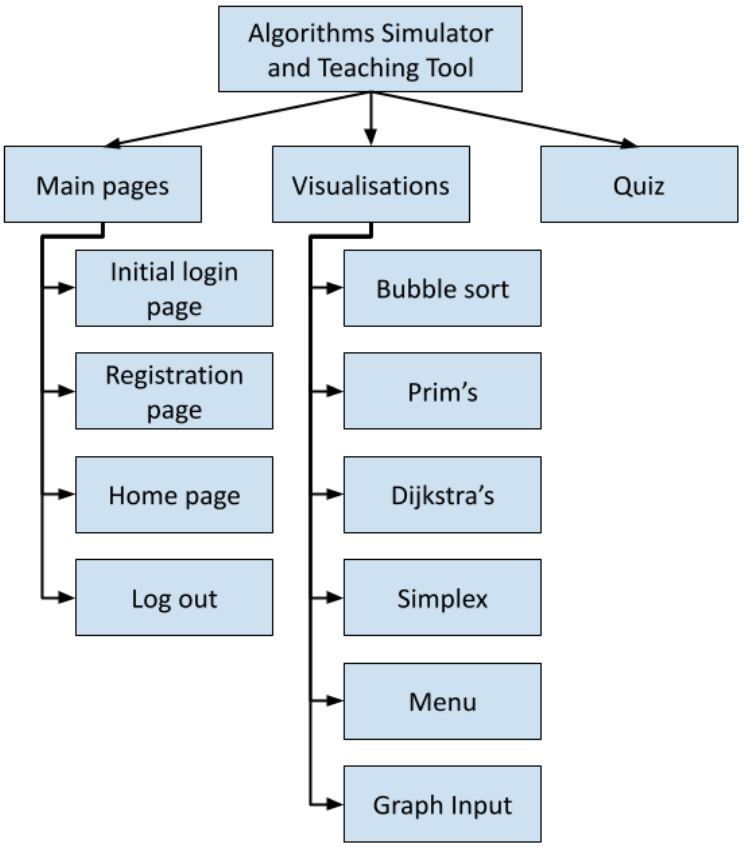
TOP DOWN MODEL



DECOMPOSITION TABLE

MODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW
Initial Login Page	This has two main sections: entering and verifying login details for existing users and accessing the registration page. Verifying the login details will require a separate subroutine for querying the database.	This page will consist of textbox entry fields for a username and password. There will also be buttons to Login and Register below these fields.
Registration Page	This has two main sections that : - fields for entering the details to create and account - validating the details and writing them to the database These sections need to be tackled separately.	This page will consist of: textbox entry fields for a username, password and password confirmation. Additionally, there will be check boxes for the 4 different algorithms implemented and a selection of a student/teacher account. Finally, there will be a Register Button and Back button.

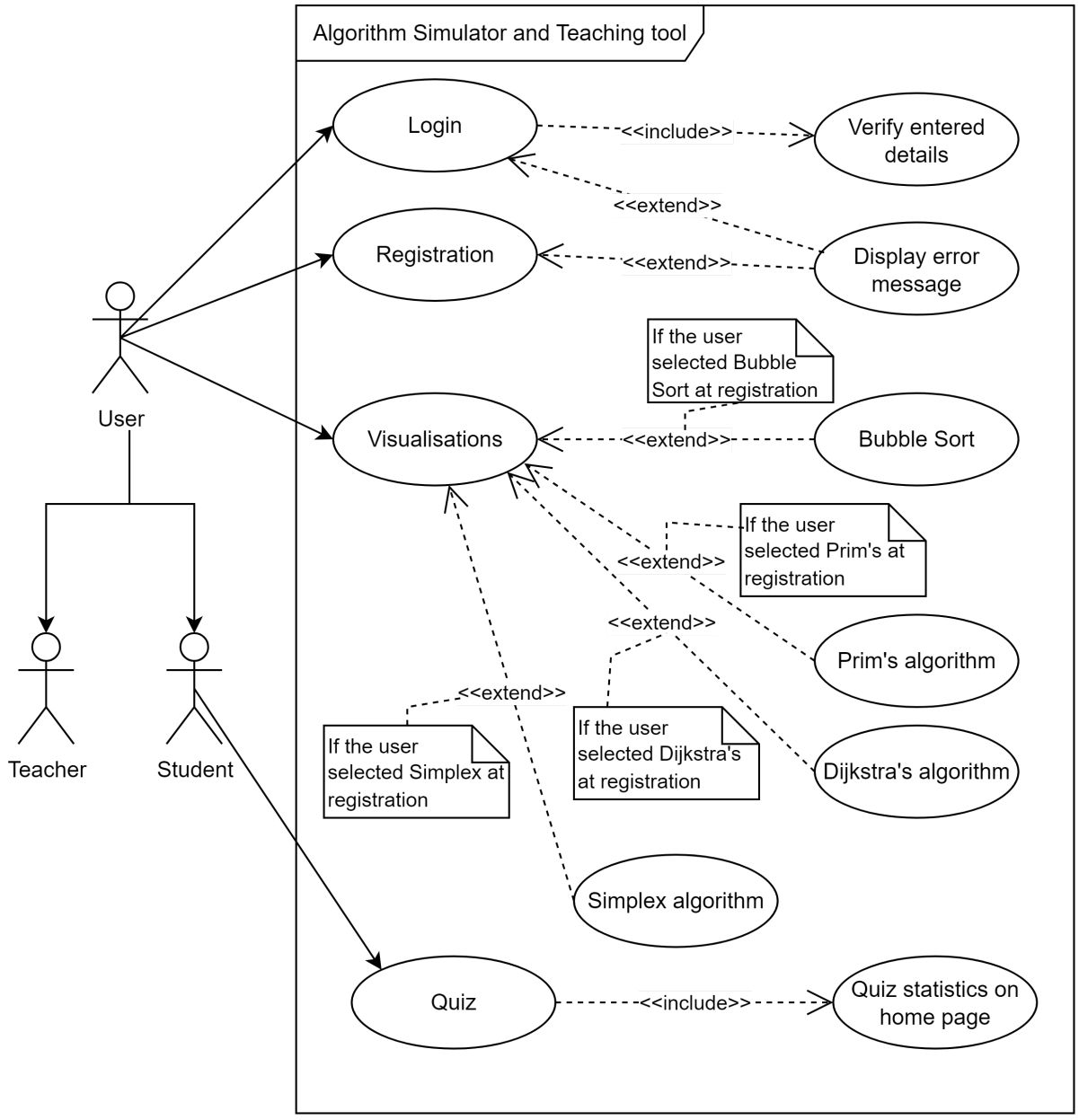
Home Page	This is the main section that users will access different algorithm visualisations from. Additionally, for student users, querying the database will be required to display the quiz statistics.	This page will consist of: a title, buttons to each selected algorithm visualisation and a logout button. If the user has a student account, there will also be a quiz button and quiz statistics bar chart.
Bubble Sort Visualisation Page	This is one subsection of the main system and has functionality that needs to be tackled separately to the rest of the system - such as displaying the results of swaps and passes on the bar chart and in text and validating the entered dataset.	This page will consist of: a title, text describing the Bubble Sort algorithm and 8 textboxes to enter a dataset of numbers to be sorted. There is also a button to add up to 4 more numbers. This unsorted list of numbers is displayed on a bar chart graph on the right hand side of the screen. Below the textboxes is the section where the outcome of each pass is shown. When the Visualise button is pressed, this begins a visualisation of the swaps and passes of Bubble Sort, with animations matching on the bar chart and the text. The execution will have the following time complexities: best case O(n), average case $O(n^2)$, worst case $O(n^2)$. It will have O(1) space complexity. Additionally, there is a 'Home' button to go back to the home page.
Prim's Algorithm Visualisation Page	This is one subsection of the main system and has functionality that needs to be tackled separately to the rest of the system - such as performing Prim's algorithm on the graph and displaying the results.	This page will consist of a title, text describing Prim's algorithm, text describing how to input a graph, a graph input section and a drop-down menu to enter the start vertex. When the visualise button is pressed, the algorithm is performed on the graph and MST edges are displayed below the start vertex. When the visualisation is performed, the vertex being visited is highlighted and at the end, all the MST edges are highlighted. The execution will have the following time complexities: best case O(E log V), average case O((V+E) log V), worst case O((V+E) log V). It will have O(E + V) space complexity.

		Additionally, there is a 'Home' button to go back to the home page.
Dijkstra's Algorithm Visualisation Page	This is one subsection of the main system and has functionality that needs to be tackled separately to the rest of the system - such as performing Dijkstra's algorithm on the graph and the table.	This page will consist of: a title, text describing Dijkstra's algorithm, text describing how to input a graph, a graph input section on the right side of the screen, a distance table on the left side of the screen, drop-down menus to enter the start and end vertices and a Visualise button. When this is clicked, the vertex being visited is highlighted on the screen and the distance table is filled in. To find the optimal route at the end, backtracking is shown on the table and the edges are highlighted on the graph. The execution will have the following time complexities: best case O(E+ V log V), average case O(E log V), worst case $O(V^2)$. It will have O(V) to O(E + V) space complexity. Additionally, there is a 'Home' button to go back to the home page.
Simplex Algorithm Visualisation Page	This is one subsection of the main system and has functionality that needs to be tackled separately to the rest of the system - such as validating entered constraints, displaying these constraints on the axes, carrying out the algorithm and displaying the results both as text and on the graph.	This page will consist of: a title, text describing the Simplex algorithm, a set of x-y axes, textboxes to enter the coefficients of the constraints and the objective function, an 'Add constraint' button and a Visualise button. When this is clicked, the constraints are plotted on the axes and a highlighted line shows the order in which the vertices of the feasible region are visited and the final values for the objective function, x and y are given at the end. The execution will have the following time complexities: best case O(m), average case O(mn), worst case $O(2^n)$. It will have O(mn) space complexity. Additionally, there is a 'Home' button to go back to the home page.
Menu	This module will be reused in all of	This is a section at the bottom of the

	the algorithm visualisation pages to allow the algorithms to be stepped through using keyboard inputs.	algorithm visualisation pages and when visualising the answer to quiz questions. It includes a 'Skip back', 'Pause' and 'Skip forward' button. Additionally, functionality to use the left and right arrow keys to step forwards and backwards through the visualisation is included for when the visualisation is paused.
Graph Input	This module will be reused in Prim's and Dijkstra's Algorithm Visualisation Pages. Therefore, a separate module, including all functionality, usability and validation required is justified.	This is a section in the Prim's and Dijkstra's algorithm sections. It includes the following functionality to input a graph: double clicking adds a vertex, clicking on a vertex and doing shift drag to another vertex adds an edge and clicking an element and pressing delete deletes that element. When adding elements, the user is prompted to enter a unique identifier for vertices and a weight for edges.
Quiz page	This subsection of the main system reuses functionality from all the algorithm Visualisation page sections but adds further functionality for users to input answers.	This page will initially include 4 buttons for the different algorithms. When pressed, each will generate a screen with a randomised dataset of 10 numbers between 1 and 50 for Bubble sort, a connected graph of 6-8 vertices with 7+ edges for Prim's and Dijkstra's algorithms and values between -10 and 10 (with both not 0) for the coefficients of the constraints and objective function for the Simplex algorithm. Then, the user will need to input the outcomes of each pass for Bubble Sort, the edges in the MST for Prim's algorithm, enter the shortest route between 2 given vertices for Dijkstra's algorithm or enter the maximum P,x,y values for the Simplex algorithm. The answer will be checked and if incorrect, the visualisation for the question will begin - see algorithm visualisation sections for this. Additionally, there is a 'Home' button to go back to the

		home page.

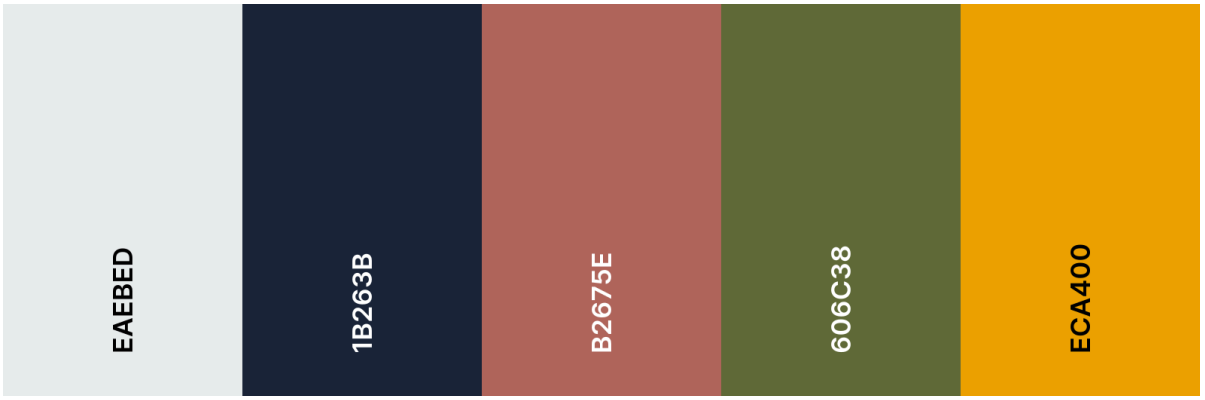
USE CASE DIAGRAM



As seen in this use-case diagram, which shows how different users - account types and algorithm selections in this case - have access to different parts of the system. For example, only student users have access and can use the Quiz functionality of this system because this is designed to test their learning, which teachers would not need. Additionally, all users can select the visualisations they want to have access to, which may be based on the exam board they study or just the algorithms available that they struggle with.

USER INTERFACE APPEARANCE

For a cohesive overall design to the system, I am planning on using the following colour scheme, with all the labels given as hex values:



I will predominantly use the blue colours - #EAEBED AND #1B263B - as the background and text colours because of the positive connotations[21] of these colours, such as them signifying intelligence, freedom, inspiration and peace. Whilst there are some negative connotations - e.g. death and mourning - in Turkey, Central Asia and China, my target demographic is students and teachers in the UK so I believe that this does not apply. The other three colours will be used in highlighting, such as showing the numbers being checked passes in Bubble Sort, highlighting the MST or shortest path for Prim's and Dijkstra's algorithms or for colour-coding the different constraints on the axes for the Simplex algorithm.

DATABASE

I originally considered using a flat file database in an unnormalised form but realised that I would have data redundancy and lack of scalability, both of which will hinder the system. Below is an example of the flat file database, with two testing accounts, I considered using:

Username	Password	Account_type	Algorithms	Correct Score	Incorrect Score
Test_accS	Testing123!	Student	Bubble Sort, Simplex	3 1	2 5
Test_accT	Testing456!	Teacher	Prim's, Dijkstra's	null	null
Billy123	1H087wow!	Student	Bubble Sort	4	7
HelloWord!!!	1m@gineThat	Student	Bubble Sort, Prim's, Dijkstra's, Simplex	2 4 5 0	6 1 4 3
123abc	cF2E/4.3U	Teacher	Prim's, Simplex	null	null
idkWhatToDo	Mi[s]XcCnj9	Teacher	Dijkstra's, Simplex	null	null

Due to the lack of scalability, I normalised its design to 1NF. Below is an example of the table, which has unique field names, values in fields being the same domain, atomic values, no identical records and a primary/composite key:

Username	Password	Account_type	Algorithm	Correct Score	Incorrect Score
Test_accS	Testing123!	Student	Bubble Sort	3	2
Test_accS	Testing123!	Student	Simplex	1	5
Test_accT	Testing456!	Teacher	Prim's	null	null
Test_accT	Testing456!	Teacher	Dijkstra's	null	null
Billy123	1H087wow!	Student	Bubble Sort	4	7
HelloWord!!!	1m@gineThat	Student	Bubble Sort	2	6
HelloWord!!!	1m@gineThat	Student	Prim's	4	1
HelloWord!!!	1m@gineThat	Student	Dijkstra's	5	4
HelloWord!!!	1m@gineThat	Student	Simplex	0	3
123abc	cF2E/4.3U	Teacher	Prim's	null	null
123abc	cF2E/4.3U	Teacher	Simplex	null	null
idkWhatToDo	Mi[s]XcCnj9	Teacher	Prim's	null	null
idkWhatToDo	Mi[s]XcCnj9	Teacher	Dijkstra's	null	null

In 1NF, the database would have a composite key made up of the Username and Algorithm fields.

Then, I normalised the database to 2NF by removing the partial dependencies in the table due to the composite key - Password and Account_type fields only depend on the username. Additionally, I created two different enrolment tables: one for teachers and one for students, which will include scores because this feature is only available to those with a Student account.

Users

PK		
Username	Password	Account_type
Test_accS	Testing123!	Student
Test_accT	Testing456!	Teacher
Billy123	1H087wow!	Student
HelloWord!!!	1m@gineThat	Student
123abc	cF2E/4.3U	Teacher
idkWhatToDo	Mi[s]XcCnj9	Teacher

Teacher Enrolment

PK	FK	
TeacherEnrollD	Username	Algorithm
1	Test_accT	Prim's
2	Test_accT	Dijkstra's
3	123abc	Prim's
4	123abc	Simplex
5	idkWhatToDo	Prim's
6	idkWhatToDo	Dijkstra's

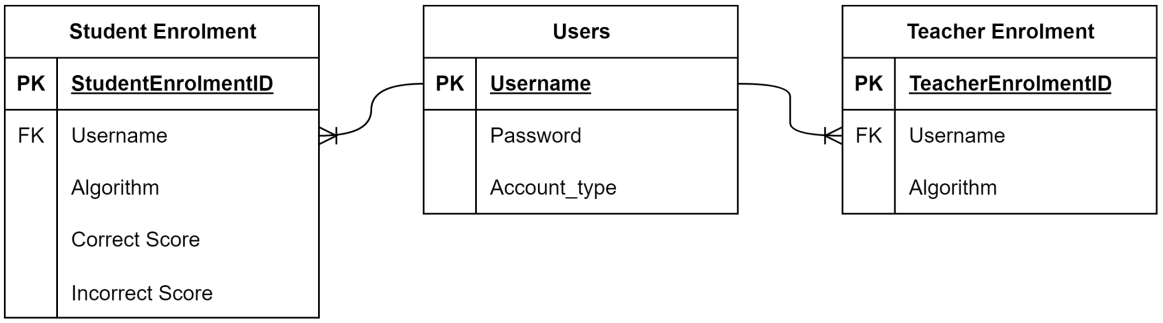
Student Enrolment

PK	FK			
StudentEnrollD	Username	Algorithm	Correct Score	Incorrect Score
1	Test_accS	Bubble Sort	3	2
2	Test_accS	Simplex	1	5
3	Billy123	Bubble Sort	4	7
4	HelloWord!!!	Bubble Sort	2	6
5	HelloWord!!!	Prim's	4	1
6	HelloWord!!!	Dijkstra's	5	4
7	HelloWord!!!	Simplex	0	3

The primary keys are shown in green. The Username fields in the Student Enrolment and Teacher Enrolment tables are foreign keys (shown in yellow) from the Users table - they are in a 1:M relationship.

The primary key, shown in yellow, for the Users table is Username. In the Algorithm Enrolment table, EnrolmentID is the primary key. Username becomes the foreign key, shown in green, in the Algorithm Enrolment table, which ensures that there is a 1:M relationship between the tables. Partial dependencies were removed and there are no transitive (non-key) dependencies so the database is in 3NF. This maintains the consistency of the data when changes are made and makes searching the data more efficient.

ENTITY RELATIONSHIP DIAGRAM



SQL QUERIES

SQL QUERY	DESCRIPTION & JUSTIFICATION	VALIDATION & JUSTIFICATION	TESTING & JUSTIFICATION
CREATE TABLE IF NOT EXISTS Users (Username TEXT NOT NULL PRIMARY KEY, Password TEXT NOT NULL, Account_type TEXT NOT NULL) ;	Creates a table to store all the registered user's data. This is required for users to be able to login.	N/A	I should be able to view this table with all the correct fields and records in DB Browser.
CREATE TABLE IF NOT EXISTS StudentEnrolment (StudentEnrolID INTEGER PRIMARY KEY AUTOINCREMENT, Username TEXT NOT NULL, Algorithm TEXT NOT NULL, CorrectScore INT NOT NULL, IncorrectScore INT NOT NULL, FOREIGN KEY (Username)	Creates a table to store student users' algorithm enrolments (algorithms they have chosen) and their quiz score for that algorithm.	N/A	I should be able to view this table with all the correct fields and records in DB Browser.

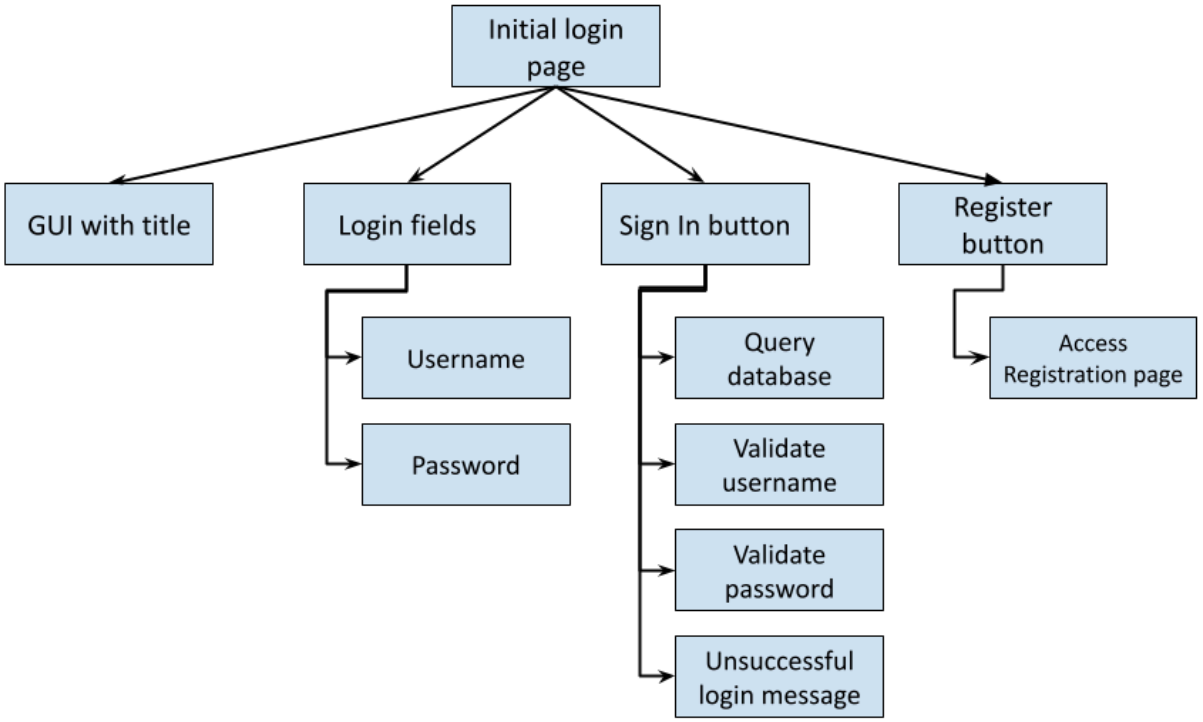
REFERENCES Users(Username)) ;			
CREATE TABLE IF NOT EXISTS TeacherEnrolment (TeacherEnrolID INTEGER PRIMARY KEY AUTOINCREMENT, Username TEXT NOT NULL, Algorithm TEXT NOT NULL, FOREIGN KEY (Username) REFERENCES Users(Username)) ;	Creates a table to store teacher users' algorithm enrollments (algorithms they have chosen).	N/A	I should be able to view this table with all the correct fields and records in DB Browser.
INSERT INTO Users (Username, Password, Account_type) VALUES (?,?,?) ;	Inserts the username, password and account type values entered by the user when registering into the Users table so that they can use this data to login.	Check that all the values have been entered. The username must be unique, the password must have at least 8 characters long, uppercase and lowercase characters, at least one number and at least one special character.	I will test this by trying to log in with the entered details after registering.
INSERT INTO StudentEnrolment (Username, Algorithm, CorrectScore, IncorrectScore) VALUES (?, ?, ?,?) ;	For a Student account, these values entered when registering will be inserted into the Student Enrolment table, with each algorithm being entered separately, allowing the correct algorithms to appear as buttons for the Visualisation buttons and for Quiz question buttons. The score is initially	The username will have been validated as being unique.	I will test this by logging in and checking that the correct buttons for the algorithms the user enroled in are displayed on the Home page and Quiz page.

	set to 0 but will be updated when quiz questions are completed correctly.		
INSERT INTO TeacherEnrolment (Username, Algorithm) VALUES (?,?) ;	For a Teacher account, these values entered when registering will be inserted into the Teacher Enrolment table, with each algorithm being entered separately, allowing the correct algorithms to appear as buttons for the Visualisation buttons	The username will have been validated as being unique.	I will test this by logging in and checking that the correct buttons for the algorithms the user enroled in are displayed on the Home page.
SELECT Username FROM Users ;	All of the Usernames are selected. Then I will search through the returned values to check that the Username is unique (when registering) or exists (for validating the Username when logging in).	N/A	I will test this by entering unique and already used usernames and checking that the correct message for unsuccessful registration is output.
SELECT * FROM Users WHERE Username = ? ;	The Username, Password and Account Type values are selected for the username entered when logging in. The password value will be used for validating the entered password. If this is valid and the home page is accessed, the account type will	The password value will be used when validating login details.	I will test this by checking that for correct usernames, correct and incorrect entered passwords are validated correctly. Additionally, I will check if the correct algorithms for the account are displayed as buttons on the home page.

	determine which table to query for the algorithms.		
UPDATE StudentEnrolment SET CorrectScore = CorrectScore + 1 WHERE Username = ? AND Algorithm = ?;	This will be for increasing the Correct Score by 1 when the user gets a question right for a certain quiz question type.	N/A	I will test this by checking that the correct score on the quiz statistics graph for that algorithm increases by 1.
UPDATE StudentEnrolment SET IncorrectScore = IncorrectScore + 1 WHERE Username = ? AND Algorithm = ? ;	This will be for increasing the Incorrect Score by 1 when the user gets a question wrong for a certain quiz question type.	N/A	I will test this by checking that the incorrect score on the quiz statistics graph for that algorithm increases by 1.
SELECT Algorithm, CorrectScore, IncorrectScore FROM StudentEnrolment WHERE Username = ? ;	This will be for displaying the quiz scores on the quiz statistics graph.	N/A	I will check that each selected algorithm is displayed on the quiz statistics graph correctly.
SELECT Algorithm FROM StudentEnrolment WHERE Username = ? ;	This will be for displaying the correct algorithms, which are selected by the user when registering, as choices for algorithm visualisations and quiz question selection.	N/A	I will check that the correct algorithms are displayed on the home page for different student accounts with different algorithm selections.
SELECT Algorithm FROM TeacherEnrolment WHERE Username = ? ;	This will be for displaying the correct algorithms, which are selected by the user when registering, as choices for algorithm visualisations.	N/A	I will check that the correct algorithms are displayed on the home page for different teacher accounts with different algorithm selections.

INITIAL LOGIN PAGE DESIGN

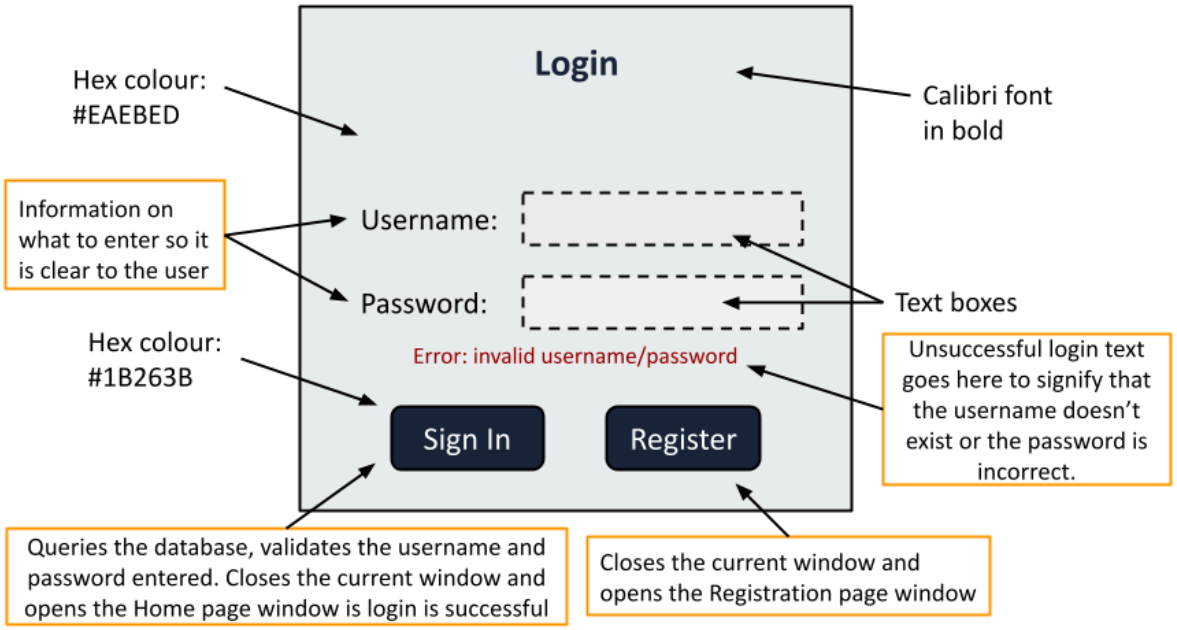
DECOMPOSITION



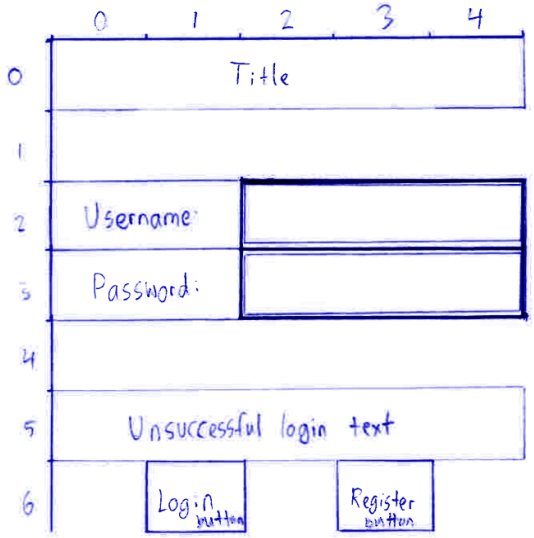
SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW
GUI with title	This will be the main page that users interact with. All the other submodules will rely on this one.	#EAEBED background colour. Text in the colour #1B263B and Calibri font. The title will be in bold at the top of the window.
Login fields	Includes the username and password fields, with the password field having placeholders. The entered values must be able to be accessed as string variables.	Username and password fields are textboxes. The password field will have placeholders. Each will be displayed with the name of the field on the left (Username or Password) and a textbox on the right.
Sign In button	This will be the main functionality submodule, therefore it is broken down into multiple sub-sections. Querying the database must occur first, which will require the use of the sqlite3 library whilst the	The Sign In button will be below the username and password entry fields in the bottom left corner. The functionality will be to query the database to check if the username exists and that the password entered

	validation sub-sections will require the use of if else statements. If the username entered does not exist in the library or the password entered is incorrect, a 'Login unsuccessful' message will appear. If validation is successful, the Initial Login page window will be closed and the Home page window will be opened. This relies on the Home page GUI having been created.	matches this. If the functionality of the Sign In button indicates a correct username and password, the current window is closed and the Home page window is opened.
Register button	This will be the way the registration page can be accessed, which will require the GUI of the Registration page to have been created.	When the user pushes this button, the Initial Login page window will close and the Registration page window will open.

GUI



As I am planning on using a grid layout for my GUI using Tkinter, below is a diagram of how the elements shown above would fit in a grid layout:



PSEUDOCODE

```
PUBLIC PROCEDURE ValidateLoginInput
      Username = value in username attribute
      Password = value in password attribute
      // check that all fields have been filled in
      IF username field is empty OR password field is empty THEN
            DISPLAY "Error: invalid username or password"
            RETURN False
      END IF
      SELECT ALL FROM Users table for given username and password
      // check if username and password are in database
      TRY
            User = SELECT * from Users for username and password
            IF user is True THEN
                  RETURN True
            ELSE
                  Display error message
                  RETURN False
      EXCEPT sqlite3.Error:
            Display error message
            RETURN FALSE
      END IF
END PROCEDURE

PUBLIC PROCEDURE loginCheck
      Validated = False
      // keep checking if validated until it is
      WHILE not Validated
            Validated = ValidateLoginInput()
      END WHILE
      CLOSE Initial login page
      OPEN Home page
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
username	string	LoginRegister class	Captures the entered user-name so that it can be validated so the user can login.	“Test_accS”
password	string	LoginRegister class	Captures the user’s entered password so that it can be validated for the user to login.	“Testing123!”
conn	sqlite3.Connection	Main and LoginRegister classes	Establishes the connection to the database for querying data.	N/A
cursor	sqlite3.Cursor	Main and LoginRegister classes	Executes SQL commands related to authenticating the user’s entries.	N/A
validated	Boolean	Local to loginCheck method	Stores the state of whether the login details have been validated so that the program will know whether to allow the user to continue to the Home page.	N/A

username	string	Local to valdiateLoginInput method	Holds a copy of the username entered by the user to allow it to be used in the validateLoginInput method.	N/A
password	string	Local to valdiateLoginInput method	Holds a copy of the password entered by the user to allow it to be used in the validateLoginInput method.	N/A
user	Tuple or None	Local to valdiateLoginInput method	Holds the results of the SQL query to allow them to be used when checking the Username and Password.	N/A

DATA STRUCTURES TABLE

NAME	DATA STRUCTURE	SCOPE	PURPOSE & JUSTIFICATION
LoginRegister	Class	Global	Manages the login and registration processes, including the interaction with the user input and the database. A class is used because it allows for the use of methods and attributes specific to the user trying to log in.

conn	sqlite3.Connection	LoginRegister and Main classes attribute	Establishes the database connection for validating the details entered by the user.
cursor	sqlite3.Cursor	LoginRegister and Main classes attribute	Executes SQLcommands for querying the database. This is required because the database must be able to be accessed for selecting values to allow the functionality of logging in to take place.
user	Tuple or None	Local to validateLoginInput method	Holds the result of the SQL query so that it can be checked for whether the user's inputs are in the database - which is for checking that the user already has an account and doesn't need to register.

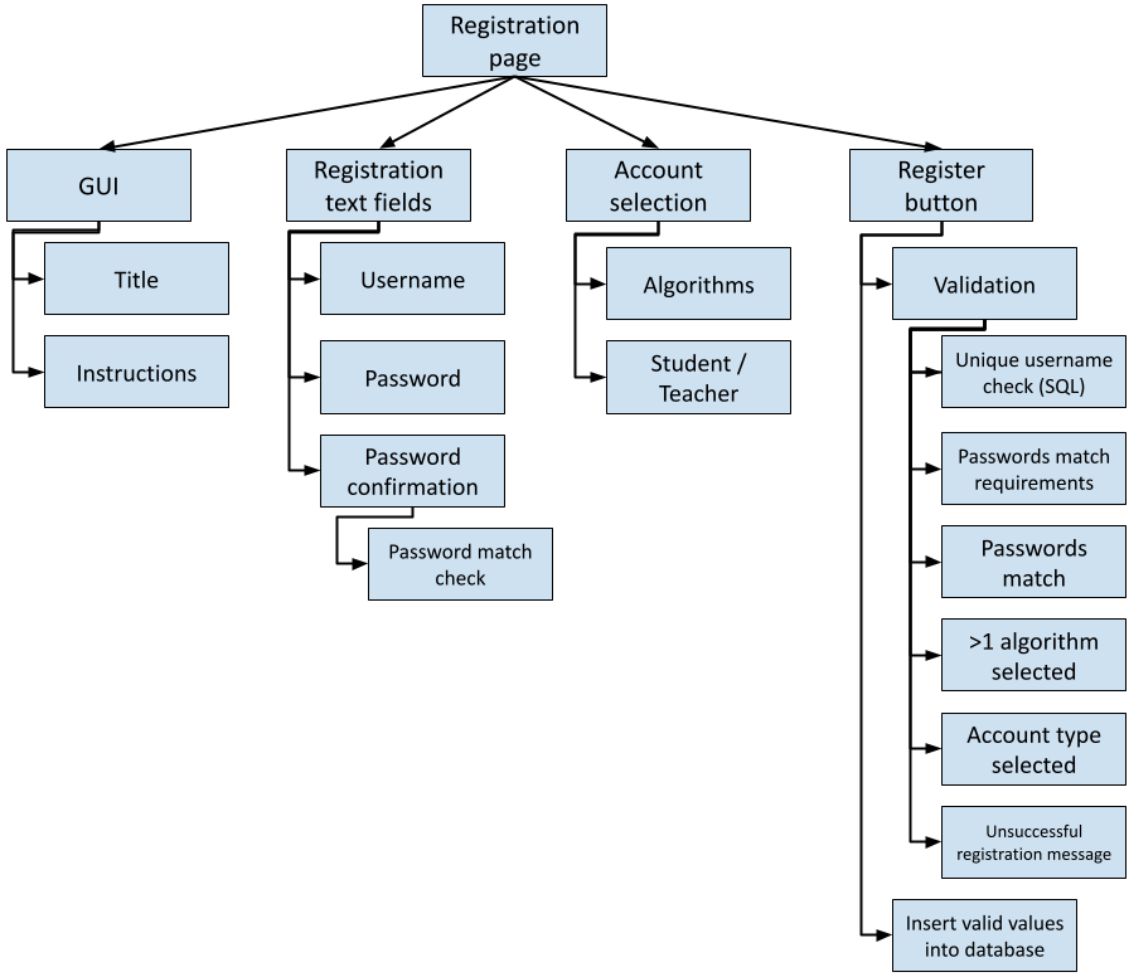
WHITE BOX TEST DATA

This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL RESULT
Log in without filling in any details.	The user should be prompted with the 'Unsuccessful login' message.	This ensures that the user must complete the full form to log in.	
Log in with a partially filled in form. E.g. username: Test_accT password blank	The user should be prompted with the 'Unsuccessful login' message.	This ensures that the user must complete the full form to log in.	
Log in with a fully filled in form but the username does not exist. E.g. username: Karu0807 password: Testing123!	The user should be prompted with the 'Unsuccessful login' message.	This ensures that the user enters the correct username and cannot log in otherwise.	
Log in with a fully filled in form but the password is incorrect for the username. E.g. username: Test_accS password: Tere_talv7	The user should be prompted with the 'Unsuccessful login' message.	This ensures that the user must enter the correct password and cannot login otherwise.	
Log in with a correct username and password. E.g. username: Test_accS password: Testing123!	The user should be logged in and redirected to the Home page.	This ensures that the user can log in if the form is completed correctly.	
Press the Register button.	Should redirect the user to the Register page.	This ensures that the 'Register' button works correctly.	

REGISTRATION PAGE DESIGN

DECOMPOSITION

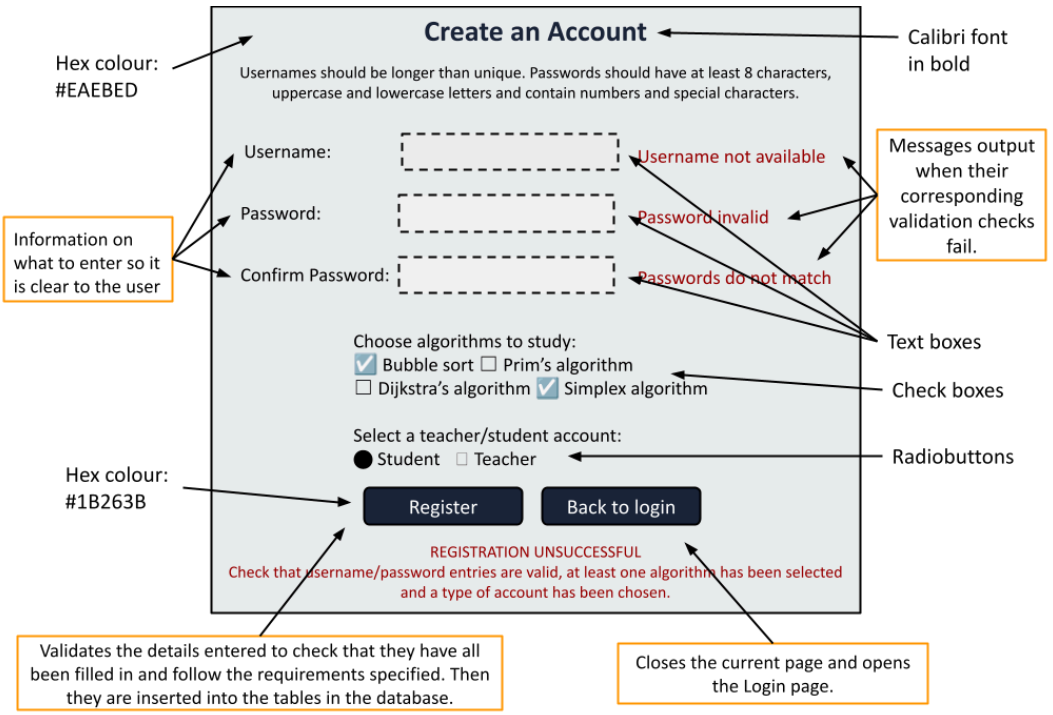


SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW
GUI with title and instructions	This will be the main page that users interact with. All the other submodules will rely on this one.	#EAEBED background colour. Text in the colour #1B263B and Calibri font. The title will be in bold at the top of the window.
Text fields	This includes the username, password and password confirmation fields - with the password fields having placeholders. The entered values must be able to be accessed as string variables.	Username, password and password confirmation fields are textboxes, each with their own label. The password fields have placeholders.

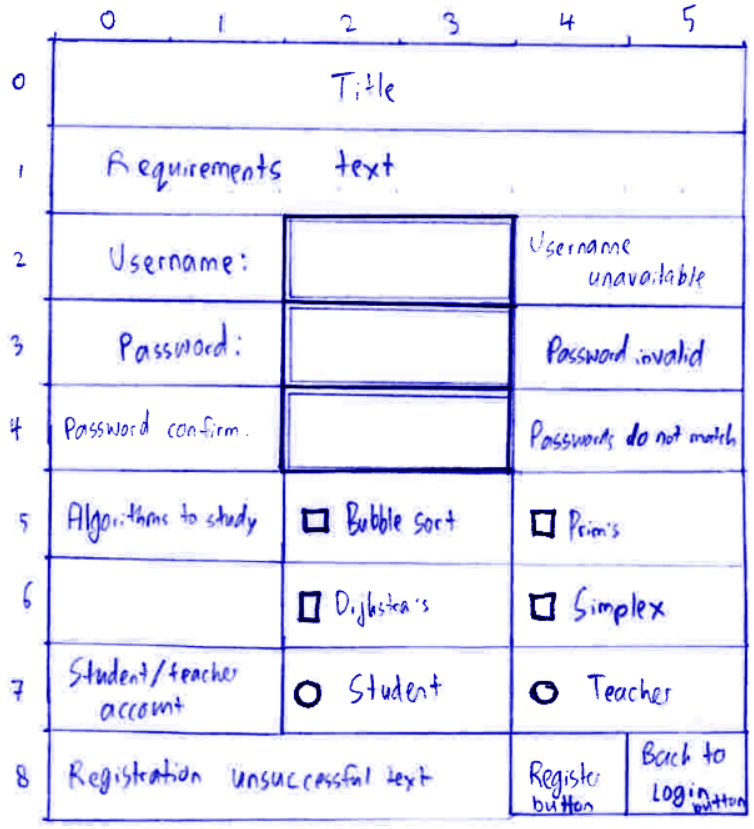
Algorithm selection	This is the way that the user can make their choice of algorithms to study, which is done using checkboxes for each algorithm.	There is a label saying 'Algorithm selection:' and checkboxes for each algorithm chosen. These will be held in variables in an array.
Student/Teacher selection	This is the way that the user can make their choice of account type, which is done using a selection.	There is a label saying 'Account type:' and radiobuttons for the two options.
Register button	This is the button that allows the user to submit the entered details. It causes validation and then inserting values into the database to begin.	This is a button element labelled 'Register'
Unique username check	This is the first part of the validation of the details. It is required as each account must be uniquely identifiable for searching in the tables.	An SQL query is performed searching for the username entered. If nothing is returned, the username has not been used already so is unique and passes the test. If an entry is returned, the validation test fails and this causes the 'Username not available' and 'Registration unsuccessful' messages to be output.
Passwords match check	This is the second part of the validation of the details. It is required as the passwords must be matching to ensure the user has entered their correct intended password.	An if condition is performed on the data from the password and password confirmation fields to ensure they are equal to each other. If they are, this validation test passes. Otherwise, this causes the 'Passwords do not match' and 'Registration unsuccessful' messages to be output.
Password valid check	This is the third part of the validation of the details and is required to check that the password is strong enough.	An if condition on the following requirements is performed: at least 8 characters long, both uppercase and lowercase characters used, at least one number and at least one special character. If this test outputs true, this validation test passes. Otherwise, the 'Password invalid' and 'Registration unsuccessful' messages to be output.
>1 algorithms selected	This is the fourth part of the validation of the details and is required so that algorithms will	An if condition will be used to check that the list of algorithm choices includes at least 1 algorithm type. If

	appear as options on the home screen and as quiz questions.	this test outputs true, this validation test passes. Otherwise, the unsuccessful registration message is output.
Account type chosen	This is the final part of the validation of the details and is required to determine whether to include the quiz functionality or not.	An if condition will be used to check an account type value has been returned from the selection. If this test outputs true, this validation test passes. This means that the details are fully validated and the program moves onto inserting the validated values into the database. Otherwise, the unsuccessful registration message is output.
Unsuccessful registration message	This is the message displayed when a validation check fails and it alerts the user to check their details.	The message will be displayed on the page and state that the registration was unsuccessful.
Insert validated values into database	This is the functionality of the registration button that performs an SQL query to insert the username, password, algorithm choices and account type into the corresponding tables.	The username, password and account type are inserted into the Users table. For a student account, for each algorithm chosen, the username, algorithm and correct/incorrect scores (0) are inserted as separate entries into the StudentEnrolment table. For a teacher account, just the username and each algorithm are inserted into the TeacherEnrolment table as separate entries.
Back to login button	This is the way the user can access the Login page again.	When the user pushes this button, the Registration page will close and the Login page will open.

GUI



As I am planning on using a grid layout for my GUI using Tkinter, below is a diagram of how the elements shown above would fit in this layout:



PSEUDOCODE

```
PUBLIC PROCEDURE validateUsernameReg
      SELECT Username FROM Users WHERE Username = entered username
      IF username matching found THEN
            RETURN False
      END IF
END PROCEDURE


PUBLIC PROCEDURE validatePasswordReg
      IF len(entered password) >= 8 AND includes uppercase and lowercase
characters AND contains at least 1 number AND contains at least 1 special
character THEN
            RETURN TRUE
      END IF
      RETURN FALSE // True if meets password criteria, False otherwise
END PROCEDURE


PUBLIC PROCEDURE registerCheck
      Valid = False

      // validate username
      validUsername = False
      WHILE not validUsername
            validUsername = validateUsernameReg()
            IF validUsername == False THEN
                  DISPLAY "Username not available"
            END IF
      END WHILE

      // validate password
      validPassword = False
      WHILE not validPassword
            validPassword = validatePasswordReg()
            IF passwords do not match AND not Validated THEN
                  DISPLAY "Passwords do not match"
                  DISPLAY "Invalid password"
            ELSE IF passwords do not match THEN
                  DISPLAY "Passwords do not match"
            ELSE IF not validPassword THEN
                  DISPLAY "Invalid password"
            ELSE
                  validPassword = True
            END IF
      END WHILE

      // validate algorithm and account type selection
      IF at least 1 algorithm selected THEN
            validAlgorithms = True
      END IF
      IF account type selected THEN
            validAccount = True
      END IF

      // all valid - enter into tables - go to home page
      // anything invalid - display error message
      IF validUsername AND validPassword AND validAlgorithms AND
      validAccount THEN
```

```
            Valid = True
            ENTER username, password, account type into Users
            IF account type == Student THEN
                  FOR algorithm IN Algorithms
                        ENTER username, algorithm, 0, 0 into
      StudentEnrolment table // the numbers for correct and incorrect quiz
      scores
                  NEXT algorithm
            ELSE
                  FOR algorithm in Algorithms
                        ENTER username, algorithm into TeacherEnrolment
      table
                  NEXT algorithm
            END IF
            CLOSE Registration page
            OPEN Home page
      ELSE
            DISPLAY "Unsuccessful registration - check valid username and
      password and that at least 1 algorithm and an account type have been
      selected"
      END IF
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
conn	sqlite3.Connection	Main and LoginRegister classes	Establishes the connection to the database for querying data.	N/A
cursor	sqlite3.Cursor	Main and LoginRegister classes	Executes SQL commands related to authenticating the user's entries.	N/A
newPassword	string	Attribute of LoginRegister	Stores the password entered by the user in the registration form so that they can be checked for whether it meets the requirements before allowing them to register.	"tEst?1324"
newConfirmpassword	string	Attribute of LoginRegister	Stores the confirmation password entered by the user in the registration form so that it can be inserted into the database.	"tEst?1324"
bubbleSortSelect	integer	Attribute of LoginRegister	Tracks whether Bubble Sort is selected - 1 is selected, 0 is not. This is because the	1

			algorithms selected must be inserted into the database.	
primSelect	integer	Attribute of LoginRegister	Tracks whether Prim's algorithm is selected - 1 is selected, 0 is not. This is because the algorithms selected must be inserted into the database.	0
dijkstraSelect	integer	Attribute of LoginRegister	Tracks whether Dijkstra's algorithm is selected - 1 is selected, 0 is not. This is because the algorithms selected must be inserted into the database.	1
simplexSelect	integer	Attribute of LoginRegister	Tracks whether the Simplex algorithm is selected - 1 is selected, 0 is not. This is because the algorithms selected must be inserted into the database.	0
accountType	string	Attribute of LoginRegister	Stores the selected account type - Student or Teacher. This is so that it can be decided whether to insert the algorithm values into the StudentEnrolment or	"Student"

			TeacherEnrolment table.	
usernameInvalid	Label	Attribute of LoginRegister	Displays an error message if the entered username is unavailable. This is because the usernames must be unique to allow for them to be used as the primary key in the Users table.	N/A
passwordInvalid	Label	Attribute of LoginRegister	Displays an error message if the entered password is invalid. The password must meet all the minimum requirements for a secure password for safety.	N/A
passwordsMatchInvalid	Label	Attribute of LoginRegister	Displays an error message if the entered passwords do not match. This is because the password chosen must be confirmed to be the one intended by the user.	N/A
invalidMessageReg	Label	Attribute of LoginRegister	Displays an error message if any registration field if invalid. This is because all entry fields must be filled in for values to be entered into the database.	N/A

usernameHolder	string	Local to registerCheck and validateUsernameReg	Holds a copy of the newUsername value entered. This is to allow it to be checked whether it is in the database or not (ie whether it is unique).	N/A
passwordHolder	string	Local to registerCheck and validatePasswordReg	Holds a copy of the newPassword value entered so that it can be checked for the minimum requirements and whether it matched the confirmation password.	N/A
confirmPasswordHolder	string	Local to registerCheck	Holds a copy of the newConfirmPassword value entered so that it can be checked against the passwordHolder in order to make sure that the user has entered their intended password.	N/A
algorithmValues	list[integer]	Local to registerCheck	Holds a copy of the values of the algorithmsChosen list so that they can be entered into the enrolment tables in the database.	N/A
usernameFound	tuple	Local to	Holds the results of the SQL	N/A

		validateUsernameReg	query so that it can be checked for whether the a matching username as been found, meaning that the username is not unique.	

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
algorithmsChosen	list	Attribute of LoginRegister	Holds the values 1 or 0 for each algorithm type so that only the chosen ones are entered into the database.
LoginRegister	Class	Global	Manages the login and registration processes, including the interaction with the user input and the database. A class is used because it allows for the use of methods and attributes specific to the user trying to register.
conn	sqlite3.Connection	LoginRegister and Main classes attribute	Establishes the database connection for validating the details entered by the user.
cursor	sqlite3.Cursor	LoginRegister and Main classes attribute	Executes SQLcommands for querying the database so that the

			username can be validated as being unique and for inserting validated entries into the database.
algorithmValues	list	Local to registerCheck	Holds a copy of the values of the algorithms chosen so that it can be checked that at least 1 algorithm has been chosen.
usernameFound	tuple	Local to validateUsernameReg	Holds the results of the SQL query for matching usernames so that it can be checked whether a value has been returned and that the username entered is unique or not.

WHITE BOX TEST DATA

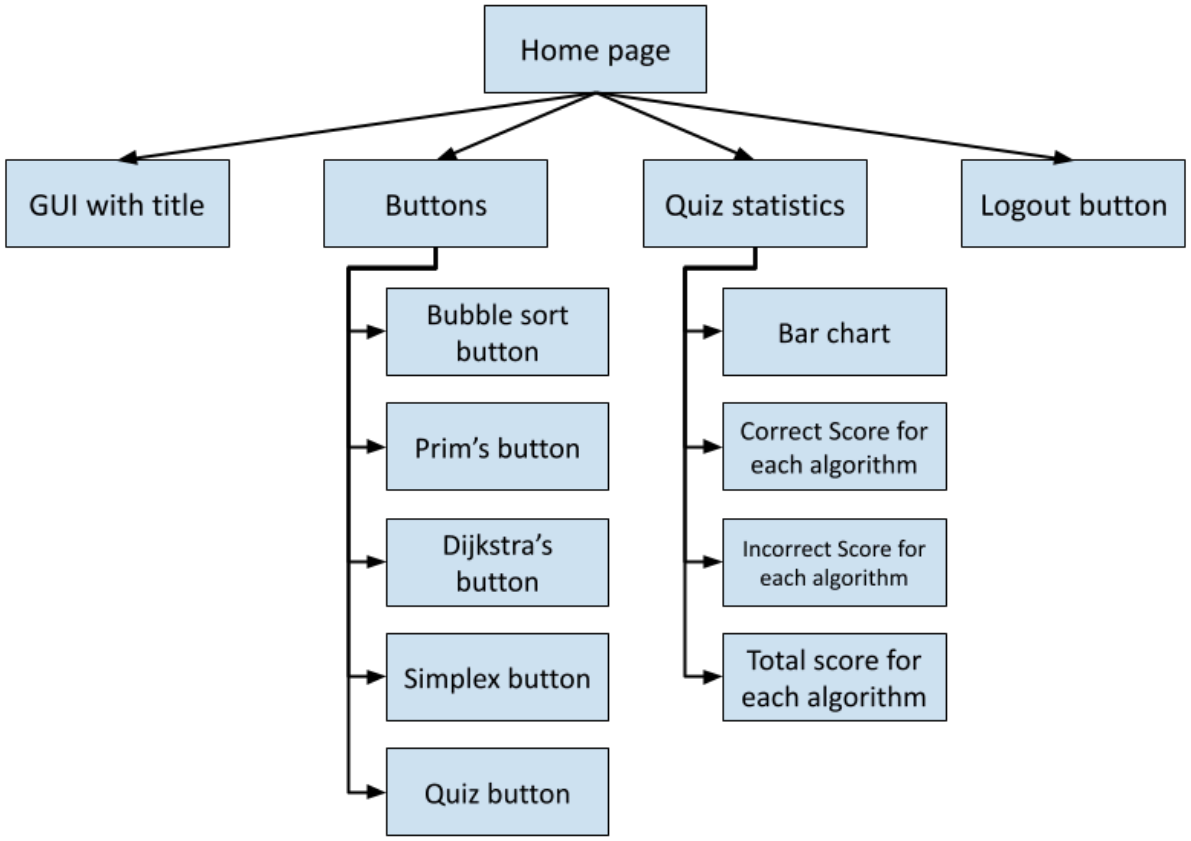
This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Register without filling in any details.	The user should be prompted with the 'Unsuccessful registration' message.	This ensures that the form is filled in fully so that all necessary details are obtained.	
Register with a partially filled out form.	Output depends on what has been filled in but should expect for the user to be prompted with the 'Unsuccessful registration' message.	This ensures that the form is filled in fully so that all necessary details are obtained.	
Register with an already used username.	The user should be prompted with the 'Username unavailable' message.	This ensures that the user has entered a unique username that can be used to query the database with.	
Register with a password that is less than 8 characters.	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	
Register with a password that is 8 characters long.	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	
Register with a password that does not contain both uppercase and lowercase characters.	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	
Register with a password that does not contain a number.	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	
Register with a password that does	The user should be prompted with the	This ensures that the user enters a strong	

not contain a special character.	'Password invalid' message.	password that meets the requirements.	
Register with the password and password confirmation fields not matching.	The user should be prompted with the 'Passwords do not match' message.	This ensures that the user must complete these fields correctly so they match and the user is sure of their password.	
Register without selecting an algorithm to study.	The user should be prompted with the 'Unsuccessful registration' message.	This ensures that at least one algorithm must be chosen.	
Register without choosing an account type.	The user should be prompted with the 'Unsuccessful registration' message.	This ensures that an account type must be selected.	
Register with fully correctly filled in details.	Registration should be successful so the registration window should be closed and the home page window should be opened.	This ensures that the user must enter fully correct details and cannot register otherwise.	
Press the 'Back to Login' button.	Should redirect the user to the Login page.	This ensures the 'Back to Login' button works correctly.	

HOME PAGE DESIGN

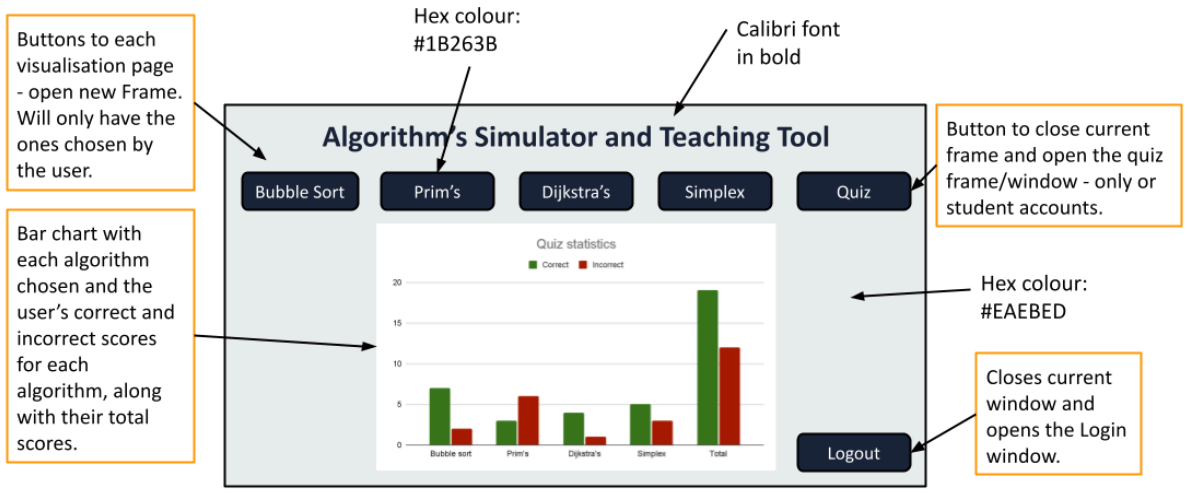
DECOMPOSITION



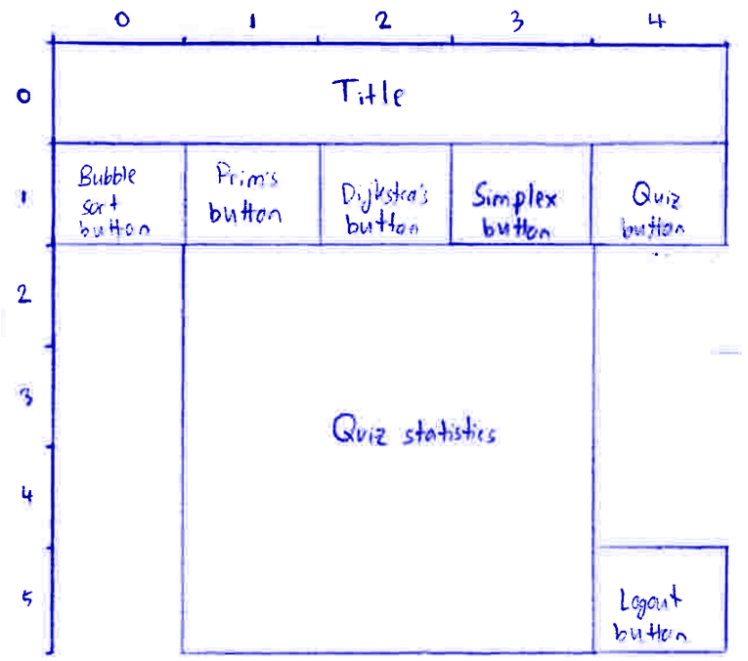
SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW
GUI with title	This will be the main page that users interact with. All the other submodules will rely on this one.	#EAEBED background colour. Text in the colour #1B263B and Calibri font. The title will be in bold at the top of the window.
Buttons for each algorithm and quiz button	Each of these buttons opens the visualisation/quiz window, which requires each of them to have been created.	#1B263B background colour and text in white. All buttons are in a line below the title.
Quiz statistics bar chart	This is the main section of the home page and requires the embedding of a matplotlib bar chart in a Tkinter window.	This will only be displayed for student accounts and will have each algorithms' correct and incorrect scores displayed.
Bar chart with	This will be one section of the quiz	The bars for each algorithm will be

Correct and Incorrect Scores next to each other for each algorithm	statistics bar chart and will require the querying of the database.	next to each other to make clear distinctions between algorithms and the correct and incorrect bars have different colours.
Bar chart with Total Correct and Incorrect Scores	This will be the other section of the quiz statistics bar chart and will require the results of the database query to be added up.	These bars are to show overall success rates. The correct and incorrect bars will be different colours.
Logout button	This will be the way the home page can be closed and the initial login page be opened again.	This button will be in the bottom right side of the screen. #1B263B background colour and text in white.

GUI



As I am planning on using a grid layout for my GUI using Tkinter, below is a diagram of how the elements shown above would fit in this layout:



PSEUDOCODE

```
// methods to close the current page and open the algorithm/quiz page
PUBLIC PROCEDURE newBubbleSortPage
      Close Home page Frame
      Open Bubble sort Frame
END PROCEDURE

PUBLIC PROCEDURE newPrimPage
      Close Home page Frame
      Open Prim's Frame
END PROCEDURE

PUBLIC PROCEDURE newDijkstraPage
      Close Home page Frame
      Open Dijkstra's Frame
END PROCEDURE

PUBLIC PROCEDURE newSimplexPage
      Close Home page Frame
      Open Simplex Frame
END PROCEDURE

PUBLIC PROCEDURE newQuixPage
      Close Home page Frame
      Open Quiz Frame
END PROCEDURE

PUBLIC PROCEDURE locations
      // find grid locations to put buttons in
```

```
      buttonLocations = []
      Iterate through algorithms chosen dictionary
            IF value == 1 THEN
                  Append key to buttonLocations
      IF account == "Student" THEN
            Append "Quiz" to buttonLocations
      RETURN buttonLocations
END PROCEDURE

PUBLIC PROCEDURE algorithmAxisValues (listType)
      // find labels to go on quiz statistics axes
      valuesList = []
      Execute SQL "SELECT Algorithm, CorrectScore, IncorrectScore FROM
StudentEnrolment WHERE Username=?", (currentAccUsername)
      IF listType == "Name" THEN
            Iterate through query results
                  Append first index to valuesList
            Append "Total" to valuesList
      ELSE IF listType == "Correct" THEN
            totalCorrect = 0
            Iterate through query results
                  Append second index to valuesList
                  totalCorrect += value
            Append totalCorrect score to valuesList
      ELSE IF listType == "Incorrect" THEN
            totalIncorrect = 0
            Iterate through query results
                  Append third index to valuesList
                  totalIncorrect += value
            Append totalIncorrect score to valuesList
      RETURN valuesList
END PROCEDURE

PUBLIC PROCEDURE plotStatistics
      // use geeksforgeeks.org for basis of this code
      Figure, axes = plt.subplots(figure size = (width, height) // decide
on values for width and height in context of GUI
      x = range(len(algorithmNames)) // x is a counter for where to place
the bars
      axes x label = "Algorithms"
      axes y label = "Score"
      title = "Quiz statistics"
      xticks([p + 0.2 for p in x]) // places the bars next to each other
      Xticklabels name = algorithmNames

      // embed in Tkinter window
      Canvas = figure in master frame
      canvasWidget = tkinter widget for canvas
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
currentAccUsername	string	HomeMain class	Stores the username of the currently logged in user so that the database can be searched.	“Test_accS”
accountType	string	HomeMain class	Stores the account type of the currently logged in user so that the correct enrolment table can be searched for algorithms.	“Teacher”
algorithmNames	List[string]	HomeMain class	Stores the names of the algorithms chosen, which are to be used for displaying the correct buttons.	[“Bubble Sort”, “Prim’s”]
correctScores	List[integer]	HomeMain class	Stores the correct scores of the different algorithms for the current user so that they can be displayed on the quiz statistics chart.	[10, 5, 2]
incorrectScores	List[integer]	HomeMain class	Stores the incorrect scores of the different algorithms for the current user so that	[3, 2, 0]

			they can be displayed on the quiz statistics chart.	
homeFrame	Frame	HomeMain class	Represents the frame that the GUI elements go in. This organises them and their grid layouts.	N/A
conn	sqlite3.Connection	Main and HomeMain classes	Establishes the connection to the database for querying data.	N/A
cursor	sqlite3.Cursor	Main and HomeMain classes	Executes SQL commands related to authenticating the user's entries.	N/A

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
algorithmNames	List[string]	Attribute of HomeMain	Holds the names of the algorithms selected by the user so that they can be used for the button texts.
correctScores	List[integer]	Attribute of HomeMain	Holds the correct quiz scores of the current user for each chosen algorithm so that they can be displayed on the quiz statistics chart.

incorrectScores	List[integer]	Attribute of HomeMain	Holds the incorrect quiz scores of the current user for each chosen algorithm so that they can be displayed on the quiz statistics chart.
buttonLocations	List	Local to locations method	Temporarily holds the algorithm names and scores for displaying so that they can be used for calculating the grid placement of the buttons.
HomeMain	Class	Global	Manages the quiz statistics for the current user (if student) and displays the correct buttons for the different algorithm visualisations (and quiz). A class is used because it allows for the use of methods and attributes specific to the user that has logged in.
conn	sqlite3.Connection	HomeMain and Main classes attribute	Establishes the database connection for validating the details entered by the user.
cursor	sqlite3.Cursor	HomeMain and Main classes attribute	Executes SQLcommands for querying the database.

WHITE BOX TEST DATA

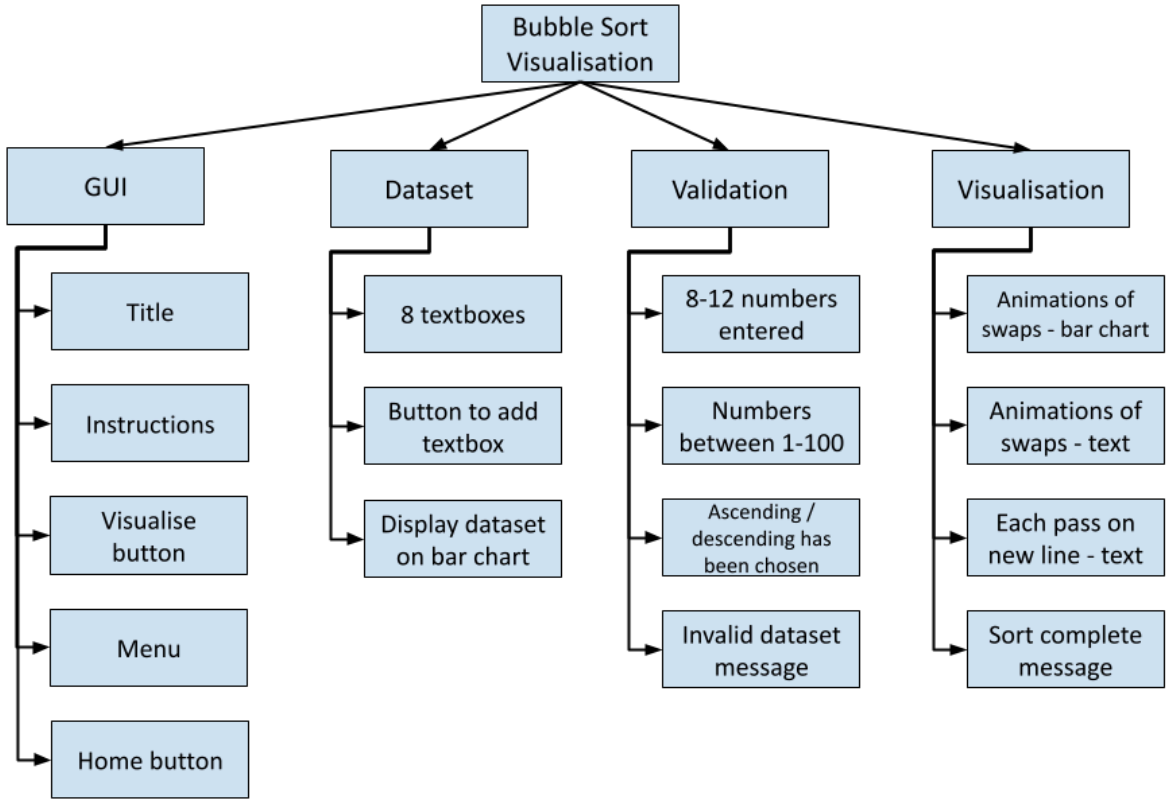
This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Log in with Student account	Home page GUI with selected algorithms buttons and quiz statistics chart displayed	This ensures that a student user has the correct algorithms that they registered with as options to visualise and quiz functionality, which includes a quiz statistics chart and a quiz button. Additionally, there should be a Logout button.	
Log in with Teacher account	Home page GUI with selected algorithms buttons displayed	This ensures that a teacher user has the correct algorithms that they registered with as options to visualise. Additionally, there should be a Logout button.	
Log in with multiple Student accounts with different algorithms choices and quiz scores already entered.	The correct algorithms are displayed as buttons and the quiz statistics are displayed correctly on the chart.	This ensures that the functionality is as expected for all student users for whatever choice of algorithms they had made when registering and whatever scores they have from the quiz section.	
Log in with multiple Teacher accounts with different algorithms choices.	The correct algorithm buttons are displayed.	This ensures that the functionality is as expected for all teacher users for whatever choice of algorithms they had	

		made when registering.	
Press the Bubble Sort button	The home page window is closed and the Bubble Sort window is opened.	This ensures that the Bubble Sort button works correctly.	
Press the Prim's Algorithm button	The home page window is closed and the Prim's algorithm window is opened.	This ensures that the Prim's algorithm button works correctly.	
Press the Dijkstra's Algorithm button	The home page window is closed and the Dijkstra's algorithm window is opened.	This ensures that the Dijkstra's algorithm button works correctly.	
Press the Simplex Algorithm button	The home page window is closed and the Simplex algorithm window is opened.	This ensures that the Simplex algorithm button works correctly.	
Press the Quiz button	The home page window is closed and the Quiz window is opened.	This ensures that the Quiz button works correctly.	
Press the Logout button	The home page window is closed and the Initial Login window is opened.	This ensures that the Logout button works correctly.	

BUBBLE SORT VISUALISATION PAGE DESIGN

DECOMPOSITION

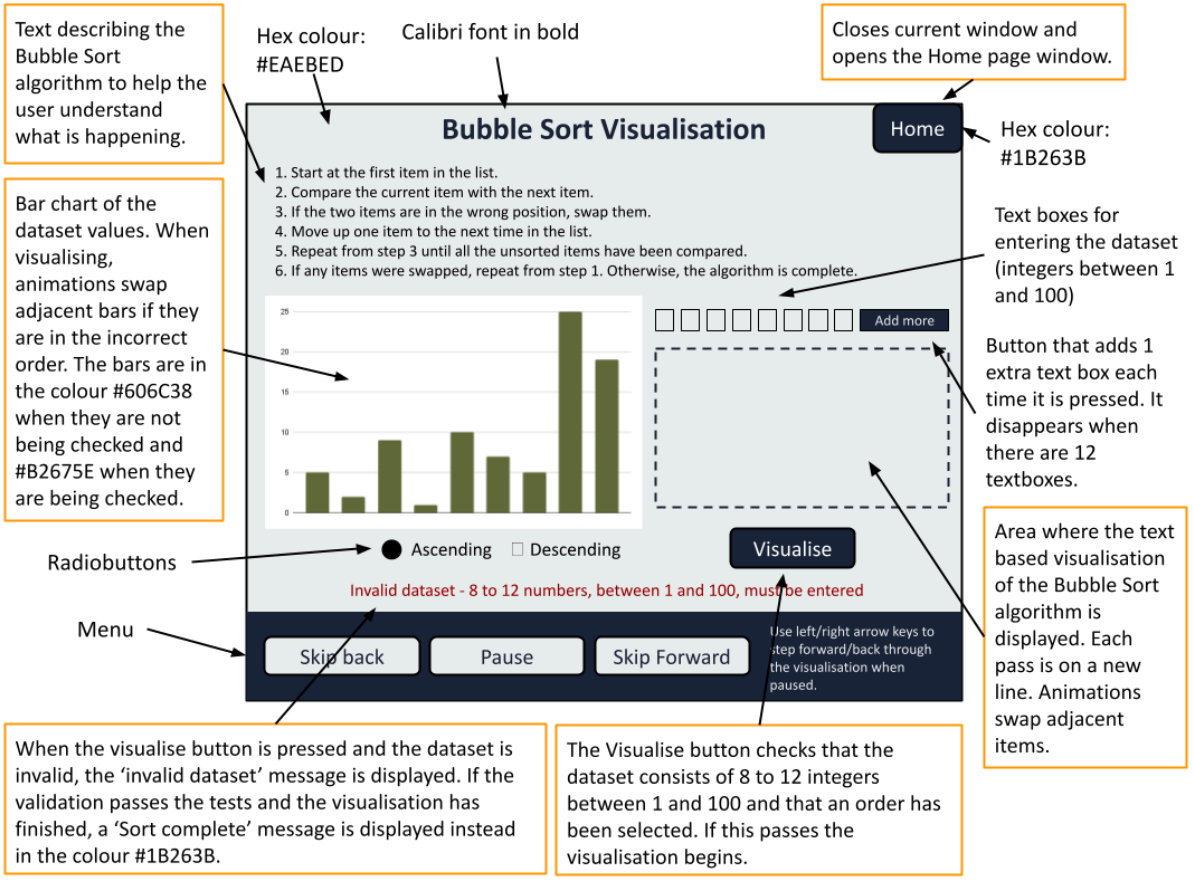


SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW
GUI with title, instructions and 'Visualise' and 'Home' buttons.	This will be the main page that users can interact with. All other submodules rely on this one.	#EAEBED background colour. Text in the colour #1B263B and Calibri font. The title will be in bold at the top of the window. Buttons will have background colour #1B263B and text colour as white. The Visualise button will begin the validation of the entered data and then visualisation of the algorithm. The 'Home' button will close the current page and open the Home page.
8 textboxes for the dataset	This will be the way that the dataset (consisting of integers) can be entered by the user.	The textboxes will be next to each other in a line. Their values will be written to an array.
Button to add textboxes	This will be the way that up to 4 more numbers can be added to the	When pressed, an extra textbox will appear next to the row of textboxes.

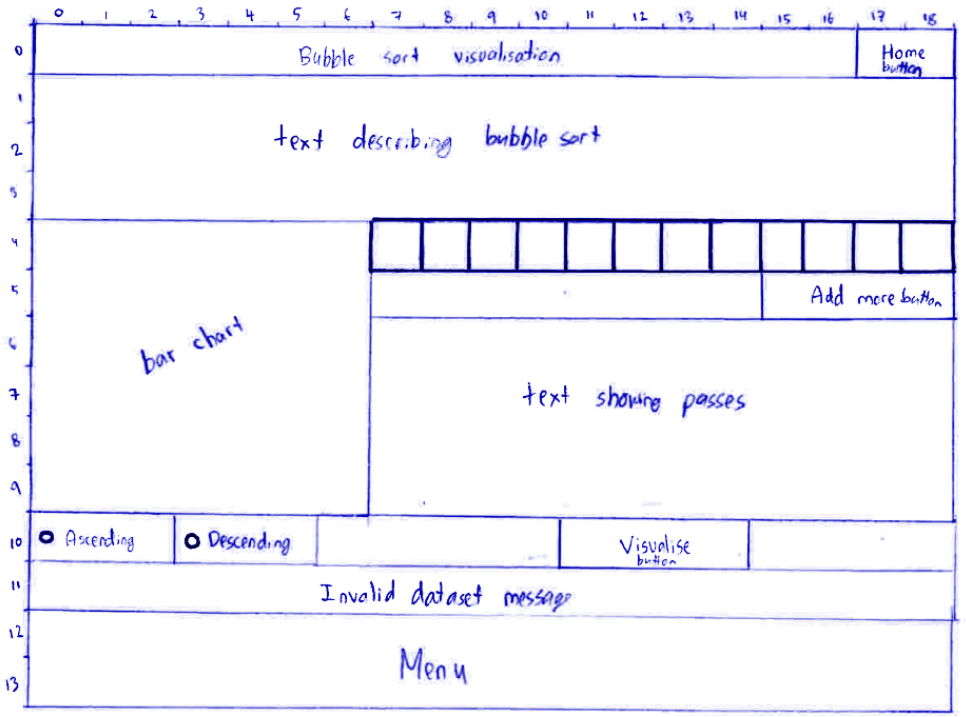
	dataset.	When the maximum number of textboxes has been reached, this button will disappear.
Dataset displayed on bar chart	This will be the way that the bar chart can be set up to be used for visualising Bubble Sort.	The numbers will be displayed in the order that they were entered. This bar chart will require the use of the Matplotlib library and will be displayed on the left side of the page. The bars will be in the colour #606C3B.
Validate that dataset has 8-12 numbers between 1 and 100.	This will be the first part of validating the inputs and is required to ensure that the dataset can be displayed clearly on the bar chart whilst also having enough numbers for users to understand the algorithm.	An if condition is performed on the length of the array storing the numbers to check that it is ≥8 and ≤12. To check that numbers are between 1 and 100, the values entered will be iterated through to check that each is between 1 and 100 (inclusive).
Ascending/descending choice made	This will be the second part of validating the inputs and is required to ensure that a choice of final order is made.	An if condition is performed on value passed into the attribute for the Radiobuttons.
Message displayed if the dataset is invalid	This is the message displayed when a validation check fails and it alerts the user to check the values entered. The type of error will be displayed.	Each validation section will be performed in an if condition. If it fails, the error type will be displayed in the message. The text will be in red.
Animations of the swaps in the Bubble Sort algorithm on the bar chart	This will be the first section of the visualisation of the Bubble Sort algorithm. It will synchronise with the text based animations.	Adjacent items will be swapped if they are in the incorrect order. If sorted to an ascending order, the larger item will move one space right and the smaller item will move one space left if they are in the wrong order. The current items being checked will be highlighted in the colour #B2675E.
Animations of the swaps in the Bubble Sort algorithm on the text	This will be the second section of the visualisation of the Bubble Sort algorithm. It will synchronise with the bar chart animations.	On each pass, the text will be copied from the previous pass and as each item is swapped, they will be animated swapping positions.
Each pass of the	This will be the third section of the	A variable will keep track of the

algorithm is on a new line of text	visualisation of the Bubble Sort algorithm. This is to make the text readable and clear to understand.	passes and this will be incremented when a new pass is reached. This will create a new line of text.
Sort complete message	This message is to indicate that the Bubble Sort algorithm has finished.	It is displayed when the algorithm has made a pass with no swaps.
Menu	This will allow the user to navigate through the algorithm.	See the MENU DESIGN SECTION.

GUI



As I am planning on using a grid layout for my GUI using Tkinter, below is a diagram of how the elements shown above would fit in this layout:



PSEUDOCODE

```
PUBLIC PROCEDURE validate
	FOR entry in self.userEntries
		TRY
			num = int(entry)
			IF 1 <= num <= 100 THEN
				// num is valid
				self.numbers.append(num)
			ELSE
				Raise ValueError
			END IF
		EXCEPT ValueError
			Display invalid message
			RETURN
	NEXT entry
	IF len(self.numbers) < 8 OR len(self.numbers) > 12 THEN
		Display invalid message
		RETURN
	END IF
	bubbleSortAnimate()
PUBLIC PROCEDURE bubbleSortAnimate
	order = self.sortOrder
	n = len(self.numbers)
	swapped = True
	// sort to ascending:
	If order == "Ascending" THEN
		WHILE swapped
			swapped = False
```

```
                  FOR j in range (n-1)
                        // check that numbers are in the wrong order
                        If self.numbers[j] > self.numbers[j+1] THEN
                              // use temp variable to swap
                              temp = self.numbers[j]
                              self.numbers[j] = self.numbers[j+1]
                              self.numbers[j+1] = temp
                              swapped = True
                              INSERT self.numbers into visualiseText
                              // give the current set of numbers and the
                              two values to highlight
                              updateChart(self.numbers, [j,j+1])
                        END IF
                  NEXT j
            ENDWHILE
      ELSE IF order == "Descending" THEN
            WHILE swapped
                  swapped = False
                  FOR j in range (n-1)
                        // check that numbers are in the wrong order
                        If self.numbers[j] < self.numbers[j+1] THEN
                              // use temp variable to swap
                              temp = self.numbers[j]
                              self.numbers[j] = self.numbers[j+1]
                              self.numbers[j+1] = temp
                              swapped = True
                              INSERT self.numbers into visualiseText
                              // give the current set of numbers and the
                              two values to highlight
                              updateChart(self.numbers, [j,j+1])
                        END IF
                  NEXT j
            ENDWHILE
      END IF
END PROCEDURE

PUBLIC PROCEDURE updateChart (numbers, swapIndices=None)
      ax.clear()
      barColours = ["blue"] * len(numbers) // change the colour later, all
bars are the same
      If swapIndices THEN
            FOR index in swapIndices
                  barColours[index] = "red" // changes colour of the
swapping bars
            NEXT index
      END IF
      sleep(0.3) // for smooth animations
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
conn	sqlite3.Connection	Main and BubbleSort classes	Establishes the connection to the database for querying data.	N/A
cursor	sqlite3.Cursor	Main and BubbleSort classes	Executes SQL commands related to authenticating the user's entries.	N/A
bubbleSortFrame	Frame	BubbleSort class	Represents the frame that the GUI elements go in.	N/A
numbers	List[integer]	BubbleSort class	Stores the user-entered numbers to be stored, which is required for sorting.	[34, 12, 45, 9, 18, 20, 2, 14]
userEntries	List[Entry]	BubbleSort class	Stores the Tkinter entry fields for user input. It is needed for creating input fields.	[Entry1, Entry2, …, Entry8]
sortOrder	String	BubbleSort class	Stores the user's sorting order choice. It is needed to determine whether to sort in ascending or descending order.	"Ascending"

addButton	Button	BubbleSort class	Adds an extra input field. It is required to allow flexibility in the dataset size.	Clicking adds new entry field
invalidMessage	Label	BubbleSort class	Displays an error message if the input is valid for providing feedback for incorrect inputs.	N/A
fig	Figure	BubbleSort class	Stores the Matplotlib figure for visualisation for displaying the animation.	N/A
ax	Axes	BubbleSort class	Stores the axes for plotting the bars. It is needed for rendering the dataset s bars on the Figure.	N/A
canvas	FigureCanvasTkAgg	BubbleSort class	This embeds the Matplotlib figure into the Tkinter UI, allowing smooth animations in Tkinter.	N/A
visualiseText	Text	BubbleSort class	This displays the sorting steps in text format to help the user understand the process.	“Step 1: [12, 34, …]

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
conn	sqlite3.Connection	BubbleSort and Main classes attribute	Establishes the database connection for validating the details entered by the user.
cursor	sqlite3.Cursor	BubbleSort and Main classes attribute	Executes SQLcommands for querying the database.
BubbleSort	Class	Global	Manages the user's dataset input, displaying the values on the bar chart and the functionality of visualising the algorithm on the dataset using the Matplotlib bar chart and the Text.
numbers	List[integer]	Attribute of BubbleSort	Stores the user's dataset input. It is needed to hold the numerical values so they can be sorted.
userEntries	List[Entry]	Attribute of BubbleSort	Stores the Entry widget objects where users input the numbers. It allows for the dynamic adding of input fields.

WHITE BOX TEST DATA

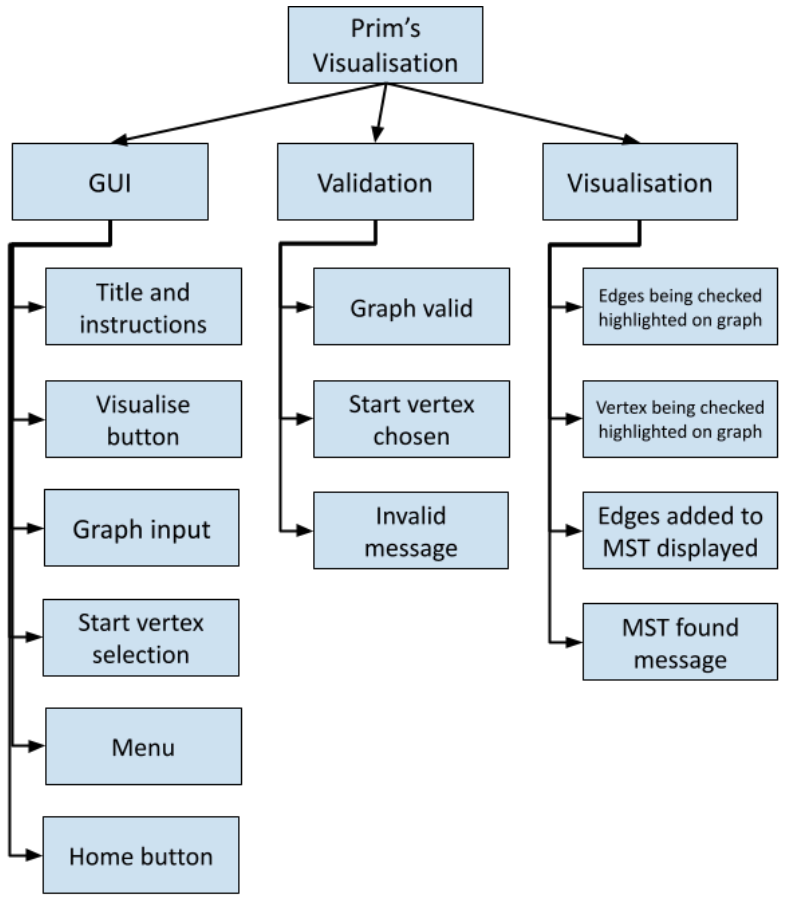
This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Visualise without entering a dataset	Invalid dataset message is displayed.	This ensures that the user must enter a set of numbers.	
Visualise with a dataset of length less that 8	Invalid dataset message is displayed.	This ensures that the testing of the length of the dataset is correct - so the dataset has >7 entries.	
Visualise with a dataset of length 8	Bubble Sort visualisation occurs correctly to sort the data both in text and on the bar chart.	This ensures that the testing of the length works correctly so that the minimum length is 8.	
Visualise with a dataset with values that are not integers	Invalid dataset message is displayed.	This ensures that the testing checks that all entries are integers.	
Visualise with a dataset with values not in the range 1-100	Invalid dataset message is displayed.	This ensures that the testing checks that all entries are in the correct range of values.	
Press the ‘Add Entry’ button	An entry field is added on the line with the rest of the entry fields. If there are already 11 entry fields, the button disappears.	This ensures that the user can enter a dataset of length greater than 8.	
Visualise with a dataset of length >8	Bubble Sort visualisation occurs correctly to sort the data both in text and on the bar chart.	This ensures that the testing of the length of the dataset is correct and allows lengths greater than 8 to be visualised.	
Visualise with a	Bubble Sort	This ensures that the	

dataset of length 12	visualisation occurs correctly to sort the data both in text and on the bar chart.	maximum length of 12 is included in the range of valid values for the length of the dataset.	
Press the 'Home' button	The Bubble Sort Visualisation page is closed and the Home page is opened.	This ensures that the 'Home' button works correctly and that the user can go back to the Home page to either log out or go to the other visualisation/quiz pages.	

PRIM'S ALGORITHM VISUALISATION PAGE DESIGN

DECOMPOSITION

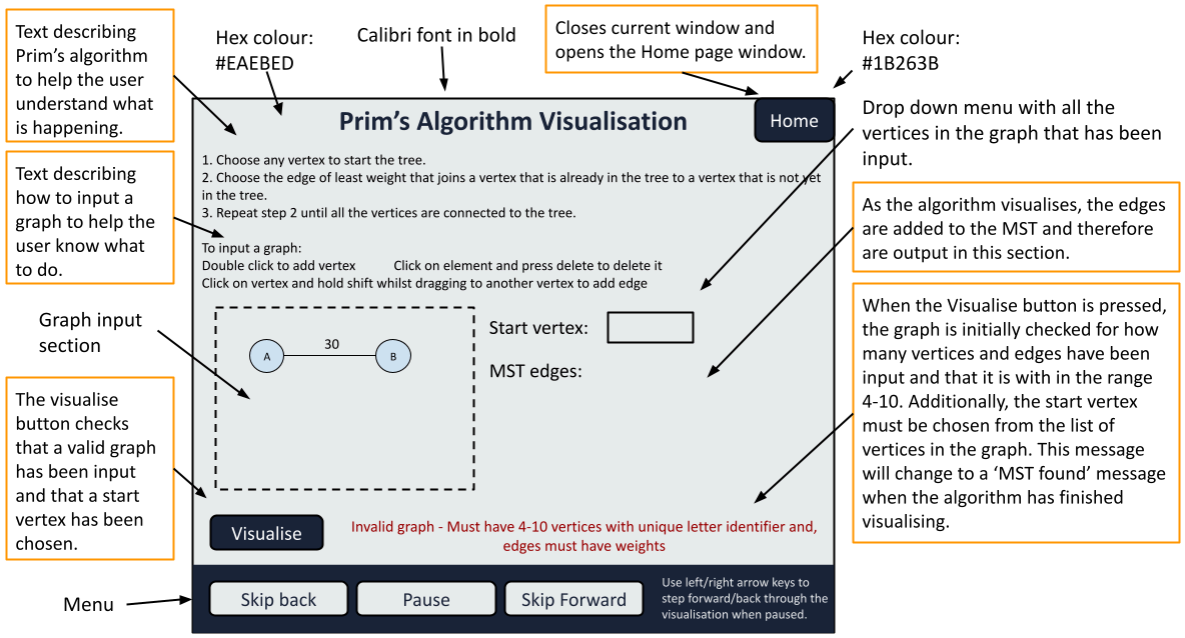


SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW

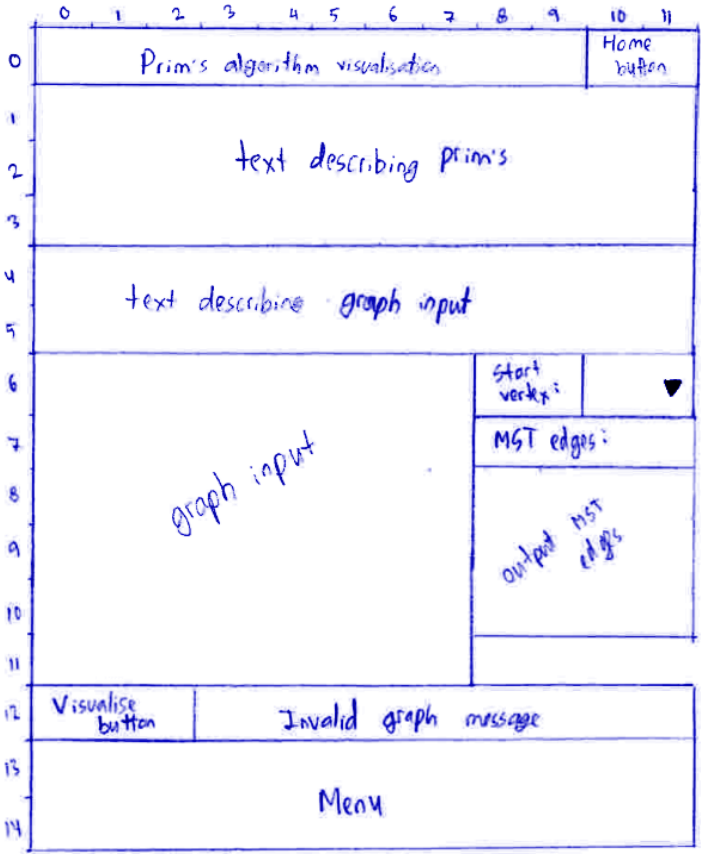
GUI with title and instructions, 'Visualise' and 'Home' buttons	This will be the main page that users can interact with. All other submodules rely on this one.	#EAEBED background colour. Text in the colour #1B263B and Calibri font. The title will be in bold at the top of the window. Buttons will have background colour #1B263B and text colour as white. The Visualise button will begin the validation of the entered data and then visualisation of the algorithm. The 'Home' button will close the current page and open the Home page.
Graph input	This will be the way that users can enter a graph for Prim's algorithm to be implemented on. The visualisation of the algorithm relies on this.	See the GRAPH INPUT DESIGN section.
Menu	This will allow the user to navigate through the algorithm.	See the MENU DESIGN SECTION.
Start vertex selection	This will be the way that the user can choose which vertex in the MST they want the algorithm to start at, which is a requirement of Prim's algorithm.	There will be a drop down menu with all the vertices in the graph.
Validation that graph is valid and start vertex chosen	This is the section before allowing the visualisation to begin dedicated to validating the inputs.	The graph will be checked that it is a graph of 4-10 vertices. Also, the graph must be connected so that each vertex must have at least 1 edge connecting it to the graph. The variable holding the start vertex is checked for a value that is a vertex in the graph.
Invalid / MST found message	This is the section of the GUI that displays the error message if a validation check fails or the message that the visualisation is complete.	This message will be displayed next to the visualise button. The text will be in red.
Vertex and edges being checked highlighted on graph	This is the main functionality of how the Prim's algorithm visualises. It focuses on the user being able to see what is being checked at each stage of the algorithm.	The vertex and edges being checked currently will be highlighted in #B2675E on the graph.
MST edges displayed as text and highlighted on	This is the second part of the functionality of the Prim's algorithm. It focuses on the user being able to	The edges added to the MST will be added on new lines in the 'output MST edges' section next to the graph

graph	clearly tell which edges have been added to the MST.	input.

GUI



As I am planning on using a grid layout for my GUI using Tkinter, below is a diagram of how the elements shown above would fit in this layout:



PSEUDOCODE

```
PUBLIC PROCEDURE validate
      // check 4 <= number nodes <= 10
      IF len(graph nodes) < 4 OR  len(graph nodes) > 10 THEN
            InvalidMessage = "Invalid graph…"
            Validated = False
            RETURN
      END IF
      // check vertices are connected
      FOR vertex in graph nodes
            IF len(graph neighbors (vertex)) == 0 THEN
                  InvalidMessage = "Invalid graph…"
                  Validated = False
                  RETURN
            END IF
      NEXT vertex
      // check weights are greater than 0
      FOR start, end, data IN graph edges
            IF data("weight") <= 0 THEN
                  invalidMessage = "Invalid graph…"
                  Validated = False
                  RETURN
            END IF
```

```
      NEXT start, end, data
      // check start vertex entered
      IF startVertexChoice == "" THEN
            Validated = False
            RETURN
      END IF
      // all checks passed if reach this point - validated
      Validated = True
      invalidMessage = ""
END PROCEDURE

PUBLIC PROCEDURE primAnimate
      mstEdges = []
      Visited = set()
      minHeap = [] // this is the queue
      visited.add(startVertex)
      FOR neighbour, edge in graph[startVertex].items()
            Weight = edge["weight"]
            // enqueue the edge:
            heappush(minHeap, (weight, startVertex, neighbour))
      NEXT neighbour, edge

      // update graph
      updatePrimGraph(mstEdges)

      WHILE len(visited) < len(graph nodes)
            IF not minHeap THEN
                  invalidMessage = "Graph not connected"
            END IF
            Weight,u,v = heappop(minHeap) // dequeue smallest
            IF v not in visited THEN
                  visited.add(v)
                  mstEdges.append((u,v)) // add edge
                  FOR neighbour, edge in graph[v].items()
                        Weight = edge["weight"]
                        heappush(minHeap, (weight, v, neighbour))
                  NEXT neighbour, edge
            updatePrimGraph(mstEdges)
END PROCEDURE

PUBLIC PROCEDURE updatePrimGraph (mstEdges)
      ax.clear()
      draw(graph, labels, ax=ax)
      Edge_labels = graph weights
      draw_labels(graph, edge_labels)
      FOR u,v in mstEdges
            Colour = #606C38 if last step, #B2675E otherwise
            draw_edges(graph, edgelist=[(u,v)], edge_color = colour)
      NEXT u,v
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
master	Tk instance	Prim class	Stores the main Tkinter window for the application.	N/A
title	Label	Prim class	Displays the title of the visualisation page so that it is clear to the user which page they are on.	N/A
primFrame	Frame	Prim class	Holds the widgets related to the Prim algorithm and the visualisation.	N/A
visualiseText	Text	Prim class	Displays the MST edges at each stage of the algorithm so that the user can clearly tell which edges have been added to the MST at each point.	N/A
invalidMessage	Label	Prim class	Displays an error message if a start vertex has not been entered or MST found when the visualisation has finished. This is so that the user is sure about what is	N/A

			causing errors or when the visualisation finishes.	
startVertexOptions	List	Prim class	Stores the vertices in the graph so that the user can choose one of them as the start vertex.	[“a”, “b”, “c”, “d”, “e”]
startVertexChoice	StringVar	Prim class	Stores the selected start vertex so that it can be used to begin executing Prim’s algorithm on the graph.	“a”
mstEdges	List of tuples	Prim class	Stores the edges in the final MST	[(“a”, “b”), (“b”, “c”)]
visited	set	Prim class	Tracks vertices visited so that Prim’s algorithm can execute properly.	{“A”, “B”, “C”}
minHeap	queue	Prim class	Stores the edges connected to the current vertex so that it can be processed for the smallest one.	[(2, “A”, “B”), (5, “B”, “C”)]
fig	Figure	Prim class	Stores the Matplotlib figure for visualisation for displaying the animation.	N/A
ax	Axes	Prim class	Stores the axes for plotting the graph. It is needed for	N/A

			rendering the graph on the Figure.	

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
graph	NetworkX Graph	Prim class	Stores the input graph and its edges and weights. This is required because it is used for the computation of the MST.
mstEdges	List of tuples	Prim class	Stores the edges in the final MST so that they can be displayed in the Text display.
visited	set	Prim class	Tracks vertices visited so that Prim's algorithm can execute properly.
minHeap	queue	Prim class	Stores the edges connected to the current vertex so that it can be processed for the smallest one.

WHITE BOX TEST DATA

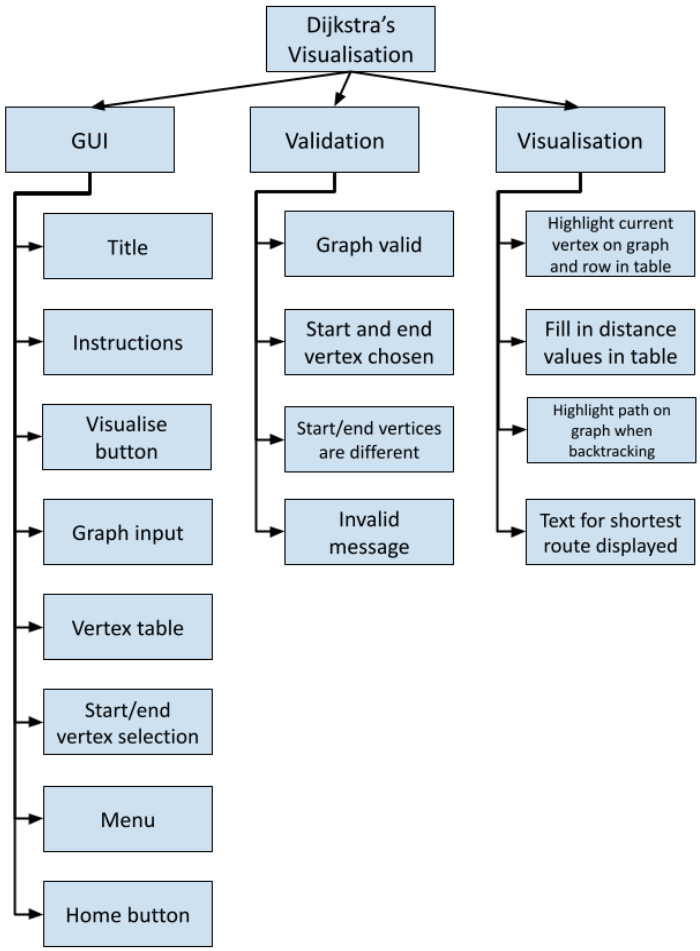
This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Press visualise without inputting a graph.	An 'Invalid graph' error message is displayed.	Confirms that a graph must be entered by the user for the visualisation to begin.	
Press visualise without entering a start vertex.	An 'Invalid - please enter a start vertex' error message is displayed.	Confirms that the start vertex must be entered for the algorithm to begin visualising.	
Visualise with a connected graph of 4 vertices and a start vertex chosen.	The visualisation of Prim's algorithm starts correctly to find the MST of the entered graph.	This is a boundary test on the minimum number of vertices that could be entered for the algorithm to visualise.	
Visualise with a connected graph of 10 vertices and a start vertex chosen.	The visualisation of Prim's algorithm starts correctly to find the MST of the entered graph.	This is a boundary test on the maximum number of vertices that could be entered for the algorithm to visualise.	
Visualise with a connected graph of 3 vertices and a start vertex chosen.	An 'Invalid graph' error message is displayed.	This is an erroneous test for if the number of vertices entered is less than the minimum.	
Visualise with a connected graph of 11 vertices and a start vertex chosen.	An 'Invalid graph' error message is displayed.	This is an erroneous test for if the number of vertices entered is more than the maximum.	
Visualise with an unconnected graph of 6 vertices and a start vertex chosen.	An 'Invalid graph' error message is displayed.	This is a test to ensure that all the vertices must have at least 1 edge connected to them, therefore	

		making the visualisation worthwhile.	
Press the Home button	The Prim's Visualisation page is closed and the Home page is opened.	This ensures that the 'Home' button works correctly and that the user can go back to the Home page to either log out or go to the other visualisation/quiz pages.	

DIJKSTRA'S ALGORITHM VISUALISATION PAGE DESIGN

DECOMPOSITION

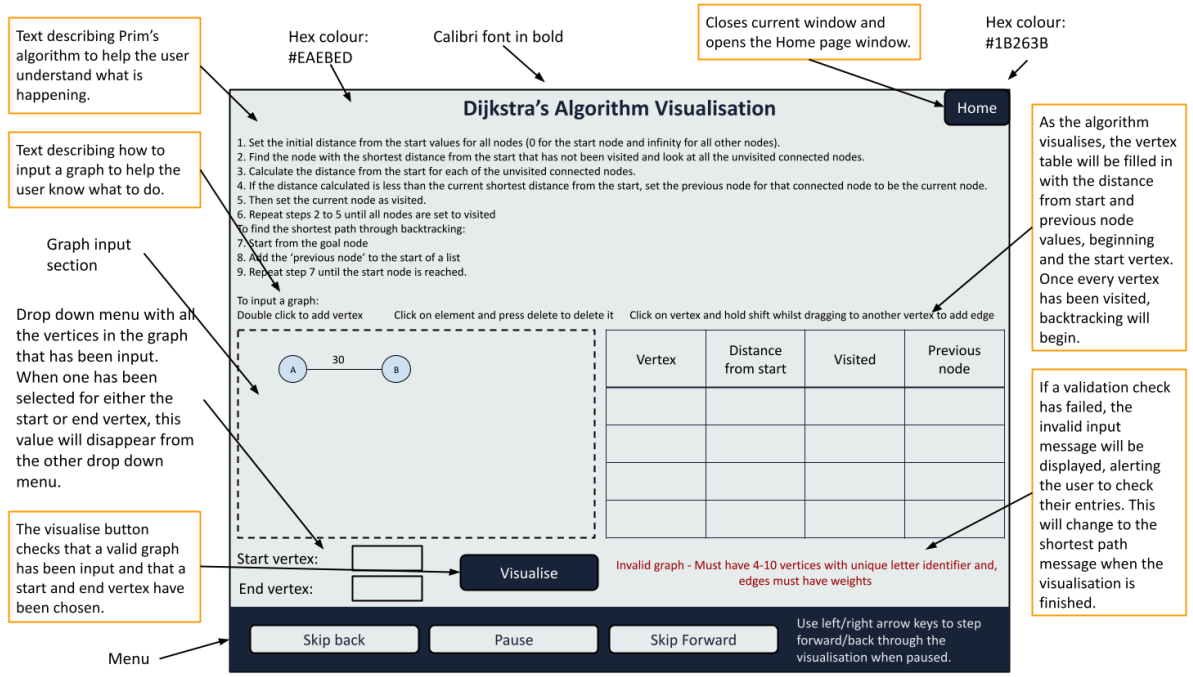


SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW

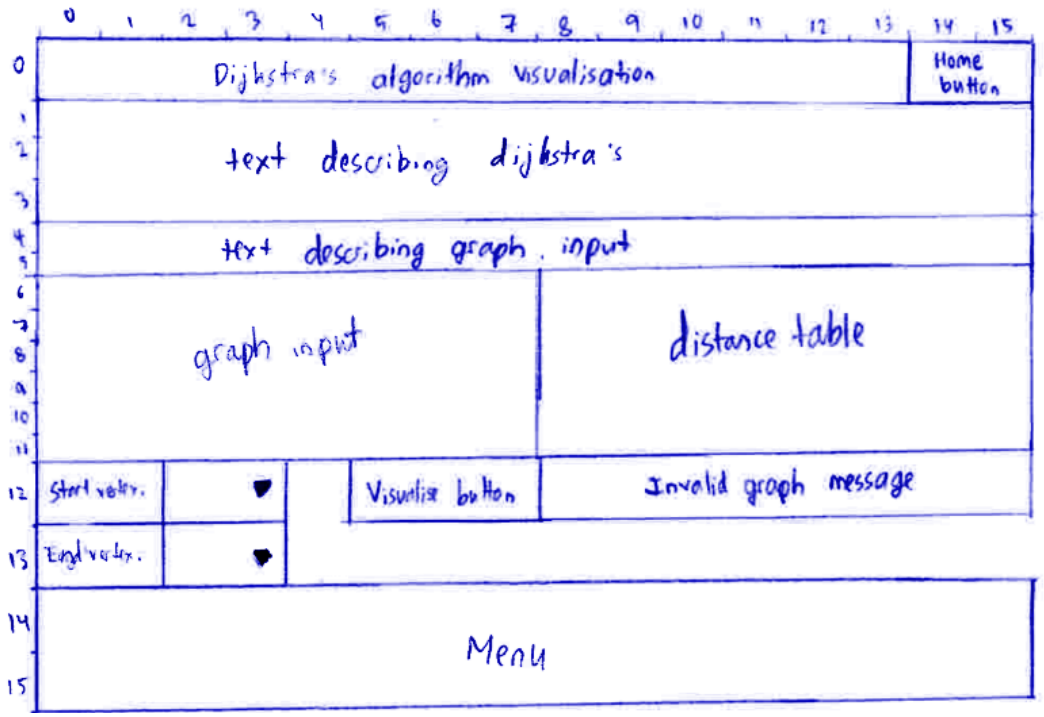
GUI with title, instructions, 'Visualise' and 'Home' buttons.	This will be the main page that users can interact with. All other submodules rely on this one.	#EAEBED background colour. Text in the colour #1B263B and Calibri font. The title will be in bold at the top of the window. Buttons will have background colour #1B263B and text colour as white. The Visualise button will begin the validation of the entered data and then visualisation of the algorithm. The 'Home' button will close the current page and open the Home page.
Graph input	This will be the way that users can enter a graph for Dijkstra's algorithm to be implemented on. The visualisation of the algorithm relies on this.	See the GRAPH INPUT DESIGN section.
Vertex/distance table	This will be one part of how the algorithm will be visualised.	There will be a table with the columns: vertex, distance from start, visited and previous vertex. Once the Visualise button has been pressed, the vertices in the graph will appear in the table and all the distances will be set to display infinity.
Start/end vertex entry	This will make the user enter a start and end vertex, both of which are required for the algorithm to find the shortest path.	There will be two drop down menus of all the vertices in the graph and the user can choose one from each. Once a vertex has been selected in one, it will disappear from the drop down menu in the other.
Menu	This will allow the user to navigate through the algorithm.	See the MENU DESIGN SECTION.
Graph and start/end vertex choice validation	This will be the validation section that is required to check that the user's inputs are correct and would allow the algorithm to function correctly.	When the Visualise button is pressed, the graph will be checked to make sure it contains between 4 and 10 vertices and each vertex has at least 1 edge connected to it. The start and end vertices will be checked for a value that is in the graph.
Invalid message	This is the section of the GUI that displays the error message if a validation check fails. It is required to alert the user to	This message will be displayed next to the visualise button. The text will be in red.

Highlighting current vertex on graph and in vertex table	This will be the first part of the visualisation of Dijkstra's algorithm. It is the functionality that makes it clear to the user which vertices are being visited, which may not be clear without the highlighting on the graph.	The current vertex will be highlighted on the graph in the colour #B2675E and the text in the vertex's row in the vertex table will be highlighted in the same colour.
Filling in the vertex table through the visualisation of the algorithm	This is the second part of the visualisation of Dijkstra's algorithm. It is focused on the main way that the algorithm functions, where checks occur for the shortest distances.	For the current vertex, the weight of each edge connected to that vertex that is to an unvisited vertex is added to the value in the current vertex's 'distance from start' value. Checks occur for each connected vertex, ensuring that the 'distance from start' is set to this value if it is less than the current value and the previous node will then be set to the current vertex. The current vertex will then be set to visited and the vertex of smallest 'distance from start' is followed. This is the way that Dijkstra's algorithm works and has time complexity O(E+VlogV) in the best and average case and $O(V^2)$ in the worst case.
When backtracking to find the shortest path, highlight the edges on the graph	This is the third part of the Dijkstra's algorithm visualisation and is the stage that finds the shortest path from the start to the end vertex.	The end vertex will be highlighted on the graph in #606C3B and its row is highlighted in the vertex table. Then the previous vertex is followed and highlighted on the graph in the same colour, along with the edge followed. This is repeated until the start vertex is reached.
When the visualisation is finished, text displays the path	This is the final functionality of the Dijkstra's algorithm visualisation and is the stage that indicates that the visualisation is finished.	The message 'Shortest path between _ and _ is: …' is displayed where the start and end vertices are given, along with the shortest path found in text form.

GUI



As I am planning on using a grid layout for my GUI using Tkinter, below is a diagram of how the elements shown above would fit in this layout:



PSEUDOCODE

A priority queue method[22] to implement Dijkstra's algorithm is the recommended version so below is the pseudocode for this section:

```
PUBLIC PROCEDURE validate
      // check 4 <= number nodes <= 10
      IF len(graph nodes) < 4 OR  len(graph nodes) > 10 THEN
            InvalidMessage = "Invalid graph…"
            Validated = False
            RETURN
      END IF
      // check weights are greater than 0
      FOR start, end, data IN graph edges
            IF data("weight") <= 0 THEN
                  invalidMessage = "Invalid graph…"
                  Validated = False
                  RETURN
            END IF
      NEXT start, end, data
      // check start vertex entered
      IF startVertexChoice == "" THEN
            Validated = False
            RETURN
      END IF
      // check end vertex entered
      IF endVertexChoice == "" THEN
            Validated = False
            RETURN
      END IF
      // all checks passed if reach this point - validated
      Validated = True
      invalidMessage = ""
END PROCEDURE

PUBLIC PROCEDURE dijkstraAnimate
      Start = self.startVertexChoice
      End = self.endVertexChoice
      Distances = {node: float("inf") for node in graph.nodes}
      Previous = {node: None for node in graph.nodes}
      Distances[start] = 0
      Queue = [(0, start)]
      Visited = set()
      WHILE queue // while there is a node to visit
            currentDist, currentNode = dequeue(queue)
            IF currentNode in visited THEN
                  // go onto next iteration
                  CONTINUE
            END IF
            visited.add(currentNode)
            Display graph with connected edges highlighted
            Update vertex table
            FOR neighbour in graph.neighbors(currentNode)
```

```
                  IF neighbour in visited THEN
                        CONTINUE
                  END IF
                  newDistance = currentDist + weight neighbour edge
                  IF newDistance < distances[neighbour] THEN
                        Distances[neighbour] = newDistance
                        Previous[neighbour] = currentNode
                        enqueue(queue, (newDistance, neighbour))
                  END IF
            NEXT neighbour
      END WHILE
      // backtracking:
      Path = []
      Current = end
      WHILE current
            path.append(current)
            Current = previous[current]
      END WHILE
      Reverse path
      Display shortest path on graph
      Display shortest path in text
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
master	Tk instance	Dijkstra class	Stores the main Tkinter window for the application.	N/A
title	Label	Dijkstra class	Displays the title of the visualisation page so that it is clear to the user which page they are on.	N/A
dijkstraFrame	Frame	Dijkstra class	Holds the widgets related to the Dijkstra algorithm and the visualisation.	N/A
tableFrame	Frame	Dijkstra class	Holds the widgets related to the vertex/distance table for the visualisation of Dijkstra's algorithm.	N/A
startVertexChoice	StringVar()	Dijkstra class	This stores the user-selected start vertex, which is needed for the execution of Dijkstra's algorithm.	"a"
endVertexChoice	StringVar()	Dijkstra class	This stores the user-selected end vertex, which is needed for determining the shortest	"D"

			path between the two	
invalidMessage	Label	Dijkstra class	This displays the shortest path between the start and end vertex and its weight once the visualisation has finished. If a user has not entered a valid graph or has not input start and end vertices, an error message will be displayed in this.	N/A
queue	List of tuples	Dijkstra class	This is a priority queue to store the values of the distance from the start and the vertex name so that the shortest value node can be used in the next iteration of Dijkstra's algorithm.	[(0, "A"), (9, "b")]
fig	Figure	Dijkstra class	Stores the Matplotlib figure for visualisation for displaying the animation.	N/A
ax	Axes	Dijkstra class	Stores the axes for plotting the graph. It is needed for rendering the graph on the Figure.	N/A

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
graph	NetworkX Graph	Dijkstra class	Stores the input graph and its edges and weights. This is required because it is used for the computation of the shortest path.
distances	Dictionary ({str: float})	Dijkstra class	This is a dictionary that stores the distances from the start vertex. This is initialised to infinity for all except the start node. It is updated as Dijkstra's algorithm is carried out.
previous	Dictionary ({str: str})	Dijkstra class	This is a dictionary that stores the previous vertex in the shortest path for backtracking. Copies of this are stored in steps at each stage of the algorithm. It is required for the final steps of execution for Dijkstra's algorithm and so that the vertex/distance table can be updated correctly.
queue	List of tuples ((float, str))	Local to dijkstraSteps	This is a priority queue and is required to determine the next node to process as Dijkstra's algorithm is carried out.
visited	Set of strings	Local to dijkstraSteps	This is an unordered collection of

			the nodes that have been processed. This is useful because it allows the program to avoid redundant calculations.
startVertexOptions	list	Dijkstra class	This stores a list of all the nodes in the graph. It is needed so that the user can select the start vertex from this list using the dropdown menu.
endVertexOptions	list	Dijkstra class	This stores a list of all the nodes in the graph. It is needed so that the user can select the end vertex from this list using the dropdown menu.

WHITE BOX TEST DATA

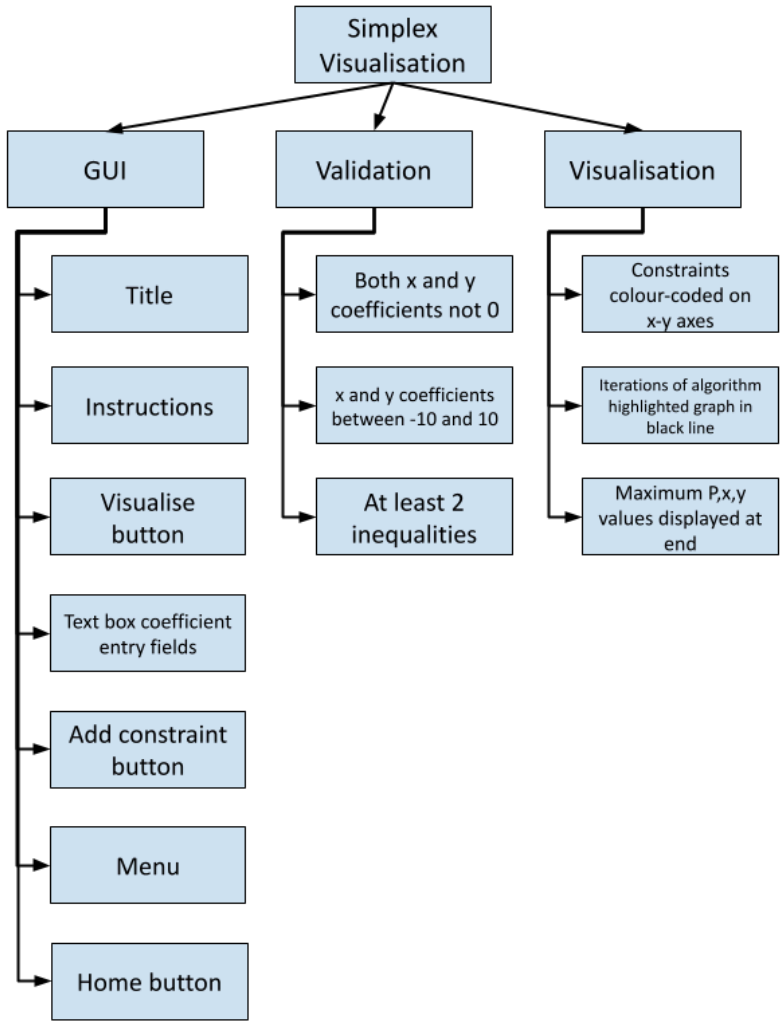
This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Press visualise without inputting a graph.	An 'Invalid graph' error message is displayed.	Confirms that a graph must be entered by the user for the visualisation to begin.	
Press visualise without entering a start vertex.	An 'Invalid - please enter a start vertex' error message is displayed.	Confirms that the start vertex must be entered for the algorithm to begin visualising.	
Press visualise without entering an end vertex.	An 'Invalid - please enter a start vertex' error message is displayed.	Confirms that the end vertex must be entered for the algorithm to begin visualising.	
Visualise with a connected graph of 4 vertices and both start and end vertices chosen.	The visualisation of Dijkstra's algorithm begins correctly to find the shortest path between the start and end vertices for the entered graph.	This is a boundary test on the minimum number of vertices that could be entered for the algorithm to visualise.	
Visualise with a connected graph of 10 vertices and both start and end vertices chosen.	The visualisation of Dijkstra's algorithm begins correctly to find the shortest path between the start and end vertices for the entered graph.	This is a boundary test on the maximum number of vertices that could be entered for the algorithm to visualise.	
Visualise with a connected graph of 3 vertices and both start and end vertices chosen.	An 'Invalid graph' error message is displayed.	This is an erroneous test for if the number of vertices entered is less than the minimum.	
Visualise with a connected graph of 11 vertices and both start	An 'Invalid graph' error message is displayed.	This is an erroneous test for if the number of vertices entered is	

and end vertices chosen.		more than the maximum.	
Visualise with an unconnected graph of 6 vertices and both start and end vertices chosen.	An 'Invalid graph' error message is displayed.	This is a test to ensure that all the vertices must have at least 1 edge connected to them, therefore making the visualisation worthwhile.	
Press the Home button	The Dijkstra's Visualisation page is closed and the Home page is opened.	This ensures that the 'Home' button works correctly and that the user can go back to the Home page to either log out or go to the other visualisation/quiz pages.	

SIMPLEX ALGORITHM VISUALISATION PAGE DESIGN

DECOMPOSITION

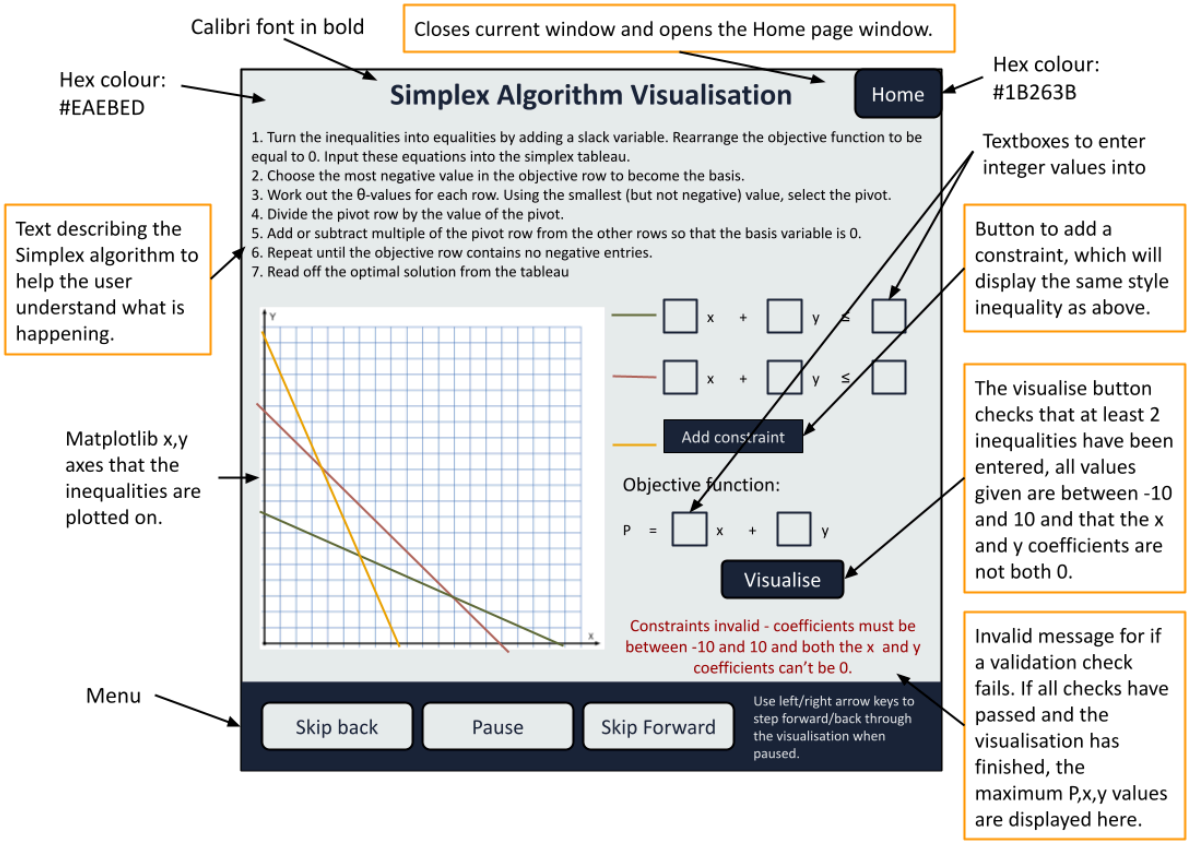


SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW
GUI with title, instructions, 'Visualise' and 'Home' buttons	This will be the main page that users can interact with. All other submodules rely on this one.	#EAEBED background colour. Text in the colour #1B263B and Calibri font. The title will be in bold at the top of the window. Buttons will have background colour #1B263B and text colour as white. The Visualise button will begin the validation of the entered data and then visualisation of the algorithm. The 'Home' button will close the current page and open the Home page.

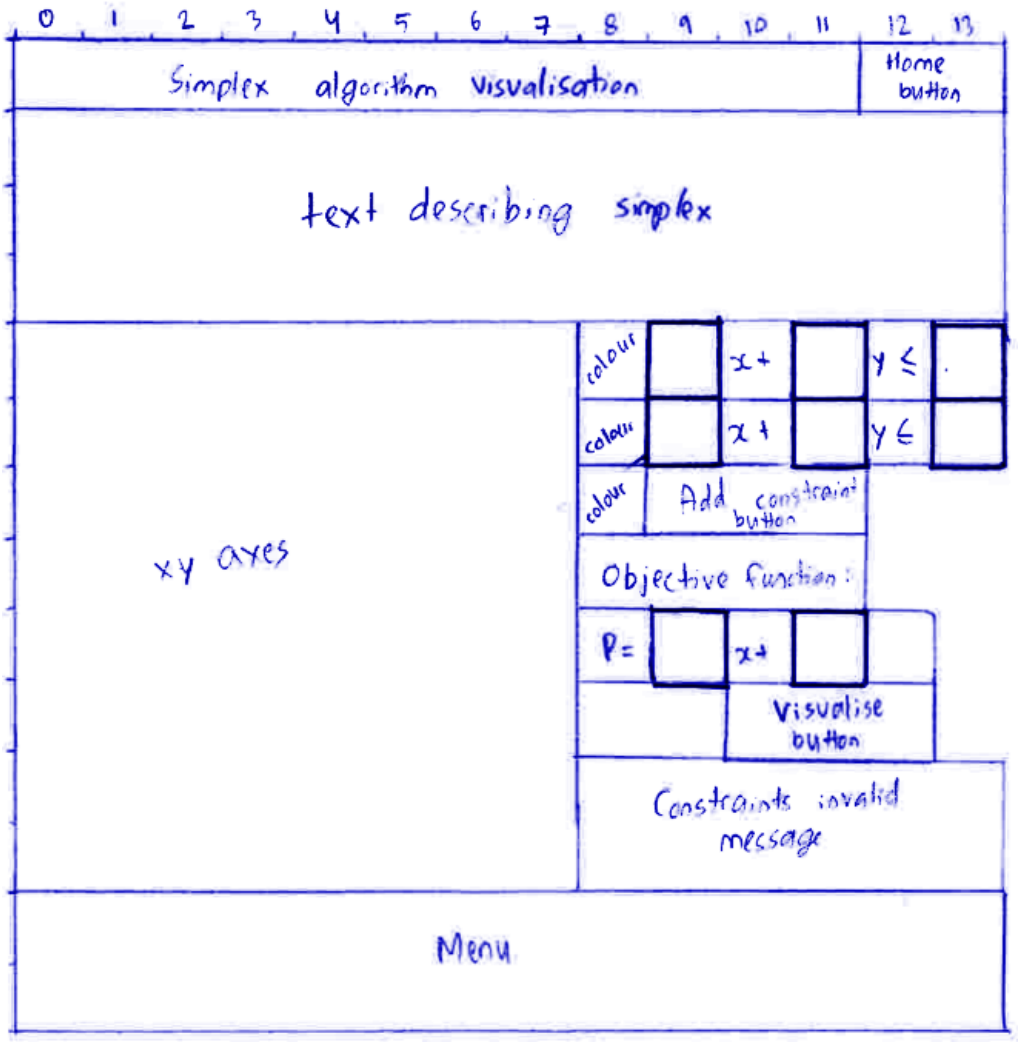
Text boxes to enter coefficients	This will be the main way that the user can input their own data to be used for the visualisation. The algorithm visualisation relies on equations to be entered.	There will be text boxes in front of the x and y text to act as coefficients and after the inequality sign, there will be another text box. These should have positive or negative integers entered.
Add constraint button	This will be the way that the user can control the number of inequalities they want to use for the visualisation.	There will be a button with background colour #1B263B and text colour as white. When pressed, a new blank inequality, like described with the text boxes section, is displayed below the other two.
Menu	This will allow the user to navigate through the algorithm.	See the MENU DESIGN SECTION.
Validation that at least 3 inequalities have been entered, x and y coefficients are not both 0 and are between -10 and 10	This will be the main functionality of the validation within this visualisation page.	Firstly, checks occur that at least 2 inequalities have been entered. Then, the coefficients for x and y are checked to make sure they are not both 0, which is required because this would cause the algorithm to not execute properly. Then, the values of all the text box entries are checked to be between 10 and -10 (inclusive), which is required because this range of values would allow the lines to be displayed clearly on the graph.
Constraints displayed and colour coded on axes	This is the way that the user's inputs can be used within the algorithm. It is a required section of how the visualisation functions.	The first inequality is displayed using the colour #B2675E and the second inequality is displayed with the colour #606C38. If the user has entered a third inequality, this is displayed on the axes using the colour #ECA400.
Black line highlights order of iterations of the Simplex algorithm as it is applied	This is the focus of how the simplex algorithm will be visualised on the axes. It requires the steps of the algorithm to be calculated beforehand.	The simplex algorithm is applied to the inequalities given. Each stage is stored. Then, the points of the feasible region that are being visited are displayed on the axes in the order calculated. A black line will follow the feasible region to each point.
Maximum P,x,y	This is the final stage of the	The values obtained from the

values displayed once found	visualisation of the simplex algorithm. It indicates that the algorithm has finished.	simplex tableau for x,y and P are displayed as text on the page when the algorithm visualisation has finished executing.

GUI



As I am planning on using a grid layout for my GUI using Tkinter, below is a diagram of how the elements shown above would fit in this layout:



PSEUDOCODE

```
PUBLIC PROCEDURE validate
      FOR constraint in constraints
            TRY
                  x = constraints["x"].get() float
                  y = constraints["y"].get() float
                  rhs = constraints["rhs"].get(0 float
                  IF x == 0.0 AND y == 0.0 THEN
                        RAISE ValueError
                  END IF
                  IF x < -10.0 OR x > 10.0 THEN
                        RAISE ValueError
                  END IF
                  IF y < -10.0 OR y > 10.0 THEN
                        RAISE ValueError
                  END IF
                  IF rhs < -10.0 OR rhs > 10.0 THEN
                        RAISE ValueError
```

```
                  END IF
            EXCEPT ValueError
                  Invalid message = "Invalid constraint"
                  RETURN
      NEXT constraint
      TRY
            X = objective["x"].get() float
            y = objective["y"].get() float
            IF x == 0.0 AND y == 0.0 THEN
                  RAISE ValueError
            END IF
            IF x < -10.0 OR x > 10.0 THEN
                  RAISE ValueError
            END IF
            IF y < -10.0 OR y > 10.0 THEN
                  RAISE ValueError
            END IF
      EXCEPT ValueError
            Invalid message = "Invalid objective"
            RETURN
      // all validation passed at this point
      createTableau()
      plotConstraints()
      simplexAnimate()
END PROCEDURE

PUBLIC PROCEDURE createTableau
      numSlackVariables = len(constraints)
      FOR constraint in constraints
            row = coefficients except rhs
            Add slack variable to row
            Add rhs value to row
            Tableau = tableau + row
      NEXT constraint
      // all constraints in tableau at this point
      objectiveRow = - values of x and y in objective
      Add 0s to objectiveRow
      Tableau = tableau + objectiveRow
END PROCEDURE

PUBLIC PROCEDURE plotConstraints
      Colours = [] // colours determined for GUI design
      Create axes
      Plot constraints with correct colours
END PROCEDURE

PUBLIC PROCEDURE simplexAnimate
      WHILE not optimal
            pivotColumn = findPivotColumn(tableau)
            pivotRow = findPivotRow(tableau, pivotColumn)
            IF pivotColumn or pivotRow == None THEN
                  // solution is unbounded
```

```
                  BREAK
            END IF
            pivot(tableau, pivotRow, pivotColumn)
            Display line on graph
      END WHILE
      // optimal solution found
      Extract solutions for x y P from tableau
END PROCEDURE

PUBLIC PROCEDURE findPivotColumn (tableau)
      RETURN index of most negative coefficient in tableau
END PROCEDURE

PUBLIC PROCEDURE findPivotRow(tableau, pivotColumn)
      Ratio = []
      FOR row i in tableau
            IF tableau[i][pivotColumn] > 0 THEN
                  Ratio[i] = tableau[i][-1] / tableau[i][pivotColumn]
            END IF
      NEXT row i
      RETURN index minimum ratio
END PROCEDURE

PUBLIC PROCEDURE pivot (tableau, pivotRow, pivotColumn)
      // make pivot element 1
      pivotElement = tableau[pivotRow][pivotColumn]
      tableau[pivotRow] divided by pivotElement
      // make column entries 0
      FOR row i not equal to pivotRow
            Multiplier = tableau[i][pivotColumn]
            Tableau[i] - multiplier * tableau[pivotRow]
      NEXT row i
      END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
master	Tk instance	Simplex class	Stores the main Tkinter window for the application.	N/A
title	Label	Simplex class	Displays the title of the visualisation page so that it is clear to the user which page they are on.	N/A
simplexFrame	Frame	Simplex class	Holds the widgets related to the Prim algorithm and the visualisation.	N/A
constraints	List of dictionaries	Simplex class	Stores the Tkinter inputs for the constraints so that their values can be accessed in execution.	[{“x”: 1, “y”: -9, “rhs”: 2}]
objective	List of floats	Simplex class	Stores the coefficients of the objective function so that they can be accessed in execution.	[3.0, -9.0]
tableau	numpy array	Simplex class	This is a 2d array that stores the simplex tableau. This is required because it allows	[[2,3,1,0,5], [-1, 1, 0, 1,2], [-3,-2,0,0,0]]

			the pivot operations to be performed during the execution of the simplex algorithm.	
x	float	Local to validate method	Stores the value of the constraint/objective's x so that it can be used in validation checks like if it is between -10.0 and 10.0.	3.6
y	float	Local to validate method	Stores the value of the constraint/objective's y so that it can be used in validation checks like if it is between -10.0 and 10.0.	-7.1
rhs	float	Local to validate method	Stores the value of the constraint's rhs so that it can be used in validation checks like if it is between -10.0 and 10.0.	3.6
pivotRow	integer	Simplex class	Stores the index of the row that the tableau will be pivoted on, allowing the functionality of the pivot method to be carried out.	1
pivotColumn	integer	Simplex class	Stores the index of the	0

			column that the tableau will be pivoted on, allowing the functionality of the pivot method to be carried out.	
multiplier	float	Local to the pivot method	Stores the value in the current row in the basic column so that a multiple of it can be subtracted from each value in the column.	2.0
ax	Axes	Simplex class	Stores the axes for plotting the bars. It is needed for rendering the dataset s bars on the Figure.	N/A
fig	Figure	Simplex class	Stores the Matplotlib figure for visualisation for displaying the animation.	N/A

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
tableau	Numpy 2d array	Simplex class	Stores the simplex tableau for

			performing row operations during the execution of the Simplex algorithm, therefore allowing the optimal solution to be found.
xPath	Tuple of float values	Local to the updateGraph method	Stores the values of the x points to show on the graph when displaying the path around the feasible region to the optimal point.
yPath	Tuple of float values	Local to the updateGraph method	Stores the values of the y points to show on the graph when displaying the path around the feasible region to the optimal point.

WHITE BOX TEST DATA

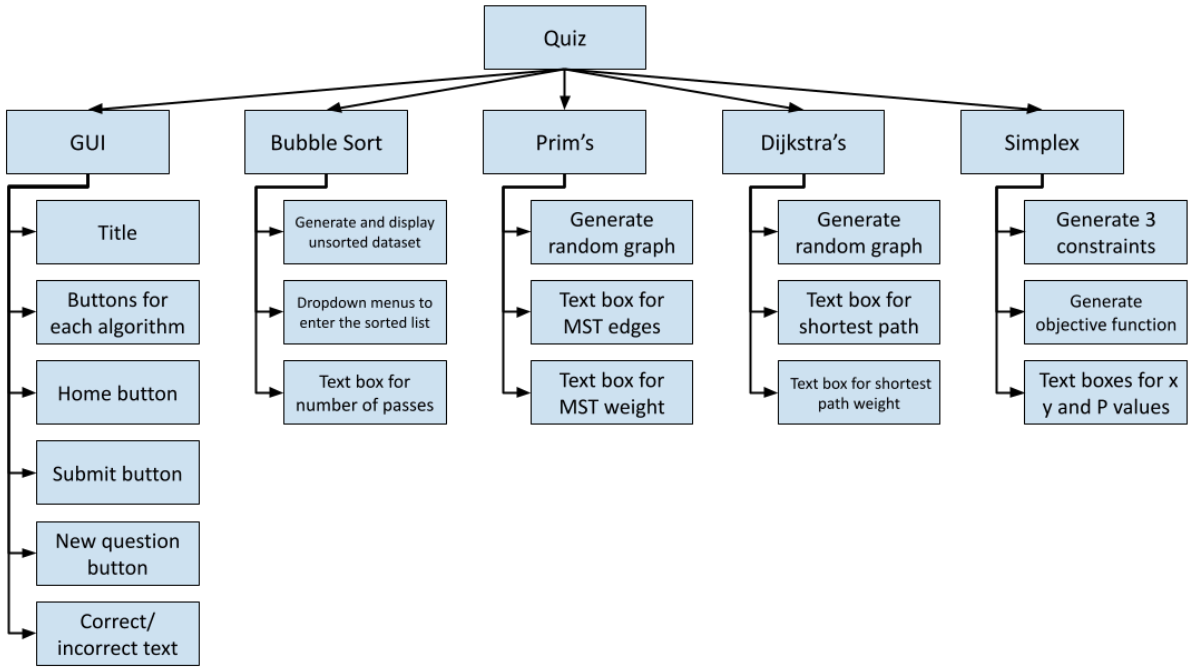
This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Press visualise without entering any constraints.	A 'Constraint invalid' error message is displayed.	Confirms that constraints must be entered by the user for the visualisation to begin.	
Press visualise without entering an objective function.	A 'Objective function invalid' error message is displayed.	Confirms that an objective function must be entered by the user for the visualisation to begin.	
Press the Add Constraint button	A constraint entry field is displayed below the other 2 entry fields. The button is no longer displayed.	Confirms that the Add Constraint button works as intended and allows an extra constraint to be entered by the user.	
Visualise with 1 valid constraint entered.	A 'Constraint invalid' error message is displayed.	Confirms that the user must enter at least 2 constraints for the visualisation to begin.	
Visualise with 2 valid constraints entered.	The visualisation of the Simplex algorithm begins correctly on the entered constraints.	Confirms that a valid input allows the visualisation to begin.	
Visualise with 3 valid constraints entered.	The visualisation of the Simplex algorithm begins correctly on the entered constraints.	Confirms that a valid input allows the visualisation to begin.	
Visualise with coefficients being greater than 10.	A 'Constraint invalid' error message is displayed.	Confirms that the coefficients are checked to be within the range -10 to 10.	
Visualise with	A 'Constraint invalid'	Confirms that the	

coefficients being less than -10.	error message is displayed.	coefficients are checked to be within the range -10 to 10.	
Visualise with both the x and y coefficients being 0.	A 'Constraint invalid' error message is displayed.	Confirms that the x and y coefficient checks occur correctly to ensure that the visualisation does not error.	
Press the Home button.	The Simplex Visualisation page is closed and the Home page is opened.	This ensures that the 'Home' button works correctly and that the user can go back to the Home page to either log out or go to the other visualisation/quiz pages.	

QUIZ PAGE DESIGN

DECOMPOSITION



SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW

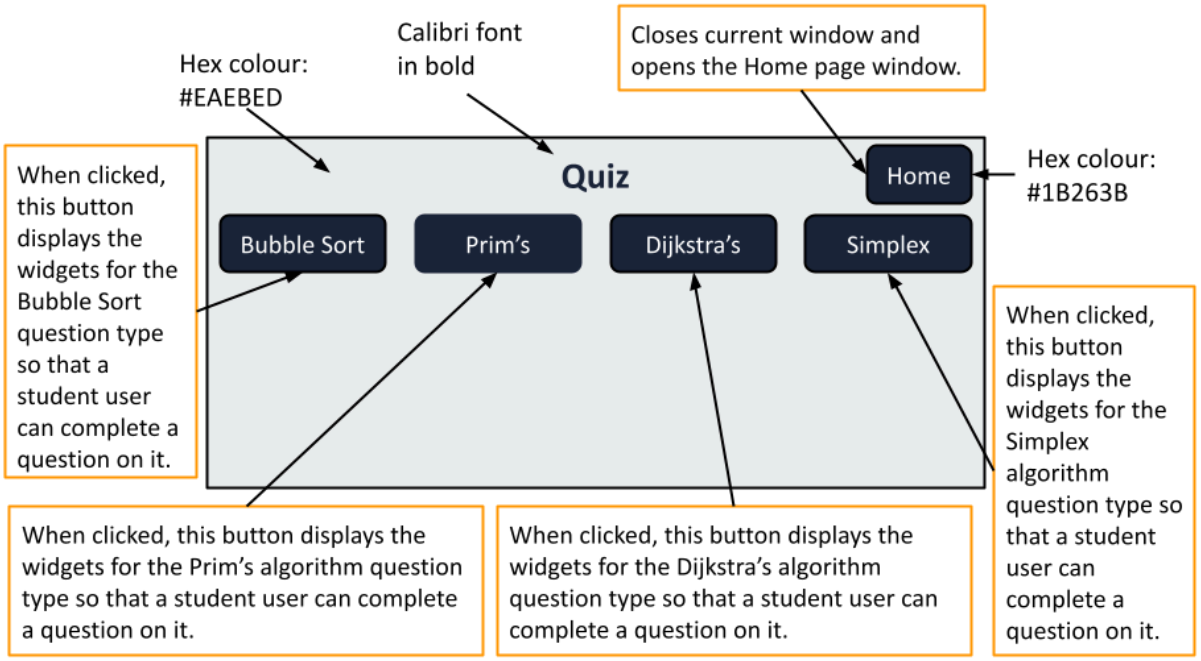
GUI with title, buttons for questions of each algorithm and a 'Home' button	This will be the main page that users can interact with. All other submodules rely on this one.	#EAEBED background colour. Text in the colour #1B263B and Calibri font. The title will be in bold at the top of the window. Buttons will have background colour #1B263B and text colour as white. The algorithm buttons will display the widgets for the specific algorithm quiz question. The 'Home' button will close the current page and open the Home page.
Submit button	This will be the way that the user can submit an answer to the question all the functionality of the quiz page relies on.	For a Bubble Sort question, the algorithm is applied to calculate the sorted list and number of passes. For a Prim's algorithm question, the algorithm is applied to calculate the MST edges and the weight. For a Dijkstra's algorithm question, the shortest path and weight are calculated. For a Simplex algorithm question, the optimal values of x y and P are calculated. For all of these question types, the values submitted are checked. The database is updated. The outcome is output as text.
New question button	This will be the way that the user can answer new questions of the same algorithm type selected. This subsection will be reused for all the algorithm quiz sections.	For a Bubble Sort question, when pressed, a new unsorted dataset is generated. For Prim's and Dijkstra's algorithm questions, new graphs are generated. For a Simplex algorithm question, 3 new constraints and an objective function are randomised. For all of the question pages, after pressing this button, all entry fields are cleared so that the user can enter their answer to the new question.
Generate and display unsorted dataset for Bubble Sort	This will be the way that questions for the Bubble Sort algorithm can be created so it is required for the functionality of the quiz page.	10 random integers between 1 and 50 inclusive are displayed in a line underneath the description of the Bubble Sort algorithm and the text saying to sort this dataset in ascending order.

Bubble Sort answer entry fields	This is the way that the user can enter their answer to the question so that it can be checked. Scores can then be updated.	There will be 10 dropdown menus in a row beneath the unsorted dataset. Each dropdown menu will have all 10 items in the dataset as options. The user should enter them in the correct ascending order. Beneath this will be a textbox to enter the total number of passes done to sort the dataset as an integer.
Generate random graph	This is a subsection to be used for generating questions for both Prim's and Dijkstra's algorithm quiz questions, meaning it will be reused. It is required for the functionality of the quiz question creation.	A connected graph of 6 to 8 vertices is created and displayed. This means that each vertex must be connected to at least one other vertex with an edge. All edges will be weighted and weights will be integers between 1 and 50. For labelling the vertices, they should be in alphabetical order beginning at A.
Prim's answer entry fields	This is the way that the user can enter their answer to the question so that it can be checked. Scores can then be updated.	Beneath the text saying the start vertex is A, there should be a textbox that the user should enter the MST edges into like this: AB, BC, CD (letter names of start and end separated with commas). Beneath this is a text box to enter the total weight of the MST as an integer.
Dijkstra's answer entry fields	This is the way that the user can enter their answer to the question so that it can be checked. Scores can then be updated.	Beneath the text saying that the start vertex is A and the end vertex is the last alphabetical letter in the list of vertex labels, there should be a textbox to enter the shortest path like this: ABCD. Beneath this is a text box to enter the weight of the shortest path.
Generate 3 constraints and objective function	This will be the way that questions for the Simplex algorithm can be created so it is required for the functionality of the quiz page.	3 constraints will be displayed in the form _x + _y ≤ _. These will be displayed as lines on the axes on the left side of the window. An objective function in the form P = _x + _y will also be displayed as text. The coefficients indicated by a _ will be between -10 and 10, with both x and y not both being 0.
Simplex answer	This is the way that the user can	These should be 3 text boxes next to

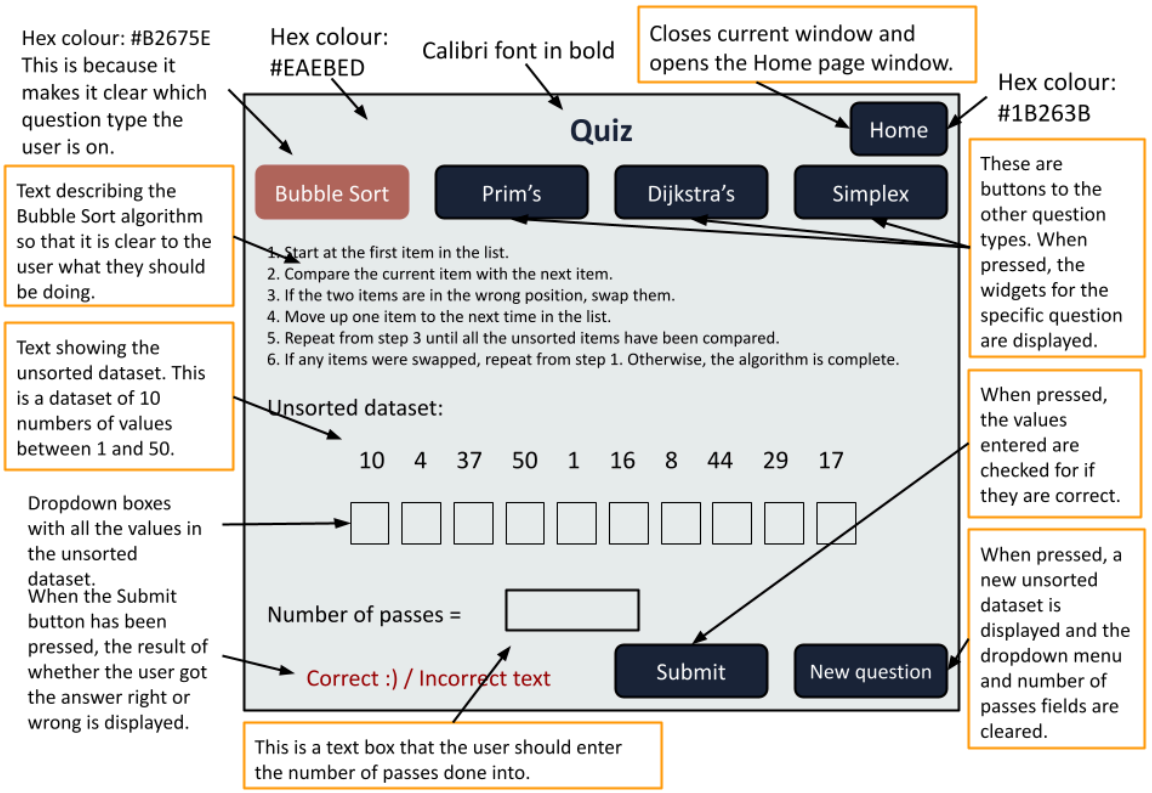
entry fields	enter their answer to the question so that it can be checked. Scores can then be updated.	text indicating to enter the optimal value of x, y and P.

GUI

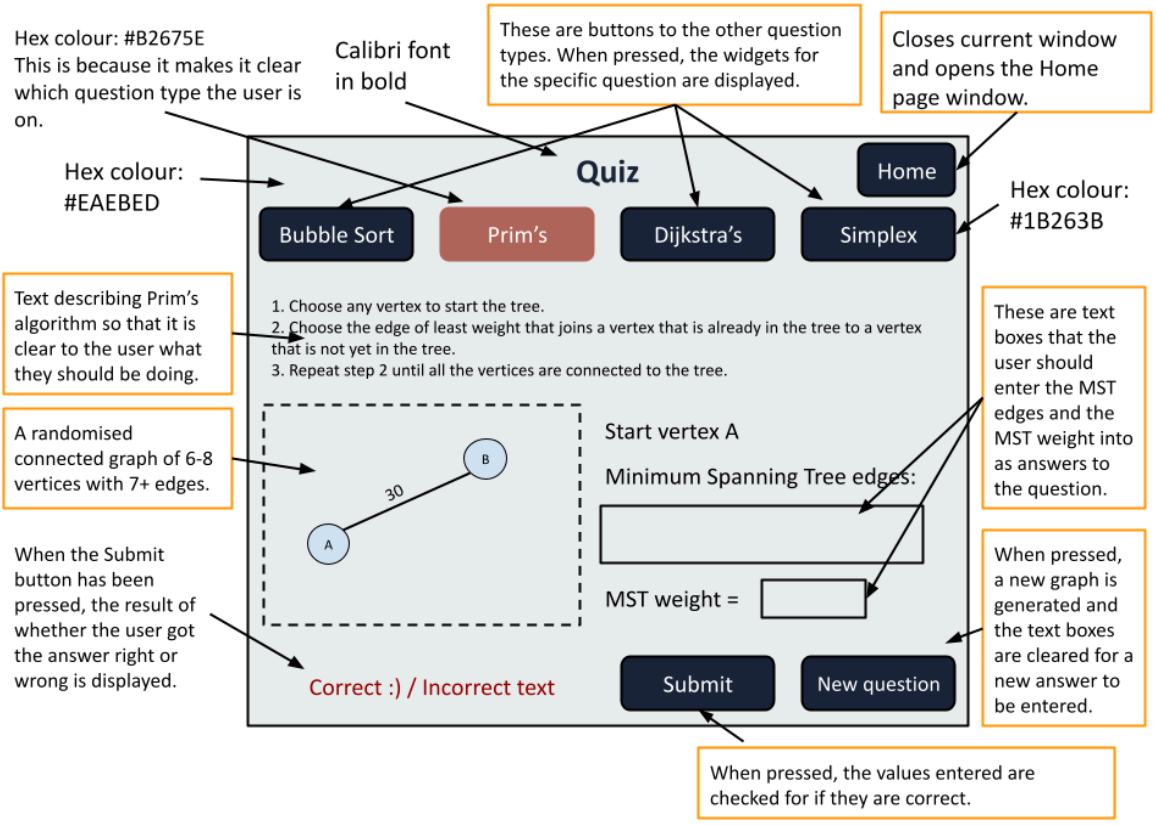
The quiz section has 5 different screens that will be able to be used by the user. Firstly, when the user presses the Quiz button on the Home page, the following is output:



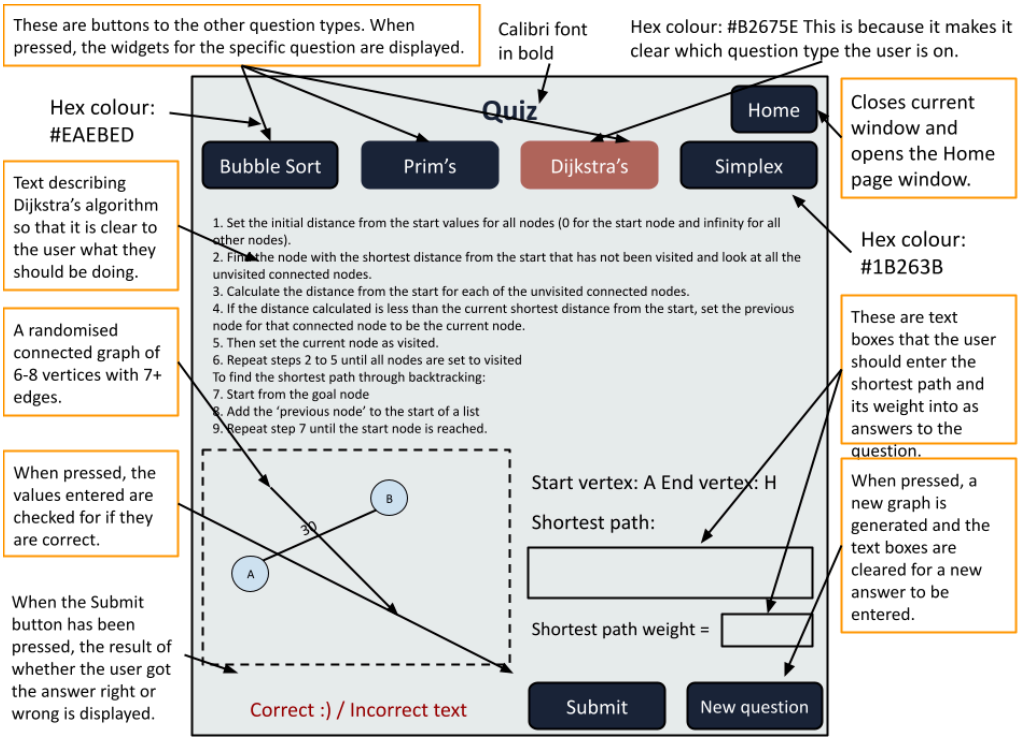
The GUI for this initial quiz page is quite plain so I may try to find something to make this page look more visually appealing during development. Then, for a Bubble Sort quiz question, the following is output:



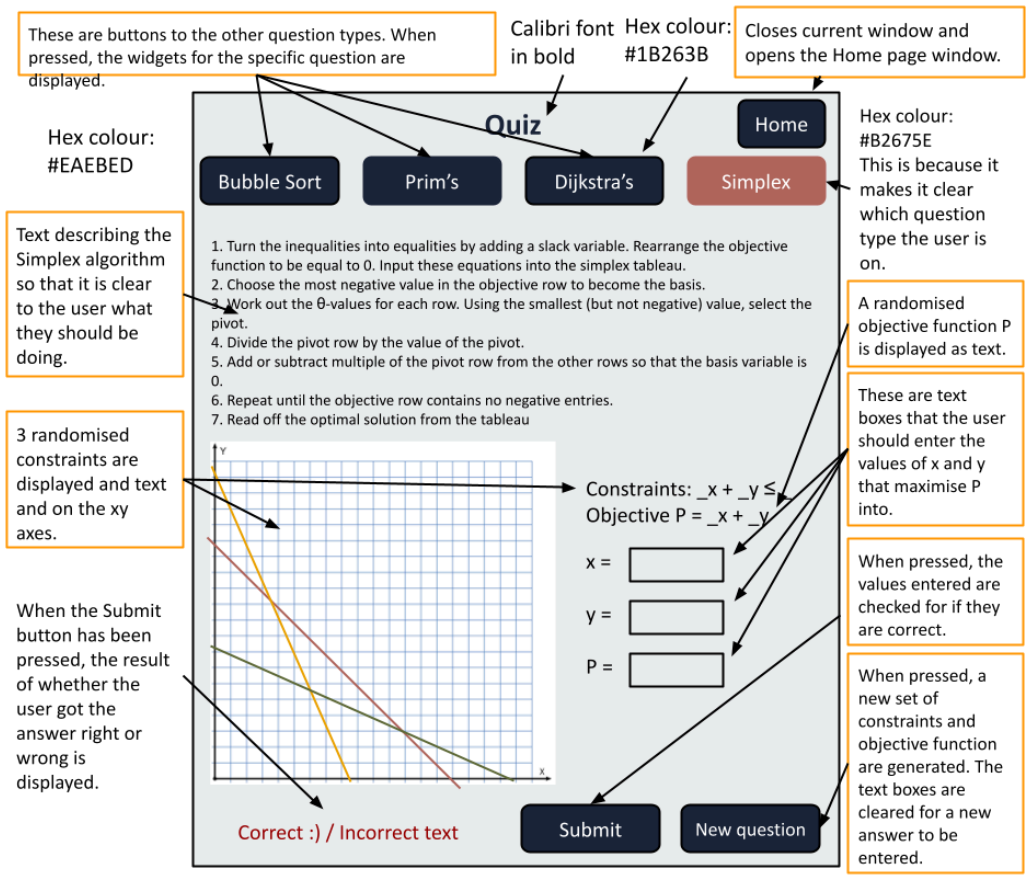
For a Prim's algorithm question, the following is output:



For a Dijkstra's algorithm quiz question, the following is output:



For a Simplex algorithm quiz question, the following is output:



PSEUDOCODE

```
PUBLIC PROCEDURE bubbleSortDataset
      quizNumbers = []
      FOR i in range(10)
            Number = randint(1,50) // use randint from random library
            quixNumbers append Number
      NEXT i
END PROCEDURE

PUBLIC PROCEDURE randomGraph
      // use prim's to make the MST
      numVertices = raindint(6,8)
      // A=65, count up from that:
      Vertices = [char(65+1) for i in range(numVertices)]
      Edges = []
      startVertex = Vertices[0]
      Visited = set([startVertex]) //
      Queue = []
      // add all edges from start
      FOR v in vertices
            IF v!= startVertex THEN
                  Weight = randint(1,50)
                  Enqueue (queue, (weight, startVertex, v))
            END IF
      NEXT v
      WHILE queue // while not empty
            Weight, u ,v = dequeue(queue)
            IF v not in visited THEN
                  Add v to visited
                  edges.append((u,v,weight))
                  FOR w in vertices
                        IF w != v AND w not in visited THEN
                              // add edge
                              Weight = randint(1,50)
                              Enqueue ( queue, (weight, v, w) )
                        END IF
                  NEXT w
            END IF
            IF len(visited) == numVertices THEN
                  // MST has been created
                  Break
      END WHILE
      // add random extra edges
      extraEdges = randint(4,10) // number
      allPoss Edges = [(u,v) for u in vertices for v in vertices if u<v]
      shuffle(allPoss Edges)
      totalEdges = extraEdges + len(edges)
      FOR u,v in allPossEdges
            IF len(edges) > = totalEdges THEN
                  // all extra edges have been added
                  Break
```

```
            END IF
            IF (u,v) not in edges and (v,u) not in edges THEN
                  Weight = randint(1,50)
                  edges.append((u,v,weight))
            END IF
      NEXT u,v
END PROCEDURE

PUBLIC PROCEDURE simplexConstraints
      quizConstraints = []
      quizObjective = []
      WHILE len(quizConstraints) < 3
            xCoef = randint(-10, 10)
            yCoef = randint(-10, 10)
            rhs = randint(-10, 10)
            IF xCoef !=0 or yCoef != 0 THEN
                  // x and y are not both 0
                  quizConstriants.append([xCoef, yCoef, rhs])
            END IF
      WHILE len(quizObjective) < 1
            xCoef = randint(-10, 10)
            yCoef = randint(-10, 10)
            IF xCoef != 0 OR yCoef != 0 THEN
                  // x and y are not both 0
                  quizObjective.append(xCoef)
                  quizObjective.append(yCoef)
            END IF

PUBLIC PROCEDURE check
      IF current question type == "Bubble Sort" THEN
            bubbleSortSteps() // use the bubbleSort class method
            correctOrder = list(steps[-1])
            correctPasses = number Passes
            IF userOrder == correctOrder and userPasses == correctPasses
THEN
                  Update text to "Correct"
                  updateScore("Correct")
            ELSE
                  Update text to "Incorrect"
                  updateScore("Incorrect")
            END IF
      END IF

      IF current question type == "Prim's" THEN
            primSteps() // use prim class method
            // get alphabetical order:
            correctMST = set(edges in primSteps[-1])
            correctWeight = sum(weights of edges in correctMST)
            IF userMST == correctMST and userWeight == correctWeight THEN
                  Update text to "Correct"
                  updateScore("Correct")
            ELSE
```

```
                  Update text to "Incorrect"
                  updateScore("Incorrect")
            END IF
      END IF

      IF current question type == "Dijkstra's" THEN
            dijkstraSteps() // use dijkstra class method
            correctPath = steps[-1]
            correctWeight = distances[end Vertex]
            IF userPath == correctPath and userWeight == correctWeight
THEN
                  Update text to "Correct"
                  updateScore("Correct")
            ELSE
                  Update text to "Incorrect"
                  updateScore("Incorrect")
            END IF
      END IF

      IF current question type == "Simplex" THEN
            simplexSteps() // use simplex class method
            correctX = steps[-1][0]
            correctY = steps[-1][1]
            correctP = steps[-1][2]
            IF userX == correctX and userY == correctY and userP ==
correctP THEN
                  Update text to "Correct"
                  updateScore("Correct")
            ELSE
                  Update text to "Incorrect"
                  updateScore("Incorrect")
            END IF
      END IF
END PROCEDURE

PUBLIC PROCEDURE updateScore (answer)
      // takes correct or incorrect as the parameter answer
      IF answer == "Correct"
            SQL query to increase CorrectScore in StudentEnrolment
      ELSE
            SQL query to increase IncorrectScore in StudentEnrolment
      END IF
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
currentQType	string	Attribute of Quiz class	Tracks the current quiz question type selected so that the correct widgets are displayed.	“Bubble Sort”
weight	integer	Local to randomGraph method	A random integer between 1 and 50 that becomes the weight for the current edge being added to the graph. This is required for creating Prim’s and Dijkstra’s quiz questions.	42
u	string	Local to randomGraph method	Tracks the vertices in the graph so that edges can be created between them. This is the start vertex for the edge. This is required for creating Prim’s and Dijkstra’s quiz questions.	“A”
v	string	Local to randomGraph method	Tracks the vertices in the graph so that edges can be created between them. This is the end vertex for the	“C”

			edge. This is required for creating Prim's and Dijkstra's quiz questions.	
xCoef	integer	Local to simplexConstraints method	This is a random integer between -10 and 10, which is required for creating the constraints and objective function for Simplex quiz questions to be generated.	2
yCoef	integer	Local to check method	This is a random integer between -10 and 10, which is required for creating the constraints and objective function for Simplex quiz questions to be generated.	-9
rhs	integer	Local to check method	This is a random integer between -10 and 10, which is required for creating the constraints for Simplex quiz questions to be generated.	0
correctOrder	List [integer]	Local to check method	This is the sorted order for Bubble Sort quiz questions. It is required for checking whether the user's input is correct or not.	[1, 2, 3, 4, 5, 6, 6, 8, 10, 15]

correctPasses	integer	Local to check method	This is the number of passes done in the execution of the Bubble Sort algorithm. It is required for checking whether the user input the correct number of passes.	3
correctMST	set	Local to check method	This is the correct final set of MST edges for performing Prim's algorithm on the graph generated. It is required for checking whether the user got the question right or wrong.	{"AB", "BC, "CD", "EH", "AG"}
correctWeight	integer	Local to check method	This is the correct weight of the MST or shortest path for the given question. It is required for checking whether the user input the correct weight or not for Prim's or Dijkstra's questions.	97
correctPath	Tuple (string)	Local to check method	This is the correct shortest path between the start and end vertices for performing Dijkstra's algorithm on the graph generated. It is required for checking	(A, B, F)

			whether the user got the question right or wrong.	
correctX	fraction	Local to check method	This is the correct optimal value of x for the generated constraints and objective function. It is required for checking whether the user got the Simplex question right or wrong.	3
correctY	fraction	Local to check method	This is the correct optimal value of y for the generated constraints and objective function. It is required for checking whether the user got the Simplex question right or wrong.	1/2
correctP	fraction	Local to check method	This is the correct optimal value of P for the generated constraints and objective function. It is required for checking whether the user got the Simplex question right or wrong.	5/2

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
Quiz	class	Global	Manages the functionality and usability features of the Quiz page. A class is used because it allows for the use of methods and attributes specific to the user that has logged in.
quizNumbers	List [integer]	Attribute of Quiz class	Stores a list of the random numbers to be sorted by the user and for checking, therefore allowing Bubble Sort questions to be completed.
quizGraph	NetworkX Graph	Attribute of Quiz class	Stores the randomised graph created so that this can be used for either Prim's or Dijkstra's algorithm quiz questions, therefore allowing them to be completed by the user and checked.
queue	queue	Local to randomGraph method	Stores the edges and their weights in a queue so that they can be processed and an MST can be created.
vertices	List [char]	Local to randomGraph method	Stores a list of all the randomised vertices in the graph being generated for Prim's and Dijkstra's

			questions.
edges	List	Local to randomGraph method	Stores all the edges in the graph being randomised for Prim's and Dijkstra's questions.
visited	set	Local to randomGraph method	Stores all the vertices that have been visited in the Prim's algorithm style of generating an MST, therefore creating a base for the extra edges to be added to make a graph that is guaranteed to be connected.
extraEdges	integer	Local to randomGraph method	Stores the number of edges to be added to the MST to make the graph that is to be used for Prim's and Dijkstra's questions.
allPossEdges	List [tuple]	Local to randomGraph method	Stores all the edges possible for the graph, which can then be shuffled and random edges from this can be added to the graph so Prim's and Dijkstra's questions can be completed.
quizObjective	List [float]	Attribute of Quiz class	Stores the coefficients of x and y for the objective function, which is required for creating and checking Simplex algorithm questions.

quizConstraints	List [list [integer]]	Attribute of Quiz class	Stores the coefficients of x y and rhs for the 3 randomise constraints so that they can be plotted on the axes and used for checking the Simplex algorithm.

WHITE BOX TEST DATA

This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Open the quiz page for an account with different algorithms chosen.	Only buttons for the selected algorithms during registration are displayed next to each other at the top of the page.	This ensures that the user's algorithm choices are followed on the quiz page, whatever the user's original choice was.	
Press the Bubble Sort button.	The Bubble Sort button is highlighted in pink and text describing the algorithm is below this. Dropdown menus for the unordered dataset are displayed below it and there is an entry field for the number of passes.	This ensures that the correct widgets required for the user to complete a Bubble Sort question are displayed and that the randomised dataset functionality works correctly.	
Submit a Bubble Sort question with both the dropdown menus and number of passes correctly filled in.	Text will appear saying 'Correct' and the correct score in the database is updated.	This ensures that the program can validate a correct answer as being correct.	
Submit a Bubble Sort question with only the dropdown menus filled in correctly.	Text will appear saying 'Incorrect number of passes' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Bubble Sort question with only the number of passes filled in correctly.	Text will appear saying 'Incorrect order' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Bubble Sort question with the dropdown menus and number of passes	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	

incorrectly filled in.			
Press the New Question button on the Bubble Sort page.	A new dataset is randomised and displayed. The dropdown menus are cleared and the new dataset becomes the options. The number of passes entry field is cleared.	This ensures that the user can generate a new question of the Bubble Sort algorithm type when they press this button.	
Press the Prim's button.	The Prim's button is highlighted in pink and text describing the algorithm is below this. A graph is displayed on the left side of the screen and there are text boxes for the MST edges and weight on the right.	This ensures that the correct widgets required for the user to complete a Prim's algorithm question are displayed and that the randomised graph functionality works correctly.	
Submit a Prim's algorithm question with both the MST and weight fields correctly filled in.	Text will appear saying 'Correct' and the correct score in the database is updated.	This ensures that the program can validate a correct answer as being correct.	
Submit a Prim's algorithm question with only the MST filled in correctly.	Text will appear saying 'Incorrect weight' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Prim's algorithm question with only the weight filled in correctly.	Text will appear saying 'Incorrect MST edges' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Prim's algorithm question with the MST and weight incorrectly filled in.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Press the New Question button on	A new graph is randomised and	This ensures that the user can generate a	

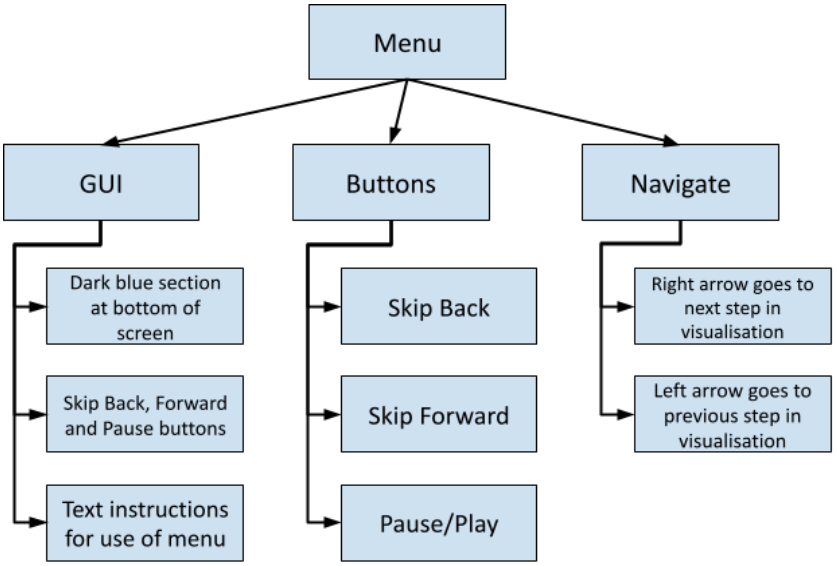
the Prim's page.	displayed. The MST and weight entry fields are cleared.	new question of the Prim's algorithm type when they press this button.	
Press the Dijkstra's button.	The Dijkstra's button is highlighted in pink and text describing the algorithm is below this. A graph is displayed on the left side of the screen and there are text boxes for the shortest path and weight on the right.	This ensures that the correct widgets required for the user to complete a Dijkstra's algorithm question are displayed and that the randomised graph functionality works correctly.	
Submit a Dijkstra's algorithm question with both the shortest path and weight fields correctly filled in.	Text will appear saying 'Correct' and the correct score in the database is updated.	This ensures that the program can validate a correct answer as being correct.	
Submit a Dijkstra's algorithm question with only the shortest path filled in correctly.	Text will appear saying 'Incorrect weight' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Dijkstra's algorithm question with only the weight filled in correctly.	Text will appear saying 'Incorrect shortest path' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Dijkstra's algorithm question with the shortest path and weight incorrectly filled in.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Press the New Question button on the Dijkstra's page.	A new graph is randomised and displayed. The shortest path and weight entry fields are cleared.	This ensures that the user can generate a new question of the Dijkstra's algorithm type when they press this button.	
Press the Simplex	The Simplex button is	This ensures that the	

button.	highlighted in pink and text describing the algorithm is below this. Constraints are displayed as text and on a set of xy axes. An objective function is displayed as text. There are text boxes for the user to enter the optimal values of x y and P.	correct widgets required for the user to complete a Simplex algorithm question are displayed and that the randomised constraints and objective functionality work correctly.	
Submit a Simplex algorithm question with the x y and P values correctly filled in.	Text will appear saying 'Correct' and the correct score in the database is updated.	This ensures that the program can validate a correct answer as being correct.	
Submit a Simplex algorithm question with only the x field filled in correctly.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Simplex algorithm question with only the y field filled in correctly.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Simplex algorithm question with only the P field filled in correctly.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Submit a Simplex algorithm question with the x y and P fields incorrectly filled in.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	
Press the New Question button on the Simplex page.	A new set of constraints and objective function are randomised and displayed. The x y and P entry fields are cleared.	This ensures that the user can generate a new question of the Simplex algorithm type when they press this button.	
Press the Home	The Quiz page is	This ensures that the	

button.	closed and the Home page is opened.	'Home' button works correctly and that the user can go back to the Home page to either log out or go to the visualisation pages.	

MENU DESIGN

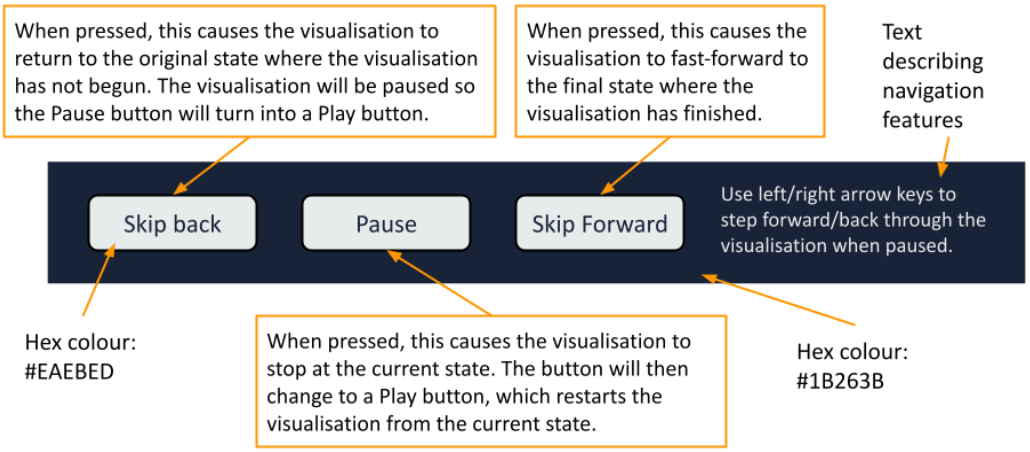
DECOMPOSITION



SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW
GUI	This will be the section of each visualisation page that contains all the usability features allowing the user to navigate the visualisation.	The Menu will be at the bottom of each visualisation page. It will have a #1B263B background colour, buttons for Skipping back and forward and a pause button. There will also be text describing how to navigate the visualisation using the arrow keys.
Skip Back button	This is one of the buttons that allows the user to navigate the visualisation.	When pressed, this button returns the visualisation to the original state - the visualisation will not have been started. The visualisation will be paused.
Skip Forward button	This is another button that allows the user to navigate the	When pressed, this button fast-forwards the visualisation to the

	visualisation.	final state (e.g. with the dataset sorted or the shortest path found and shown). It causes all sleep delays to be ignored.
Pause/Play button	This is another button that allows the user to navigate the visualisation by stopping it and restarting it.	When pressed, the visualisation will stop at the current state in the visualisation. It will then change to a 'Play' button, which causes the visualisation to restart from where it was left paused.
Navigation	This allows the user to navigate the visualisation at their own pace by using their arrow keys.	When the visualisation is paused, pressing the right arrow key will cause the next step in the visualisation to be shown. When the left arrow key is pressed, the previous step in the visualisation is shown.

GUI



For the algorithm visualisations, I will pass the start row and width of the menu into the class, therefore allowing me to choose the location of each button within the Menu for each visualisation.

PSEUDOCODE

```
PUBLIC PROCEDURE skipBack
      currentStep = 0 // return visualisation to the first step
      isPaused = True
      IF has attribute ("updateChart") THEN
            // bubble sort
            updateChart(steps[currentStep])
      ELSE IF has attribute … THEN
```

```
            // prim's
            updatePrimGraph(steps[currentStep])
      ELSE IF has attribute … THEN
            // dijkstra's
            updateDijkstraGraph(steps[currentStep])
            Update text in vertex table
      ELSE IF has attribute ("updateSimplexGraph") THEN
            // simplex
            updateSimplexGraph(steps[currentStep])
      END IF
      invalidMessage = ""
END PROCEDURE

PUBLIC PROCEDURE skipForward
      currentStep = len(steps) -1 // go to final step
      isPaused = True
      IF has attribute ("updateChart") THEN
            // bubble sort
            updateChart(steps[currentStep])
      ELSE IF has attribute ("updatePrimGraph") THEN
            // prim's
            updatePrimGraph(steps[currentStep])
      ELSE IF has attribute ("updateDijkstraGraph") THEN
            // dijkstra's
            updateDijkstraGraph(steps[currentStep])
            Update text in vertex table
      ELSE IF has attribute ("updateSimplexGraph") THEN
            // simplex
            updateSimplexGraph(steps[currentStep])
            Invalid message = "optimal solution at … "
      END IF
      invalidMessage = ""
END PROCEDURE

PUBLIC PROCEDURE togglePlayPause
      isPaused = not isPaused // switch state to the opposite
      playPauseButton.configure(if ifPaused text="Pause" else "Play")
      IF not isPaused THEN
            IF has attribute ("updateChart") THEN
                  // bubble sort
                  bubbleSortAnimate()
            ELSE IF has attribute ("updatePrimGraph") THEN
                  // prim's
                  primAnimate()
            ELSE IF has attribute ("updateDijkstraGraph") THEN
                  // dijkstra's
                  dijkstraAnimate()
            ELSE IF has attribute ("updateSimplexGraph") THEN
                  // simplex
                  simplexAnimate()
            END IF
      END IF
```

```
END PROCEDURE

PUBLIC PROCEDURE stepBack
      IF isPaused and currentStep > 0 THEN
            currentStep -= 1
            IF has attribute (“updateChart”) THEN
                  updateChart(steps, steps indices)
                  // indices are current swapping items
            ELSE IF has attribute (“updatePrimGraph”) THEN
                  // prim’s
                  updatePrimGraph(steps[currentStep])
            ELSE IF has attribute (“updateDijkstraGraph”) THEN
                  // dijkstra's
                  updateDijkstraGraph(steps[currentStep])
                  Update text in vertex table
            ELSE IF has attribute (“updateSimplexGraph”) THEN
                  // simplex
                  updateSimplexGraph(steps[currentStep])
            END IF
      END IF
END PROCEDURE

PUBLIC PROCEDURE stepForward
      IF isPaused and currentStep > 0 THEN
            currentStep += 1
            IF has attribute (“updateChart”) THEN
                  updateChart(steps, steps indices)
                  // indices are current swapping items
            ELSE IF has attribute (“updatePrimGraph”) THEN
                  // prim’s
                  updatePrimGraph(steps[currentStep])
            ELSE IF has attribute (“updateDijkstraGraph”) THEN
                  // dijkstra's
                  updateDijkstraGraph(steps[currentStep])
                  Update text in vertex table
            ELSE IF has attribute (“updateSimplexGraph”) THEN
                  // simplex
                  updateSimplexGraph(steps[currentStep])
                  IF is final step THEN
                        Invalid message = “optimal solution at … “
                  END IF
            END IF
      END IF
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
master	Tk	Attribute of Menu class	This is the main window that is passed into the class, which is required to ensure the proper Tkitner user interface hierarchy.	N/A
rowStart	integer	Attribute of Menu class	Stores the starting row index for the menu, ensuring the correct placement in the grid layout for the specific algorithm visualisation.	12
menuLength	integer	Attribute of Menu class	Defines the number of the columns the menu spans, ensuring the proper layout in the user interface for the specific algorithm visualisation.	19
currentStep	integer	Attribute of Menu class	Keeps track of the current step in the visualisation process, which is required to allow navigation between steps.	5

isPaused	Boolean	Attribute of Menu class	Determines whether the visualisation is currently running or is paused, so that the user's can navigate between steps.	True/False
playPauseButton	Button	Attribute of Menu class	Stores the play/pause button to allow the visualisation states to be toggled between dynamically.	N/A
steps	list	Attribute of Menu class	Holds the sequence of steps in algorithm execution, allowing users to navigate between them.	[[3,1,4,5,6,7,9,14], …]

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
menuFrame	Frame	Attribute of Menu class	Groups lal the menu-related Tkinter widgets into one frame for a structured GUI.
steps	list	Attribute of Menu class	Holds the sequence of steps in algorithm execution, allowing users to navigate between them.

WHITE BOX TEST DATA

This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

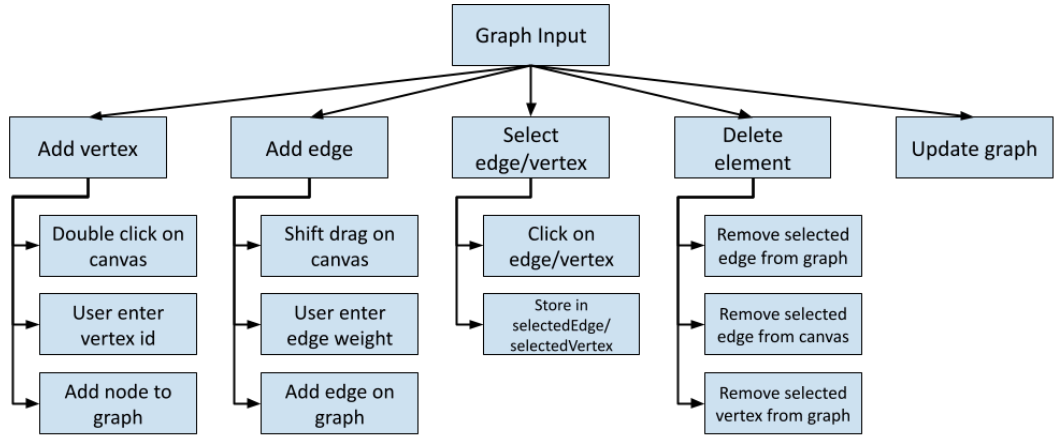
TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Press the 'Skip Forward' button for the Bubble Sort visualisation	The text-based visualisation displays all the swaps and the bar chart updates to show the final sorted values.	Confirms that the 'Skip Forward' button works as intended for the Bubble Sort visualisation.	
Press the 'Skip Backward' button for the Prim's visualisation	The text-based visualisation displays the final edges in the MST and the graph has the MST edges highlighted in #606C38	Confirms that the 'Skip Forward' button works as intended for the Prim's visualisation.	
Press the 'Skip Forward' button for the Dijkstra's visualisation	The vertex table displays the final values in each of the rows and the graph has the shortest path highlighted in #B2675E.	Confirms that the 'Skip Forward' button works as intended for the Dijkstra's visualisation.	
Press the 'Skip Forward' button for the Simplex visualisation	The final path from the origin to the optimal vertex is highlighted on the graph and the values of x y and P are displayed as text.	Confirms that the 'Skip Forward' button works as intended for the Simplex visualisation.	
Press the 'Skip Back' button for the Bubble Sort visualisation	The dataset remains the same but the initial, unsorted values are displayed as text and on the bar chart.	Confirms that the 'Skip Back' button works as intended for the Bubble Sort visualisation.	
Press the 'Skip Back' button for the Prim's visualisation	The graph remains the same but the highlighting on the graph is removed.	Confirms that the 'Skip Back' button works as intended for the Prim's visualisation.	

Press the 'Skip Back' button for the Dijkstra's visualisation	The graph remains the same but the highlighting on the graph is removed. The vertex table returns to its original blank state.	Confirms that the 'Skip Back' button works as intended for the Dijkstra's visualisation.	
Press the 'Skip Back' button for the Simplex visualisation	The blank graph is displayed with just the constraints shown. The text is blank.	Confirms that the 'Skip Back' button works as intended for the Simplex visualisation.	
Press the 'Pause' button for the Bubble Sort visualisation	The visualisation stops, the button changes from 'Pause' to 'Play' (or vice versa).	Confirms that the 'Pause' button works as intended for the Bubble Sort visualisation.	
Press the 'Pause' button for the Prim's visualisation	The visualisation stops, the button changes from 'Pause' to 'Play' (or vice versa).	Confirms that the 'Pause' button works as intended for the Prim's visualisation.	
Press the 'Pause' button for the Dijkstra's visualisation	The visualisation stops, the button changes from 'Pause' to 'Play' (or vice versa).	Confirms that the 'Pause' button works as intended for the Dijkstra's visualisation.	
Press the 'Pause' button for the Simplex visualisation	The visualisation stops, the button changes from 'Pause' to 'Play' (or vice versa).	Confirms that the 'Pause' button works as intended for the Simplex visualisation.	
Use the right and left arrow keys for the Bubble Sort visualisation	The visualisation goes back/forward one step in the visualisation on both the bar chart and as text.	Confirms that the arrow key functionality works as intended for the Bubble Sort visualisation.	
Use the right and left arrow keys for the Prim's visualisation	The visualisation goes back/forward one step in the visualisation on both the graph and text.	Confirms that the arrow key functionality works as intended for the Prim's visualisation.	

Use the right and left arrow keys for the Dijkstra's visualisation	The visualisation goes back/forward one step in the visualisation on both the graph and the vertex table.	Confirms that the arrow key functionality works as intended for the Dijkstra's visualisation.	
Use the right and left arrow keys for the Simplex visualisation	The visualisation goes back/forward one step in the visualisation on the graph. If the final stage is reached, the values of P x and y are displayed as text.	Confirms that the arrow key functionality works as intended for the Simplex visualisation.	

GRAPH INPUT DESIGN

DECOMPOSITION

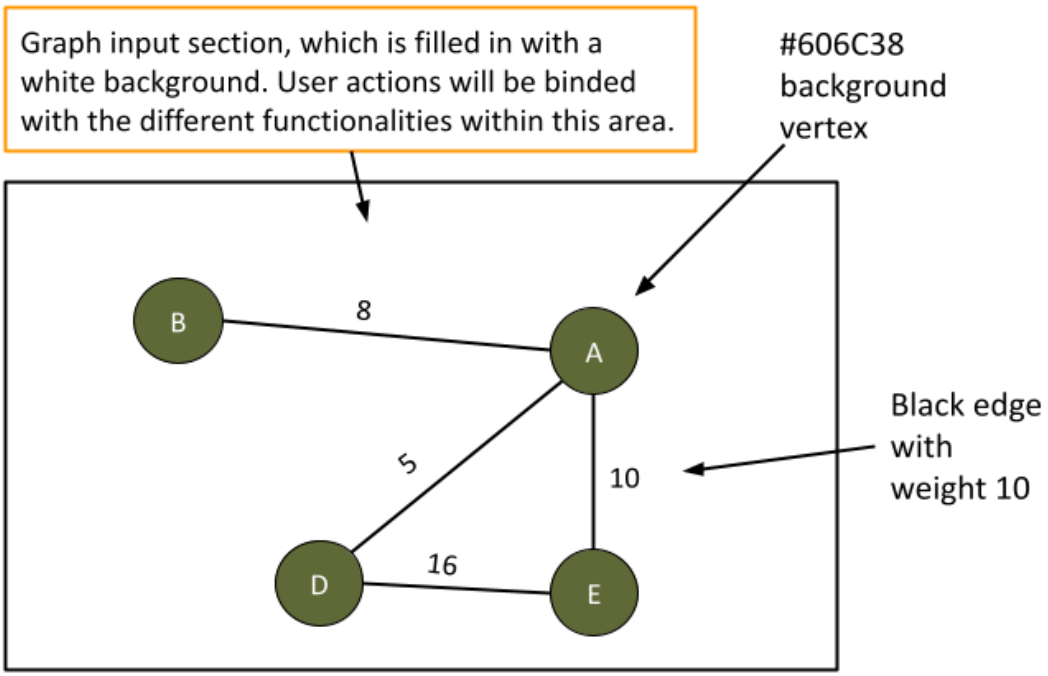


SUBMODULE	REASON FOR DECOMPOSITION	BRIEF OVERVIEW
Add vertex	This is the first section of the functionality of the Graph Input section. This functionality can be further decomposed into the user's entries and the displaying of the vertex on the graph.	When the user double clicks within the graph, a pop up will ask the user to enter a unique vertex ID. A check will occur that this ID has not been used before. Then, the vertex will be displayed on the graph as a circle, filled with the colour #606C38, with the ID in the centre.
Add edge	This is the second section of the functionality of the Graph Input section. This functionality can be further decomposed into the user's entries and the displaying of the	When the user does shift+ drag between two vertices, a pop up will ask the user to enter a weight that is greater than 0. Then, the edge will be displayed as a black line on the

	edge on the graph.	graph between the two vertices and the weight on the edge.
Select edge or vertex	This is the third section of the functionality of the Graph Input section and is required for the deleting of an element to take place.	Checks will occur for if a user clicking on an element occurs. If it does occur, the vertex or edge will be stored in the selectedVertex or selectedEdge attribute.
Delete element	This is the fourth section of the functionality of the Graph Input section and is required to allow the user to remove an element from the graph.	If the user presses the delete key and an element has been selected, that element will be removed from the graph. If a vertex is being deleted and an edge is connected to it, the edge will also be deleted.
Update graphs	This is the subsection of the Graph Input section that ensures that the user interface remains up-to-date if a change has been made.	When one of the functions add vertex, add edge or delete element have occurred, this method will be called to ensure that the graph displays the correct version of the graph that is currently being stored.

GUI

As this Graph Input section is a part of the Prim's and Dijkstra's visualisation pages, the diagram of the GUI is quite simplistic but shows what I expect a graph to look like when the user has entered one. The simplicity is due to the fact that this decomposed subsection of the project is to create a smaller, functionality focused reusable piece of code.



PSEUDOCODE

```
PUBLIC PROCEDURE addVertex
      IF len(graphNodes) < 10 THEN // nodes must be between 4-10
            vertexID = dialog("Vertex Input", "Enter ID:")
            IF len(vertexID)==1 AND vertexID not in graphNodes THEN
                  graph.add_node(vertexID, pos=(x,y))
                  updateVisualisation()
            END IF
      END IF
END PROCEDURE

PUBLIC PROCEDURE selectCreateEdge
      FOR node, (x,y) in graph
            IF selectedVertex is None THEN
                  selectedVertex = node
            ELSE IF selectedVertex != node THEN
                  targetVertex = node
                  weight = dialog("Weight input", "Enter weight:")
                  IF weight > 0 THEN // enforce weight to be positive
integer
                        graph.add_edge(selectedVertex,targetVertex,
weight)
                        updateVisualisation()
                  END IF
                  // clear previous selections so doesn't keep adding edge
                  selectedVertex = None
                  targetVertex = None
            END IF
```

```
            RETURN
      NEXT node, (x,y)

      FOR edge in graph
            IF edge pressed THEN
                  selectedEdge = edge
                  RETURN
            END IF
      NEXT edge
END PROCEDURE

PUBLIC PROCEDURE deleteElement
      IF selectedVertex not None THEN
            IF node, (x,y) selected THEN
                  graph.remove_node(node)// specific to NetworkX
                  updateVisualsiation()
                  selectedVertex = None // clear previous selection
            END IF
      IF selectedEdge not None THEN
            FOR edge in graph edges
                  IF edge pressed THEN
                        graph.remove_edge(selectedEdge)//remove
connections
                        updateVisualisation()
                        selectedEdge = None // clears previous selection
                        RETURN
                  END IF
            NEXT edge
      END IF
END PROCEDURE

PUBLIC PROCEDURE updateVisualisation
      // note specifics will be applied to the NetworkX library
      ax.clear()
      Draw graph
END PROCEDURE
```

VARIABLES AND VALIDATION TABLE

VARIABLE	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION	SAMPLE TEST DATA
master	Tk	Attribute of GraphInput class	Reference to the main Tkinter window, allowing the UI and widgets to be created.	N/A
graph	Graph	Attribute of GraphInput class	Stores the graph representation using the NetworkX library. This is to allow the tracking of nodes and edges.	N/A
selectedVertex	string	Attribute of GraphInput class	This stores the currently selected vertex. It is needed for edge creation and deletion.	“A”
selectedEdge	tuple	Attribute of GraphInput class	This stores the currently selected edge for deletion.	(“A”, “B”)
vertexID	string	Local to addVertex method	This stores the unique identifier for a vertex being added, which must be a unique single character. It is needed to allow vertices to be created.	“C”

weight	integer	Local to selectCreateEdge method	This stores the weight entered for an edge being created. The value must be an integer greater than 0.	5
targetVertex	string	Attribute of GraphInput class	This stores the target vertex, which is needed to allow edges to be created.	“B”

DATA STRUCTURES TABLE

NAME	DATA TYPE	SCOPE	PURPOSE & JUSTIFICATION
graph	nx.Graph()	Attribute of the GraphInput class	Holds the graph structure using NetworkX, storing the nodes and edges, therefore allowing them to be displayed.
pos	dict	Attribute of the GraphInput class	Stores the positions of the nodes, allowing visualisation. This is required because the NetworkX library is being used.
edgeLabels	dict	Attribute of the GraphInput class	Stores the weight of each edge in the graph.

WHITE BOX TEST DATA

This is the white box test data, which the programmer uses to test and know how to fix the errors based on their knowledge of the code.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	ACTUAL OUTPUT
Double click on the canvas	Pop up to enter the vertex ID appears.	Confirms that the functionality to add a vertex works as intended.	
Enter a single character as the ID	Vertex created and displayed on the graph.	Confirms that a valid input for the vertex ID allows a vertex to be added.	
Enter multiple characters as the ID	Error message so the vertex is not created.	Confirms that an invalid ID input does not allow a vertex to be created.	
Enter an already used ID as the vertex ID.	Error message so the vertex is not created.	Ensures that all vertices have unique IDs, therefore confirming that an invalid ID input does not allow a vertex to be created.	
Hold shift and drag between two vertices.	A pop up to enter the edge weight appears.	Confirms that the functionality to add an edge works as intended.	
Enter a positive integer as the weight.	An edge between the two selected vertices is created.	Confirms that a valid weight input allows an edge to be displayed on the graph.	
Enter a float/real number for the weight.	An error message appears and the edge is not created between the two vertices.	Confirms that an invalid weight input prevents an edge from being created.	
Enter a negative number for the weight.	An error message appears and the edge is not created	Confirms that an invalid weight input prevents an edge from	

	between the two vertices.	being created.	
Select a vertex and press delete.	The vertex is removed from the graph, as well as any connected edges.	Confirms that the functionality to remove a vertex works as intended.	
Select an edge and press delete.	The edge is removed from the graph.	Confirms that the functionality to delete an edge works as intended.	

BLACK BOX TEST DATA

This is the type of testing where the software is tested without the testers being aware of the internal structure of the code. One of my stakeholders will be carrying out this testing.

SUCCESS CRITERIA	INPUT	EXPECTED RESULT	JUSTIFICATION	REAL RESULT
[1][a,b,c,e]	Open the program.	The login page should load.	This test ensures that the program starts properly and displays the correct window on startup.	
[2][a,b,c,d,e,f,g,h]	Attempt to register successfully.	Once the Register button has been pressed, if all inputs are valid, the home page will be opened. Otherwise, messages will display what has been incorrectly input.	This ensures that the user can register correctly and gain access to the rest of the system.	
[1][d] [3][c,d]	Attempt to login successfully for a student account.	Once the Login button has been pressed, the home page will be opened if the inputs were correct. On the home page, there will be a quiz statistics chart and	This ensures that a student user can log in successfully and gain access to the rest of the system if they have already registered. It also ensures that the correct	

		a button to the quiz page. Otherwise, a message will display if an input is incorrect.	type of home page is displayed.	
[1][d]	Attempt to login successfully for a teacher account.	Once the Login button has been pressed, the home page will be opened if the inputs were correct. Otherwise, a message will display if an input is incorrect.	This ensures that a teacher user can log in successfully and gain access to the rest of the system if they have already registered. It also ensures that the correct type of home page is displayed.	
[3][e]	Attempt to log out.	The login page will be displayed.	This ensures that a user can exit the system properly.	
[3][b] [4][a,b,c,d, e,k]	Open Bubble Sort Visualisaiton page.	The Bubble Sort Visualisation page will open, with fields to enter the dataset into and a bar chart, on which the dataset will be displayed.	This ensures that the Bubble Sort page can be accessed successfully, therefore enabling the visualisation of the algorithm.	
[4][d,f,g,h,i ,j]	Input a dataset and visualise the Bubble Sort algorithm.	An invalid dataset input will result in an error message being output. Otherwise, the sorting of	This ensures that the visualisation functionality and the validation for the Bubble Sort algorithm	

		the dataset will occur in both the text version and on the bar chart.	execute properly.	
[4][l] [9][a,b,c,d, e]	Use the menu to navigate through the visualisation of Bubble Sort.	Pressing Pause stops the visualisation at the current state. Pressing the Skip Back / Forward button changes the visualisation to the initial / final stage - either the initial unsorted dataset or the final sorted dataset. Pressing the arrow keys changes the visualisation to the previous or next step.	This ensures that the menu functions correctly for a Bubble Sort visualisation.	
[3][b] [5][a,b,c,d, e,j]	Open the Prim's Algorithm Visualisaiton page.	The Prim's Algorithm Visualisation page will open, with fields to enter a graph and a start vertex.	This ensures that the Prim's Algorithm page can be accessed successfully, therefore enabling the visualisation of the algorithm.	
[5][f,g,h,i] [10][a,b,c, d,e]	Input a graph and visualise Prim's algorithm.	An invalid dataset input will result in an error message being output. Otherwise, the visualisation of finding	This ensures that the visualisation functionality and the validation for Prim's algorithm execute properly.	

		the MST is executed on the graph and in text.		
[5][k] [9][a,b,c,d, e]	Use the menu to navigate through the visualisation of Prim's algorithm.	Pressing Pause stops the visualisation at the current state. Pressing the Skip Back / Forward button changes the visualisation to the initial / final stage - either the initial unsorted dataset or the final sorted dataset. Pressing the arrow keys changes the visualisation to the previous or next step.	This ensures that the menu functions correctly for a Prim's algorithm visualisation.	
[3][b] [6][a,b,c,d, e,f,k]	Open the Dijkstra's Algorithm Visualisation page.	The Dijkstra's Algorithm Visualisation page will open, with fields to enter a graph, start and end vertices and a blank vertex table.	This ensures that the Dijkstra's Algorithm page can be accessed successfully, therefore enabling the visualisation of the algorithm.	
[6][g,h,i,j] [10][a,b,c, d,e]	Input a graph and visualise Dijkstra's algorithm.	An invalid dataset input will result in an error message being output. Otherwise, the visualisation of finding the shortest path	This ensures that the visualisation functionality and the validation for Dijkstra's algorithm execute properly.	

		between the start and end vertices is executed on the graph and in text.		
[6][l] [9][a,b,c,d, e]	Use the menu to navigate through the visualisation of Dijkstra's algorithm.	Pressing Pause stops the visualisation at the current state. Pressing the Skip Back / Forward button changes the visualisation to the initial / final stage - either the initial unsorted dataset or the final sorted dataset. Pressing the arrow keys changes the visualisation to the previous or next step.	This ensures that the menu functions correctly for a Dijkstra's Algorithm visualisation.	
[3][b] [7][a,b,c,d, e,j]	Open the Simplex Algorithm Visualisation page.	The Simplex Algorithm Visualisation page will open, with fields to enter coefficients for constraints and an objective function.	This ensures that the Simplex Algorithm page can be accessed successfully, therefore enabling the visualisation of the algorithm.	
[7][f,g,h,i]	Input coefficients for the constraints and visualise the Simplex algorithm.	An invalid dataset input will result in an error message being output. Otherwise, the visualisation of finding	This ensures that the visualisation functionality and the validation for the Simplex algorithm execute properly.	

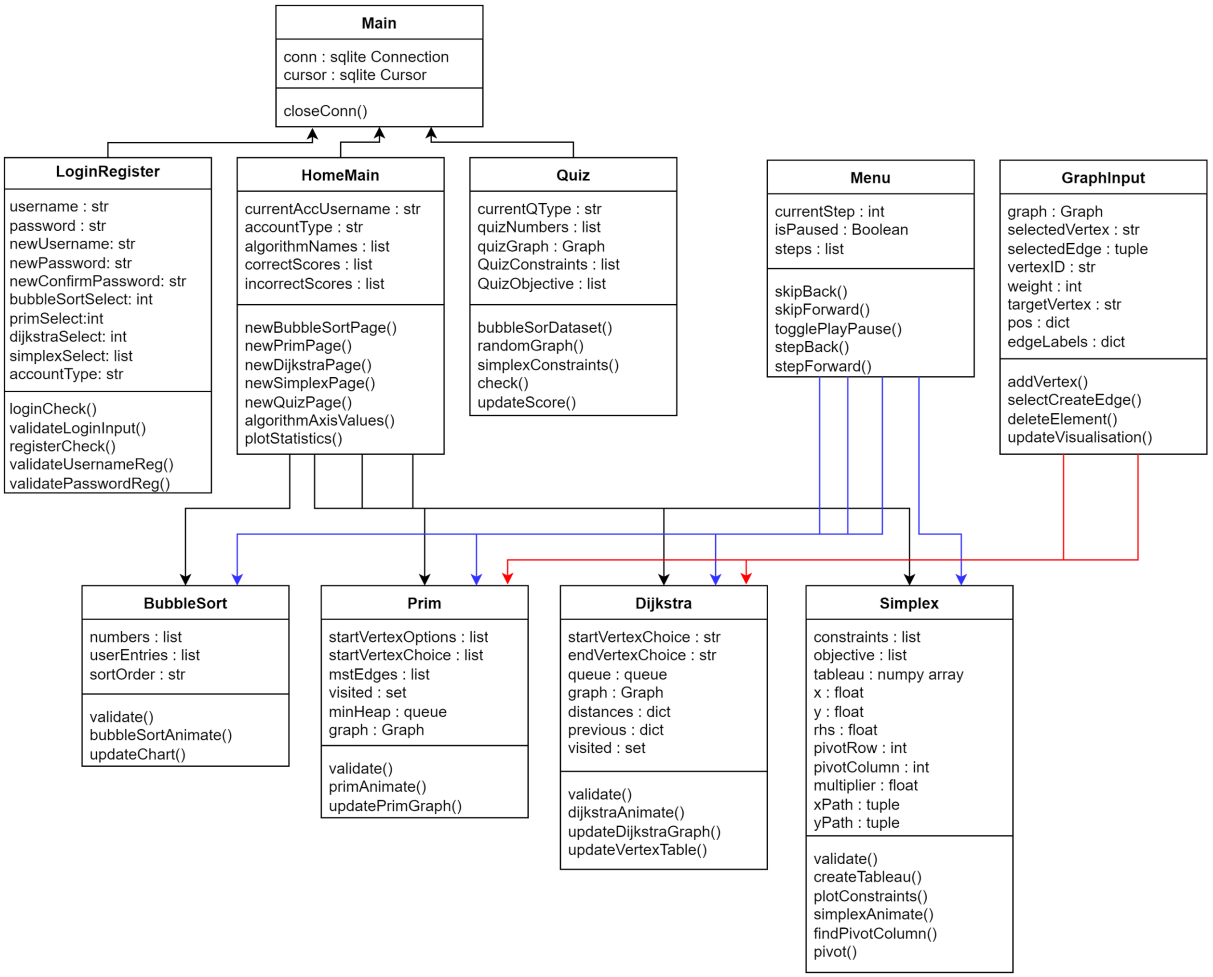
		the optimal value for the constraints and objective function entered is executed on the axes.		
[7][k] [9][a,b,c,d, e]	Use the menu to navigate through the visualisation of the Simplex Algorithm.	Pressing Pause stops the visualisation at the current state. Pressing the Skip Back / Forward button changes the visualisation to the initial / final stage - either the initial unsorted dataset or the final sorted dataset. Pressing the arrow keys changes the visualisation to the previous or next step.	This ensures that the menu functions correctly for a Simplex Algorithm visualisation.	
[8][a,b,m]	Open the Quiz page.	The quiz page will open, with a title and buttons for each of the algorithm types chosen by the user when registering.	This ensures that the user can correctly access the Quiz page so that they can test themselves as part of learning.	
[8][c,d,e]	Complete a quiz question for the Bubble Sort algorithm.	An unordered dataset is displayed as text and as options in dropdown menus. There is also a field to enter the number	This ensures that the user can complete a question for the Bubble Sort algorithm type, therefore allowing users	

		of passes. When the user submits an answer the outcome of whether they got it right or wrong is displayed and the database is updated accordingly.	to practice this algorithm.	
[8][f,g]	Complete a quiz question for Prim's Algorithm.	A graph is randomised and displayed. Next to it are text boxes to enter the MST edges and MST weight into. When the user submits their answers, the outcome of whether they got it right or wrong is displayed and the database is updated accordingly.	This ensures that the user can complete a question for the Prim's algorithm type, therefore allowing users to practice this algorithm.	
[8][h,i]	Complete a quiz question for Dijkstra's Algorithm.	A graph is randomised and displayed. Next to it are text boxes to enter the shortest path and the shortest path's weight into. When the user submits their answers, the outcome of whether they got it right or wrong is displayed and the	This ensures that the user can complete a question for the Dijkstra's algorithm type, therefore allowing users to practice this algorithm.	

		database is updated accordingly.		
[8][j,k]	Complete a quiz question for the Simplex Algorithm.	3 constraints and an objective function are randomised and the constraints are displayed on the graph. There are text boxes to enter the optimal x y and P values into. When the user submits their answers, the outcome of whether they got it right or wrong is displayed and the database is updated accordingly.	This ensures that the user can complete a question for the Simplex algorithm type, therefore allowing users to practice this algorithm.	

SOLUTION APPROACH

For the programming of this system, I have decided to predominantly use an Object Oriented Programming approach because it is advantageous to systems with a modular design and makes adding new features easier because classes can be extended. However, the complexity in programming the system is increased, which may make the development of the system longer. Because of the OOP approach, I have split the system into modules, which will each be classes. Below is a diagram to illustrate their structure:



The blue highlighting of the arrows indicates that the BubbleSort, Prim, Dijkstra and Simplex classes all inherit the Menu class and the Prim and Dijkstra classes also inherit the GraphInput class, which is shown with the red highlighting.