

# Algorithms Simulator and Teaching Tool

---

# PROJECT CONTENTS

<b>PROJECT CONTENTS</b>	<b>2</b>
<b>ANALYSIS</b>	<b>7</b>
PROJECT DEFINITION AND PROBLEM RECOGNITION	7
RESEARCH	8
THE ALGORITHMS	8
BUBBLE SORT	8
PRIM'S ALGORITHM	8
DIJKSTRA'S ALGORITHM	9
THE SIMPLEX ALGORITHM	10
BIG O VALUES FOR THE ALGORITHMS	11
THE IMPORTANCE OF VISUALISATION IN LEARNING	12
STAKEHOLDERS	13
QUESTIONNAIRE	14
COMPUTATIONAL THINKING & METHODS	14
THINKING ABSTRACTLY	14
THINKING AHEAD	16
THINKING PROCEDURALLY	17
THINKING LOGICALLY	17
THINKING CONCURRENTLY	17
COMPUTATIONAL METHODS	17
BACKTRACKING	18
DATA MINING	18
HEURISTICS	18
PERFORMANCE MODELLING	18
PIPELINING	18
VISUALISATION	19
EXISTING SYSTEMS RESEARCH	19
UNIVERSITY OF SAN FRANCISCO'S DATA STRUCTURE VISUALISATIONS WEBSITE	19
GILP	22
DIJKSTRA'S ALGORITHM VISUALIZER - danvbyjan.com WEBSITE	24
PROPOSED ESSENTIAL FEATURES	24
INITIAL LOGIN PAGE	25
REGISTRATION PAGE	25
HOME PAGE	25
BUBBLE SORT VISUALISATION PAGE	25
PRIM'S ALGORITHM VISUALISATION PAGE	26

DIJKSTRA'S ALGORITHM VISUALISATION PAGE	27
SIMPLEX ALGORITHM VISUALISATION PAGE	28
QUIZ PAGE	28
LIMITATIONS AND FEASIBILITY	29
ECONOMIC	29
TIME	29
TECHNICAL	30
POLITICAL	30
LEGAL	30
HARDWARE AND SOFTWARE REQUIREMENTS	31
HARDWARE	31
SOFTWARE	31
SUCCESS CRITERIA	32
[1] INITIAL LOGIN PAGE	32
[2] REGISTRATION PAGE	33
[3] HOME PAGE	34
[4] BUBBLE SORT VISUALISATION PAGE	35
[5] PRIM'S ALGORITHM VISUALISATION PAGE	36
[6] DIJKSTRA'S ALGORITHM VISUALISATION PAGE	37
[7] SIMPLEX ALGORITHM VISUALISATION PAGE	39
[8] QUIZ PAGE	40
[9] MENU	42
[10] GRAPH INPUT	43
<b>DESIGN</b>	<b>44</b>
DECOMPOSITION OF THE PROBLEM	44
TOP DOWN MODEL	44
DECOMPOSITION TABLE	45
USE CASE DIAGRAM	49
USER INTERFACE APPEARANCE	50
DATABASE	50
ENTITY RELATIONSHIP DIAGRAM	52
SQL QUERIES	52
INITIAL LOGIN PAGE DESIGN	56
DECOMPOSITION	56
GUI	57
PSEUDOCODE	58
VARIABLES AND VALIDATION TABLE	59
DATA STRUCTURES TABLE	60
WHITE BOX TEST DATA	62
REGISTRATION PAGE DESIGN	63

---

DECOMPOSITION	63
GUI	66
PSEUDOCODE	67
VARIABLES AND VALIDATION TABLE	69
DATA STRUCTURES TABLE	73
WHITE BOX TEST DATA	75
HOME PAGE DESIGN	77
DECOMPOSITION	77
GUI	78
PSEUDOCODE	79
VARIABLES AND VALIDATION TABLE	81
DATA STRUCTURES TABLE	82
WHITE BOX TEST DATA	84
BUBBLE SORT VISUALISATION PAGE DESIGN	86
DECOMPOSITION	86
GUI	88
PSEUDOCODE	89
VARIABLES AND VALIDATION TABLE	91
DATA STRUCTURES TABLE	93
WHITE BOX TEST DATA	94
PRIM'S ALGORITHM VISUALISATION PAGE DESIGN	95
DECOMPOSITION	95
GUI	97
PSEUDOCODE	98
VARIABLES AND VALIDATION TABLE	100
DATA STRUCTURES TABLE	102
WHITE BOX TEST DATA	103
DIJKSTRA'S ALGORITHM VISUALISATION PAGE DESIGN	104
DECOMPOSITION	104
GUI	107
PSEUDOCODE	108
VARIABLES AND VALIDATION TABLE	110
DATA STRUCTURES TABLE	112
WHITE BOX TEST DATA	114
SIMPLEX ALGORITHM VISUALISATION PAGE DESIGN	116
DECOMPOSITION	116
GUI	118
PSEUDOCODE	119
VARIABLES AND VALIDATION TABLE	122
DATA STRUCTURES TABLE	125

---

WHITE BOX TEST DATA	126
QUIZ PAGE DESIGN	127
DECOMPOSITION	127
GUI	130
PSEUDOCODE	133
VARIABLES AND VALIDATION TABLE	136
DATA STRUCTURES TABLE	140
WHITE BOX TEST DATA	143
MENU DESIGN	147
DECOMPOSITION	147
GUI	148
PSEUDOCODE	148
VARIABLES AND VALIDATION TABLE	151
DATA STRUCTURES TABLE	152
WHITE BOX TEST DATA	153
GRAPH INPUT DESIGN	155
DECOMPOSITION	155
GUI	156
PSEUDOCODE	157
VARIABLES AND VALIDATION TABLE	159
DATA STRUCTURES TABLE	160
WHITE BOX TEST DATA	162
BLACK BOX TEST DATA	164
SOLUTION APPROACH	172
<b>DEVELOPMENT</b>	<b>173</b>
PROTOTYPE 1 - INITIAL LOGIN AND REGISTRATION PAGES	173
ITERATION 1 - SETTING UP THE DATABASE AND USER INTERFACE	173
ITERATION 1 - DEVELOPER REVIEW	177
ITERATION 2 - LOGIN AND REGISTRATION FUNCTIONALITY	177
ITERATION 2 - DEVELOPER REVIEW	206
ITERATION 3 - REFINING THE GUI	206
ITERATION 3 - DEVELOPER REVIEW	209
PROTOTYPE 1 - STAKEHOLDER REVIEW	209
PROTOTYPE 1 - SUCCESS CRITERIA REVIEW	210
PROTOTYPE 2 - HOME AND BUBBLE SORT VISUALISATION PAGES AND MENU	217
ITERATION 1 - HOME PAGE AND SETTING UP VISUALISATION AND QUIZ PAGES	217
ITERATION 1 - DEVELOPER REVIEW	243
ITERATION 2 - BUBBLE SORT FUNCTIONALITY	243
ITERATION 2 - DEVELOPER REVIEW	258
ITERATION 3 - REFINING THE BUBBLE SORT GUI AND DEVELOPING THE MENU	258

---

ITERATION 3 - DEVELOPER REVIEW	283
PROTOTYPE 2 - STAKEHOLDER REVIEW	283
PROTOTYPE 2 - SUCCESS CRITERIA REVIEW	284
PROTOTYPE 3 - PRIM'S AND DIJKSTRA'S VISUALISATION PAGES AND GRAPH INPUT	294
ITERATION 1 - FUNCTIONALITY OF GRAPH INPUT SECTION	296
ITERATION 1 - DEVELOPER REVIEW	317
ITERATION 2 - PRIM'S VISUALISATION PAGE FUNCTIONALITY AND GUI	317
ITERATION 2 - DEVELOPER REVIEW	356
ITERATION 3 - DIJKSTRA'S VISUALISATION PAGE FUNCTIONALITY AND GUI	356
ITERATION 3 - DEVELOPER REVIEW	401
PROTOTYPE 3 - STAKEHOLDER REVIEW	401
PROTOTYPE 3 - SUCCESS CRITERIA REVIEW	403
PROTOTYPE 4 - SIMPLEX VISUALISATION AND QUIZ PAGES	419
ITERATION 1 - SIMPLEX VISUALISATION PAGE FUNCTIONALITY AND GUI	419
ITERATION 1 - DEVELOPER REVIEW	454
ITERATION 2 - QUIZ PAGE: BUBBLE SORT AND SIMPLEX	455
ITERATION 2 - DEVELOPER REVIEW	492
ITERATION 3 - QUIZ PAGE: PRIM'S AND DIJKSTRA'S	493
ITERATION 3 - DEVELOPER REVIEW	524
PROTOTYPE 4 - STAKEHOLDER REVIEW	524
PROTOTYPE 4 - SUCCESS CRITERIA REVIEW	525
<b>EVALUATION</b>	<b>543</b>
TESTING	543
FUNCTION	543
ROBUSTNESS	552
USABILITY	581
USABILITY FEATURES	581
STAKEHOLDER FEEDBACK	585
REVIEW AND EVALUATION	586
LIMITATIONS AND MAINTENANCE	587
POST DEVELOPMENT	588
<b>FINAL CODE</b>	<b>589</b>
<b>REFERENCES</b>	<b>648</b>

# ANALYSIS

## PROJECT DEFINITION AND PROBLEM RECOGNITION

Many complex algorithms are studied as part of A-Level Computer Science and in the A-Level Further Maths Decision Maths modules. For my project, I am planning on creating a tool that teachers of Computer Science/Further Maths and students studying the module can use to visualise and go through algorithms step-by-step with the solutions laid out comprehensively. I am planning on implementing four algorithms in my project, with solutions being represented in text based and graphical methods. These algorithms are:

- The Bubble Sort algorithm for sorting a list of data
- Prim's algorithm for finding the Minimum Spanning Tree of a connected graph
- Dijkstra's algorithm for finding the shortest path between two nodes in a weighted graph
- The Simplex algorithm for finding the optimal solution to an optimisation problem using the simplex tableau

Users of this software will be able to input their own list of data for Bubble sort, input a graph by dragging and dropping nodes and connecting these nodes for Prim's and Dijkstra's and input inequalities and equations for the Simplex algorithm.

My intention for this project is to assist students in understanding how these algorithms work both visually and through text-based solutions and allow them to test themselves through a quiz tool where the user can choose a type of algorithm, solve the question themselves and then input the answer, obtaining feedback from the system. This system will also allow teachers to explain the algorithms more easily as students have graphical representations to follow through, aiding their understanding and the teachers' explanations.

I am aiming for this software to have a user login system, graphical and text-based implementations of each algorithm and a quiz tool for each algorithm, which will write the data into the database and can be used to suggest which algorithm students should practise.

Next, I will research the four algorithms in depth to fully understand what they do and how the text-based solutions should be set out. I will also discuss with teachers of Computer Science/Decision Maths and students studying these subjects to find out what I should aim to achieve with creating this learning tool.

## RESEARCH

### THE ALGORITHMS

#### BUBBLE SORT

Bubble sort, which is studied in both Computer Science A-Level and Further Maths A-Level Decision Maths modules, is used to sort a dataset by comparing each item with the item adjacent to it, swapping them if they are in the incorrect order. The Bubble Sort algorithm is as follows:

1. Start at the first item in the list.
2. Compare the current item with the next item.
3. If the two items are in the wrong position, swap them.
4. Move up one item to the next time in the list.
5. Repeat from step 3 until all the unsorted items have been compared.
6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

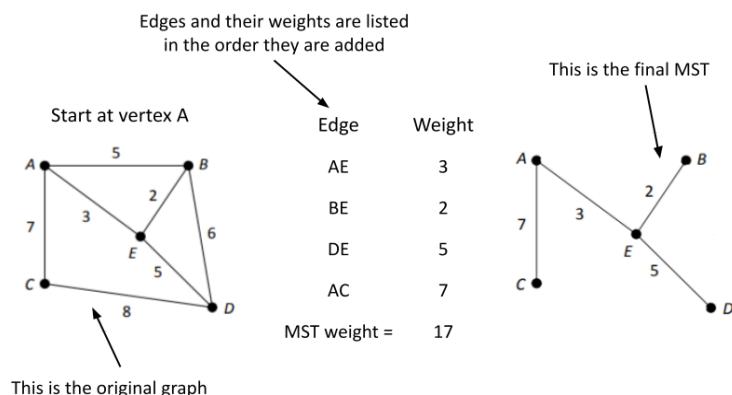
This sorting algorithm is not very efficient with a best case time complexity of  $O(n)$  and a worst case time complexity of  $O(n^2)$ . Therefore, I will limit the size of the data sets that users can input to about 10 items.

#### PRIM'S ALGORITHM

Prim's algorithm, which is one of the algorithms only studied in Further Maths A-Level Decision Maths modules, is designed to find the minimum spanning tree<sup>1</sup> of a graph. A spanning tree is a subgraph which includes all the vertices of the graph and is also a tree, which is a connected graph with no cycles (loops). A minimum spanning tree (MST) is a spanning tree such that the total length of its edges is as small as possible. There are two methods for Prim's algorithm. The first method, which uses a graph, is as follows:

1. Choose any vertex to start the tree.
2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
3. Repeat step 2 until all the vertices are connected to the tree.

Here is an annotated example of the solution to this first method of Prim's algorithm:



The second method, which uses a distance matrix (a matrix that shows the distances between vertices), is as follows:

1. Choose a starting vertex and delete the row corresponding to that vertex.
2. Label the column corresponding to the start vertex with 1 and ring the smallest undeleted entry in that column. If there is a choice, pick any.
3. Delete the row corresponding to the entry just ringed.
4. Label the column corresponding to the entry just ringed with the next label number.
5. Ring the lowest undeleted entry in all labelled columns.
6. Repeat steps 3 - 5 until all rows are deleted. The ringed entries represent the edges in the MST.

Here is an annotated example of Prim's algorithm using a distance matrix. It uses the same graph as above:

This is the distance matrix version of the graph used in the previous example for Prim's algorithm. Start at vertex A as done before.

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>A</b>	–	5	7	–	3
<b>B</b>	5	–	–	6	2
<b>C</b>	7	–	–	8	–
<b>D</b>	–	6	8	–	5
<b>E</b>	3	2	–	5	–

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>A</b>	–	5	7	–	3
<b>B</b>	5	–	–	6	2
<b>C</b>	7	–	–	8	–
<b>D</b>	–	6	8	–	5
<b>E</b>	3	2	–	5	–

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>A</b>	–	5	7	–	3
<b>B</b>	5	–	–	6	2
<b>C</b>	7	–	–	8	–
<b>D</b>	–	6	8	–	5
<b>E</b>	3	2	–	5	–

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>A</b>	–	5	7	–	3
<b>B</b>	5	–	–	6	2
<b>C</b>	7	–	–	8	–
<b>D</b>	–	6	8	–	5
<b>E</b>	3	2	–	5	–

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
<b>A</b>	–	5	7	–	3
<b>B</b>	5	–	–	6	2
<b>C</b>	7	–	–	8	–
<b>D</b>	–	6	8	–	5
<b>E</b>	3	2	–	5	–

1. Delete row A as it is the starting vertex and ring the smallest edge.

2. The smallest edge from step 1 is labelled.

This is what the distance matrix will look like at the end. All vertices have been labelled. The same MST as before is created.

As stated when considering the needs of my stakeholders, a graphical representation is vital to students understanding the algorithms. Therefore, the first method for Prim's algorithm could be better for students to understand the algorithm because the second method with the distance matrix can look more complicated as there are more numbers. However, in terms of coding the solution, the distance matrix seems to lend itself more to data structures like 2D arrays. Overall, considering that students must be able to solve problems using both methods, I may try to implement both methods in my project as it is designed to help students understand the algorithms and some may understand better looking at one method than the other.

## DIJKSTRA'S ALGORITHM

Dijkstra's algorithm, which is studied in both Computer Science A-Level and Further Maths A-Level Decision Maths modules, is designed to find the shortest path between two vertices for a weighted graph. A path is a finite set of edges where no edge appears more than once and a weighted graph is a graph where edges have weights (or distances) associated with them. Dijkstra's algorithm is as follows:

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
3. Calculate the distance from the start for each of the unvisited connected nodes.
4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
5. Then set the current node as visited.
6. Repeat steps 2 to 5 until all nodes are set to visited.

Then to find the shortest path through backtracking:

7. Start from the goal node
8. Add the ‘previous node’ to the start of a list
9. Repeat step 7 until the start node is reached.

Here is an annotated example of the table that is used for Dijkstra’s algorithm in Computer Science A-Level:

The table shows the state of nodes A, B, C, D, and E. Node A is the start node. The 'Visited' column indicates whether a node has been visited (Yes) or not (No). The 'Previous node' column is empty. An annotation 'This is the start node in this case' points to node A. Another annotation 'This will be changed to Yes once a node has been visited' points to the 'Visited' column for node A.

Node	Distance from start	Visited	Previous node
A	0	No	
B	$\infty$	No	
C	$\infty$	No	
D	$\infty$	No	
E	$\infty$	No	

Dijkstra’s algorithm is represented differently in some Further Maths A-Level Decision Maths modules however I have decided to use the implementation used in Computer Science A-Level.

## THE SIMPLEX ALGORITHM

The Simplex algorithm, which is only studied in Further Maths A-Level Decision maths modules, is designed to find the optimal solution to a set of inequalities. Below is an annotated example of a simplex tableau:

The simplex tableau is a grid of numbers representing a linear programming problem. Annotations explain various parts of the tableau:

- The basic variable column contains the variables that are not zero for the optimal solution.
- Decision variables ( $x, y$  or  $z$ ) are in the next column.
- Slack variables (these turn the inequalities into equalities and are given the letters  $r/s/t$ ) are in the next column.
- These are the solutions to the constraints.
- Constraints are labeled  $r, s$  and  $P$ .
- Objective function is labeled  $P$ .
- Pivot is indicated by an arrow pointing to the  $-\frac{1}{3}$  value in the  $P$  row.
- $-\frac{1}{3}$  is the most negative value in the objective row so the pivot will be in this column.
- $3$  is the smallest which is why  $x$  will replace  $s$  as a basic variable.
- $6 \div \frac{2}{3} = 9$  and  $4 \div \frac{4}{3} = 3$  are calculations for the  $\theta$  values.

basic variable	$x$	$y$	$r$	$s$	value	$\theta$ values
$r$	$\frac{2}{3}$	1	$\frac{1}{3}$	0	6	$6 \div \frac{2}{3} = 9$
$s$	$\frac{4}{3}$	0	$-\frac{1}{3}$	1	4	$4 \div \frac{4}{3} = 3$
$P$	$-\frac{1}{3}$	0	$\frac{4}{3}$	0	24	

The algorithm is as follows:

1. Turn the inequalities into equalities by adding a slack variable to each inequality. Rearrange the objective function ( $P$ ) to be equal to 0. Input the coefficients of all the variables and the solutions to the constraints and objective function into the simplex tableau.
2. Choose the most negative value in the objective row of the simplex tableau to become non-zero (the basis).
3. Work out the  $\theta$ -value for this variable for each row. This is done by dividing the value for that row by the basis value in that row. Select the smallest value that is not less than zero. Where the row and column meet becomes the pivot.
4. Divide the pivot row by the value of the pivot, making the pivot equal to 1.
5. Add or subtract multiples of the pivot row from all other rows so the basis variable is 0.
6. Repeat from step 1 until the objective row contains no negative entries.
7. To read off the optimal solution from the tableau, the basic variables are equal to the values in the value column. The equation of the objective row will be written as  $P = \text{optimal value} - \text{sum of the products of the variables with positive values in the objective row and their respective coefficients}$ . The optimal values of the variables in the objective function is 0 in order to maximise  $P$ .

The above method is for maximising the objective function. It is also possible to minimise it if you maximise the negative of the objective function. Then, after following the above steps, the minimum objective value will be the negative of the objective value in the tableau.

### BIG O VALUES FOR THE ALGORITHMS

For Bubble Sort,  $n$  is the size of the dataset. For Prim's algorithm and Dijkstra's algorithm,  $V$  is the number of vertices and  $E$  is the number of edges. For the Simplex algorithm,  $m$  is the number of constraints and  $n$  is the number of variables.

ALGORITHM	TIME COMPLEXITY	SPACE COMPLEXITY	EXPLANATION
Bubble Sort	Best: $O(n)$ Average: $O(n^2)$ Worst: $O(n^2)$	$O(1)$	For time complexity, in the best case the data set is already sorted so only one pass is required to check every item. As the algorithm has a nested loop, this makes the time complexity polynomial as the time taken to execute both loops increases with the size of the data set. No extra memory is required, meaning it is an in-place sort, so space complexity is constant.
Prim's	Best: $O(E \log V)$ Average: $O((V+E) \log V)$ Worst: $O((V+E) \log V)$	$O(E + V)$	The best case scenario is when the graph is already a MST but if it isn't, the time complexity depends on the number of edges. Prim's algorithm uses a

			priority queue/min-heap and a visited set. The space complexities of these are then combined to get the overall space complexity. <sup>2</sup>
Dijkstra's	Best: $O(E + V \log V)$ Average: $O(E \log V)$ Worst: $O(V^2)$	$O(V)$ to $O(E + V)$	Dijkstra's algorithm has a FOR loop nested in a WHILE loop. This means the time complexity is at worst case polynomial but depending on how optimised the graph is (e.g. very sparse), the time complexity can be reduced. The space complexity of Dijkstra's algorithm depends on the data structures used. <sup>3</sup>
Simplex	Best: $O(m)$ Average: $O(mn)$ Worst: $O(2^n)$	$O(mn)$	The best case scenario is that the algorithm converges quickly to the optimal solution and in the average case, it may take some pivot steps to reach the optimal solution. In the worst case, the algorithm may oscillate between vertices of the feasible region without making quick progress towards the optimal vertex. For the space complexity, a $(m+1)$ by $(n+1)$ tableau is usually used so the space required is proportional to $mn$ .

## THE IMPORTANCE OF VISUALISATION IN LEARNING

Visualisation is the possibility of perceiving and processing information in a graphical form. This is important in education because it focuses the concentration of students, reduces the burden of teachers, deepens and extends the use of spoken word and supports the remembering of the contents of the visualisation. Colours and forms of the basic elements of the visualisation are important for highlighting information, illustrating the context and referencing the connections. Page layout, like symmetry or rhythm, are key to defining the structure and logic to the visualisation.<sup>4</sup>

## STAKEHOLDERS

My first group of stakeholders is students. A student studying Decision Maths modules in A-Level Further Maths could use my project to step through and understand problems covered in class or

exemplar problems already set up in the software, which they could also use to test themselves for revision and then check by following through the solution. They could also use this tool to check their answers to homework problems by inputting their own data set, graph or inequalities for these algorithms and find where exactly they went wrong if their answer does not match the solution given in homework answers because of the ability to go through the steps one-at-a-time to find the exact point when they made the error. As all four of these algorithms are studied in most A-Level Further Maths Decision Maths modules, my project is very suitable for this target audience because they will be able to use the full functionality of the system to support their learning and revision. Another member of this project target audience is Computer Science students because they study Bubble Sort and Dijkstra's algorithm as part of their course. Whilst Computer Science students will likely not make use of the full functionality of the system as they do not study Prim's algorithm or the Simplex algorithm, this software would still be beneficial to supporting their learning and revision.

My second group of stakeholders is Computer Science and Maths teachers. A teacher could use my project during lessons to support students struggling to understand the algorithms that the software will support. The teacher can show the software on the board and the visual aspect of the software will support the students' understanding as the teacher would not need to rely on a theoretical understanding of the algorithms being sufficient for the students to apply the algorithms in questions. In addition, this tool could be used by teachers if they had to teach these algorithms in virtual learning environments as they can easily go through the steps of the algorithm and go back to steps that students struggled to understand without needing to write down or erase anything, which is a struggle when teaching virtually.

A-Level students, aged 16-19, are my target demographic because of the increasing numbers of students choosing to study Computer Science and Further Maths at A-Level. From JCQ's report<sup>5</sup> on 2024's A-Level results, students studying Computer Science increased from 11124 in 2019 to 20370 in 2024, whilst students studying Further Maths increased from 14572 in 2019 to 18082. Below are the statistics for number of students and percentages for each grade (from A\* to U) for both subjects:

26	A Level	Computing	2024	20370	5.8	24.0	45.9	67.1	84.7	95.1	100
27	A Level	Computing	2023	18306	5.3	22.2	44.4	65.8	83.4	94.8	100
28	A Level	Computing	2019	11124	3.4	17.9	40.3	63.3	83.4	94.7	100
83	A Level	Mathematics (Further)	2024	18082	28.7	58.4	79.1	89.8	95.3	98.2	100
84	A Level	Mathematics (Further)	2023	15080	29.2	58.5	77.0	88.5	94.7	98.0	100
85	A Level	Mathematics (Further)	2019	14527	24.7	53.5	73.7	86.6	94.3	97.9	100

Due to the increasing numbers<sup>6</sup> of students and the stagnant or decreasing numbers of Maths and Computer Science teachers (which is being seen in Scotland<sup>7</sup>), this tool would be very useful to these teachers as it is important to help students do as well as they can in these exams by aiding their comprehension and practice of the complex algorithms in Decision Maths and Computer Science, which many may find challenging.

## QUESTIONNAIRE

I interviewed one of my stakeholders - an A-Level Further Maths D1 student - to find out the basic requirements from the user about what they would want to see in a visualisation system for the algorithms they study.

**Developer:** Was it difficult to understand the algorithms you have studied in D1?

**Stakeholder:** Not really because our teacher explained it well but sometimes I forget how they work, which makes it difficult to do questions.

**Developer:** If you were to have a visualisation tool that simulates each algorithm, do you think this would be/have been better for your learning?

**Stakeholder:** Yes because the plain English algorithm can be a bit unclear. With graphs and pictures, it would be clearer and more organised.

**Developer:** What features would you like to see in a visualisation for Bubble Sort?

**Stakeholder:** Showing how the algorithm works step-by-step could be useful and different colours for the values being swapped would be useful. I have found that textbooks aren't very clear about this so a system like this could be more helpful.

**Developer:** For Prim's algorithm (executed graphically), what would you like to see in a visualisation to help your understanding?

**Stakeholder:** Colour coding would be helpful to see what exactly is going on.

**Developer:** For Dijkstra's algorithm, what features would you want in a visualisation?

**Stakeholder:** Definitely step-by-step animations would help me understand Dijkstra's better.

**Developer:** And what would you want to see in a visualisation of the Simplex algorithm?

**Stakeholder:** I'm not really sure but I find the Simplex tableau confusing and unclear sometimes so maybe just drawing the lines on a graph like in linear programming.

**Developer:** If you could input your own data, graph or constraints, how would you use this system?

**Stakeholder:** Checking answers!

**Developer:** And finally, is the aesthetic important to the system?

**Stakeholder:** Not necessarily the colours but I think it needs to look organised.

From this interview, it seems that the main features that I should aim to implement in this system are colour-coding in the visualisations, animations in the visualisation of Dijkstra's algorithm, x-y axes for the Simplex algorithm and an organised GUI.

## COMPUTATIONAL THINKING & METHODS

### THINKING ABSTRACTLY

Abstraction is the process of separating ideas from reality by hiding unnecessary details and highlighting details that are important in the context. It is needed to improve the clarity of what is being conveyed. One advantage of abstraction is that it enables for more efficient design during software development, meaning that it reduces the time needed to be spent on the project and it saves system memory. However, abstraction can detract from the appeal of the program as it may make it too simplistic.

TYPE OF ABSTRACTION	JUSTIFICATION
---------------------	---------------

Representational	This is taking out unnecessary details from a problem and only displaying the most essential information. As my project focuses on the exam-style of questions involved in these A-Levels, I do not have the context of a real-world system to remove. However, the questions that are a part of these A-Levels (as they have to be a part of an exam) will be removed, meaning that the start and end nodes will just be highlighted in a different colour for Dijkstra's algorithm, just the data set will be given for Bubble Sort, just the graph and distance matrix will be given for Prim's algorithm, and there will be no formulating of linear programs required for the Simplex algorithm. Instead, I will provide input fields for Bubble Sort, Prim's algorithm and the Simplex algorithm. The advantage of using this type of abstraction is that it makes the problem much easier to visualise and solve as understanding language in the formulation of linear programs for Simplex is not required. However, this may oversimplify the problem as this software is designed to help students learn and revise how to answer questions in exams, which have context and written portions involved.
Generalisation	This type of abstraction is where you group things in terms of a set of common characteristics. Some common features to multiple algorithms that I will include in my project can have general subroutines or functions that can be called multiple times. For example, both Prim's and Dijkstra's algorithms require a graph input system that can be called when one of these algorithms is being displayed. The login system and quiz data have a common feature of extracting and writing data to a database so functions/subroutines or OOP could be used for these features. The advantage of this is that it can make coding faster as code can be reused. However, if the code is overgeneralised, excessive programming time may be needed to make it usable for the different purposes.
Procedural	This is where code is simplified so that it can be used for many different purposes. Procedural abstraction can be used when I am writing the pseudocode for general outlines of what each part of the decomposed problem will do, such as instead of writing the code for finding the minimum weight edge for every time an edge needs to be added to the tree (which is equal to the number of nodes). The advantage of this is that code can be reused, which reduces coding time. However, you aren't required to understand all the code you are using with procedural abstraction, which may make debugging the code difficult.
Functional	This is where functions are used to abstract a process, hiding the complexity and focusing on the values returned. I can use functional abstraction for breaking down the main portion of my code, such as different buttons leading to different pages or functions calculating and returning the objective row values, finding the pivot, calculating the θ-values and performing the row operations. Having functions returning these values allows me to focus on the logic of how to combine them

	together to carry out the Simplex algorithm. The advantage of this is that you don't need to understand the code written in the functions but this can make debugging more difficult.
Data	This is where you separate how a compound data object is used from the details on how it is constructed. One of the complex data structures that I will be using in this project is a graph, which is an abstraction of an adjacency matrix, which is an abstraction of a dictionary. With the use of graphs in Prim's and Dijkstra's algorithms, I am using data abstraction to separate understanding and designing how this complex data structure is going to be used in my project from the details of how it will be implemented in my project. The advantage of this is that you don't need to write code for similar data structures, which saves time. A disadvantage is that there may be limitations with the use of other data structures.
Problem Reduction	This is the process of generalising a problem, removing unnecessary parts until the problem is one that has already been solved, which makes coding simpler but more difficult to debug if you don't understand the code that has been reused from solutions to these simpler problems. Due to the nature of my project idea, all of the algorithms I am planning on implementing already have solutions and ways of visually implementing and coding them.
Information Hiding	This is where systems hide their complexity behind an interface. The system will hide the complexity of the coded solutions to these algorithms, as well as how they are graphically represented, and the login, quiz and data collection features from the user behind the interface that the user can interact with. The advantage of this is that it improves the usability of the system. A disadvantage is that additional code is required to create the interface, which increases development time.

## THINKING AHEAD

Thinking ahead is a way to maximise code efficiency and minimise errors by considering the inputs, outputs and preconditions as well as the reusable components involved in a system. The advantages of documenting these are that it makes program components reusable as the necessary checks before executing a function are already documented. Additionally, it makes the program easier to debug. Some of the inputs my system will require include usernames and passwords, nodes and edges through drag-and-drop and numerical values for the inequalities in Simplex or the list of values for Bubble Sort. For outputs, the different algorithms will have their own specific graphical outputs according to their specifications, which are outlined in the research section. For the login/registration system, a 'Login/Registration successful/unsuccessful' message will be output. The main precondition for my system is that all of the algorithms have an input, e.g a dataset, to use.

## THINKING PROCEDURALLY

Thinking procedurally is concerned with decomposing the problem - breaking it down into smaller, more manageable chunks. This involves identifying the components of a problem and the steps to solve a problem, along with identifying any sub-procedures. The advantage of this is that it makes designing and then developing a solution easier as each chunk - module, function, procedure or method - can be considered separately. In the design and development of my system, I will use decomposition through stepwise refinement in a top down structure diagram when beginning the design process. The main way I will break the system down is into the registration and login systems, the algorithm visualisation systems, which includes the four different algorithms I am planning on implementing, and the quiz system. Each of these main components will be further subdivided until each sub-problem can be implemented as a subroutine.

## THINKING LOGICALLY

Thinking logically is concerned with the points in a solution where a decision needs to be made. It combines logical operators and data sets to produce logical expressions that can be used to identify possible outcomes. In my system, thinking logically can be applied to the login system as validation is required to check the user's inputs, such as their username and password because this involves Boolean conditions. This would be beneficial to my system because validation would help prevent the system from being vulnerable for cyber-attacks.

## THINKING CONCURRENTLY

Concurrency is where an application is making progress on more than one task at the same time. Thinking concurrently is a skill which involves working out which parts of a program can be developed to be processed at the same time and which parts are dependent on others. The advantages of this are that it allows users to interact with applications while other tasks are running in the background, which is known as reactive programming, and that long-running tasks won't delay short-running tasks. However, concurrent tasks could corrupt the consistent state of a program, causing a deadlock, which is where tasks suspend indefinitely, and consume large amounts of resources as scheduling and synchronisation are required for threads to be processed on different cores. This system is designed to have visualisations of the different algorithms being implemented. Therefore, use of a graphical processing unit (GPU) is required. GPUs are made up of cores that are very efficient at processing large amounts of visual data in parallel. In addition, GPUs are good for processing complex mathematical operations, like calculations using floating point, which may be used in the Simplex algorithm depending on the numbers input for the constraints.

## COMPUTATIONAL METHODS

A computational approach will be ideal for this system because of the repetitive nature of some of these algorithms, therefore automating them to aid users with understanding them as the computer can do the calculations, therefore making these algorithms computable. Additionally, the system is intended to be a visual representation, which a computational approach is beneficial for.

---

## BACKTRACKING

Backtracking is the process of incrementally building towards a solution. The advantages of this are that it guarantees a solution if one exists, it is efficient and is easy to implement. However, backtracking is slow for complex problems and it may not find the best solution. Dijkstra's algorithm is an example of how backtracking can be used in this system. Once all of the nodes have been visited, the program should backtrack from the end node to the start node in order to find the shortest path.

## DATA MINING

Data mining is the concept of analysing vast amounts of data gathered from a variety of sources in order to discover new information or trends. This is not applicable to my system.

## HEURISTICS

This is an approach to problem solving that allows us to make use of our experiences to find a 'good enough' solution to a problem. The advantage of this is that it is usually possible to find a solution close to the best solution available but it will not guarantee that you will find the 'best' solution. As this system is designed as a revision and learning support tool, limits will be set on the inputs for the data sets or graphs entered, which ensures that the algorithms being used are tractable - can be run quickly enough to be practical and useful. Therefore, heuristics will not be applicable to the system.

## PERFORMANCE MODELLING

This is the process of approximating how well models perform using mathematics and simulations. The advantage of this is that you are able to predict problems and act on them before the problems actually occur in the real world. However, if the rules that made up the model are wrong then it will produce incorrect results. As this system is not designed to be a multiplayer or multi user system, performance modelling for stress testing is not going to be applicable. Additionally, as this system will likely never be released to the public, performance modelling on the login/registration system or the algorithm input system will likely not be required apart from for testing with erroneous data.

## PIPELINING

Pipelining is a process that allows for projects to be delivered faster by dividing modules into individual, independent tasks. The advantage of pipelining is that it increases the number of instructions completed per clock cycle. However, pipelining increases the complexity of the code, instruction latency increases and if the code is written to have lots of jumps and decisions, the pipeline would need to be flushed multiple times, which is inefficient. As I am planning on writing the code in Python, the handling of multiple threads is not a concern when writing the code. In addition, libraries like Matplotlib and NumPy, which I will likely be using for this system, make use of parallel processing more than pipelining.

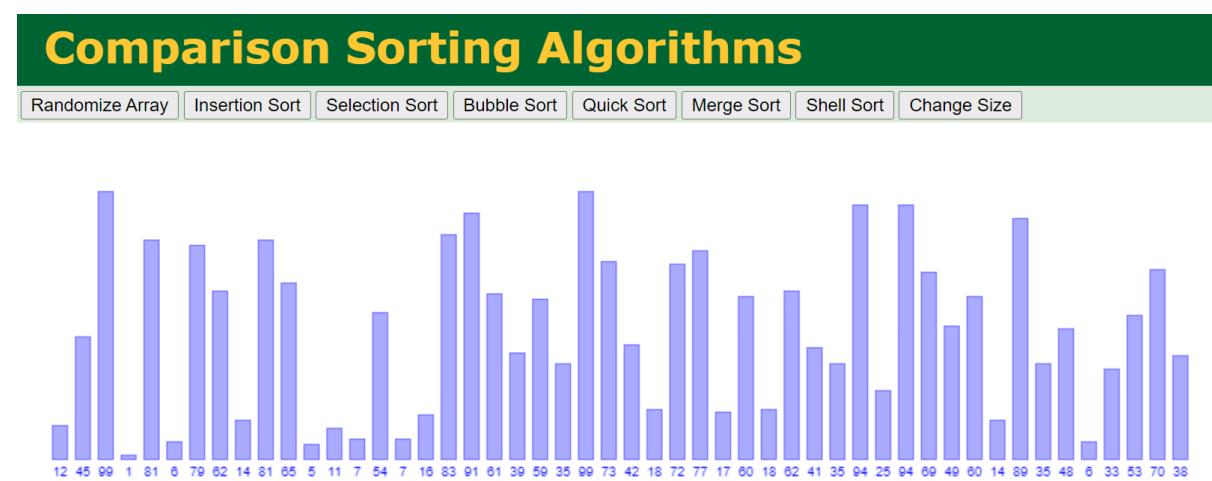
## VISUALISATION

Visualisation includes a variety of techniques (diagrams, images, graphs, flow charts, etc.) that aim to illustrate what a problem entails, and how to approach its solution. This system is inherently designed to use visualisation as this is a better approach to helping students retain information<sup>8</sup> and therefore succeed with applying these algorithms in programs or written questions. The advantage of using visualisation is that it simplifies the concepts of the algorithm being applied, in this case. However, visualisation does not explain why something is the way it is. Therefore, this tool would only be beneficial if students have an understanding - even vague understanding - of the specific algorithm being visualised.

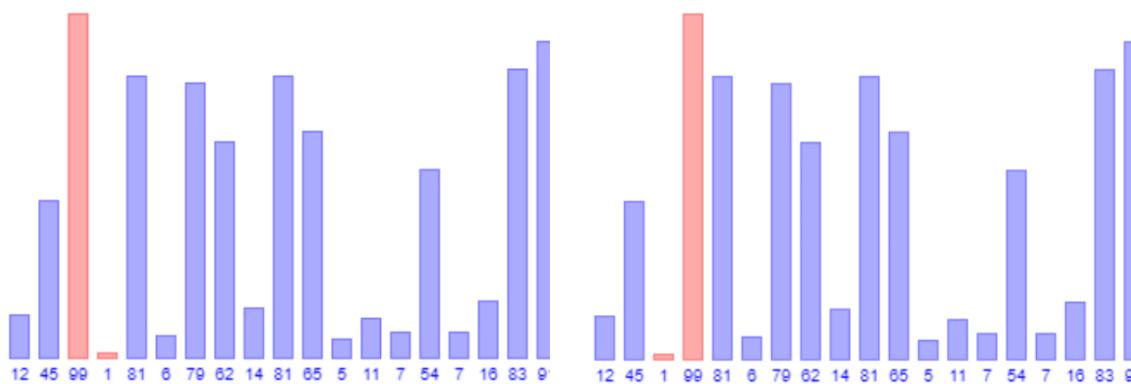
## EXISTING SYSTEMS RESEARCH

### UNIVERSITY OF SAN FRANCISCO'S DATA STRUCTURE VISUALISATIONS WEBSITE

One example of an existing system similar to my project is the University of San Francisco's 'Data Structure Visualizations' website<sup>9</sup>, which has many visualisations for sorting and graph algorithms. For example, below is a screenshot of how the Bubble Sort algorithm is visualised with a bar chart style graph with random numbers to be sorted:

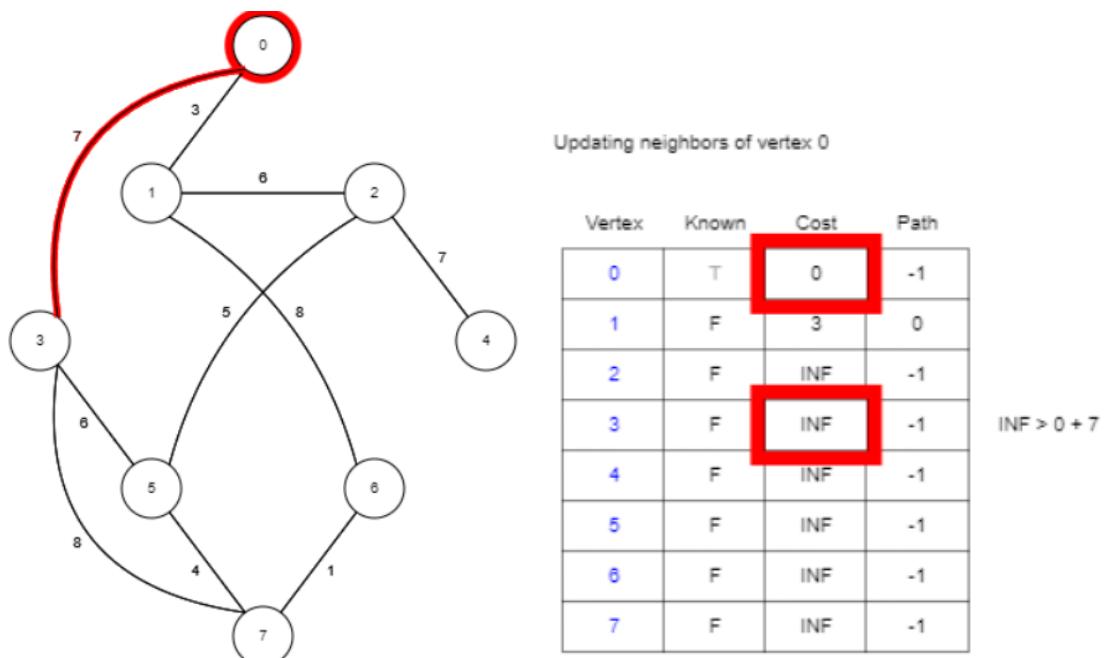


Using the Bubble sort algorithm, below is an example of the numbers 99 and 1 being compared and then switched.



The advantage of this type of visualisation as part of the simulation is that users can clearly and quickly see which number is larger, rather than just the characters for the numbers. In addition, some students using this system may have learning difficulties and this type of size based visualisation may be more beneficial to them than just a lot of numbers being switched around, which is what the text based solution represents. A disadvantage of this bar chart style is that if the range of the numbers of very large, for example 0.1 to 100, and some of the numbers are similar, for example, 23.4 and 23.5, it may be difficult to differentiate between the numbers on the bar chart as they may appear to be the same size. Therefore, the written solution is a necessary part of the output of the solution to Bubble Sort.

With Dijkstra's algorithm, the user enters a start vertex. The node currently being visited is clearly highlighted in red, alongside a table with the boxes being clearly highlighted. The calculation checking the distance from start or the 'Cost' is also shown through text.



Additionally, the backtracking portion of Dijkstra's algorithm is graphically shown through the highlighted box changing to the node that the current node is pointing to for the shortest distance.

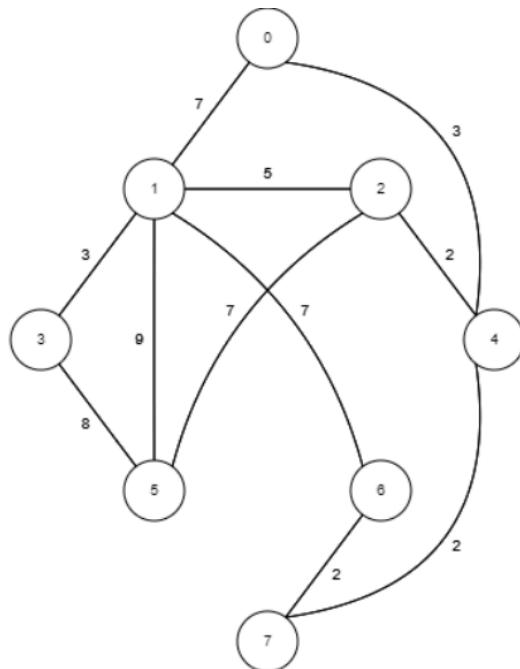
Furthermore, this implementation calculates all of the shortest routes from the start node and displays all of them.

Vertex	Known	Cost	Path
0	T	0	-1
1	T	3	0
2	T	9	1
3	T	7	0
4	T	16	2
5	T	13	3
6	T	11	1
7	T	12	6

0  
0 1  
0 1 2  
0 3  
0 1 2 4

One advantage of this solution for visualising Dijkstra's algorithm is that the red highlighting makes it easy to follow through the table and the graph because it highlights the current node and then the edges to all the unvisited connected nodes. A disadvantage of this type of visualisation is that as the nodes are all labelled as numbers, the table may not be very clear in terms of quickly differentiating between the rows for different nodes. So, I would use letters for each of the nodes instead.

With Prim's algorithm, a start vertex is entered by the user. A table, similar to the one used in the representation of Dijkstra's algorithm, is used for updating neighbouring nodes.



Updating neighbors of vertex 2

Vertex	Known	Cost	Path
0	T	0	-1
1	F	5	2
2	T	2	4
3	F	INF	-1
4	T	3	0
5	F	INF	-1
6	F	INF	-1
7	F	2	4

Vertex 4 known

One advantage of this implementation of Prim's algorithm is that the edges being checked are clearly highlighted, which makes it easier to follow. However, considering the way the algorithm would be applied when writing it down, either through writing down the edge of least weight at each stage of adding nodes or through using the distance matrix, the table is not very beneficial to understanding

this. Whilst the way it is represented comes from a Computer Science approach where a focus is on checking which edge has the lowest weight, I think that as this algorithm is studied in Further Maths and not Computer Science, an approach that highlights the edges being checked but doesn't have this table, instead just lists out the edges - similar to the first method for Prim's - or uses the distance matrix.

In addition, all the algorithm visualisation pages have a menu as shown below.

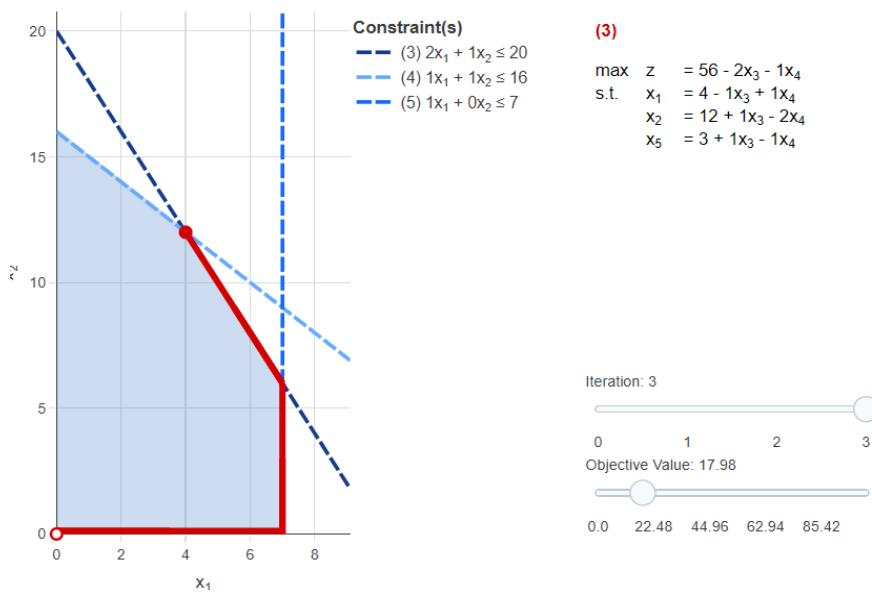


The 'Skip Back' button causes the visualisation to go back to the start whilst the 'Skip Forward' button goes to the end. The 'Step Back' button goes back to the previous step so, for example, in Bubble Sort this is the previous comparison. The 'Step Forward' button does the same but goes to the next step instead. Both of these buttons can only be used when the visualisation is paused and are greyed out when the Visualisation is playing. The 'Play' or 'Pause' button restarts or pauses the visualisation. The animation speed slider changes the speed of the animation and the change canvas size controls change the size of the web page but not the different elements that are part of the visualisation.

One advantage of this menu for these visualisations is that the uses of all the buttons are clear. Although users may want to use the step forward/back buttons to go through steps and pause at their own pace, which they are able to do with these buttons, they are quite slow to use, which may be something that puts them off. So, if I were to implement something similar in my system, left and right arrow keyboard inputs may be better suited to this as they can be faster to use. Additionally, the 'Change Canvas Size' controls do not seem to be a very useful feature for any of the algorithms I am planning to implement as the elements displayed already fit on the screen.

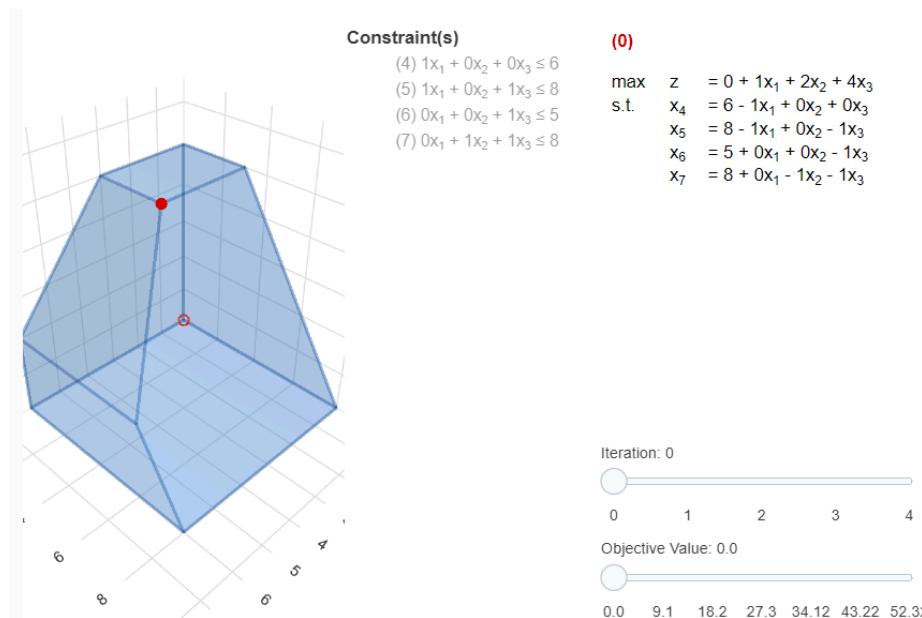
## GILP

An example of an existing system for visualising the Simplex algorithm is GILP<sup>10</sup> (Geometric Interpretation of Linear Programs). This is a python package that utilises Plotly for visualising the geometry of linear programs and was developed for a course at Cornell University. This package allows you to define linear programs and then has a function that will visualise the constraints as lines or planes on a graph and allows you to step through the iterations of simplex by showing which vertex of the feasible region is visited. For example, below is an example from GILP:



Colour coding of the different constraints, shading of the feasible region and a red line to follow through the iterations of the simplex algorithm are used, along with a slider to allow the user to step through, all make this implementation of visualising the Simplex algorithm very clear to look at and understand. However, unless the package is installed, the end user is not able to input constraints, therefore not making this very beneficial for allowing students, who may not be able to code given that this is only studied in Further Maths A-Level Decision topics, to learn with multiple examples that they input. Therefore, whilst this is a good implementation in terms of how it is laid out, my implementation will need to be quite different to this example as there are other features I want this system to have.

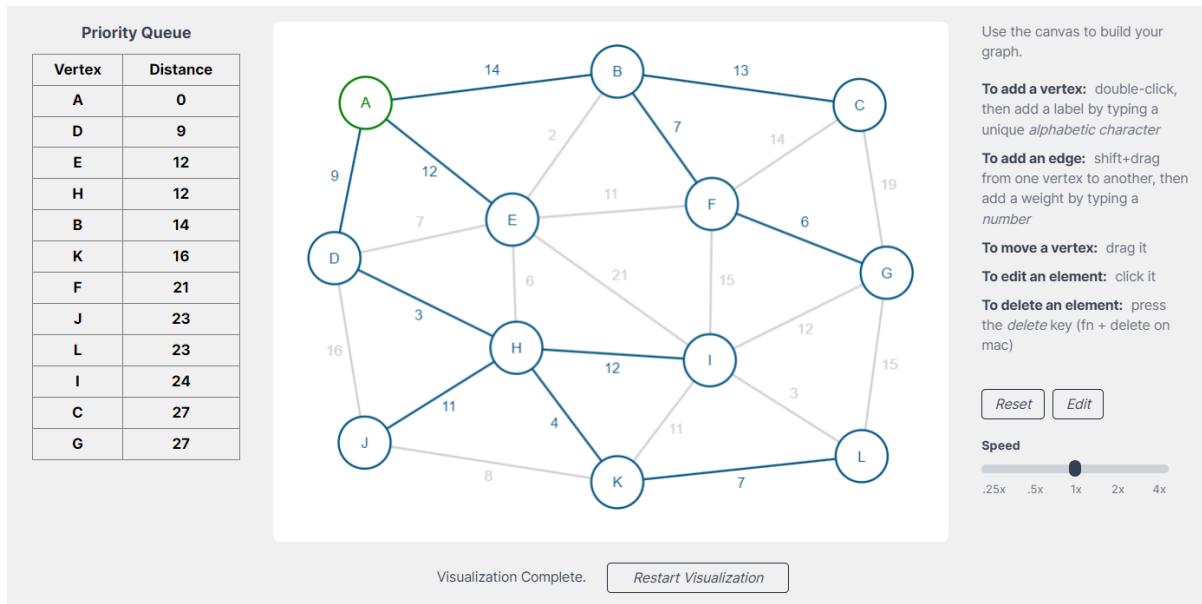
Another thing to consider with GILP is that it allows constraints with 3 variables to be visualised. An example is shown below:



Similar to the 2 dimensional example shown above the colour coding and highlighting of the path followed to find the optimal solution are advantaged to this implementation but the lack of ability for the end user to input constraints is a disadvantage. Whilst this may be a very beneficial tool to students and teachers, I have already decided to limit my implementation of visualising the Simplex algorithm to 2 variables due to time constraints for development.

## DIJKSTRA'S ALGORITHM VISUALIZER - danvbyjan.com WEBSITE

This is a website<sup>11</sup> that allows users to input graphs or use pre-built graphs and then applies Dijkstra's algorithm to it. Instead of a table with cost and path, this system has a Priority Queue table with the nodes and the shortest distance to that node, which is updated as each node is visited. An example of the finished version of Dijkstra's algorithm being applied is shown below:



One advantage of this system is the ability for users to input their own graphs by double clicking to input a node and doing shift-drag between nodes to input an edge. As one of the goals for implementing Dijkstra's and Prim's algorithms is to let the user input their own graph, this way is a helpful guide on how this could be applied. On the other hand, I think that the Priority Queue is not very beneficial for students' understanding of applying these algorithms because it doesn't show the previous node that connects to the current one, therefore not being very useful for backtracking, which is an important final step for Dijkstra's.

## PROPOSED ESSENTIAL FEATURES

After the analysis of similar existing systems [and the questionnaire], I have compiled the features that are essential for this system.

## INITIAL LOGIN PAGE

The initial login page will be quite minimalistic with the title ‘Algorithms Simulator and Teaching Tool’ as a focus. There will also be a username field, a password field, which will have black dots as placeholders in order to ensure security, a login button and registration button. If the login button is pressed and login fails, a message will appear saying that ‘Login was unsuccessful. Check the username/password entered are correct. If you have not registered before, click the Register button.’. If login is successful, the home page will be accessed. If the registration button is clicked, the registration page will be accessed.

## REGISTRATION PAGE

The registration page will also be quite simple with the system title, a username field, a password field and password confirmation field (both of which have black dots as placeholders). If the passwords do not match, a message will come up saying ‘Passwords do not match’. If the password does not meet the following minimum security requirements, a message will appear saying which requirement(s) are not met:

- At least 8 characters long
- Uppercase and lowercase characters must be used
- At least one number
- At least one special character

There will also be checkboxes for which algorithms (Bubble sort, Prim’s algorithm, Dijkstra’s algorithm, Simplex algorithm) the user wants to be able to use - note that they must select at least one. Additionally, the user must select either a Student or Teacher account type. When the ‘Register’ button is clicked, all of this data is written to a database and the home page is accessed. If registration fails, a message will be output saying ‘Registration failed - please check everything has been filled in correctly and try again’.

## HOME PAGE

The home page will have the title of the system, some text describing what the system does, and buttons to access the visualisation pages of all the algorithms selected by the user to study. If the user has a student account, a graph of their quiz statistics will be displayed, including the number of correctly and incorrectly answered questions of each type of algorithm they have done. This is because it is helpful for students to be able to keep track of the algorithms they struggle with and should practise, something that doing revision with paper and pen is not useful for.

Finally, there should be a ‘Logout’ button that logs the user out of their account and returns them to the Initial Login page. If the system is closed, the user is also logged out.

## BUBBLE SORT VISUALISATION PAGE

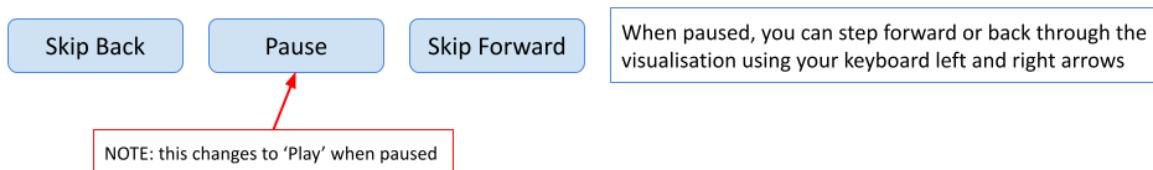
The Bubble Sort Visualisation page will display the following initially:

- A title of ‘Bubble Sort Visualisation’

- Text describing the algorithm in step-by-step plain English instructions
- 8 text boxes to enter numbers into and a button to add more. There will be a limit to 12 numbers in the data set and each number must be between 1 and 100.
- A selection of ascending or descending ordering
- ‘Home Page’ button back to the home page

When data to be sorted has been entered, this will be displayed on a bar chart because users can clearly and quickly see which number is larger, rather than just the characters for the numbers. In addition, some students using this system may have learning difficulties and this type of size based visualisation may be more beneficial to them than just a lot of numbers being switched around.

Additionally, there will be a ‘Start Visualisation’ button to start the visualisation of the Bubble Sort algorithm. If this button is pressed and a data set of at least 8 numerical values has not been entered, an error message will be displayed. During comparisons in each pass, the animations on the bar chart will match with a text-based visualisation of the items and the items being compared are highlighted in a different colour on the bar chart. This is an advantage as it makes it clear which items are being compared so there is no confusion by users. Once the visualisation has started, the menu can be used. An diagram (with annotations in red) of the menu is shown below:



The ‘Skip Back’ button will start the visualisation again from the beginning. The ‘Skip Forward’ button will fast forward the visualisation to the end (when the sort is complete). The ‘Pause’ button, which will be a ‘Play’ button when paused, will stop execution and instead will allow the user to step forward or back through the visualisation using their left and right arrow buttons. This is an advantage of the system because it allows students to learn at their own pace and find exactly where they went wrong if they are checking homework, for example. Additionally, this would allow teachers to explain each step one at a time but not have to write it all out themselves. When the sort has finished, a message will appear on screen saying ‘Sort Complete’.

## PRIM’S ALGORITHM VISUALISATION PAGE

The Prim’s Algorithm Visualisation page will display the following initially:

- A title of ‘Prim’s Algorithm Visualisation’
- Text describing the algorithm in step-by-step plain English instructions
- A graph input field
- Instructions for creating a graph
- ‘Home Page’ button back to the home page

The graph input field will have the following functionality:

- A double click adds a vertex and the user must input a unique identifier of 1 character. The number of vertices must be between 4 and 10

- Clicking on a vertex and doing shift drag to another vertex adds an edge. The user must input a weight for the edge
  - To delete an element (vertex or node), click on the element and press delete on the keyboard
- Additionally, there will be a ‘Start Visualisation’ button that starts the visualisation of Prim’s algorithm on that graph. As edges are checked, they will be highlighted on the graph and if they are added to the MST, they will also be displayed as text. When the visualisation is finished, a message should appear to say ‘MST found’ and the edges in the MST should stay highlighted on the graph. If the ‘Start Visualisation’ button is pressed and a valid graph has not been entered, an error message will appear.

There will also be a menu field, as described in the features of the Bubble Sort Visualisation page, with the ‘Step Forward’ and ‘Step Back’ buttons allowing the user to go through the checking of the different edges connected to the current edge.

Furthermore, as discussed in the Research section, a matrix representation version of Prim’s algorithm would be beneficial as it is studied in the Edexcel Further Maths Decision 1 module but I think that this may not be added due to time constraints and so may become a feature possibly added in Future Development.

## DIJKSTRA’S ALGORITHM VISUALISATION PAGE

The Dijkstra’s Algorithm Visualisation page will display the following initially:

- A title of ‘Dijkstra’s Algorithm Visualisation’
- Text describing the algorithm in step-by-step plain English instructions
- A graph input field
- Instructions for creating a graph
- A blank table for Dijkstra’s algorithm, including the Vertex, Distance from the Start Vertex, Known and Shortest Path columns, as described in the Research section
- ‘Home Page’ button back to the home page

The functionality of the graph input field will be the same as it should be for the Prim’s Algorithm Visualisation Page but as vertices are added, they should appear in the table. There will be an additional field to enter the start and end vertices, which should be different. When the ‘Start Visualisation’ button is pressed, as vertices are visited, they should be highlighted on the graph and in the table and values should be input in the table. When the backtracking portion of Dijkstra’s algorithm is reached, the edges should be highlighted on the graph. When the visualisation is finished, a message should appear with the text for the shortest path from the start to the end vertex. If the ‘Start Visualisation’ button is pressed and a valid graph has not been entered or the start and end vertices are not entered or are the same, an error message will appear.

There will also be a menu field, as described in the features of the Bubble Sort Visualisation page, with the ‘Step Forward’ and ‘Step Back’ buttons allowing the user to go through the adding of the different values in the table one-at-a-time.

## SIMPLEX ALGORITHM VISUALISATION PAGE

The Simplex Algorithm Visualisation page will initially display:

- A title of ‘Simplex Algorithm Visualisation’
- Text describing the algorithm in step-by-step plain English instructions
- A blank graph of the positive x and y axes
- Textboxes to enter number coefficients for the constraints and a ‘less than or equal to’ inequality symbols, so the constraints will appear in the form:  $_x + y \geq _$
- A button to add a constraint (start with two but can add a third constraint)
- Similar text boxes to add coefficients for the objective function, which is maximised in the form:  $P = _x + _y$
- ‘Home Page’ button back to the home page

The advantage of having textbox inputs for the coefficients is that it makes validation of numerical inputs easier. The coefficients cannot both be 0 for x and y and the values must be between -10 and 10. If there is no numerical input for any input field and the ‘Start Visualisation’ button is pressed, an error message will be displayed.

When the ‘Start Visualisation’ button is pressed, the lines for the constraints will be plotted on the graph, with  $<$  constraints being dotted lines and  $\leq$  constraints being full lines and all the lines being colour-coded. The points at the vertices of the Feasible Region of the graph should appear as coordinates when the cursor goes on them. As vertices in the Feasible Region are visited, a highlighted line on the graph should follow the path until the maximum point is reached. When the visualisation is finished, text should appear saying ‘Maximise P to + when  $x=_$  and  $y=_$ ’.

Additionally, there will also be a menu field, as described in the features of the Bubble Sort Visualisation page, with the ‘Step Forward’ and ‘Step Back’ features allowing the user to go through the iterations of the Simplex Algorithm and see the path taken on the graph.

## QUIZ PAGE

The ‘Quiz’ page will only be available to users with a student account. The initial page will display the following:

- A title of ‘Quiz’
- Buttons for each of the algorithm types the user has selected to study
- ‘Home Page’ button back to the home page

When a button has been pressed, a randomised question will be displayed for that algorithm. Below is a table of the randomised values and the functionality of each possible algorithm:

ALGORITHM	RANDOMISED DATA FOR QUESTION	FUNCTIONALITY
Bubble Sort	10 random numerical values between 1 and 50	There will be 10 drop down menus in line with each number value as an option and users should enter

		the final sorted outcome and the number of passes it took.
Prim's Algorithm	A connected graph of 6-8 vertices with 7+ edges	The user should enter the edges in the MST as text (e.g. AB or BA) and the total weight of the MST.
Dijkstra's Algorithm	A connected graph of 6-8 vertices with 7+ edges and start and end vertices (which should be different).	The user should enter the shortest route between the start and end vertices and the weight of the path.
Simplex Algorithm	Inequalities for constraints and the objective function, with coefficients being between -10 and 10, but not both 0 for x and y. These inequalities will be displayed on a graph.	The user should enter the maximum value of P and the corresponding x and y values.

Additionally, there will be a ‘Check’ button that will output whether the user got the right answer or not. Next to the ‘Check’ button will be a ‘New Question’ button that randomises a new dataset, graph or set of constraints and objective function for the same algorithm type.

The values for the number of questions correctly or incorrectly answered for that question will be updated in the database, allowing the quiz statistics graph to display accurate information.

## LIMITATIONS AND FEASIBILITY

### ECONOMIC

There is not much economical pressure on this project as I am creating it, meaning that there is no cost for software to consider.

### TIME

Due to the time constraints and deadlines that I have, being an A-Level student myself, some of my current proposed features may not end up being implemented. In addition, some other features, such as other algorithms like the Route Inspection algorithm or other sorting algorithms like Quick Sort. These may be implemented in post-development to improve the system.

### TECHNICAL

As I am planning on writing my code in Python, all of the source code will need to either be interpreted or compiled.

Interpreters translate the code line by line, converting them into machine code. The advantage of interpreters is that the locations of syntax errors are easily detected, making it easier for the

programmer to debug. However, it is slower to translate the code, and this needs to be done every time the code is run, and that the interpreter needs to be installed on the user's computer.

Compilers compile the code into one executable file. The process of compilation is as follows:

1. Lexical Analysis - remove comments and unnecessary spaces and then replace keywords, constants and identifiers with tokens. These build up a symbol table.
2. Syntax Analysis - Variables are checked to make sure they have been correctly declared and contain the correct data type and operations are checked to ensure that they are appropriate for the type of variable being used.
3. Code Generation - generate machine code.
4. Optimisation - makes the program more efficient so it runs faster and uses fewer resources.  
Libraries are added to the code.

An advantage of compiling the code is that the end user does not need to have an interpreter installed on their computer and that it can be faster for larger files. However, all errors are only displayed at the end of compilation, which makes debugging harder for the programmer as locations of errors may not be given.

Considering the benefits to the programmer, and that I am not planning on releasing the code to the public, I think it would be better to interpret the code as it would allow any errors to be found and fixed faster. Therefore, any end users will need to have an interpreter for Python 3.10 and the Tkinter, matplotlib, networkX and sqlite3 libraries installed.

## POLITICAL

The target demographic for this system is A-Level Computer Science and Further Maths students. The algorithmic thinking skills system will help to develop will put these students in a strong position to go on to work in fields related to national security, such as in GCHQ, and technological expansion, which is of interest to the government as they set out a new industrial strategy, which can be seen in the Labour Party's general election manifesto<sup>12</sup>.

## LEGAL

As part of the login system and the quiz statistics system, I will be collecting data about users. Therefore, this system must adhere to the Data Protection Act 2018<sup>13</sup>, which states that data must be:

- Used fairly, lawfully and transparently
- Used for specified and explicit purposes
- Used in a way that is adequate, relevant and limited to only what is necessary
- Accurate and kept up-to-date
- Handled in a way that ensures appropriate security

If I were to sell this software, the Copyright Designs and Patents Act 1998 would also need to be considered. However, as I am not planning on releasing the software, this is not something that I will think about.

## HARDWARE AND SOFTWARE REQUIREMENTS

### HARDWARE

I am planning on writing my code in Python. The minimum system specification<sup>14</sup> for Python 3.10 are:

- 1GHz single-core processor
- 2GB of RAM
- 10GB of free disk space
- Basic operating system like Windows 7/8/10+, macOS or Linux

However, as I am planning on having graphics and animations involved in the visualisations of the algorithms so the GPU must be able to run 24 frames per second, which is the standard movie frame rate, although I may decide to make this slower in development and testing for the user to understand what is happening better. Additionally, I will use 24 bit colour depth, as humans cannot see better than that. Therefore, the target machine will be a standard laptop or PC.

My PC's specifications are: AMD Ryzen 5 5625U with Radeon Graphics (6 cores and 12 threads), 2.30GHz, 8GB RAM.

### SOFTWARE

For development, I will be using Thonny 4.1.4<sup>15</sup> because of the debugging features and step in and out functionality. Additionally, my operating system is Windows 11 Home. Therefore, this system will be optimised to work on these specifications and operating system, which is justified as Windows has 73% of the market share<sup>16</sup> for desktop and laptop devices.

I am planning on writing the code for this system in Python and interpreting the code. Python 3.10 has many libraries that are useful for plotting x,y graphs and connected graphs, like matplotlib and NetworkX, which is useful for algorithms like Prim's, Dijkstra's and Simplex, which I am planning on implementing. Additionally, the advantage of interpreting the code is that debugging the code becomes easier for the programmer as the locations of syntax errors are output when found and that the user can personalise the code to their own needs as they will need the code and this can be released with an open licence. However, a requirement is that the end user will need to have an interpreter for Python 3.10 and the following libraries installed on their computer:

- Tkinter<sup>17</sup> - this will be used as the main GUI that the users interact with. All the graphs from matplotlib or NetworkX can be input into Tkinter user interfaces.
- Matplotlib<sup>18</sup> - has features for plotting equations on axes, which will be useful for the Simplex algorithm, and for bar charts, which will be useful for Bubble sort and the quiz statistics section of the Home page.
- NetworkX<sup>19</sup> - has features for creating and displaying directed graphs, which will be used in Prim's and Dijkstra's algorithms.
- Sqlite3<sup>20</sup> - this will be used to allow SQL commands to be executed on a database in SQLite, which will be used for creating accounts, checking the correct login details are entered and changing the number of questions users got right/wrong in the quiz section.

I may find some other libraries to use later on in development so I will make it clear what the final software requirements are in the Evaluation section.

## SUCCESS CRITERIA

### [1] INITIAL LOGIN PAGE

Criteria [a,b,c,e] refer to usability and criteria [d] refers to the functionality of the Initial Login page.

- [a] There will be text greeting the user with the title of the system 'Algorithms Simulator and Teaching Tool'. I will test this by checking the page and making sure this appears.
- [b] There will be two data input fields - Username and Password - that should appear below the title. I will test this by checking that they appear on the page and that text can be entered into them.
- [c] When data is entered into the Password field, black dots appear as placeholders. I will test these by entering different passwords into the Password field to make sure placeholders appear instead of the text.
- [d] There will be a Sign In button below the two input fields. When it is pressed, the database is queried and the values are compared to make sure they are correct for the user to log in and continue to the Home Page. If incorrect details were entered, a message will pop up saying 'Login was unsuccessful. Check the username/password entered are correct. If you have not registered before, click the Register button.'. I will test this by entering correct and incorrect usernames and passwords and checking that the functionality of clicking the Sign In button is correct for all cases.
- [e] There will be a Register button next to the Sign In button. When it is clicked, the user will be sent to the Registration page. I will test this by clicking the Register button and checking that this occurs.

### [2] REGISTRATION PAGE

Criteria [a,b,d,e,f,g,i] refer to usability whilst criteria [h,i] refer to functionality in the Registration page.

- [a] There will be text greeting the user saying 'Create an Account' and the following requirements: 'Usernames should be unique.' and 'Passwords should have at least 8 characters, uppercase and lowercase letters and contain numbers and special characters.'. I will test this by checking that when the Registration page is accessed, this text is displayed.
- [b] A Username input field, Password input field and Password Confirmation input field appear below each other and below the text. I will test this by checking that when the Registration page is accessed, these fields are displayed and text can be entered into them.

- [c] When data is entered into the Username input field, text will say ‘Username not available’ if the entered username has already been used. I will test this by having premade accounts and checking that when I enter their usernames, this message displays and when I enter a unique username, it does not.
- [d] When data is entered into the Password Confirmation input field, text will display whether it matches the data in the Password field. I will test this by entering matching and different passwords into the fields and checking that only text is output if the passwords do not match.
- [e] The Password and Password Confirmation fields have black dots as placeholders. I will test this by entering different passwords into both fields to make sure placeholders appear instead of the text.
- [f] Check boxes will appear for the four different algorithms implemented. I will test this by checking that when the Registration page is accessed, these appear and algorithms can be checked.
- [g] A student/teacher selection will appear for the type of account being created. I will test this by checking that when the Registration page is accessed, this appears and only one of these options can be chosen.
- [h] A Register button will appear below the student/teacher account choice. When pressed, the Username, Password and Password Confirmation fields should be validated against the rules detailed in criteria [2][a] and for the two password entries matching. Additionally, validation that at least one algorithm has been checked and that a type of account has been chosen. If all these checks have passed successfully, all of this data should be written to a database and continue to the Home page. If any of the validation checks is unsuccessful, a message will pop up saying ‘Registration unsuccessful. Check that username/password entries are valid and that at least one algorithm has been checked and a type of account has been selected.’. I will test this by checking that when the Registration page is reached, the Register button appears. For testing functionality, I will enter details that fully/partially/don’t meet the criteria and try pressing the Register button, checking that the functionality is correct.
- [i] A Back button will appear at the bottom right corner of the Registration page. When clicked, this returns the user to the Initial Login page. I will test this by making sure that when the Registration page is reached, this button appears in the correct place and that pressing it successfully returns the user to the Initial Login page.

### [3] HOME PAGE

Criteria [a,b,e] refer to usability whilst criteria [b,c,d] refer to functionality in the Home page.

- [a] There will be text greeting the user with the title of the system ‘Algorithms Simulator and Teaching Tool’. I will test this by checking the page and making sure this appears.
- [b] There will be a button for each of the algorithm visualisation pages (out of the Bubble Sort Visualisation, Prim’s Algorithm Visualisation, Dijkstra’s Algorithm Visualisation and Simplex Algorithm Visualisation) that the user checked upon registration - see criteria [2][h]. When each is clicked, the user will be sent to the corresponding page. I will test this by creating accounts with different combinations of algorithm choices and making sure that when the button is clicked, the correct page is accessed.
- [c] If the user has a student account, there will be a Quiz button. When clicked, this should take the user to the Quiz page. It should appear below the algorithm visualisation page buttons. To test this, I will create student and teacher accounts and check that this button only appears for student accounts and that clicking it takes the user to the Quiz page successfully.
- [d] If the user has a student account, there will be a bar chart of the user’s quiz statistics, including the number of correctly and incorrectly answered questions of each type of algorithm they have completed. This data should be read from the database. I will test this by creating accounts with different numbers of correctly and incorrectly answered questions of each type and check that the data is output correctly on the bar chart on the Home page.
- [e] There will be a Logout button in the bottom right corner of the page. When clicked, the user should be logged out of their account and return the Initial Login page. I will test this by checking that when the Home page is reached, this appears in the correct place and that clicking it has the correct functionality, as described.

### [4] BUBBLE SORT VISUALISATION PAGE

Criteria [a,b,c,d,e,f,k,l] refer to usability whilst criteria [d,f,g,h,i,j] refer to functionality in the Bubble Sort Visualisation page.

- [a] There will be text greeting the user with the title of the page ‘Bubble Sort Visualisation’. I will test this by checking the page and making sure this appears.
- [b] There will be text describing the Bubble Sort algorithm in steps in plain English and the requirements for the dataset. I will test this by checking the page and making sure this appears.
- [c] There will be 8 text boxes on the right half of the screen that the user can enter numbers into. To test this, I will check that these appear and that numbers can be entered.

- [d] There will be a button in line with the text boxes to add 1 more text box. This will only appear until there are 12 text boxes. To test this, I will check that this button appears and that up to 4 text boxes can be added, the button disappears when there are 12 text boxes and that numbers can be entered into the text boxes.
- [e] There is an ascending/descending order selection that the user must make. To test this, I will check that on the Bubble Sort Visualisation page this is displayed and the selection functions as intended.
- [f] There is a Start Visualisation button below the text describing the algorithm. Pressing this will initially validate that the dataset is made up of eight numbers between 1 and 100 (inclusive). Additionally, it is checked that a choice of ascending or descending order is made. If this validation fails, a message will appear saying ‘Dataset not valid’ or ‘Order not chosen.’. If this validation is successful the entered dataset will appear on a bar chart, located on the left of the screen, and start the visual execution of the Bubble Sort algorithm. To test this, I will check that the button appears on the screen in the correct place and that the visualisation appears as it should. Additionally, to test the validation of the dataset, I will input different values that are/aren’t accepted and try not choosing an order to check that it functions correctly.
- [g] Between each comparison in the Bubble Sort algorithm, there should be a check for a click of the Skip Forward, Skip Back and Pause buttons. Additionally, with each comparison, the two items being compared should be highlighted in red on the bar chart and swap places if they are different. To test this, I will add messages to say that these checks have been completed and I will visually check that the highlighting and swapping is correct.
- [h] For each pass, the text for all the numbers should be on one line. To test this, I will make sure that when the visualisation has started, the passes are on one line in the text section.
- [i] For each comparison, if there is a swap needed, there will be an animation of the two items swapping on both the bar chart and the text numbers
- [j] When the visualisation is finished, a message will be displayed saying ‘Sort Complete’. To test this, I will check that when the dataset has been sorted, this message is displayed.
- [k] There is a Back button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.
- [l] There is a menu - criteria [9] - at the bottom of the screen. To test this, I will check that it is displayed in the correct place and functions as intended.

## [5] PRIM'S ALGORITHM VISUALISATION PAGE

Criteria [a,b,c,d,e,f,g,j,k] refer to usability whilst criteria [f,g,h,i,j] refer to functionality in Prim's algorithm visualisation page.

- [a] There will be text greeting the user with the title of the system 'Prim's Algorithm Visualisation'. I will test this by checking the page and making sure this appears.
- [b] There will be text describing Prim's algorithm in steps in plain English. I will test this by checking the page and making sure this appears.
- [c] Below the text describing Prim's algorithm, there is text describing how to input a graph - see criteria [10]. I will test this by checking the page and making sure this appears.
- [d] On the left of the screen, there will be a graph input section - see criteria [10]. To test this, I will check that it is displayed on the correct side of the screen and functions as intended.
- [e] On the right side of the screen, there will be a start vertex drop-down menu to choose a vertex from the input graph. I will test this by checking that it is displayed in the correct place on the screen and that all the vertices in an entered graph are in the drop-down menu and can be selected. I will do this by inputting different valid graphs and ensuring that this is correct for all of them.
- [f] There is a Start Visualisation button below the text describing the graph inputs. Pressing this will initially check if a valid graph has been entered and that a start vertex has been chosen. If this validation is unsuccessful, a message will be displayed saying either 'Graph not valid' or 'Choose start vertex'. If validation is successful, the visualisation of Prim's algorithm will begin. To test this, I will try pressing the Start Visualisation button, having input different valid and invalid graphs, with or without a start vertex entered. This is to check that the correct output is displayed for each case.
- [g] As the visualisation of Prim's algorithm is executing, the edges being checked should be highlighted in red. To test this, I will input different graphs and run the visualisation, ensuring that the correct edges are being checked.
- [h] As the visualisation of Prim's algorithm is executing, the edges added to the MST and their weights should be displayed as text on the right side of the screen. I will test this by inputting different graphs and running the visualisation, ensuring that the correct edges and values are displayed, which I will work out by hand.
- [i] When the visualisation is finished, a message will appear saying 'MST found' and the edges in the MST should be highlighted in red on the input graph. To test this, I will enter different graphs and check that running the visualisation completes with this step.

- [j] There is a Home button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.
- [k] There is a menu - criteria [9] - that appears at the bottom of the screen. To test this, I will check that it is displayed in the correct place and functions as intended.

## [6] DIJKSTRA'S ALGORITHM VISUALISATION PAGE

Criteria [a,b,c,d,f,g,k,l] refer to usability whilst criteria [e,g,h,i,j] refer to functionality in Dijkstra's Algorithm Visualisation page.

- [a] There will be text greeting the user with the title of the system 'Dijkstra's Algorithm Visualisation'. I will test this by checking the page and making sure this appears.
- [b] There will be text describing Dijkstra's algorithm in steps in plain English. I will test this by checking the page and making sure this appears.
- [c] Below the text describing Dijkstra's algorithm, there is text describing how to input a graph - see criteria [10]. I will test this by checking the page and making sure this appears.
- [d] On the left of the screen, there will be a graph input section - see criteria [10]. To test this, I will check that it is displayed on the correct side of the screen and functions as intended.
- [e] On the right side of the screen, there will be a vertex table including columns for vertex, distance from start vertex, visited (T/F) and path. To test this, I will check that when the Dijkstra's visualisation page is accessed, this table is displayed in the correct place.
- [f] Below the table, there will be a start vertex drop-down menu and an end vertex drop-down menu to choose a vertex from the input graph. The vertices chosen should be different. I will test this by checking that it is displayed in the correct place on the screen and that all the vertices in an entered graph are in the drop-down menu and can be selected. I will do this by inputting different valid graphs and ensuring that this is correct for all of them.
- [g] Next to the start vertex drop-down menu, there will be a Start Visualisation button. Pressing this will initially check if a valid graph has been entered and that different start and end vertices have been chosen. If this validation is unsuccessful, a message will be displayed saying either 'Graph not valid' or 'Choose different start and end vertices. If validation is successful, the vertex table is filled in with the vertices and the start vertex's distance from the start value is set to 0. Then, the visualisation will begin. To test this, I will try pressing the Start Visualisation button, having input different valid and invalid graphs, with or without a

start vertex entered. This is to ensure that the vertex table is filled in correctly and to check that the correct output is displayed for each case.

- [h] As vertices are visited in Dijkstra's algorithm, they will be highlighted on the graph in red and the cells in the vertex table will be highlighted as they are updated. This is because colour-coding makes it clear to the user why certain values are being updated in the table. To test this, I will work out by hand the order the vertices should be visited and the state of the vertex table at each stage for a number of different valid graphs in order to check that the highlighting is correct at all stages.
- [i] For the backtracking portion of Dijkstra's algorithm, the edges being added to the path should be highlighted on the graph as they are added (i.e. from end vertex to start vertex). To test this, I will run the visualisation on different valid graphs and ensure that the order of backtracking and highlighting is correct and match up.
- [j] When the visualisation is finished, text should appear stating the shortest route from the start to the end vertex and the path should remain highlighted on the graph. To test this, I will run the visualisation on different valid graphs and ensure that the correct shortest route is displayed.
- [k] There is a Back button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.
- [l] There is a menu - criteria [9] - that appears at the bottom of the screen. To test this, I will check that it is displayed in the correct place and functions as intended.

## [7] SIMPLEX ALGORITHM VISUALISATION PAGE

Criteria [a,b,c,d,e,f,g,j] refer to usability whilst criteria [d,f,g,h,i,k] refer to functionality in the Simplex Algorithm Visualisation page.

- [a] There will be text greeting the user with the title of the system 'Simplex Algorithm Visualisation'. I will test this by checking the page and making sure this appears.
- [b] There will be text describing the Simplex algorithm in steps in plain English. I will test this by checking the page and making sure this appears.
- [c] On the right side of the screen, there are two constraint entry fields. This includes a text box for the x coefficient, a text box for the y coefficient and a text box for the constant. These will display as the following  $_x + _y \geq _$  with the  $_$  signifying a text box in this case. I will test this by checking that these are displayed when the page is accessed and that numbers can be entered into the text boxes.

- [d] Below the 2 constraint entry fields, there will be an 'Add constraint' button. When pressed, this will display a constraint entry field - as described in [7][c] - below the 2 existing entry fields, with the button disappearing. I will test this by checking that when the page is accessed, the button is displayed and that clicking it will make the button disappear and a constraint entry field appear, which will be tested as done in [7][c].
- [e] Below the 'Add constraint' button, there will be text box entry fields for the x and y coefficients of the objective function. This will be displayed in the form  $P = _x + _y$ . I will test this by checking that this is displayed when the page is accessed and that numbers can be entered into the text boxes.
- [f] Below the objective function entry field, there will be a 'Visualise' button. When the user presses this, the entered constraints and objective function will be validated, checking that the numbers have been entered and that they are between -10 and 10, with the x and y coefficients not both being 0. If this fails, a message will appear saying 'Constraints/objective function not valid'. If validation is successful, the constraints will be displayed on the x,y graph on the screen and the visualisation of the Simplex algorithm will begin. To test this, I will enter different valid and invalid data into the text boxes and check that the correct output is displayed.
- [g] On the right side of the page, below the text describing the algorithm, there will be a graph of the positive x and y axes. On the graph, the entered constraints will be plotted as straight lines and colour coded with red, green and blue. To test this, I will enter different valid constraints and check that they are plotted correctly on the axes.
- [h] As the iterations of the Simplex algorithm occur, a highlighted bold black line should appear on the axes showing which vertices of the feasible region are visited with each iteration. To test this, I will run the visualisation with different valid constraints and check whether or not the order of visiting the vertices and the final maximising vertex is correct by comparing it to a calculation done by hand using the simplex tableau.
- [i] When the visualisation is finished, a message should be output saying 'Maximise P to \_ when  $x = _$  and  $y = _$ ', with x and y being the coordinate values and P being the value calculated using these x and y values. To test this, I will enter different constraints and objective functions and check that the correct values are output in this message.
- [j] There is a Back button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.
- [k] There is a menu - criteria [9] - that appears at the bottom of the screen. To test this, I will check that it is displayed in the correct place and functions as intended.

## [8] QUIZ PAGE

Criteria [a,b,d,e,g,i,k,m] refer to usability whilst criteria [b,c,f,h,j,l,m] refer to functionality in the Quiz page module.

- [a] There will be text greeting the user with the title of the system ‘Quiz’. I will test this by checking the page and making sure this appears.
- [b] There will be a button for each of the algorithms (out of the Bubble Sort Visualisation, Prim’s Algorithm Visualisation, Dijkstra’s Algorithm Visualisation and Simplex Algorithm Visualisation) that the user checked upon registration - see criteria [2][h]. When each is clicked, the page will change to a quiz question of that algorithm. I will test this by creating accounts with different algorithms selected and check that the correct ones appear on the screen and that they generate the correct type of question - see criteria [8][c-k].
- [c] For the Bubble Sort algorithm question, a random dataset of ten values between 1 and 50 is generated and displayed on the right side of the screen as a list.
- [d] Below the generated dataset are 10 drop down menus of the values in the dataset. This is because this eliminates the issues from users typing the wrong number. The user should enter the final sorted outcome of the Bubble Sort algorithm being performed on the given dataset. I will test this by checking generating multiple questions and checking that the datasets are random and that the correct values appear in the drop down menus.
- [e] Below the drop down entry fields will be a text box for the user to enter the number of passes that occurred into. To test this, I will randomise multiple questions and check that this appears correctly.
- [f] For Prim’s algorithm, a random connected graph of 6-8 vertices with 7+ edges will be generated as a graph for the user to perform Prim’s algorithm on. Each of the vertices will be labelled with the letters A-H. This graph will be on the left side of the screen. The start vertex will also be listed as A. I will test this by choosing this type of question multiple times and checking that the graph displayed is random.
- [g] On the right side of the screen for Prim’s algorithm will be a text box for the user to enter the edges in the MST into like this: “AB, BC, CD”. Below this text box will be another one to enter the value of the weight of the MST into. I will test this by randomising multiple questions and checking that these appear as expected.
- [h] For Dijkstra’s algorithm, a random connected graph of 6-8 vertices with 7+ edges will be generated as a graph for the user to perform Dijkstra’s algorithm on. Each of the vertices will be labelled with the letters A-H. This graph will be on the left side of the screen. The start and end vertices will be listed as A and the last alphabetical letter vertex in the graph. I will

test this by choosing this type of question multiple times and checking that the graph displayed is random.

- [i] On the right side of the screen for quiz questions of Dijkstra's algorithm will be a text box that the user can enter the shortest path into in the format "ABCD". Below this will be a text box that the user can enter the value of the weight of the shortest path into. I will test this by selecting this type of questions and checking that these text boxes are displayed in the correct place.
- [j] For the Simplex algorithm, 3 constraints and an objective function will be randomised, with coefficients being between -10 and 10, but not both 0 for x and y. These will be displayed on the right side of the screen and plotted on a graph on the left side of the screen. I will test this by selecting this type of question and checking that the correct constraints, with coefficients in the correct range, are displayed and plotted.
- [k] Below the objective function on the right side of the screen for the Simplex algorithm will be a text box for P, a text box for x and a text box for y. The user must enter the values of P, x and y that maximise the objective function. I will test this by selecting this type of question and checking that these text boxes are displayed in the correct place and that numbers can be entered into them.
- [l] Below the question displays will be a Check button that the user can press. This will check whether the answer entered is correct by running the algorithm on the randomised data. Then, text will display whether the user got the answer correct or not. The user's quiz statistics will also be updated in the database. I will test this by getting questions right and wrong (by working them out by hand) and checking that the output is correct.
- [m] There is a Back button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.

## [9] MENU

Criteria [a] refers to usability whilst criteria [b,c,d,e] refer to functionality in the Menu module.

- [a] The Menu will be at the bottom of the screen for each visualisation of the algorithms. It will have 3 buttons - 'Skip Back', 'Pause' and 'Skip Forward' - along with text to describe the instructions for keyboard arrow inputs for stepping forward/back throughout algorithms. I will test that these all appear in the correct place for each algorithm's visualisation page and that the appearance is as designed.
- [b] The 'Skip Forward' button will skip forward through the visualisation to the final solution. For Bubble Sort, this will be all the text-based passes and a bar chart visualisation that is ordered. For Prim's algorithm, this will be the highlighted MST on the graph and the list of

edges on the right side of the screen. For Dijkstra's algorithm, this will be the completed vertex table and the shortest path highlighted on the graph. For the Simplex algorithm, this will be the highlighted path of the vertices in the feasible region, including the maximum objective function value, x and y values being stated. To test this, I will press the 'Skip Forward' button for each of the different algorithms, on different datasets/graphs/constraints, and check that the functionality is as intended.

- [c] The 'Skip Back' button will return the screen to the initial state of the algorithm's visualisation, which is where the original state of the dataset, graph of constraints will be displayed. To test this, I will press this button for all the algorithm visualisations, with a variety of datasets/graphs/constraints, and check that its functionality is as intended.
- [d] The 'Pause' button will stop the execution of the visualisation and will retain the state of the different variables and graphics. When the visualisation is paused, the 'Pause' button will change to a 'Play' button, which will restart the visualisation from where it was left off. To test this, I will press this button - both when 'Pause' and 'Play' - at different points in the visualisation and check that it functions as intended for all the different algorithms, using a variety of datasets/graphs/constraints.
- [e] When the user presses their keyboards left or right arrow, the visualisation will stop executing at the set rate and will instead go onto the next/previous step when the user presses the right/left arrow key. To test this, I will press these keys at different points in the execution in the visualisations for each algorithm, using a variety of datasets/graphs/constraints, and check that they function correctly.

## [10] GRAPH INPUT

Criteria [a,b,d,e] refer to usability whilst ciretia [a,b,c,d,e] refer to functionality in the Graph Input module.

- [a] A graph input space will be on the right side of the screen for Prim's and Dijkstra's algorithms. The following functionality will only work when they occur with the mouse in this space. To test this, I will try to perform the different functions both in and out of the space and check that the correct output appears on the screen, along with the programmed version of the graph being correct.
- [b] When the user double clicks in the graph input space, a vertex will be added. This vertex will have a text box that the user must enter a unique letter into. If a letter is entered that has already been used, the outline of the vertex will turn red. Once a unique letter has been entered, the user can click off the vertex or drag the vertex around the graph input space - connected edges will stay connected to the vertex. To test these functionalities, I will try to double click to create vertices in different parts of the graph input space, try entering different characters (with some being more than 1 letter, some being non-unique characters

and some being valid characters) and try dragging vertices around the graph input space. With all these tests, I will check that the function is as intended.

- [c] When 10 vertices have been added in the graph input space, double-clicking will no longer add any more vertices. I have limited graphs to this size due to the time complexity of the different algorithms. Additionally, the graph must have at least 4 vertices, which will be checked when the ‘Start Visualisation’ button is pressed for either visualisation. To test these, I will try creating graphs of different sizes and check whether the corresponding functionality and outputs are correct.
- [d] To add an edge, the user will need to hold shift on their keyboard and drag between two vertices. When this has been done, a text box will appear, connected to the edge, that will make the user input a weight for the edge. Validation will occur for a number being entered for each weight, with the edge turning red if an invalid entry has been made. To test this, I will try to connect edges between different vertices, ensuring that they are only added when intended and appear correctly, and that entering different values for the weight (valid numbers and invalid text) will result in the correct output.
- [e] To delete any element (vertex or edge), the user will need to click on the element, which should appear on the screen as highlighted, and press delete on their keyboard. To test this, I will generate different graphs and try to delete different edges and vertices, ensuring that this occurs correctly and that it is updated in the graph in the code correctly.

## DESIGN

### DECOMPOSITION OF THE PROBLEM

Decomposition is the process of breaking down a problem into smaller subproblems. The advantage of this is that it makes the problem easier to solve and program. Below is a top down model and a decomposition table of the system, including the reasons for decomposing each subsection and a brief overview of what each subsection will include - both functionality and usability. Then, each subsection will be further decomposed and a design for the section will be composed.