms.

If you carefully notice, there could be several small sublists to be sorted when you apply Quicksort on a large list. 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 list gets “small enough”, and then use selection sort to sort the small lists.
- Use Quicksort until the list gets “small enough”, and then use insertion sort to sort the small lists.
- Use Quicksort to "mostly" sort the list, i.e., use Quicksort to sort the list until a cut-off size is reached, 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 Quicksort to sort the list until a cut-off size is reached, 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. Also, there should only be a single pdf file uploaded on the Learning Central.

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: Queue and Stack Data Structures

Q2-A. You should create two methods for a data structure implementing a *Queue* as a circular array. Your data structure should have the class name MyArrayQueue. The two methods that should be implemented are:

a) Adding an element to the queue:

public void enqueue(**Object** theElement) {…}

[**12 marks**]

(Functionality: 8, Design: 2, Presentation: 2)

b) Deleting an element from the queue and returning the deleted element:

public Object dequeue() {…}

[**8 marks**]

(Functionality: 5, Design: 2, Presentation: 1)

In both methods exceptions should be handled properly. For example, what happens when adding an element to a full circular queue? This will be explained in the lecture, please refer. There will be skeleton code for MyArrayQueue available on Learning central that contains the MyArrayQueue class with some fully implemented methods. It also includes signatures of two methods that you should implement: enqueue(Object theElement) & dequeue(). Please read the method docblocks (comments) carefully. You can reuse any implemented methods in the provided skeleton code. You ARE NOT allowed to change any parts of the implemented methods of the provided code. There will be penalty for making any changes. This class/skeleton should only be used for attempting Q2-A.

Q2-B. Report [**10 marks**]

You should also submit a pdf file with no more than 500 words that have a short-written justification for your design of the program reflecting on the algorithm and efficiency of the code. It should contain a screenshot showing an example of the output for part 2.

Assessment Criteria

Credit will be awarded against the following criteria.

PART 1:

High 1st (80%+)–[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.

1st(70-79%)–[Q1-A, B, C] Fully working codes that demonstrate very good understanding of the assignment problem and scenarios using a relevant Java approach. Very Good formatting of input/output, catch exception handling, the application deals well with invalid user input and doesn't crash for any data. Very good and consistent code layout. Appropriate use of comments. [Q1-D] All required outputs. Clear and appropriate write-up with all justified choices.

2.i (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.

2.ii (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.

3rd/Pass (40-49%) - [Q1-A, B, C] Faulty codes with wrong implementation, which can be run using few changes, very poorly demonstrating the understanding of the assignment problem and scenarios and not 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.

Marginal Fail (30-39%) - [Q1-A, B, C] Faulty codes with wrong implementation, poorly demonstrating the understanding of the assignment problem and scenarios without using a relevant Java approach. Very 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

Fail (<29%) - [Q1-A, B, C] Faulty codes with wrong implementation and/no submission.. [Q1-D] Poor write-up with no/bad justified choices and/or no submission

PART-2:

Criteria for assessment for part Q2-A (code) are as follows:

Marks	Functionality	Design	Presentation
High 1st (80%+)	Fully working code that demonstrates an excellent understanding of the assignment problem using a relevant java approach.	Excellent design with proper use of appropriate types, program control structures and classes from the core API and carefully considers their use (names chosen for classes, methods, and variables should convey the purpose and meaning of the named entity).	Excellent use of in line comments in the code and excellent readability (means things like placement of curly braces, code indentation, wrapping of long lines, layout of parameter lists, etc.)
1st (70-79%)–	Fully working code that demonstrates an excellent understanding of the assignment problem using a relevant java approach.	Excellent design with proper use of appropriate types, program control structures and classes from the core API and carefully considers their use (names chosen for classes, methods, and variables should convey the purpose and meaning of the named entity).	Excellent use of in line comments in the code and excellent readability (means things like placement of curly braces, code indentation, wrapping of long lines, layout of parameter lists, etc.)
2.i (60-69%)	All required functionality is met, and the code is generally working properly with some minor errors.	Good design with proper use of appropriate types, program control structures and classes from the core API and carefully considers their use (names chosen for classes, methods, and variables should convey the purpose and meaning of the named entity).	Good use of in line comments in the code and good code readability (means things like placement of curly braces, code indentation, wrapping of long lines, layout of parameter lists, etc.)
	Some of the functionality	Moderate design with the use of appropriate types, program	Some in line comments and moderate code

2.ii (50-59%)	developed with incorrect output and minor errors.	control structures and classes from the core API and carefully considers their use (names chosen for classes, methods, and variables should convey the purpose and meaning of the named entity).	readability (means things like placement of curly braces, code indentation, wrapping of long lines, layout of parameter lists, etc.)
3rd /Pass (40-49%)	Some of the functionality developed with incorrect output and major errors.	Low design with incomplete classes and data structures and their use (names chosen for classes, methods, and variables should convey the purpose and meaning of the named entity).	in line comments are missing at requires places and low code readability (means things like placement of curly braces, code indentation, wrapping of long lines, layout of parameter lists, etc.)
Marginal Fail (30-49%)	Faulty functions with wrong implementation and incorrect output.	Poor design with incomplete and/or incorrect classes and data structures and their use (names chosen for classes, methods, and variables should convey the purpose and meaning of the named entity).	Poor/no comments and poor code readability (means things like placement of curly braces, code indentation, wrapping of long lines, layout of parameter lists, etc.)
Fail (0-29%)	Faulty functions and/or no submission	Poor design and/or no submission	No comments and/or no submission

Criteria for assessment for part Q2-B (report) are as follows:

Mark	**Assessment Criteria**
High 1st (80%+)	An excellent and original discussion is provided. Your points are justified with excellent quality writing and screenshots.
1st (70-79%)	A very good and original discussion is provided. Your points are justified with very good quality writing and screenshots.
2.i (60-69%)	A good and original discussion is provided. Your points are justified with good quality writing and screenshots.
2.ii (50-59%)	A good discussion is provided, and points are justified with screenshots or inconsistent screenshots submitted.
3rd /Pass (40-49%)	Some discussion is provided, and points are not justified with screenshots or inconsistent screenshots submitted.
Marginal Fail (30-39%)	Limited or no discussion and no screenshot evidence submitted
Fail (<29%)	No discussion and/or no submission.

Help and Support

Questions about the assessment can be asked on https://stackoverflow.com/c/comsc/ and tagged with ‘CM1210’ along with lecturer’s name: Surya Thottam Valappil or Rhodri Morris.

Support for the programming elements of the assessment will be available in the lab classes and drop-in lab sessions along with sessions particularly run for coursework support

Feedback

Feedback on your coursework will address the above assessment criteria. Feedback and marks will be returned via Learning Central.