EST 1892 LSBU

Coursework Specification

CW2_Specification_CSI_4_DSA_20/21

Read this coursework specification carefully, it tells you how you are going to be assessed, how to submit your coursework on-time and how (and when) you'll receive your marks and feedback.

Module Code	CSI-4-DSA
Module Title	DATA STRUCTURES AND ALGORITHMS
Lecturer	MIKE CHILD
% of Module Mark	50%
Distributed	26 MAR 2021
Submission Method	Submit online via this Module's Moodle site
Submission Deadline	23:00 Friday 14 May 2021
Release of Feedback & Marks	Feedback and provisional marks will be available in the Gradebook on Moodle within 15 working days of submission.

Coursework Aim:

To demonstrate understanding of the module subject matter and the ability to apply it, including critical analysis and judgement. To achieve this, you are provided with an example application that must be analysed and evaluated. You are then required to discuss the applicability of a range of data structures and algorithms from the module material to creating different implementations of the same functionality. Finally you are required to propose and then implement an improved design.

Commented [GU1]: typo

Coursework Details:

Type:	Report on analysis, design and development task
Word Count:	No formal word count is required but for general guidance no more than 1500 words would be expected. The word count will always be ambiguous because the requirement for some embedded code will skew it. There are no specific penalties for exceeding or failing to reach the word count guidance.
Presentation:	▪ Any quoted material must be referenced, and a bibliography provided. Formal referencing is not an assessed criteria – however the identification of technical resources consulted in order to solve specific described problems *is* an assessed criteria (see main task specification) ▪ Work must be submitted as a Word document (.doc/docx) or a PDF. ▪ Course work must be submitted using Arial font size 11 (or larger if you need to), with a minimum of 1.5 line spacing ▪ Your student number must appear at the front of the coursework. Your name must **not** be on your coursework.
Referencing:	Harvard Referencing should be used, see your Library Subject Guide for guides and tips on referencing.
Regulations:	Make sure you understand the University Regulations on expected academic practice and academic misconduct. Note in particular: ▪ Your work must be your own. Markers will be attentive to both the plausibility of the sources provided as well as the consistency and approach to writing of the work. Simply, if you do the research and reading, and then write it up on your own, giving the reference to sources, you will approach the work in the appropriate way and will cause not give markers reason to question the authenticity of the work. ▪ All quotations must be credited and properly referenced. Paraphrasing is still regarded as plagiarism if you fail to acknowledge the source for the ideas being expressed. **TURNITIN:** When you upload your work to the Moodle site it will be checked by anti-plagiarism software.

Learning Outcomes

This coursework will fully or partially assess the following learning outcomes for this module.

On completion of the module you will be able to:

- Describe different algorithms and the means used to measure their performance.
- Analyse programming problems and identify appropriate algorithmic solutions.
- Develop software to solve relatively complex problems and assess alternative solutions.
- Apply critical and analytic reasoning to tasks.

Assessment Criteria and Weighting

LSBU marking criteria have been developed to help tutors give you clear and helpful feedback on your work. They will be applied to your work to help you understand what you have accomplished, how any mark given was arrived at, and how you can improve your work in future.

Full details of the criteria, interpretation, weightings and rubric to be applied are included in the main task specification below.

How to get help

We will discuss this Coursework Specification in class. However, if you have related questions, please contact Mike Child, childm@lsbu.ac.uk as soon as possible.

Resources

Course materials and online documentation that has been cited in the tutorial exercises are expected to be the main resources required for this assignment.

Commented [GU2]: grammar… rephrase

Quality assurance of coursework specifications

Coursework specifications within CSI division go through internal (for new modules with 100% coursework also through external) moderation. This is to ensure high quality, consistency and appropriateness of the coursework as well as to share best practice within the CSI division.

Details of the moderators for this coursework specification are below:

Moderated (internal)	Bugra, Muhammad, Emeka
Moderated (CSI lead)	George Ubakanma
Signed off by (HoD/DHoD)	George Ubakanma

----------For Internal use by CSI lead only----------

Changes required to CW?	~~Yes~~, No *
Examples of good practice	

*** if changes are required, moderator to complete the below:**

List of changes required	[These needs to be met before signoff can be achieved]
ML Response	[ML response, date]
Moderator Response	[ML response, date]

Coursework Assignment

This is the second of two coursework assignments for this module and is worth 50% of the module marks.

This assignment involves the analysis of a case study application as follows.

The application receives reports of vehicles breaking speed limits that are created and sent by speed cameras at various remote locations. The reports are instances of the class Report (see below) and must be stored in internal data structures of some kind. The application must offer two further functions:

- A method to supply a sorted collection of Report objects for a period of time specified by a start and end time.
- A method to supply a sorted collection of Report objects that relate to a specific camera.

You are provided an implementation of this application in Java and your tasks are to analyse this implementation, discuss other ways it could be done, propose a design and attempt to implement that design in code. These tasks are described in detail below.

The Provided Application

You are provided with the following Java class source files:

Report – this class defines the report objects that the application needs to receive and store internally.

TimeID – this class consists of a timestamp and a camera ID number. The report class has a field that contains a TimeID object that acts as a unique identifier for the report. TimeID objects implement the Comparable interface and thus can be sorted into a natural order based on ascending timestamps. Where two TimeID objects have the same timestamp, the camera ID number determines their order.

ProvidedImplementation – this class is an implementation of the application. It provides the required functionality by offering three methods:

- ***void receiveReport(Report report)*** – which receives reports coming from speed cameras and adds them to the internal data structures of the application.
- ***Collection<Report> reportsByPeriod(long start, long end)*** – which returns a time-sorted collection of all the Report objects whose timestamps lie between the provided start and end values.
- ***Collection<Report> reportsForCamera(int cameraID)*** – which returns a time-sorted collection of all the Report objects that came from the camera with the given cameraID number.

It also has a main method which demonstrates the application running. The demonstration creates 100 random reports and calls the receiveReport method with each of them, printing each report as it does so. It then calls the reportsByPeriod method selecting a period of roughly 1/3 of the period the reports were generated for, and prints the resulting collection. It then calls the reportForCamera method with the camera ID 12 and prints the resulting collection. The reports are all generated with camera IDs between 10 and 20 so this should usually result in some reports being returned.

ReportTimeIDComparator – this is a Comparator class that compares two report objects on the basis of their TimeID values. This allows Reports to be sorted according to their timestamp and camera ID and is used in the ProvidedImplementation application to do so.

Utils – this class contains some static functions that are used by the provided implementation but on the whole are not necessary for you to use. It provides the methods to create random Report objects. It does contain one method that might be useful for some possible implementations of th application:

- ***int timeIDKeyedBinarySearch(List<Report> reports, TimeID timeID)*** – this method performs a binary search of the given list of reports, that must have been sorted by TimeID previously, looking for a report that matches the given TimeID object. If a matching report exists, the index it is found at is returned. If no such report exists, the position at which it would have been found if it did is returned, encoded as(-position - 1).

  This method is similar to the one you encountered in "Exercises-8-Java-Collections-and-Binary-Searches" and is in fact implemented using the same code within the Utils class.

  Note also that you should not feel obliged to use this method in your improved implementation of the application – although improvements can be made by using it there are even better solutions that do not use it!

More detailed information about some of these classes follows:

The Report class

```
public class Report {
  public TimeID timeID;
  public int speedLimit;
  public String carRegistration;
  public int speedRecorded;
  /**
   * Returns a String representing the values in this object so that is can
   * be directly printed with System.out.println();
   */
  public String toString() {
    return timeID.toString() +
    "\nSpeed limit: " + speedLimit + " mph" +
    "\nCar registration: " + carRegistration +
    "\nSpeed recorded: " + speedRecorded + " mph";
  }
}
```

The Report class is very simple and consists only of four public fields and an implementation of the toString method. This is not good design from a Java object-oriented point of view, but it means it is very simple and makes it similar to a C struct.

The speedLimit field contains the speed limit in force at the location of the particular speed camera that generated the report (for example, 30mph, 40mph etc.). The carRegistration field contains the registration of the car that was photographed breaking the speed limit. The speedRecorded field contains the speed the speeding car was actually recorded as doing. The timeID field requires a little more explanation. This field is of type TimeID, a class that contains two fields of its own, consisting of the timestamp at which the speeding event took place and a unique camera ID number. Together, these two things uniquely identify the report object, distinguishing two reports that occur at the same moment but at different cameras. The TimeID class is described below.

The TimeID class

As explained above the TimeID class has two instance fields: a *long* timestamp representing the time, and an *int* camera ID number. It also has a constructor that accepts values for these two fields to make it easy to construct one in a single statement. It also implements the Comparable interface, meaning that two TimeID objects can be compared for order. The comparison takes place in the *compare* method and compares the two TimeIDs first by their timestamps, and second (only if their timestamps are equal) by their camera ID. Thus if camera 12 and camera 140 generate TimeID objects for the same moment, the one from camera 12 will be sorted before the one from camera 140. The methods upperBound and lowerBound create and return instances of TimeID that are suitable for defining time ranges for all cameras (these contain the highest and lowest camera ID respectively to ensure this is the case). Although the TimeID class appears complicated, the most important thing is to understand that it allows Report objects to have a unique and sortable ID field.

```
/**
 * A class containing a time stamp and a camera ID number.
 * As a camera can only ever make one Report at any given instant,
 * the Report's TimeID is a unique identifier of that Report.
 * <p>
 * TimeID's have a natural ordering based on the timestamp first,
 * and then the numerical value of the camera ID in the case that
 * two TimeID objects have the same timestamp.
 */
public class TimeID implements Comparable<TimeID> {
  // constant values to define the number of possible cameras
  public static final int MAX_CAMERA_ID = 9999;
  public static final int MIN_CAMERA_ID = 1;
  // class instance fields
  public int cameraID;
  public long timestamp;
  /** Constructor. */
  public TimeID( long timestamp, int cameraID) {
    this.timestamp = timestamp;
    this.cameraID = cameraID;
  }
  /**
   * Implements the Comparable<TimeID> interface to make TimeIDs orderable.
   */
  public int compareTo(TimeID other) {
    if (this.timestamp < other.timestamp) {
      return -1;
    } else if (this.timestamp > other.timestamp) {
      return 1;
    } else if (this.cameraID < other.cameraID) {
      return -1;
    } else if (this.cameraID > other.cameraID) {
      return 1;
    } else {
      return 0;
    }
  }
  /**
   * Returns an instance of TimeID with the given timestamp suitable to use
   * as the upper bound (end) for a time period for selection purposes.
   */
  public static TimeID upperBound( long timestamp) {
    return new TimeID( timestamp, MAX_CAMERA_ID + 1);
  }
  /**
   * Returns an instance of TimeID with the given timestamp suitable to use
   * as the lower bound (start) for a time period for selection purposes.
   */
  public static TimeID lowerBound( long timestamp) {
    return new TimeID( timestamp, MIN_CAMERA_ID - 1);
  }
  /**
   * Returns a String representing the values in this object so that is can
   * be directly printed with System.out.println();
```

```
   */
  public String toString() {
    return "Time: " + timestamp + " (" + String.format("%tc", timestamp) + ")\nCamera: " + cameraID;
  }
}
```

Tasks

1. You are required to analyse the ProvidedImplementation class giving an overview of how it works, the time complexity achieved for each of the three main functions and the various operations that they consist of, and a critical appraisal of the ways in which this is or is not an efficient implementation.

2. You are required to discuss the applicability of each of the data structures and algorithms listed below, for possible implementations of each of the application's functions:

   Array based lists, Linked lists, Sorting, Binary searching, Binary search trees and Hash maps.

   You should explain in what ways they could be used together to implement the functionality using different approaches, and what time complexities could be achieved by doing so. In doing so, you can assume that any sorting is accomplished with an efficient algorithm that offers O(n log n) time and works for both array and linked lists. You can also assume that any binary search tree is implemented to be self-balancing (either as an AVL or red/black implementation). **You do not need to, and should not, provide coding for this discussion**; it should be theoretical in nature, however it needs to be specific. You must address each function of the application in turn, discuss all possible approaches and compare them.

3. As the outcome of the discussion you have provided, you should propose what you believe to be the most efficient overall approach to fulfil each of the three main functions (receive report, reports for time period in time order, reports for camera in time order), using an assumption that the application should **prioritise a fast response for the two report queries over the performance of the method that receives reports**. Make sure you explain why you make the proposals you do.

4. Finally you should attempt to implement the approach described in part 3 as a Java application, along the lines of the ProvidedImplementation class (you can copy this and then change the code as you wish). The final code should be embedded in your report and you should clearly explain how it corresponds to the design you have proposed. Should you find that you cannot work out how to implement that design in Java, you should explain why and if you can make another approach from your discussions work you can use that instead.

Important things to keep in mind!

- The different data structures we look at in this module each have their own specific use cases based on the performance they can offer for various functions (such as adding, removing, finding, sorting or maintaining order). This is what you should be thinking carefully about in approaching this coursework.
- The provided implementation stores all received reports in a list, but another possibility is to use more than one data structure in order to optimise access times.
- Data structures contain references to the items they notionally contain. This means that a data structure can contain references to any kind of item, including other data structures. You can have lists of lists, or maps of lists or lists of maps and so on.
- You should make sure that it is clear that you know which data structures are used by the various Java collection classes when discussing the code to avoid ambiguity in your writing.

Do say "the TreeMap class uses a binary search tree which means it offers…" rather than "The TreeMap offers…"

- The main criteria being assessed is the degree to which you have absorbed the module's subject matter, can apply it to specific problems, can analyse relevant case studies and critically evaluate code in the context of this knowledge. This is more important than being able to implement a correctly functioning program and you should not spend all your time trying to get your code working at the expense of writing about the issues involved.

You should refer directly to lecture slides, exercises and any other materials you use to help you with the assignment. This is a directly assessed criteria (practical competence, see below).

You should not include the text of the tasks given above. Failing to follow these guidelines will impact the mark awarded (personal and professional development, see below).

There is no set word count but your report is unlikely to need to exceed 1500 words.

All work will be submitted to the TurnItIn plagiarism detection service. Your work should be entirely your own. Collaboration between students is not permitted. You are reminded that if another student submits some or all of your work without your permission you will be held responsible for this, as the University cannot tell who actually did the work; this is treated as collaboration and each student that is involved will be held accountable. For this reason **you are strongly advised *not* to share your work with other students**, and ***not* to publish parts of your work in progress on the VLE forum when seeking advice from tutors**.

Marking Criteria

This assignment will be marked using an adaptation of the University's new standardised marking criteria. It is important that you pay attention to the criteria that will be applied and address them in the text of your report.
Note that marks are awarded for the following main criteria:

1. **Subject Knowledge (35%)**

   Understanding and application of subject knowledge. Contribution to subject debate.
   Mainly assessed by your written explanation of all tasks.
2. **Critical Analysis (15%)**

   Mainly assessed by the rationale you give for your judgement on the most appropriate data structures for this case study.
3. **Testing and Problem-Solving Skills (30%)**

   Design, implementation, testing and analysis of product/process/system/idea/solution(s) to practical or theoretical questions or problems.
   Assessed mainly on the basis of your analysis of the provided code and your own coded implementation.
4. **Practical Competence (10%)**

   Skills to apply theory to practice or to test theory.
   Assessed on documented evidence of your use of technical documentation (for example, course materials and specific exercises or other technical materials) in working out how to accomplish the assignment.
5. **Personal and Professional Development (10%)**

   Management of learning through self-direction, planning and reflection
   Assessed on the basis of the quality of your submitted report.

RUBRIC

Please note the criteria weightings under each criteria.

Criteria	Outstanding 100-80%	Excellent 79-70%	Very good 69-60%	Good 59-50%	Satisfactory 49-40%	Inadequate 39-30%	Very poor 29-0%
Subject Knowledge Understanding and application of subject knowledge. Contribution to subject debate. **35%**	Shows sustained breadth, accuracy and detail in understanding key aspects of subject. Contributes to subject debate. Awareness of ambiguities and limitations of knowledge.	Shows breadth, accuracy and detail in understanding key aspects of subject. Contributes to subject debate. Some awareness of ambiguities and limitations of knowledge.	Accurate and extensive understanding of key aspects of subject. Evidence of coherent knowledge.	Accurate understanding of key aspects of subject. Evidence of coherent knowledge.	Understanding of **key aspects** of subject. Some evidence of coherent knowledge.	Some evidence of superficial understanding of subject. Inaccuracies.	Little or no evidence of understanding of subject. Inaccuracies.
Critical Analysis Analysis and interpretation of sources, literature and/or results. Structuring of issues/debates. **15%**	Outstanding demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Excellent demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Very good demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Good demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Demonstration of critical analysis of the **key** possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Trivial demonstration of critical analysis of the possible design strategies that could be used to meet the software requirements, and evaluation of the approaches chosen.	Little or no critical analysis has been demonstrated.
Testing and Problem-Solving Skills Design, implementation, testing and analysis of **product/process/system/idea/solution(s)** to practical or theoretical questions or problems **30%**	Outstanding implementation/analysis and description of all required software, with near perfectly organised, formatted and documented source code.	Excellent implementation/analysis and description of all required software, with well organised, formatted and documented source code provided.	Competent implementation/analysis and description of all required software, with well organised, formatted and documented source code.	Implementation/analysis and description of all required software, with well organised, formatted and documented source code, with some missing/incorrect functionality or poor quality.	Implementation/analysis and description of most of the required software, with well organised, formatted and documented source code, with some missing/incorrect functionality or poor quality.	Implementation/analysis and description of only part of the required software, with well organised, formatted and documented source code, with some missing/incorrect functionality or poor quality.	Little or no implementation/analysis and description.
Practical Competence Skills to apply theory to practice or to test theory **10%**	Outstanding descriptions of factual information, programming techniques or theoretical explanations being found in technical or theoretical reference material.	Excellent explicit descriptions of all factual information, programming techniques or theoretical explanations that were found in technical or theoretical reference material.	Good explicit descriptions of all factual information, programming techniques or theoretical explanations that were found in technical or theoretical reference material.	Reasonable descriptions of most factual information, programming techniques or theoretical explanations that were found in technical or theoretical reference material.	Basic examples of the **main** factual information, programming techniques or theoretical explanations that were found in technical or theoretical reference material.	Some trivial examples of factual information, programming techniques or theoretical explanations being found in technical or theoretical reference material.	Little or no evidence of factual information, programming techniques or theoretical explanations being found in technical or theoretical reference material.
Personal and Professional Development Management of learning through self-direction, planning and reflection **10%**	Outstanding report organisation, structure, presentation, narrative voice and language.	Excellent report organisation, structure, presentation, narrative voice and language.	Very good report organisation, structure, presentation, narrative voice and language.	Good report organisation, structure, presentation, narrative voice and language.	Satisfactory report organisation, structure, presentation, narrative voice and language.	Poor report organisation, structure, presentation, narrative voice and language.	Report does not constitute a serious attempt at the assignment.