RMIT UNIVERSITY

Algorithms and Analysis
COSC 1285/2123
Assignment 1

Assessment Type	Group assignment. Groups as allocated and notified on Canvas. Submit online via Canvas → Assignments → Assignment 1. Marks awarded for meeting requirements as closely as possible. Clarifications/updates may be made via announcements/relevant discussion forums.
Due Date	Friday 16th April 2021, 11:59pm
Marks	30

1 Overview

One of the frequent phrases we hear during the COVID-19 panademic are epidemic modelling and contract tracing. As the virus is transmitted human to human, one of the effective methods to stop an outbreak is to identify the close contacts of an infected individual, and isolate all of them for a period of time (typically 14 days). To estimate the impact of outbreaks, epidemic modelling is an extremely important tool. A graph is a natural representation for these close contacts, tracing outbreaks and for epidemic modelling.

When we represent these networks as a graph, the vertices in such a graph represent people and edges represents close contact relationship.

In class, we studied three methods to represent the graph, the adjacency list, adjacency matrix and vertex/edge list representations. There is a fourth type of representation called *incident matrix* (see below for details). The performance of each representation varies depending on the characteristics of the graph. In this assignment, we will implement the adjacency list, adjacency matrix and incident matrix representations for the relational parts, and evaluate on how well they perform when modelling close contacts and for epidemic modelling. This will assist with your understanding of the tradeoffs between data structure representations and its effect when implementing operations or solving problems using algorithms and data structures.

2 Learning Outcomes

This assessment relates to all of the learning outcomes of the course which are:

- CLO 1: Compare, contrast, and apply the key algorithmic design paradigms: brute force, divide and conquer, decrease and conquer, transform and conquer, greedy, dynamic programming and iterative improvement;
- CLO 2: Compare, contrast, and apply key data structures: trees, lists, stacks, queues, hash tables and graph representations;

- CLO 3: Define, compare, analyse, and solve general algorithmic problem types: sorting, searching, graphs and geometric;
- CLO 4: Theoretically compare and analyse the time complexities of algorithms and data structures; and
- CLO 5: Implement, empirically compare, and apply fundamental algorithms and data structures to real-world problems.

3 Background

3.1 Networked SIR Model

There are several well-known epidemic models, but the one that we will focus on in this assignment is the (Networked) SIR model. The SIR epidemic model assumes each individual has three states - *Susceptible (S)* to the disease, *Infected (I)* by the disease and *Recovered (R)* to the disease. The progression of states is assumed to be susceptible (S) to Infected (I) to Recovered (R). Initially some in the initial population are infected and all others are susceptible. Once recovered, an individual has immunity and will not be further infected. There are also other models, like SIS (Susceptible, Infected, Susceptible), but for the purpose of this assignment, we will stick with the most fundamental model, the SIR model, as others are really variations on this.

In the traditional SIR model, there is assumption of full mixing between individuals - i.e., any individual can potentially infect another. The model then derives differential equations that estimates the changes in the total number of susceptible, infected and recovered individuals over time, and these are proportional to the numbers of the other two states - e.g., change in number of infected individuals is dependent on the number of susceptible and the number of recovered individuals. It is able to model some diseases relatively well, but is inaccurate for others and misses out on local spreading and effects. Hence a networked SIR model was proposed, where instead of full mixing between individuals, infected individuals can now only infect its neighbours in the graph/network, which typically represents some form of "close contact".

The networked SIR model works as follows.

At each timestep, each susceptible individual can be infected by one of its infected neighbours (doesn't matter which one, as an individual is infected regardless of who caused it). Each infected neighbour has a chance to infect the susceptible individual. This chance is typically modelled as an infection parameter α - for each chance, there is α probability of infection. In addition, at each timestep each infected individual have a chance of recovering, either naturally or from medicine and other means (this model doesn't care which means). Let the recovery parameter β represent this, which means there is β probability that an infected individual will recover and change to recovered state. There is no transition for susceptible to recovered (which could be interesting, e.g., to model vaccination, but left for future assignments).

As you can probably infer, the more neighbours one has, the higher potential for infection. Hence one reason for the general advice from the health departments to reduce the number of close contacts. Also wearing masks reduces the infection probability, again reducing the spread of an outbreak. In this assignment, you will get an opportunity to experiment with these parameters and see what differences they make.

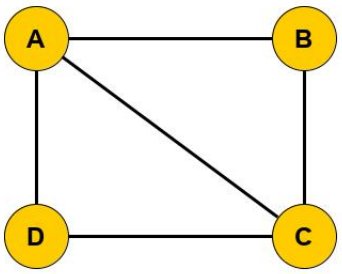
For more information about SIR and epidemic modelling, have an initial read here `https://en.wikipedia.org/wiki/Compartmental_models_in_epidemiology`. We will also release some recordings talking more about the model on Canvas, particularly providing more details about how to run the SIR model to simulate an epidemic, please look at those also when they become available.

3.2 Incidence Matrix Representation

The ***incidence matrix*** represents a graph as a set of vertices and list of edges incident to each vertex as a 2D array/matrix. More formally, let the incidence matrix of a *undirected* graph be an `n x m` matrix A with `n` and `m` are the number of vertices and edges of the graph respectively. The values in the matrix A follows the following rules:

- Each edge e_k is represented by a column in A.
- Each vertex is represented by a row in A.
- Let the two incident vertices to an edge e_k be v_i and v_j. Then the matrix entries $A_{i,k}$ and $A_{j,k}$ are both $= 1$.
- $A_{i,k} = 0$ for all other non-incident edges.

For example, the following graph:



has its incidence matrix as below:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{ccccc} AB & AC & AD & BC & CD \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

For further readings about this graph representation, please see `https://en.wikipedia.org/wiki/Incidence_matrix`.

4 Assignment Details

The assignment is broken up into a number of tasks, to help you progressively complete the project.

Task A: Implement the Graph Representations, their Operations and the Networked SIR model (10 marks)

In this task, you will implement data structures for representing *undirected*, *unweighted* graphs using the *adjacency list*, *adjacency matrix* and *incidence matrix* representations. Your implementation should also model vertices that have SIR model states associated with them. Your implementation should support the following operations:

- Create an empty undirected graph (implemented as a constructor that takes zero arguments).
- Add a vertex to the graph.
- Add an edge to the graph.
- Toggle the SIR state on vertices.
- Delete a vertex from the graph.
- Delete an edge from the graph.
- Compute the k hop neighbours of a vertex in the graph.
- Print out the set of vertices and their SIR states.
- Print out the set of edges.

In addition, we want you to implement the (networked) SIR epidemic model to simulate how a virus can spread through a population.

Note that you are welcome to implement additional functionality. When constructing our solutions to the assignment, we have found that adding some methods was helpful, particularly for implementing the SIR model. But the above functionalities are ones you should complete and ones you will be assessed on.

Data Structure Details

Graphs can be implemented using a number of data structures. You are to implement the graph abstract data type using the following data structures:

- Adjacency list, using an array of linked lists.
- Adjacency matrix, using a 2D array (an array of arrays).
- Incidence matrix, using a 2D array (an array of arrays).

For the above data structures, you must program your own implementation, and not use the LinkedList or Matrix type of data structures in java.utils or any other libraries. You must implement your own nodes and methods to handle the operations. If you use java.utils or other implementation from libraries, this will be considered as an invalid implementation and attract 0 marks for that data structure. The only exceptions are:

- if you choose to implement a map of vertex labels to a row or column index for the adjacency matrix or incidence matrix, you may use one of the existing Map classes to do this.
- if you choose to implement a map of vertex labels to its associated SIR state, you may use of the existing Map classes to do so.

Operations Details

Operations to perform on the implemented graph abstract data type are specified on the command line. They are in the following format:

```
<operation> [arguments]
```

where operation is one of {AV, AE, TV, DV, KN, PV, PE, SIR, Q} and arguments is for optional arguments of some of the operations. The operations take the following form:

- `AV <vertLabel>` – add a vertex with label 'vertLabel' into the graph. Has default SIR state of susceptible (S).
- `AE <srcLabel> <tarLabel>` – add an edge with source vertex 'srcLabel', target vertex 'tarLabel' into the graph.
- `TV <vertLabel>` – toggle the SIR state of vertex 'vertLabel'. Togglng means go to the next state, i.e., from S to I, from I to R (if in R, remain in R).
- `DV <vertLabel>` – delete vertex 'vertLabel' from the graph.
- `DE <srcLabel> <tarLabel>` – remove edge with source vertex 'srcLabel' and target vertex 'tarLabel' from the graph.
- `KN <k> <vertLabel>` – Return the set of all neighbours for vertex 'vertLabel' that are up to k-hops away. The ordering of the neighbours does not matter. See below for the required format.
- `PV` – prints the vertex set of the graph. See below for the required format. The vertices can be printed in any order.
- `PE` – prints the edge set of the graph. See below for the required format. The edges can be printed in any order.
- `SIR <infected seed vertices, delimited by ;> <infection probability> <recover probability>` – Run the networked SIR model simulation, using the current graph as the network, the specified seed vertices as additional set of infected vertices (in addition to any existing in the current graph) and the two infection and recover

probabilities. Runs the model simulation to completion, which means two things a) there are no more vertices with Infected (I) state and there are no changes in the number of infected or recovered in the latest iteration of the model; or b) if condition there are still infected vertices but there has been no changes in the number of infected or recovered for 10 iterations, then can stop the simulation. Outputs the list of nodes infected at each iteration, see below for format.

- `Q` – quits the program.

k-hop Neighbour operation format details The format of the output of the neighbour operation for vertex 'A' should take the form:

```
A: neighbour1 neighbour2 ...
```

If a vertex has no neighbours, then the list of neighbours should be empty. The node 'A' should not be in the list of neighbours, and the graphs can be assumed to have no self-loops (self loops don't necessarily make sense for epidemic modelling).

Print vertex operation format details The print vertex operation output the vertices and associated SIR state in the graph in a single line. The line should specifies all the valid vertex (indices) in the graph.

```
(vertex1,state1) (vertex2,state2) (vertex3,state3) ...
```

where state1 is the SIR state of vertex 1 etc.

Print edge operation format details The print edge operation output the edges in the graph in over a number of lines. Each line specifies an edge in the graph, and should be in the following format:

```
srcVertex tarVertex
```

SIR operation format details The networked SIR model file output format is each line represents one iteration of the process, and each line will record the number of newly infected and recovered vertices (newly infected or recovered nodes are ones that become infected or recovered in this iteration). The format is as follows:

```
<iteration number> :  [<space separated, list of newly infected
vertices>] :  [<space separated, list of newly recovered vertices>]
```

Example of all operations As an example of the operations, consider the output from the following list of operations:

AV A
AV B
AV C
AV D
AV E
AV F
AV G

```
AV H
AE A B
AE C B
AE B D
AE A E
AE D C
AE F A
AE B F
AE C G
AE G H
AE E G
AE F G
KN 1 A
KN 2 A
KN 1 F
DV C
DE B A
DE G H
KN 1 H
TV B
TV E
TV B
PV
PE
SIR A;F 0.8 0.5
Q
```

The output from the four neighbour operations ('KN 1 A', 'KN 2 A' 'KN 1 F', 'KN 1 H') should be (remember that the order these neighbourhoods do not matter so if your output differs from this ordering but has the same set of vertices, that is fine):

```
A: B E F
A: B C D F E G
F: A B G
H:
```

The output from the print vertices operation (PV) could be (remember that the order doesn't matter):

```
(A,S) (B,R) (D,S) (E,I) (F,S) (G,S) (H,S)
```

The output from the print edges operation (P E) could be (remember that the order doesn't matter):

```
A E
E A
F A
A F
B F
F B
```

```
D B
B D
E G
G E
F G
G F
```

The output from the 'SIR A;F 0.8 0.5' operation could be (note that it is a stochastic model, so you can get different results for different runs, so use the following more for what the output format should look like):

```
1: [G] : []
2: [] : [A]
3: [] : []
4: [] : []
5: [] : [F G]
6: [] : []
```

Testing Framework

We provide Java skeleton code (see Table 1 for the important files) to help you get started and automate the correctness testing. You may add your own Java files to your final submission, but please ensure that they compile and can be run on the core teaching servers.

Notes

- Consider the specifications here carefully. If you correctly implement the "Implement me!" parts and follow the output formating for operations like PV, PE and SIR, you in fact do not need to do anything else to get the correct output formatting. The main class in the provided skeleton code will handle this.
- The implementation details are up to you, but you must implement your own data structures and algorithms, i.e., do not use the built-in structures in Java unless for the purposes outlined in the box above.
- If you develop on your own machines, please ensure your code compiles and runs on the university's core teaching servers. You don't want to find out last minute that your code doesn't compile on these machines. If your code doesn't run on these machines, we unfortunately do not have the resources to debug each one and cannot award marks for testing.
- All submissions should compile with no warnings on **Oracle Java 1.8** - this is the version on the core teaching servers.
- Any clarifications on the Canvas FAQ or Assignment 1 Update page will override the specifications here if they are conflicting.

file	description
`RmitCovidModelling.java`	Main class. Contains code that reads in operation commands from stdin then executes those on the selected graph implementation. Also will format the output as required. *No need to modify this file.*
`ContactsGraph.java`	Interface class for the graph representations. It contains the common interface/methods that you'll need to implement for the various representations. *Generally no need to modify this file, but you might want to add helper functionality, and here would be one place to do so for common ones across all representations..*
`AdjacencyList.java`	Code that implements the adjacency list implementation of a graph. Complete the implementation (implement parts labelled "Implement me!").
`AdjacencyMatrix.java`	Code that implements the adjacency matrix implementation of a graph. Complete the implementation (implement parts labelled "Implement me!").
`IncidenceMatrix.java`	Code that implements the incidence matrix implementation of a graph. Complete the implementation (implement parts labelled "Implement me!").
`SIRState.java`	Code that implements the three vertex states of a SIR model. *No need to modify this file.*
`SIRModel.java`	Code that implements the networked SIR model. Complete the implementation (implement parts labelled "Implement me!").

Table 1: Table of Java files.

Task B: Evaluate your Data Structures and SIR model (20 marks)

In this second task, you will evaluate your implemented structures in terms of their time complexities for the different operations and different use case scenarios. Scenarios arise from the possible use cases of contract tracing and epidemic modelling on a graph. In addition, run some epidemic modelling simulations to evaluate your structures within an application (epidemic modelling and simulation) and to explore more about the effect of the model parameters on epidemic spreading.

Write a report on your analysis and evaluation of the different implementations. Consider and recommend in which scenarios each type of implementation would be most appropriate. Report the outcomes of your SIR epidemic model simulations. The report should be **10 pages or less**, in font size 12. See the assessment rubric (Appendix A) for the criteria we are seeking in the report.

Graph generators

To evaluate the data structures, we will generate a number of graphs, then apply a number of operations on them, see the Use Case Scenarios section next. In this section, we describe more about how to generate the graphs, and see on Canvas for further details and videos.

There are many ways to generate graphs, but we will focus on two:

- Random (Erdos-Renyi) – there is a probability for generating an edge between any pair of vertices.
- Scale-free – graphs whose degree distribution follow a power law (see `https://en.wikipedia.org/wiki/Scale-free_network`).

We suggest to use Pajek[1], which is available on multiple platforms and has visualisation, which allows one to look at the generated graphs. But it also has basic graph generation functionality and doesn't require programming to use, unlike more powerful graph libraries[2].

Use Case Scenarios

Typically, you use real usage data to evaluate your data structures. However, for this assignment, **you will write data generators** to enable testing over different scenarios of interest. We are also interested in the effect of the average degree of the graph[3] on these scenarios. There are many possibilities, but for this assignment, consider the following scenarios:

Scenario 1 k-hop Neighbourhoods: When doing contract tracing and/or epidemic modelling, computing neighbourhoods is an important function. In this scenario, the graph is not changing, but important operations such as k-hop neighbourhood (recall this is all neighbours up to k-hops away) are requested.

You are to evaluate the performance of the the k-hop neighbourhood implementations, as the average degree of the evaluated graph and k are varied.

Scenario 2 Dynamic Contact Conditions: In the real world, the contact conditions between people are likely not static and there will be need to update these over time. In this scenario, the contacts are changing (been added and deleted). In this scenario, you are to evaluate the performance of your implementations in terms of:

- edge additions
- edge deletions

You are to evaluate the performance the edge operations as the average degree and size (number of vertices) of the initial graph (before edge changes) are varied.

[1] `http://mrvar.fdv.uni-lj.si/pajek/` the website looks a bit dated, but this is one of the well known social network analysis tools.

[2] Networkx `https://networkx.org/` – if you know Python, you are more than welcome to use this instead.

[3] Average number of neighbours per node, which both graph generator models allows to be specified

Scenario 3 Dynamic People Tracing: As time goes on, the people been traced will change. Although it is possible to have a large graph that tries to capture every possible person, it is computationally difficult to do so and run simulations on it. In this scenario, you are to evaluate the performance of your implementations in terms of:

- vertex addition
- vertex deletion

You are to evaluate the performance the vertex operations as the average degree and size (number of vertices) of the initial graph (before vertex changes) are varied.

SIR Model Epidemic Simulation

In addition to evaluating how well the graph representations perform for the contract tracing and epidemic modelling simulations, we also want to experiment with how the networked SIR model works and performs.

Run epidemic simulation for different graph types (Erdos-Renyi and Scale-free), different seed initialisations and different infection and recover probabilities. For performance evaluation, evaluate each of the three data structures on a subset of the possible simulation parameters and compare how they perform (in terms of timing efficiency).

We don't expect a comprehensive analysis, but want you to explore and gain a better understanding of epidemic modelling and what it implies for how to limit the spread of epidemics.

Consider how you might present the results, but at a minimum plot the following:

- Number of susceptible, infected and recovered after every iteration.

Data Generation

When generating the vertices and edges to add or remove and find neighbourhoods for, the distribution of these elements, compared to what is in the graph already, will have an effect on the timing performance. However, without the usage and query data, it is difficult to specify what this distributions might be. Instead, in this assignment, uniformly sample from a fixed range, e.g., 0 to max vertex label of your graph when generating the vertices and edges for removing and adding and k-hop neighbourhoods, and a different range (e.g., greater than the largest vertex label when adding vertices (we do not want to repeatingly add vertices that are in the graph already).

For generating graphs with different initial average degrees and sizes, you can either use Pajek or your favourite graph generator, we will not prescribe one particular approach. Whichever method you decide to use, remember to generate graphs of different average degrees to evaluate on. Due to the randomness of the data, you may wish to generate a few graphs with the same average degrees and sizes and take the average across a number of runs when performing the performance evaluation and analysis.

Analysis

In your analysis, you should evaluate each of your representations and data structures in terms of the different scenarios outlined above.

Note, you may be generating and evaluating a significant number of graphs and evaluation datasets, hence we advise you to get started on this part relatively early.

5 Report Structure

As a guide, the report could contain the following sections:

- Explain your data and experimental setup. Things to include are (brief) explanations of the data scenarios you decide to evaluate on (e.g., how many additions/removals operations did you evaluate and why these), size of the graphs, the range of average degrees tested (add some brief explanation of why this range selection), describe how the scenarios were generated (a paragraph and perhaps a figure or high level pseudo code suffice), which approach(es) you decide to use for measuring the timing results, and briefly describe the fixed set(s) you sampled from to generate the elements for addition, removal of vertices and edges and k-hop neighbourhood of the graphs.

- Evaluation of the data structures using the generated data (different scenarios, average degree, graph size). Analyse, compare and discuss your results. Provide your explanation on why you think the results are as you observed. You may consider using the known theoretical time complexities of the operations of each data structure to help in your explanation if useful.

- Summarise your analysis as recommendations, e.g., for this certain data scenario of this average degree (and size), I recommend to use this data structure because... We suggest you refer to your previous analysis to help.

- Evaluate the different parameters of the SIR model and its effect on the number of susceptible, infected and recovered individuals over time. Which combination of parameters has the fastest infection spread, which one has the most extensive number of infected? How does the average timing compare between your three graph representation? Can you explain why?

6 Submission

The final submission will consist of three parts:

- Your **Java source code** of your implementations. **Include only source code and no class files**, we will compile your code on the core teaching serers. Make sure that your assignment can run only with the code included! That is, it should be self contained, including all the skeleton code needed to run it.

  Your source code should be placed into in a flat structure, i.e., all the files should be in the same directory/folder, and that directory/folder should be named as `Assign1-<your student number>`. Specifically, if your student number is s12345, when `unzip Assign1-s12345.zip` is executed then all the source code files should be in directory Assign1-s12345.

- Your **data generation code**. Create a sub-directory/sub-folder called "generation" within the Java source file directory/folder. Place your generation code within that folder. We will not run the code, but may examine their contents.

- Your **written report for part B** in PDF format, called "assign1.pdf". Submit this separately to a Turnitin submission.

Note: **submission of the report and code will be done via Canvas**. We will provide details closer to the submission deadline. We will also provide details on how the setup we use to test the correctness of your code will be like, so you can ensure your submitted structure will conform to the automated testing we will perform.

7 Assessment

The project will be marked out of 30. Late submissions will incur a deduction of 3 marks per day, and no submissions will be accepted 5 days beyond the due date and will attract 0 marks for the assignment.

The assessment in this project will be broken down into two parts. The following criteria will be considered when allocating marks.

Implementation (10/30):

- You implementation will be assessed based on the number of tests it passes in our automated testing.
- While the emphasis of this project is not programming, we would like you to maintain decent coding design, readability and commenting, hence commenting and coding style will make up a portion of your marks.

Report (20/30):

The marking sheet in Appendix A outlines the criteria that will be used to guide the marking of your evaluation report[4]. Use the criteria and the suggested report structure (Section 5) to inform you of how to write the report.

8 Team Structure

This project should be done in **pairs** (group of two). If you have difficulty in finding a partner, post on the discussion forum or contact your lecturer. If you want to do the assignment individually, please **contact one of your lecturers first and obtain his approval**.

In addition, please submit what percentage each partner made to the assignment (a contribution sheet will be made available for you to fill in), and submit this sheet in your submission. The contributions of your group should add up to 100%. If the contribution percentages are not 50-50, the partner with less than 50% will have their marks reduced. Let student A has contribution X%, and student B has contribution Y%, and $X > Y$. The group is given a group mark of M. Student A will get M for assignment 1, but student B will get $\frac{M}{\frac{X}{Y}}$.

This semester we will also institute some additional group management initiatives, including registration and not allowing further group changes in the week before the due date. We will detail more about this on Canvas.

[4]Note for the marking guide, if one of the criteria is not demonstrated a tall, then 0 marks will be awarded for that criteria.

9 Academic integrity and plagiarism (standard warning)

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods
- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the following: `https://www.rmit.edu.au/students/student-essentials/rights-and-responsibilities/academic-integrity`.

We will run both code and report similarity checks.

10 Getting Help

There are multiple venues to get help. First point of call should be Canvas, assignment FAQ (on Canvas), recordings about it and the discussion forum. There will also be weekly online consultation hours for the assignment (see Canvas for times). In addition, you are encouraged to discuss any issues you have with your Tutor or Lab Demonstrator. Please **refrain from posting solutions** to the discussion forum.

A Marking Guide for the Report

Design of Evaluation (Maximum = 5 marks)	**Analysis of Results** (Maximum = 10 marks)	**Report Clarity and Structure** (Maximum = 5 marks)
5 marks Data generation and experimental setup are well designed, systematic and well explained. All suggested scenarios, data structures and a reasonable range of parameters were evaluated. Each type of test was run over a number of runs and results were averaged.	10 marks Analysis is thorough and demonstrates understanding and critical analysis. Well-reasoned explanations and comparisons are provided for all the data structures, scenarios and parameters. All analysis, comparisons and conclusions are supported by empirical evidence and possibly theoretical complexities. Well reasoned recommendations are given.	5 marks Very clear, well structured and accessible report, an undergraduate student can pick up the report and understand it with no difficulty.
3.75 marks Data generation and experimental setup are reasonably designed, systematic and explained. There are at least one obvious missing suggested scenarios, data structures or reasonable parameters. Each type of test was run over a number of runs and results were averaged.	7.5 marks Analysis is reasonable and demonstrates good understanding and critical analysis. Adequate comparisons and explanations are made and illustrated with most of the suggested scenarios and parameters. Most analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Reasonable recommendations are given.	3.75 marks Clear and structured for the most part, with a few unclear minor sections.
2.5 marks Data generation and experimental setup are somewhat adequately designed, systematic and explained. There are several obvious missing suggested scenarios, data structures or reasonable parameters. Each type of test may only have been run once.	5 marks Analysis is adequate and demonstrates some understanding and critical analysis. Some explanations and comparisons are given and illustrated with one or two scenarios and parameters. A portion of analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Adequate recommendations are given.	2.5 marks Generally clear and well structured, but there are notable gaps and/or unclear sections.
1 marks Data generatio and experimental setup are poorly designed, systematic and explained. There are many obvious missing suggested scenarios, data structures or reasonable parameters. Each type of test has only have been run once.	2 marks Analysis is poor and demonstrates minimal understanding and critical analysis. Few explanations or comparisons are made and illustrated with one scenario and parameter setting. Little analysis and comparisons are supported by empirical evidence and possibly theoretical analysis. Poor or no recommendations are given.	1 marks The report is unclear on the whole and the reader has to work hard to understand.