

**Department of Computing**

**Algorithms and Data Structures**

**(55-508810-AF-202425)**

**Name: J. A. Rowden**

**Student ID: 33004175**

**Table of Contents**

[1. Project 1 3](#_Toc178928233)

[1.1 Computational Complexity 3](#_Toc178928234)

[1.2 Observation of rate of growth 3](#_Toc178928235)

[1.3 Incorporation of formative feedback 4](#_Toc178928236)

[2. Project 2 5](#_Toc178928237)

[2.1 Algorithms in an Algorithmic Language, such as ADL 5](#_Toc178928238)

[2.2 Data Structures 7](#_Toc178928239)

[2.3 Software and its Presentation, including testing (and video link) 7](#_Toc178928240)

[2.4 Descriptive Report, including artefacts 10](#_Toc178928241)

[2.4.1 Transitioning algorithms to implementation 10](#_Toc178928242)

[2.4.2 Problem-solving strategy 11](#_Toc178928243)

[2.5 Incorporation of formative feedback 13](#_Toc178928244)

[3. Project 3 14](#_Toc178928245)

[3.1 **Data Structure selection** 14](#_Toc178928246)

[3.2 Developed Algorithms 14](#_Toc178928247)

[3.2.1 Algorithm1: Influencers in Unweighted Network 14](#_Toc178928248)

[Time Complexity of Algorithm1: Discuss about the time complexity of Algorithm1 that you have developed. 14](#_Toc178928249)

[3.2.2 Algorithm2: Influencers in Weighted Network 14](#_Toc178928250)

[Time Complexity of Algorithm2: Discuss about the time complexity of Algorithm2 that you have developed. 14](#_Toc178928251)

[3.3 Implementation 15](#_Toc178928252)

[3.3.1 Implementation details 15](#_Toc178928253)

[3.3.2 Publishing your project 15](#_Toc178928254)

[3.3.3 Evaluation 15](#_Toc178928255)

[3.4 Reflection 15](#_Toc178928256)

[3.4.1 Video 15](#_Toc178928257)

[3.4.2 Feedback 15](#_Toc178928258)

[4. Project 4 16](#_Toc178928259)

[4.1 Introduction to Bin Packing Problem 16](#_Toc178928260)

[4.2 Random Mutation Hill Climbing 16](#_Toc178928261)

[4.3 Algorithm Design 16](#_Toc178928262)

[4.3.1 Solution Representation 16](#_Toc178928263)

[4.3.2 Fitness Function Analysis 16](#_Toc178928264)

[4.3.2 Small Change Strategy 16](#_Toc178928265)

[4.3.4 Data Structure 16](#_Toc178928266)

[4.3.5 Algorithm Pseudocode 16](#_Toc178928267)

[4.3.6 Experiment Strategy 16](#_Toc178928268)

[4.4 Results 16](#_Toc178928269)

[4.5 Testing 16](#_Toc178928270)

[4.6 Software and its Presentation, including testing (and video link) 16](#_Toc178928271)

[4.7 Incorporation of formative feedback 17](#_Toc178928272)

# 1. Project 1

## Computational Complexity

You are expected to provide computational complexity expressed in terms of Big-O. We are not interested in verbal explanation. We are much more interested in mathematical justification expressed in terms of a polynomial expression, e.g.,

*NOTE to SELF: Insert Scan then anotate*

## Observation of rate of growth

Let the two functions be known as:

F(n) = n2

G(n) = 2n/4

To observe the rate of growth of these functions so I can comment on them I have devised 2 methods.

Method 1: table of values

|  |  |  |
| --- | --- | --- |
| n | f(n) | g(n) |
| 1 | 1 | 0.5 |
| 5 | 25 | 8 |
| 10 | 100 | 256 |
| 15 | 225 | 8192 |
| 20 | 400 | 262144 |
| 25 | 625 | 8388608 |
| 30 | 900 | 2.68E+08 |

*Data generate by MS excel spreadsheet (LINK?) .*

Method2: graphics calculator application

*Using* [*https://www.geogebra.org/calculator*](https://www.geogebra.org/calculator) *:*

*A screenshot of a graph

Description automatically generated*

*Note: x is used as a substitute for n and is limited to the positive scale as negative complexity does not need to be considered.*

Analysis

As shown best by the graph g(n) is better for small values of n - up till n = 8 (see intersect point B) . Note I’m ignoring the first intercept point A as data sets normally consist of 1 or more items, not of ~ .062 of an item. However past a data set size of 8, as best shown by the table if values f(n) is significantly better than g(n) as the difference between f(n) and g(n) grows exponentially large.

## 1.3 Incorporation of formative feedback

Provide evidence of how you evaluated and acted on the formative feedback you received from your tutors, including the tutor’s name and the timestamp.

|  |  |  |  |
| --- | --- | --- | --- |
| Date and Time | Name of the Tutor | Feedback of the tutor | How the comment is addressed |
| 04/10/24 ~ 1040 | Mehemet | Excellent work | Kept doing what I was doing |
|  |  |  |  |

# 2. Project 2

## 2.1 Algorithms in an Algorithmic Language, such as ADL

I have designed two separate algorithms and corresponding data structures (A&DS) to solve the this problem, based on depth first and breadth first traversals of graphs to avoid confusion on which algorithms go with which data structures I will list common A&DS and then under separate headings the A&DS for each approach.

Common Data Structures.

**2D array – used for Maze Representation:**

* The maze doesn’t change, only accessed so the lack of dynamics doesn’t matter
* Space efficient
* Easily accessed
* *Note:* Though I represented the maze using a 2D array I treated the maze more like a graph in both solutions with adjacent 0s being connected and ignoring the 1s.

**Endless Stack – used to store chain of visited nodes/positions**

* Custom data type
* A dynamic stack as I would not be able to tell how big it need to be before running the algorithm and this size would change with different mazes and algorithms.
* Used as I wanted a data structure that preserved the order on which items where added to it so the route would not be compromised
* *Note:* both ADL approaches use the endless stack but during development of the depth first approach I realised I didn’t need the extra functionality of the stack as this was handled by the recursive nature of the algorithm so I reverted to using a list.
* **Functions:**
  + **isEmpty()**
  + **push() – adds an item to the top of the stack**
  + **pop() – returns and removes the top item on the stack**
  + **convertToList() – returns in list form for display**
  + **Count() – returns integer value of number of item in stack.**

**List – Used to store visited nodes and then the chain of visited nodes after removal of endless stack from the depth first approach.**

* Dynamic for same reasons as endless stack
* Easily accessed

Common Algorithms

**InRange – checks if a position that algorithm was looking at was inside the map.**

* Used to avoid errors
* Used to recognize the exit.
* Implemented in a parent class of both approaches classes.
* *Note:* this was implemented during development after receiving feedback from lecturer hence no ADL

Depth First Data Structures

**Array – Used for Positions/ positions**

* The maze doesn’t change dimensions, so the lack of dynamics doesn’t matter
* Space efficient compared to an object approach
* Easily accessed

Depth First Algorithms

In a low level pseudo code :

function lookArround(array currentLocation, array maze ){

// returns an array of all coordinates 1 place away form currentLocation

int x <- currentLocation - 1

int y <- currentLocation + 1

array zerosLocation <- {}

for y to y + 3 {

for x to x + 3 {

if maze[x,y] == 0 and maze[x,y] != currentLocation {

zerosLocation.add({x,y})

} else if maze[x,y] != 1 {

zerosLocation.add({-1,-1}) // -1,-1 signifies end

}

x <- x + 1

}

x <- current location - 1 // resets x to end s

y <- y - 1

}

return zerosLocation

}

function findRoute(array currentPos, EndlessStack previous, list visited, array maze){

// returns to exits if found if not returns empty array.

array options <- lookArround(currentPos, maze)

previous.add(currentPos)

vistited.add(currentPos)

//check if there's not any options i.e. dead end

if (options == null){

return {}

} else{

// check if exit found

for each option in options {

if (option == {-1,-1} and previous != null){

previous.add(currentPos)

return previous

}

// if not check paths

for each option in options {

if( !visited.isIn(option)){

EndlessStack route <- findRoute(option, previous, visited maze)

if (route != null){

return route

}

}

}

if ( previous == null) {

print("no exit")

}esle{

return {}

}

}

}

Depth First Data Structures

**Queue – Used to store nodes to visit next**

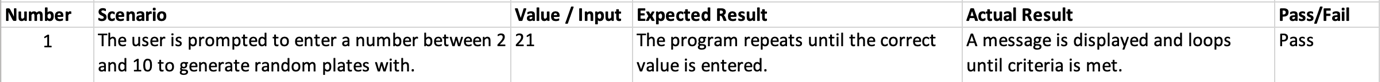
## 2.3 Software and its Presentation, including testing (and video link)

In this section, you are required to explain your software’s functionality and how you tested it. To keep the word count to a minimum, it is sufficient to present your software in the form of a video recording. **Do not forget to ensure the video recording is accessible by us If we cannot access your video recording, you will get zero marks for that section. Therefore, please make sure that you test your video link by asking someone else accessing your video recording.**Refer to the assessment brief for more information on the format that must be followed during recording, e.g.,

[*https://youtu.be/nR\_NoYhzUIs*](https://youtu.be/nR_NoYhzUIs)

In your video recording, you will need to demonstrate how you tested the software and what results you obtained using the acceptance tests outlined in this section, e.g.,

**Acceptance tests**

****

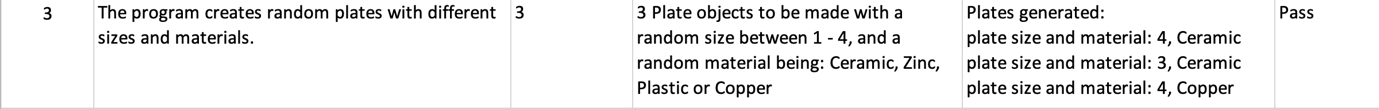
Text

Description automatically generated



Graphical user interface

Description automatically generated



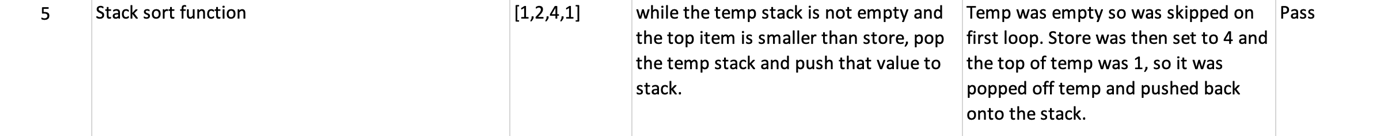
Text

Description automatically generated



Graphical user interface, text, application

Description automatically generated

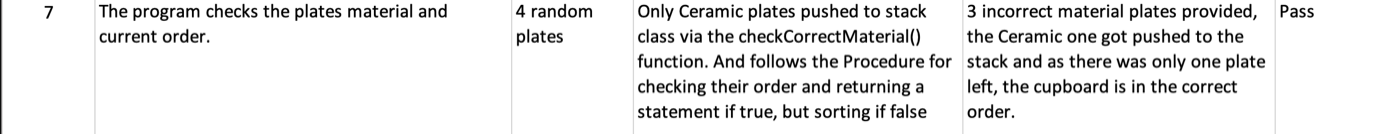
Text

Description automatically generated



Graphical user interface, text

Description automatically generated



Text

Description automatically generated

## 2.4 Descriptive Report, including artefacts

In this section, you are required to elaborate on how you devised your algorithm and developed the corresponding implementation in C#. Specifically, we are interested in your description of how you transformed your algorithm(s) into a corresponding implementation in C# and what strategy you employed to address the issues in this project (see below).

### 2.4.1 Transitioning algorithms to implementation

*Writing the code from ADL to Java was very simple, it does include more print statements for the program, however the same implementation is there as shown in the ADL above. The main difference is the stack class has a member variable called ‘item’, which stores the last item that has been popped from the stack. Apposed to directly storing the item from the pop() method returning a value.*

*Here is a diagram of the StackClass that I have created. This is the main class, however in the coded version there is some extra print functions for visual purposes only in the console application, so I have not included them here.*

Table

Description automatically generated

*The entirety of the coded Stack class will be shown in the video presentation. However here is the implemented sort() method designed to sort the plates in order with the largest at the bottom and smallest on top.*

Text

Description automatically generated

*As well as the procedure of creating the randomised plates and checking the material and initial order.*

Text

Description automatically generated

### 2.4.2 Problem-solving strategy

*To reorder the plates, I had to take into consideration the Time-Space Trade which made me to choose stack for my data representation, apart from the choosing the right data structure I had to think of how I will represent the plates, the easiest way to do that is to refer to the plates as integers since we know that we only have 4 types of plates that are desirable we can go for 1, 2, 3, 4 and any other number is an undesired plate.*

*Solution 1:*

*The first solution that I came with is space efficient, my algorithm consists of using two stacks, the first will contain the hand of plates in any order, the second stack will be used to sort the plates in the right order, the sorting process is simple:*

*The first step is to move the plates from stack 1 to stack 2 and getting rid of the unwanted plate types.*

*The second step is to move back the plates from stack 2 to stack 1 but we will be following some rules:*

*When we move a plate from stack 2 to stack 1 we need to check either plate type that we are moving is bigger or equal to the top of the first stack if it is we just push it there, if the plate type is less than the top of the first stack we will move the top of the first stack to the second stack and compare again the top of the first stack with our plate, every time we push an item from stack 1 to 2 we are having a count that increase every time, until we fulfil the condition, when we do we are pushing back the plates from stack 2 to stack 1 that have been moved for the sorting using a for loop using the count. We are repeating the process if the second stack is not empty.*

*The complexity of our algorithms:*

*F(n) = n+ (n-j).2k*

*So, O(f(n)) = n + (n-j).2k*

*n = initial number of plates*

*j = is the number unwanted plates*

*k = is the number of plates inside stack 1 when we want to move a plate from stack 2 to stack 1, we multiply it by 2 because we are moving them twice: from stack 1 to stack 2 then from stack 2 to stack 1*

*We are taking k as the worst-case scenario.*

*Here is a diagram representing our sorting process:*

Diagram, engineering drawing

Description automatically generated

Solution 2:

*The second solution that I came with is a time efficient solution but that consumes space in parallel, I am basically using 5 different stacks an initial stack where the plates are placed in any order and four other temporary stacks which will be used to sort the plates, we have four type of plates that are acceptable so each temporary stack is responsible for storing a specific type of plates, so what the algorithm is doing is pushing the plates from the initial stack to the temporary stack for example, if the plate is type 1 it will be stored in temporary stack 1, if the plate type is 2 it will be stored in temporary stack 2 etc …, any undesirable plate will be popped out from the initial stack and won’t be stored anywhere, when there will be no more plates in the initial stack we will push back to the stack using while loops the plates type number 1 then number 2 then number and lastly number 4. The big O notation for this algorithm is:*

*n represents the initial number of plates in our stack*

*j represents the number of unwanted plates*

*We have the first while loop which will run for n number of time to store the plates in the temporary stacks and since we are not putting back the unwanted plates in our stack, we subtract it from the total number of plates*

## 2.5 Incorporation of formative feedback

Provide evidence of how you evaluated and acted on the formative feedback you received from your tutors, including the tutor’s name and the timestamp.

|  |  |  |  |
| --- | --- | --- | --- |
| Date and Time | Name of the Tutor | Feedback of the tutor | How the comment is addressed |
|  |  |  |  |
|  |  |  |  |

# 3. Project 3

## 3.1 **Data Structure selection**

This section describes the data structures used to represent the graph using xxxxx.

Provide a discussion on time-space trade-off of the data structures.

# 3.2 Developed Algorithms

# Algorithm1: Influencers in Unweighted Network

***// Methods for graph construction***

***// Write comment of what the next steps do***

*Procedure jjjjjj(IN aa, IN b)*

*declare j,jj,jjj…*

***// Algorithm1***

*CALL jjjjjj(\_,\_) // Prints direct cities of any given input*

*end*

### Time Complexity of Algorithm1: Discuss about the time complexity of Algorithm1 that you have developed.

## 3.2.2 Algorithm2: Influencers in Weighted Network

***// Methods for graph construction***

***// Write comment of what the next steps do***

*Procedure jjjjjj(IN aa, IN b)*

*declare j,jj,jjj…*

***// Algorithm2***

*CALL jjjjjj(\_,\_) // Prints direct cities of any given input*

*end*

### Time Complexity of Algorithm2: Discuss about the time complexity of Algorithm2 that you have developed.

# Implementation

## 3.3.1 Implementation details

Provide a paragraph explaining the implementation details

## 3.3.2 Publishing your project

Publish your project code in GitHub, make [JayaTangirala (github.com)](https://github.com/JayaTangirala) as collaborator and place the GitHub link of the project here for ready reference.

## 3.3.3 Evaluation

An evaluation of the implementation's scalability to the road network available at [euroroad | Road Networks | Network Data Repository (networkrepository.com)](https://networkrepository.com/road-euroroad.php) where Edges represent roads and nodes are intersections where roads meet.

# Reflection

## 3.4.1 Video

Make a video presentation explaining the working of the developed program in C# for not more than 10 minutes.

## 3.4.2 Feedback

Provide evidence of how you evaluated and acted on the formative feedback you received from your tutors, including the tutor’s name and the timestamp.

|  |  |  |  |
| --- | --- | --- | --- |
| Date and Time | Name of the Tutor | Feedback of the tutor | How the comment is addressed |
|  |  |  |  |
|  |  |  |  |

# 4. Project 4

## 4.1 Introduction to Bin Packing Problem

*This section requires you to document your understanding of the subject based on your findings and discoveries from desk research. You must cite your references in this section.*

## 4.2 Random Mutation Hill Climbing

*This section requires you to discuss your understanding of the method. You may conduct desk research for this section and must cite them.*

## 4.3 Algorithm Design

*In this section, you must explicitly explain the designed algorithm for solving the problem using Random Mutation Hill Climbing. You may add your own sub-sections under this section, such as:*

### 4.3.1 Solution Representation

Discuss how you are going to model or represent the problem’s solution. Use diagrams and explain them if necessary.

### 4.3.2 Fitness Function Analysis

This section should have discussed several (at least 2) proposed fitness functions for the problem. Provide evidence of which fitness function is the best/suitable for your work and explain it.

### 4.3.2 Small Change Strategy

Explain your strategy for small change. Use diagrams and elaborate on them.

### 4.3.4 Data Structure

Discuss the data structure you will use for the work. This should cover the data structure for the solution representation, the dataset itself, and the main program of the experiment.

### 4.3.5 Algorithm Pseudocode

Write the pseudocode for the algorithm you use for the work.

### 4.3.6 Experiment Strategy

Outline your experiment strategy for the work. This can be how many times you will run for the experiment. How many iterations will you use in the experiment, and how will you present the results? Will it be getting the average or only the best run result?

## 4.4 Results

*Tabulate the results and plot graphs in this section. Discuss them. You may add sub-sections in this section based on your content.*

## 4.5 Testing

*Provide evidence for testing, e.g., unit and system testing.*

## 4.6 Software and its Presentation, including testing (and video link)

*All you need is to attach a link for the video recording. Do not forget to ensure the video recording is accessible by us.* **If we cannot access your video recording, you will get zero marks for that section. Therefore, please ensure you test your video link by asking someone else to access your video recording.**

## 4.7 Incorporation of formative feedback

Provide evidence of how you evaluated and acted on the formative feedback you received from your tutors, including the tutor’s name and the timestamp.

|  |  |  |  |
| --- | --- | --- | --- |
| Date and Time | Name of the Tutor | Feedback of the tutor | How the comment is addressed |
|  |  |  |  |
|  |  |  |  |