A2 Computer Science Project

Contents:

Analysis

Problem Definition

There are many different sorting and searching algorithms used in the computer science industry today. Sorting algorithms provide students with some backbone concepts of computer science such as data structures, big O notation and efficiency. Some of these are taught in GCSE and all the way up to A-level and university classes and may appear to be confusing to some students, especially younger ones.
This solution will present a graphical representation of some common algorithms that are both efficient and inefficient. It will highlight that there is no ‘perfect’ algorithm and many choices are based on the context of the data used. This solution is intended for educational purposes.

Stakeholders

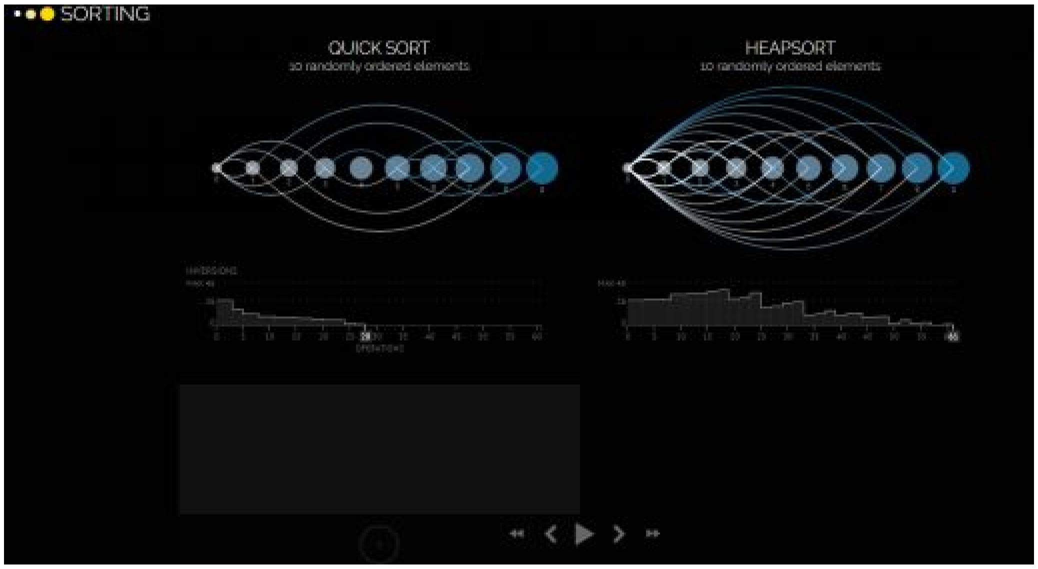
The stakeholders are made up of those with an interest in Computer Science. With those that are just curious about the subject to those that are studying it at GCSE, Vocational and A-Level. Potential users are those that have an interest in computing and/or maths and want to know how sorting and searching algorithms work, as well as how they perform in comparison to each other.
Teachers and tutors may use this also, to teach their students about these algorithms in a simple way. Users will select an algorithm, give it data to populate the array or have it randomly generated and watch how each one performs. Students will use this program for a few minutes at a time, depending on the algorithm used, how complex the array is and the number of algorithms they want to look at.
This could be a useful revision tool as it gives students quick access to a variety of algorithms from a laptop / desktop pc.
The designer will maintain the code, keep it neat and easy to understand and respond to the feedback of stakeholders/users.

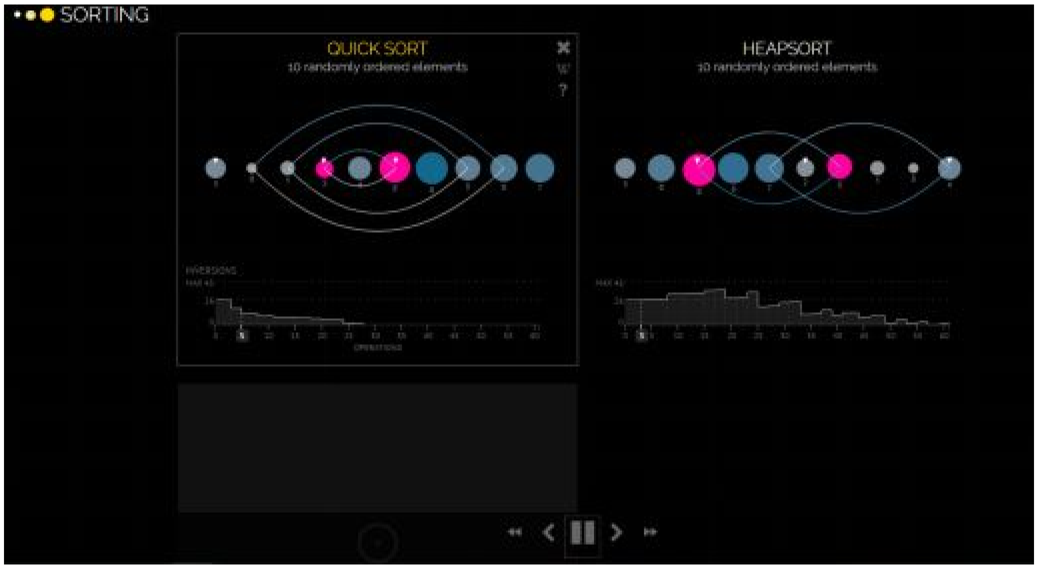
Existing Solutions

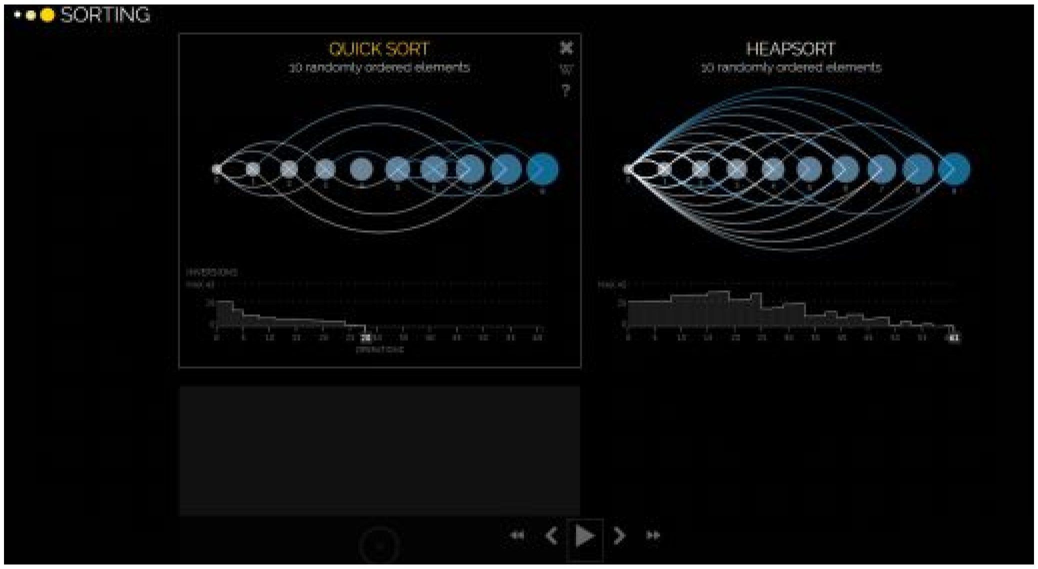
Two current solutions are sorting.at and visualgo.net/bn/sorting. These are both web-based solutions to sorting algorithms. Both of them have a limited amount of input available to the user. Sorting.at can have multiple sorts running at the same time, thanks to this it is easier to see which sort performs better depending on context.
However, it may seem a little complicated at first but it does link to the respective Wikipedia article to explain each sort.
visualgo.net’s representation is much more intuitive and the pseudocode for each sort goes alongside the visuals as it runs. It also gives more freedom to the user with the data being used.

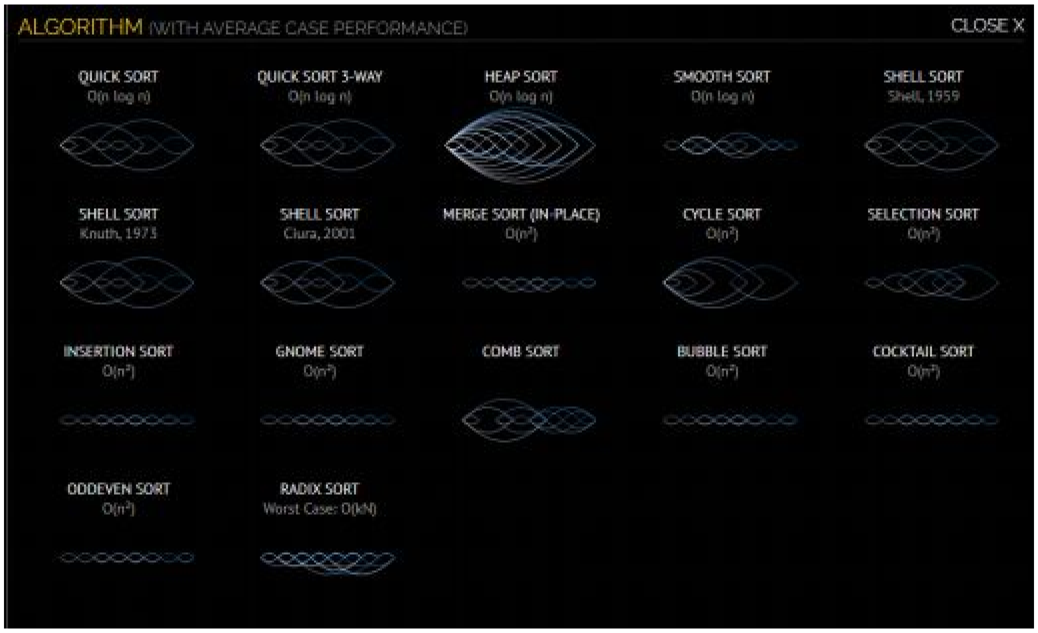
From research into existing solutions, I have concluded that my own solution would benefit largely from having pseudocode alongside each sort to aid the explanation process, this is because users are likely to be in the age range of 14 - 18 (and those new to coding concepts) and these users may find it much easier to understand basic pseudocode rather than code in a specific language alongside it. Pseudocode also allows for users to carry it over into another programming language if they wish to try code something similar for themselves.

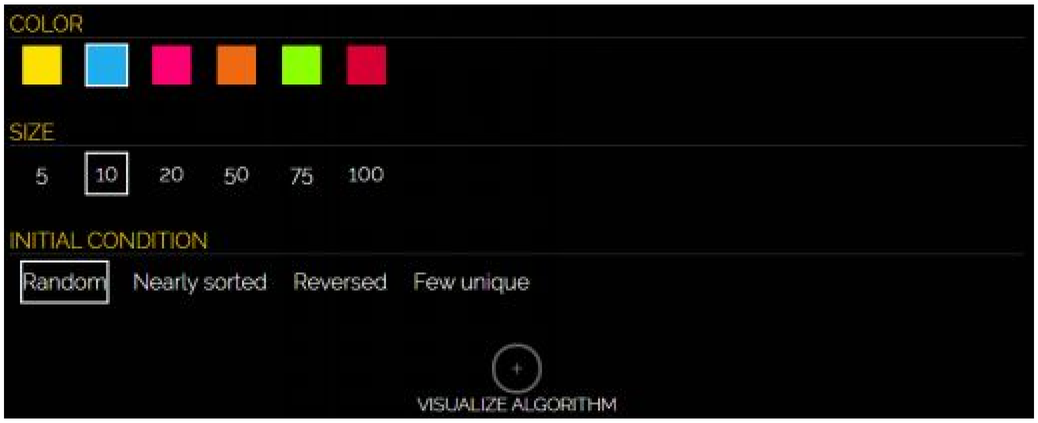
Sorting.at










COLOR

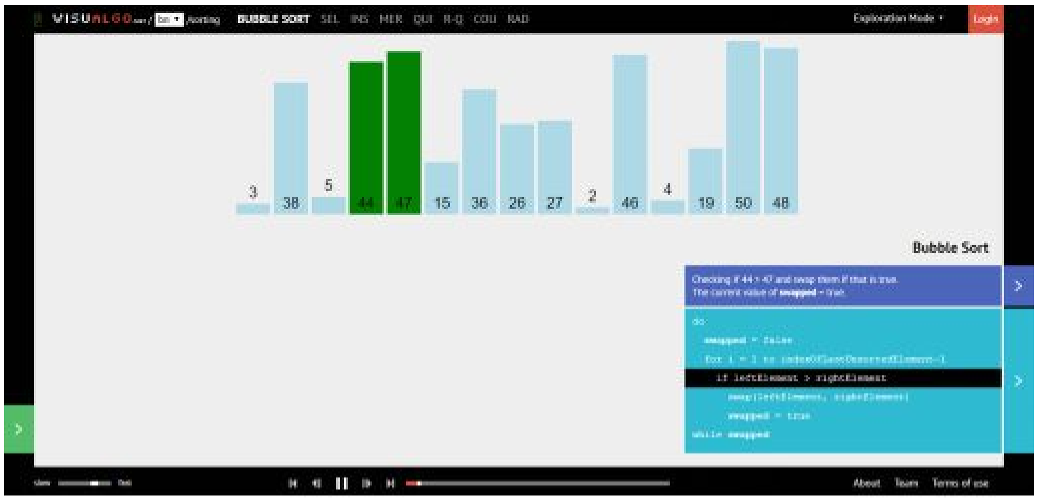
🟨 [✓ 🟦] 🟪 🟧 🟩 🟥

SIZE

5 [✓ 10] 20 50 75 100

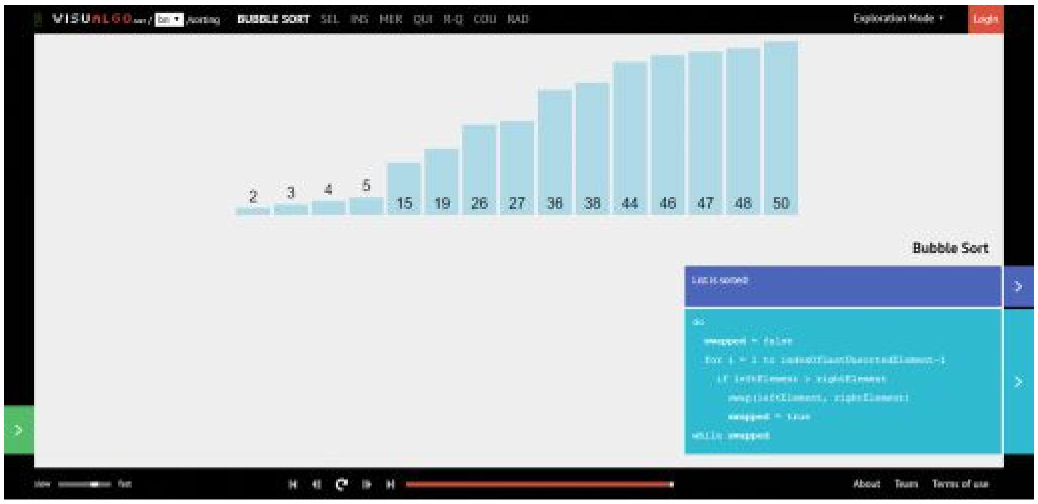
INITIAL CONDITION

[✓ Random] Nearly sorted Reversed Few unique visualgo.net/bn/sorting



Checking if 44 > 47 and swap them if that is true.
The current value of **swapped** = true.

```
do
  swapped = false
  for i = 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap(leftElement, rightElement)
      swapped = true
while swapped
```



List is sorted

```
do
  swapped = false
  for i = 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap(leftElement, rightElement)
      swapped = true
while swapped
```

Investigation

A questionnaire was posted online and shown to clients in order to gather more data on what potential users wanted from the program. The questions and results are as follows:

- Should both sorting and searching algorithms be in the same program?
  - Yes, put them both together (11)
  - No, make a separate program for each (8)

- how much data would you like to be usable in the program?
  - 10 integers (1)

- 20 integers (4)
- 30 integers (1)
- 40 integers (0)
- 50 integers (4)
- 100 integers (7)
- Other (please specify) (1)
  - As much as the user wants

- Please list any sorting/searching algorithms you would like to see/know of. N/A if you know none
  - Pancake sort (1)
  - Stooge sort (1)
  - Comb sort (1)
  - Bubble sort (9)
  - Merge sort (4)
  - Insertion sort (2)
  - Quick sort (1)
  - Bogosort (2)
  - Stalin sort (1)
  - Binary search (4)
  - Linear search (4)
  - Fibonacci search (1)
  - As many as possible (1)
  - N/A (5)

- If you own a desktop/laptop computer, what are it's specifications? (RAM and clock speed + cores)
  - <2GB RAM and <1.8Ghz quad-core (6)
  - >= 2GM RAM and >1.8Ghz quad-core (10)
  - N/A (1)
  - Other (please specify) (2)
    - 4GB RAM 1.7Ghz dual-core (1)
    - 8GB RAM 1.7Ghz dual-core(1)

- With a visual representation of sorting and searching algorithms using bars, what colour should the bars and background be?
  - Dark grey background and light blue/purple bars (1)
  - Black background and white bars with different colour to highlight current element (1)
  - Black background and white bars (2)
  - Black background and red bars (1)
  - Grey background and red bars (1)
  - Grey background and white bars (1)
  - Anything with good contrast (2)

    - Green bars and white background (1)
    - Blue bars and white background (2)
    - Customisable by user (1)
    - Orange bars and black background(1)
    - N/A (4)

- Should users be able to input their own data or should it be randomised?
    - Yes, users should be able to input their own data (14)
    - No, data should be randomised (4)
    - N/A (1)

- Should pseudocode (simplified version of code) be shown alongside each sort/search?
    - Yes (13)
    - No (5)
    - N/A (1)

- What font and font colour should be used?
    - Roboto, white (2)
    - Ariel, black (2)
    - Comic sans, red (1)
    - Comic sans, black (2)
    - Black colour (1)
    - Carrier, black (1)
    - Jetbrains mono, black (1)
    - Consolas, black (1)
    - User decides (1)
    - Times new roman, black (1)
    - N/A (6)

- How large should integers used in the program be allowed to be?
    - <10 (1)
    - <20
    - <30
    - <40 (1)
    - <50 (3)
    - <100 (11)
    - Other (please specify) (3)
        - >100 (1)
        - <65535 (1)
        - As much as the user wants (1)

- Should results be stored in a text file to be referred to afterwards?
    - Yes (14)

■ No (4)
■ N/A (1)

Solution Requirements and Success Criteria

The solution requirements and success criteria consist of data collected from the research on existing solutions, the research survey and my own thoughts.

1. To show a list of all searches and sorts available to the user for ease of use.
   1.1. Font size must be large enough so that all users can read it.
   1.2. Font must be clear so that all users can read. For example, no joke fonts such as Wingdings.
   1.3. Font, background and bars must be in contrasting colours. For example, a black background with white bars (grey when selected) and blue text

2. To allow users to enter their own data if they want to know how a certain data set is processed with each sort, or let it be randomised.
   2.1. data must be integers - strings cannot be sorted, floats cannot due to restraints of library being used.
   2.2. data must be positive - negative integers would not work, due to nature of how displaying bars will work.
   2.3. integers must not be higher than 100 - due to visual restraints.
   2.4. array must contain more than one item - otherwise it will always be sorted
   2.5. array must contain no more than 100 items - due to time restrictions, some algorithms can be slow
   2.6. Duplicate data is allowed

3. To provide a visual representation of the data, this makes it easier for those who don't understand how sorts/searches work.
   3.1. bars must be clear to see - can easily see where bars are and their relative size.
   3.2. animation must appear fluid - can't be too slow, may not be possible depending on technical specifications of the user's device.
   3.3. bars will be white before the sort begins, red when not sorted, green when fully sorted and grey when highlighted.

4. To provide pseudocode for each sort to allow users to replicate each sort/search in their preferred programming language if they wish to.
   4.1. text of pseudocode must appear alongside the visual representation - to know which sort goes with each code
   4.2. text must follow some of the same rules as the sort/search list: large and clear. - so users can read it
   4.3. colour will be orange - to stand out from other text.

5. To store results in a text file for sorts to allow users to look back on previous data.
   5.1. store steps of each sort in the file - see step by step afterwards what happened
   5.2. file will be .txt - this keeps file sizes to a minimum, and all devices can read .txt files.

Computational Methods

The program will be created using a divide and conquer approach. It will be split up into several procedures that will each handle an aspect of the solution. This will allow for decomposition and code reuse as one procedure can be used at several points in the program. For example, creating an array will happen a lot during the program depending on how many algorithms the user wants to try. Because of this a procedure will be created to make arrays that can then be called at various stages in the program's execution. Additionally, merge sort and quick sort utilise the divide and conquer approach themselves.

A core concept of the sorting and searching algorithm world is Big O notation, the algorithms used in the program are tractable as none go above O(n^2). As such the use of a heuristic approach is unnecessary. The algorithms used in this program will take a reasonable time to complete e.g. no O(n!) time complexity.

Another instrumental concept in this solution is visualisation, by the end of program development, users will be able to see in a simple - abstracted (reduced from complex inner workings of algorithm to simple bars sized relatively to each other) - way what happens in the various algorithms. This allows (some) complex procedures to be reduced to a simple array of bars, providing a straightforward view for users. This will be especially helpful for younger users who may be interested in the topic but don't yet have the knowledge to understand the inner workings of each algorithm.

System Requirements

- 1.8 GHz or faster processor. Quad-core or better recommended
- 2 GB of RAM; 8 GB of RAM recommended (2.5 GB minimum if running on a virtual machine)

-Taken from https://docs.microsoft.com/en-us/visualstudio/releases/2019/system-requirements
→ Actual requirements will be substantially less than this

- A computer using the Windows OS because the file will be .exe so not usable on Linux, Mac or mobile phones

Design

Objective

The interface should look clean and be user friendly. The following are the ways in which I intend to achieve the objective, these were obtained partly through the research survey - though are subject to change until the solution is complete.

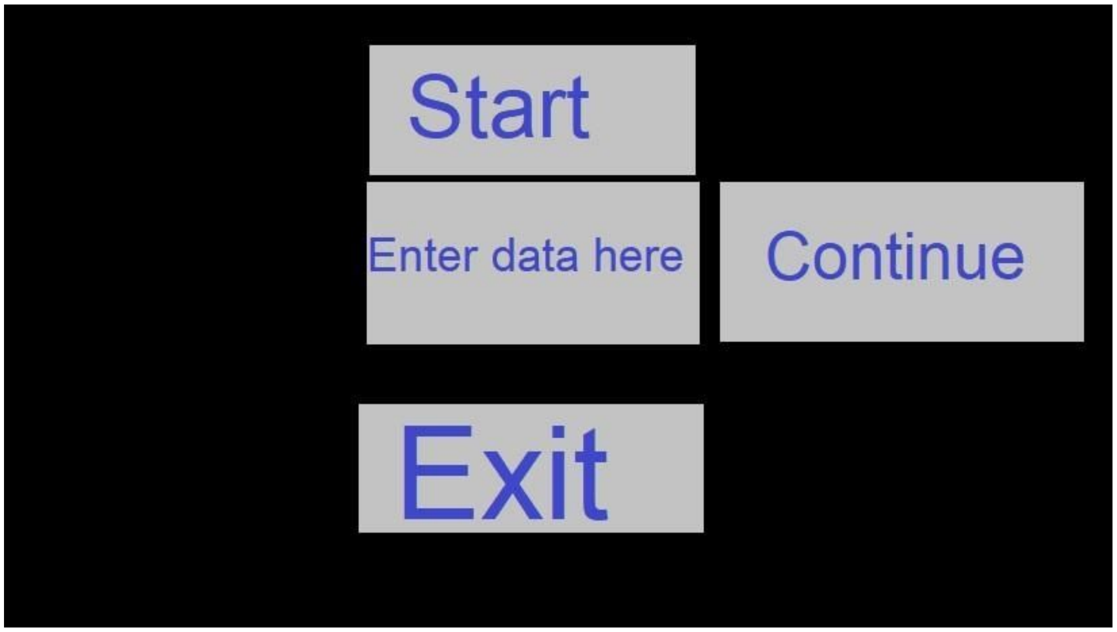
1. Black background so that the screen is not too bright so comfortable for night use
2. White bars to act as a contrast to the background
3. Red bars for when the list is not sorted
4. Green bars for when the whole list is sorted
5. Grey bars for when bar(s) are being selected
6. Grey boxes around menu buttons
7. Yellow (previously blue) text in Comic Sans - Roboto is unavailable in Visual Studio - for menu and more contrast for easy viewing
8. Orange text for pseudocode to separate it from the rest

Interface Design

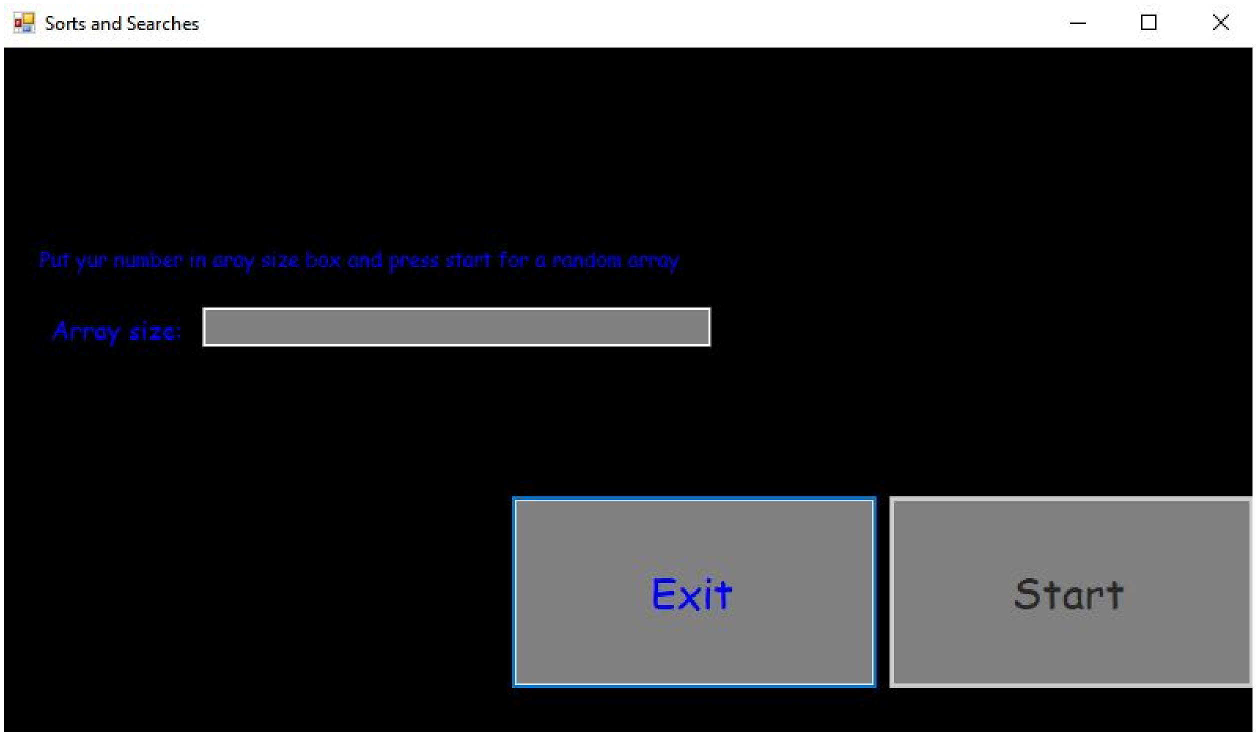
These are some basic designs for the main menu and sort and search screen. These were designed based on the questionnaire sent to stakeholders and other potential users, they were designed with simplicity in mind.

Main Menu

Original design



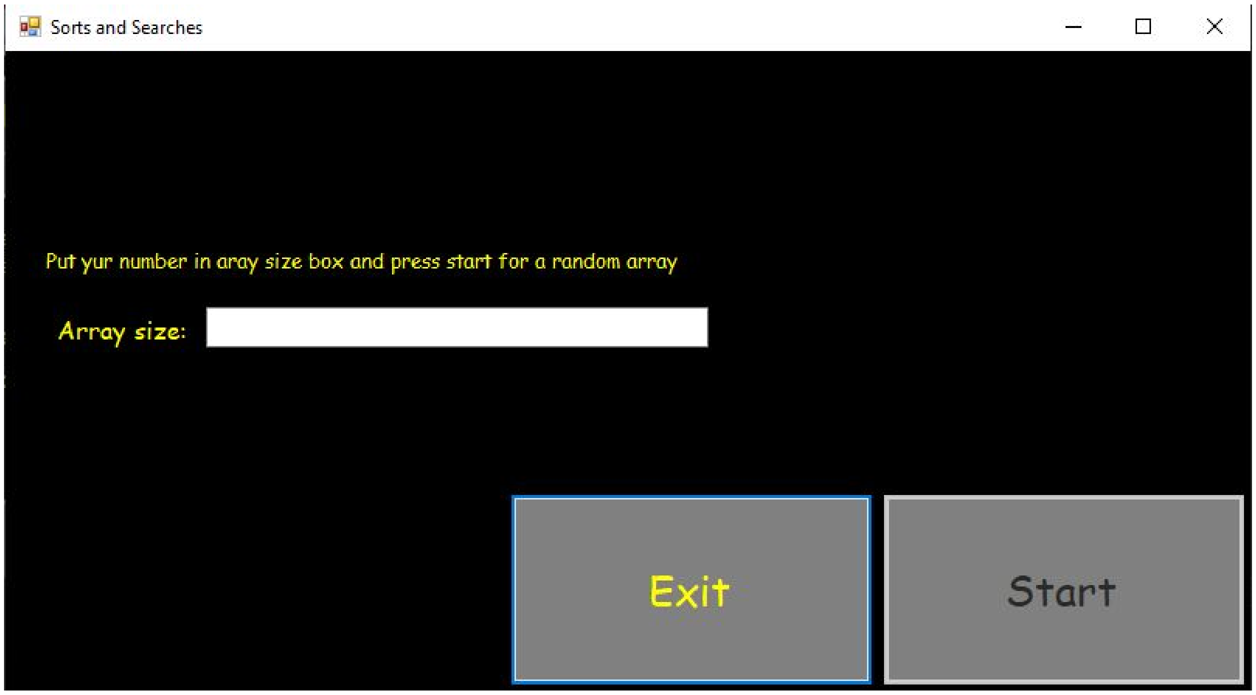
Version 1.1



Put yur number in aray size box and press start for a random array

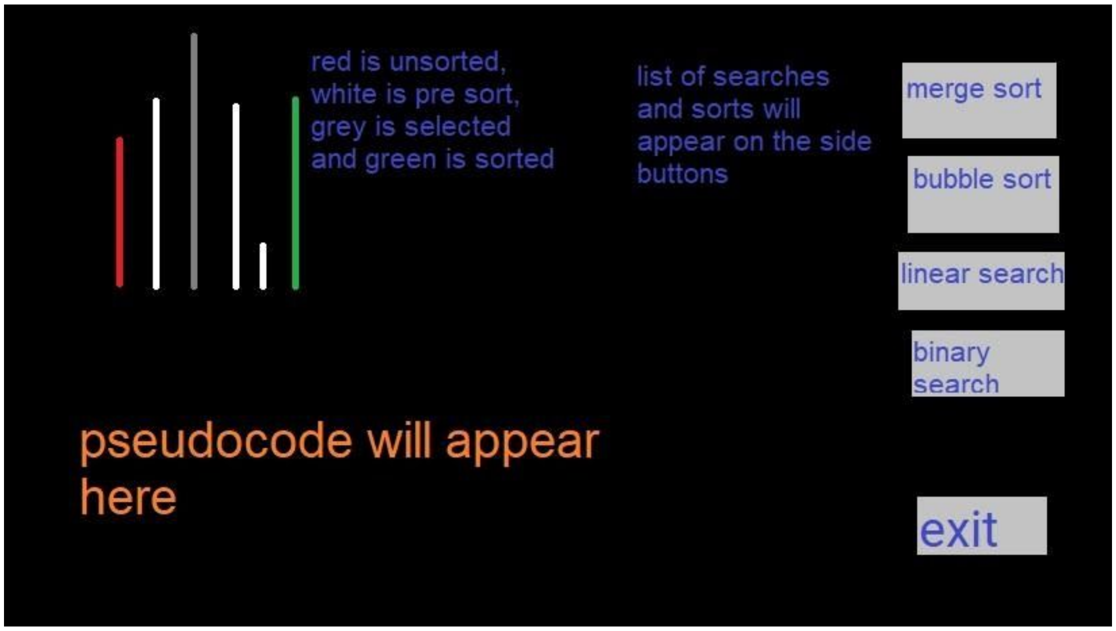
Array size:

Version 1.2

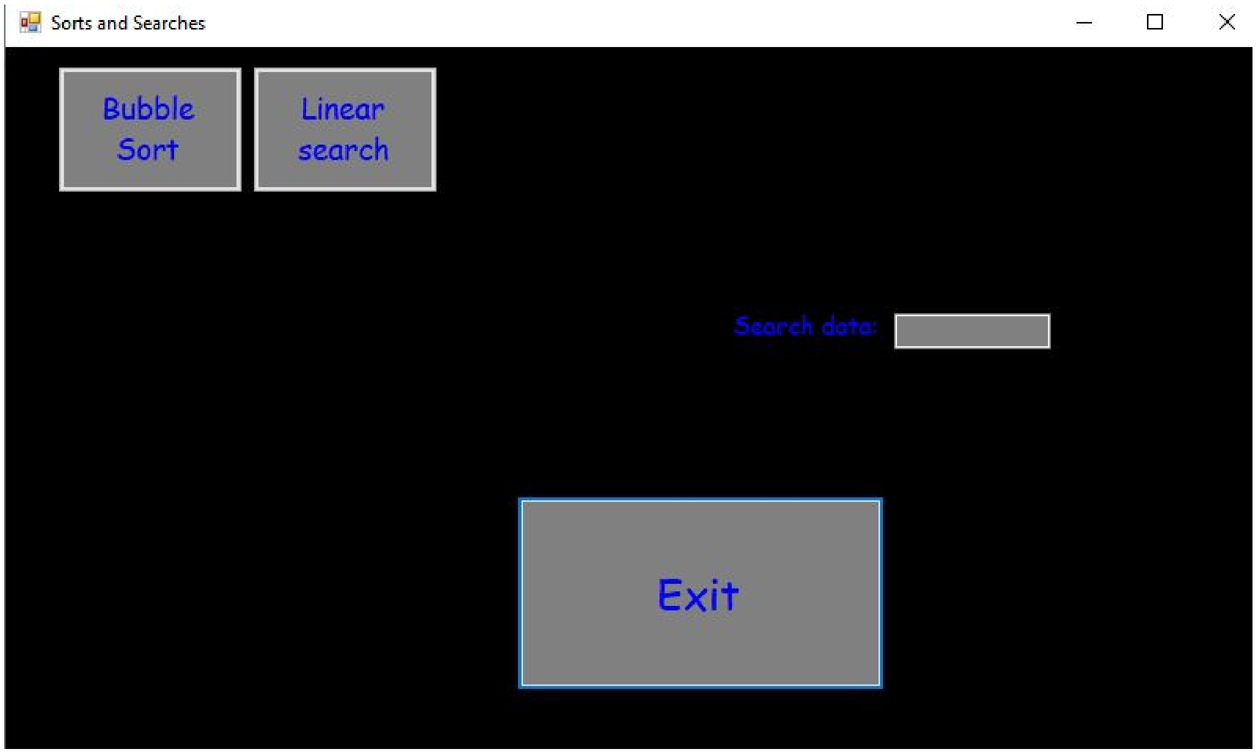

Sorts and Searches
Put yur number in aray size box and press start for a random array
Array size:
Exit
Start

Sorts and searches

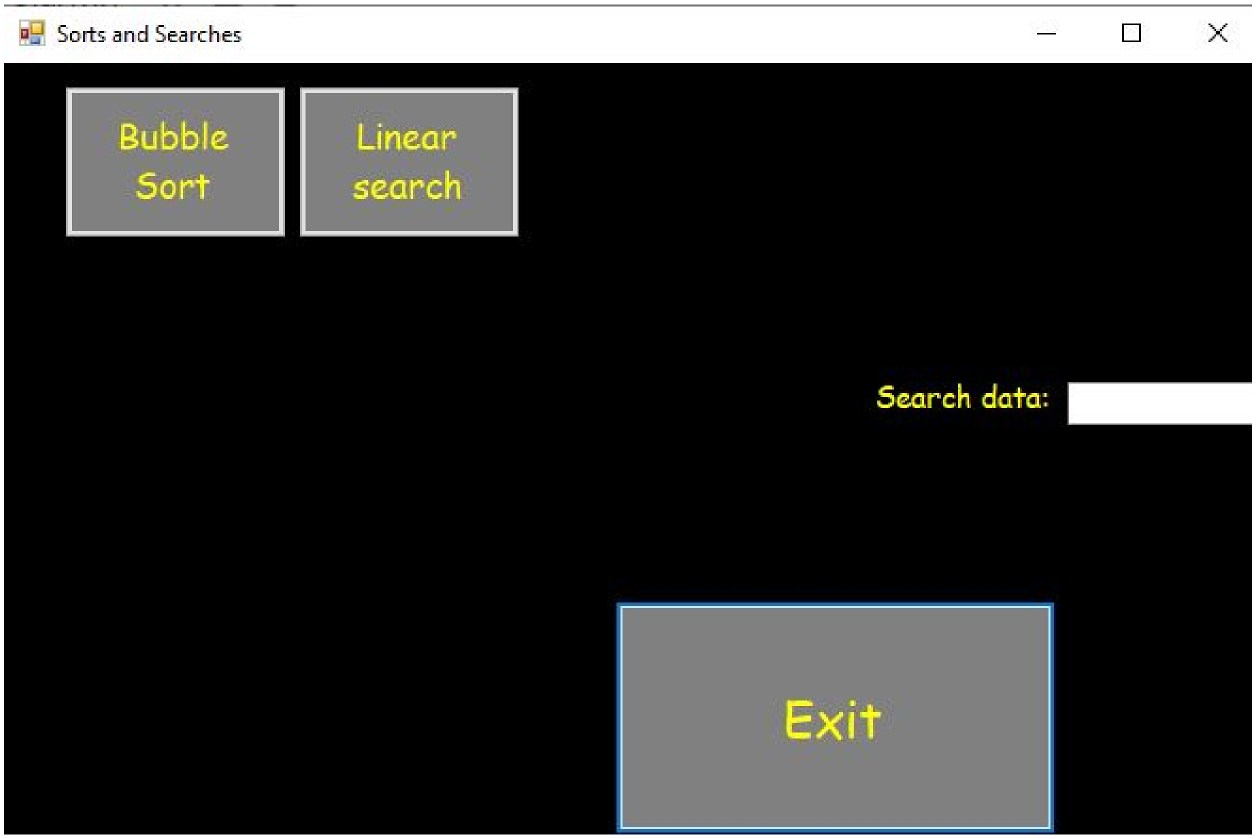
Original design


red is unsorted, white is pre sort, grey is selected and green is sorted
list of searches and sorts will appear on the side buttons
merge sort
bubble sort
linear search
binary search
pseudocode will appear here
exit

Version 1.1


Sorts and Searches
Bubble Sort
Linear search
Search data:
Exit

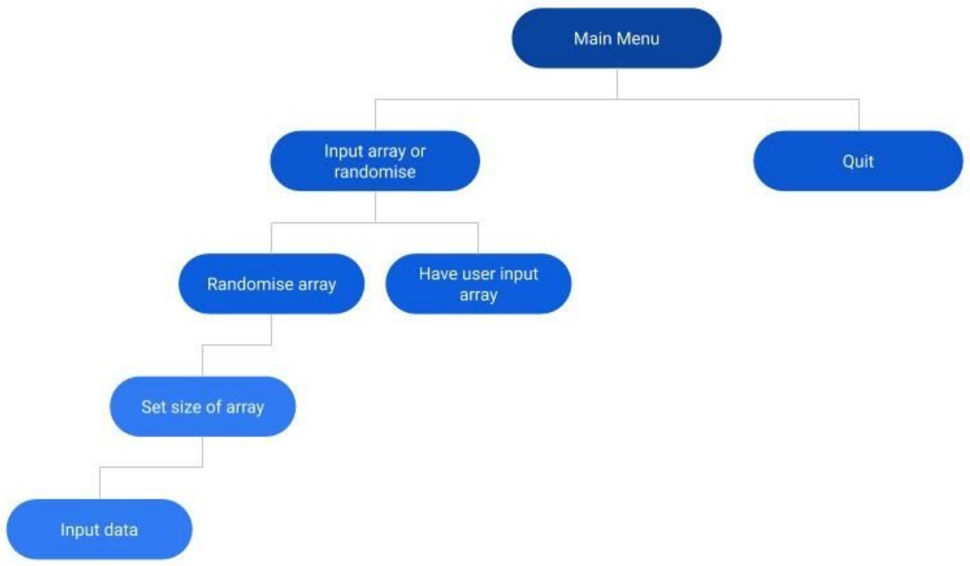
Version 1.2



Decomposition and Flowcharts

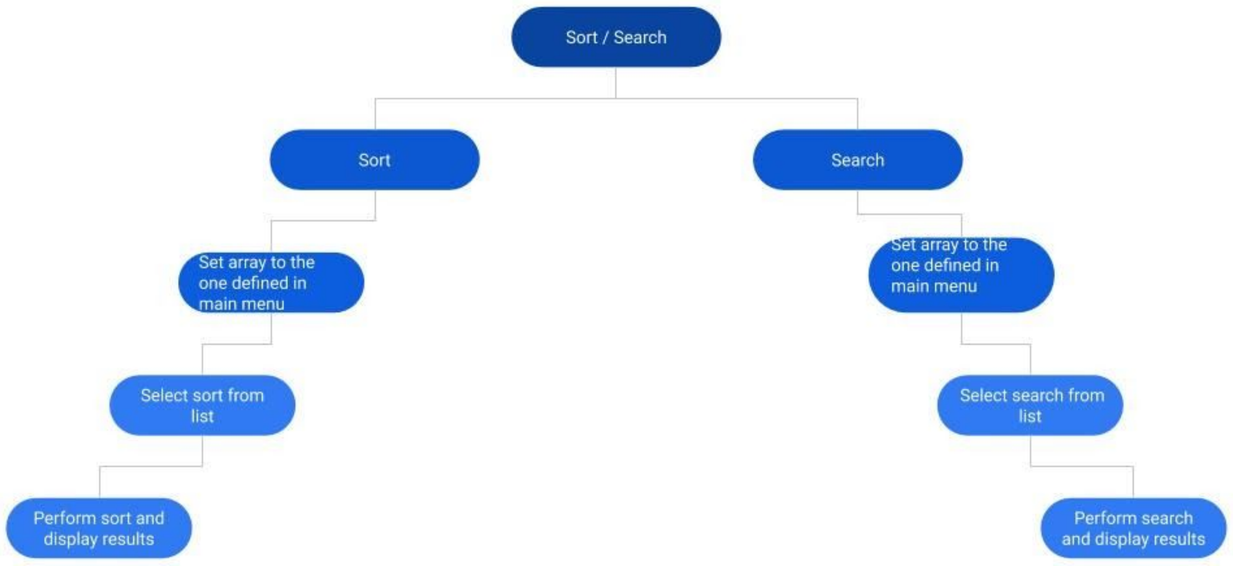
Below are the decompositions and flowcharts of the program. These outline the main features of it in a simple format.

Main Menu Decomposition

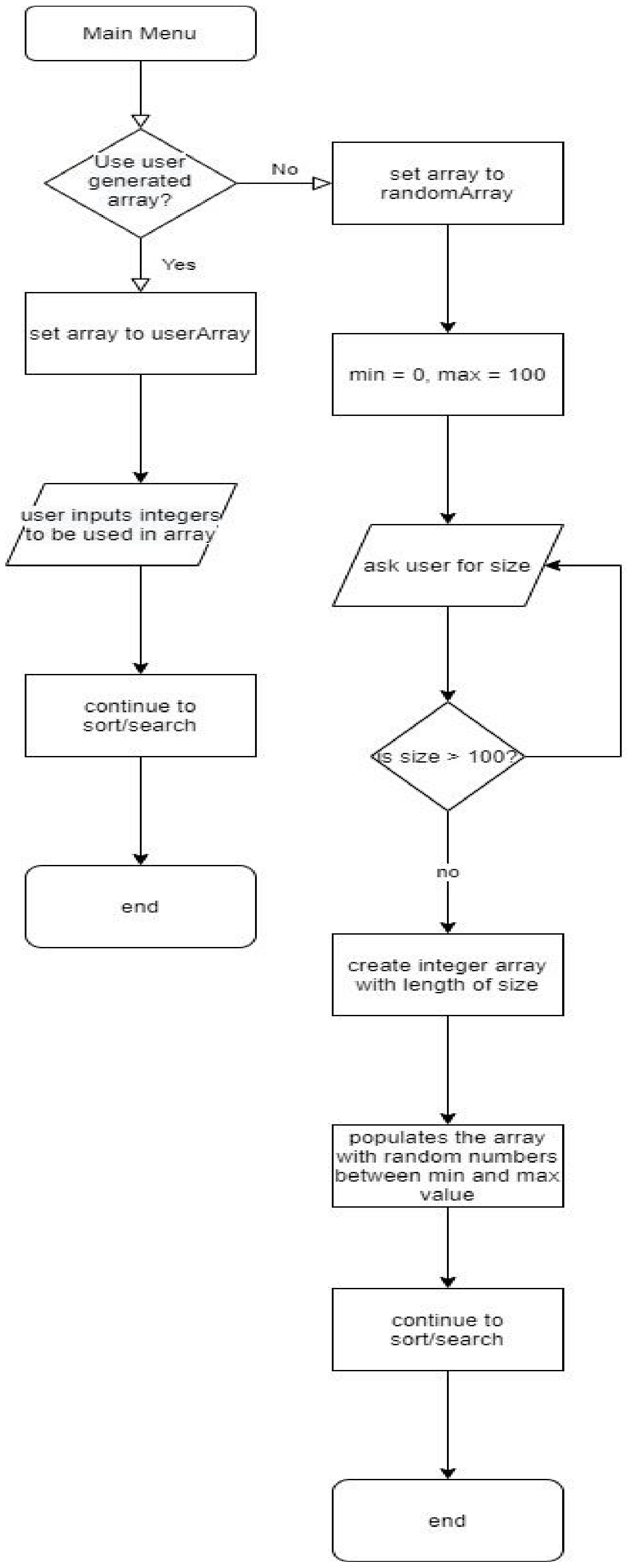


Sort/Search decomposition

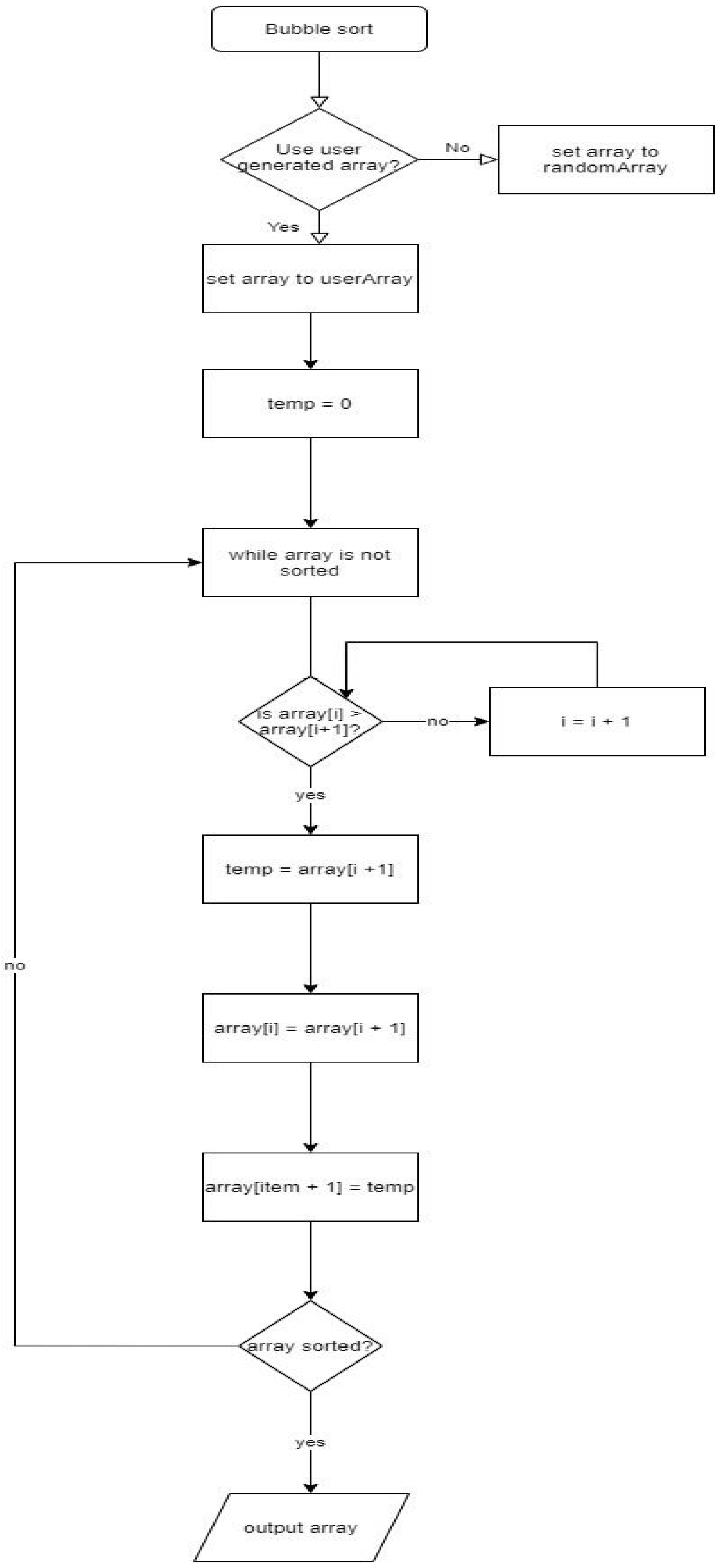
Details of one of each (will be updated with others) sort and search will be displayed through pseudocode.


Sort / Search
Sort
Search
Set array to the one defined in main menu
Set array to the one defined in main menu
Select sort from list
Select search from list
Perform sort and display results
Perform search and display results

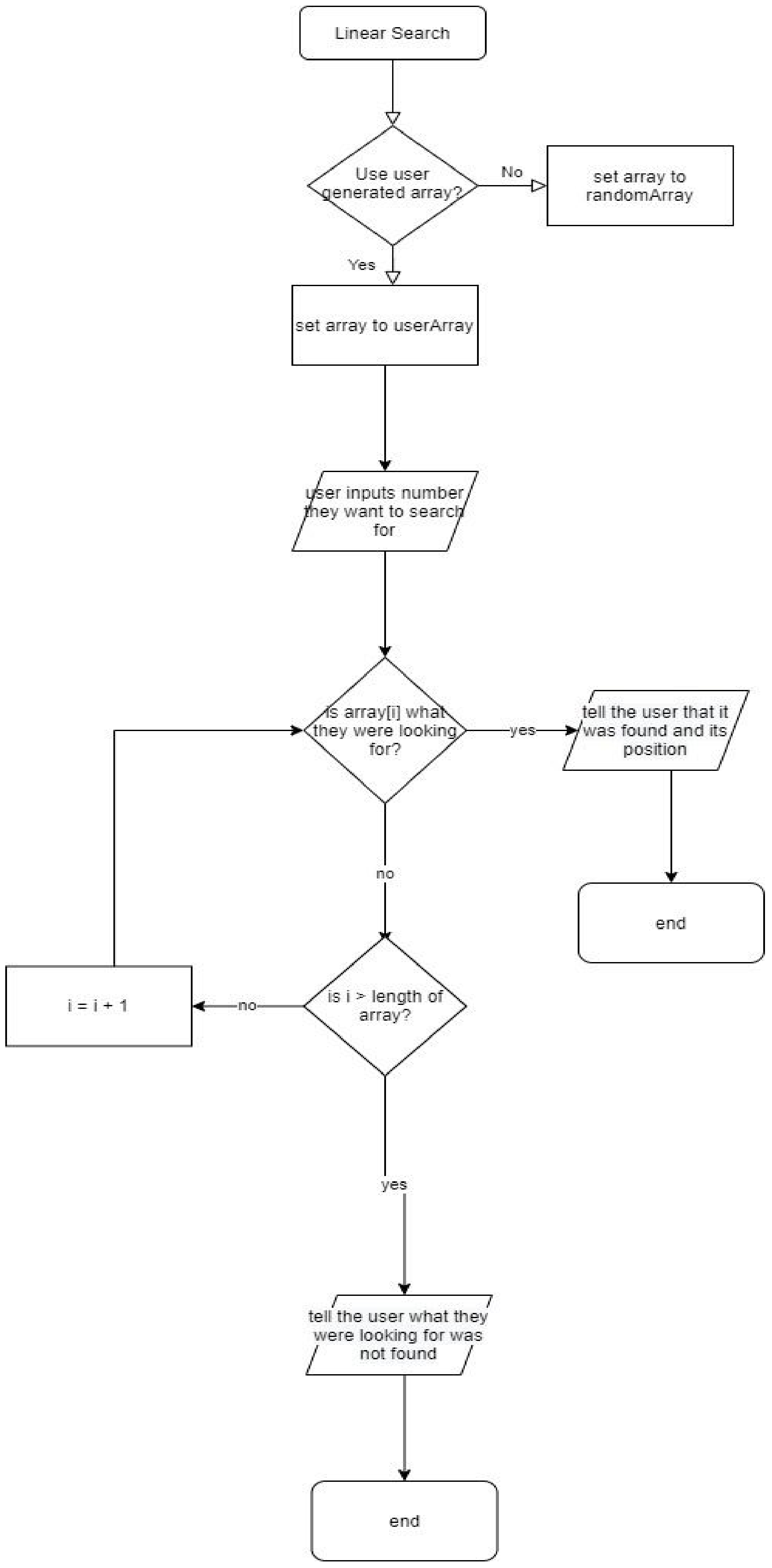
Main Menu Flowchart


Main Menu
Use user generated array?
No
Yes
set array to randomArray
set array to userArray
min = 0, max = 100
user inputs integers to be used in array
ask user for size
continue to sort/search
is size > 100?
no
end
create integer array with length of size
populates the array with random numbers between min and max value
continue to sort/search
end

Bubble Sort Flowchart


Bubble sort
Use user generated array?
No
set array to randomArray
Yes
set array to userArray
temp = 0
while array is not sorted
is array[i] > array[i+1]?
no
i = i + 1
yes
temp = array[i +1]
array[i] = array[i + 1]
array[item + 1] = temp
array sorted?
no
yes
output array

Linear Search Flowchart


Linear Search
Use user generated array?
No
set array to randomArray
Yes
set array to userArray
user inputs number they want to search for
is array[i] what they were looking for?
yes
tell the user that it was found and its position
end
no
is i > length of array?
no
i = i + 1
yes
tell the user what they were looking for was not found
end

Data Dictionary

Pseudocode Data Dictionary: - (actual code Data Dictionary will be below it once development of the coded solution has begun, the pseudocode one will act as a framework for the real one)

Identifier	**Description**	**Data type**	**Validation / specification**	**Scope**
strRandomOrUser	Will be used to store whether the user wants to choose their own array or have it randomly generated by the program	string	"random" or "user" - case will not matter, program will convert it to lowercase	MainMenu
intLength	Stores the length of the randomly generated array	integer	Less than 100	MainMenu
constMin	Stores the lowest number that the random number generator will pick	constant	0	MainMenu
constMax	Stores the highest number that the random number generator will pick	constant	100 - due to visual/memory limits	MainMenu
randomArray	An array of the random numbers	array	Length is decided by user Integers only	MainMenu, BubbleSort, LinearSearch

strData	Stores the integers picked by the user	string	should only contain integers within the string	MainMenu
userArray	Stores the integers picked by the user in an array	array	Integers only	MainMenu, BubbleSort, LinearSearch
strSplit	Splits the string of inputs into individual integers using the .split() function	string	It should turn the string into individual integers	MainMenu
strContinue	Stores if the user wants to continue with the program or exit	string	"continue" or "exit" - case will not matter, program will convert it to lowercase	MainMenu
intTemp	A temporary variable	integer	0 to begin with. Will always be an integer	BubbleSort
strArrayChoic e	Stores if the user wants to use random array or user array	string	Y/N case doesn't matter	BubbleSort, LinearSearch
array	Stores all the integers to be sorted	array	integers only	BubbleSort, LinearSearch
intFind	Stores the number that the user wants to search for	integer	between 0 and 100	LinearSearch

Sample Pseudocode

This section contains some sample pseudocode from each area of the program. These have been written as if to be used in a ‘console application’. However, the solution will be a ‘windows forms application’, as such the pseudocode will not strictly follow how the C# code will look, due to the GUI nature of the program rather than a CLI application.

Main Menu

```
//Main menu

Start MainMenu

strRandomOrUser = input (“Enter your own array data or use a randomised array. Please enter random or user - case is not important”) //tells the user they have the choice of choosing their own array and using a random one and gets them to choose


If randomOrUser == “random” do //if the user chooses a random array
    intLength = input (“How long do you want the array to be? Max = 100 due to visual restraints”) //asks the user how many integers they want in their array

    constMin =0 //sets the lowest possible number to 0
    constMax = 100 //sets the highest possible number to 100

    randomArray[] = int[length] //creates an integer array with length specified by the user

    For i in length do //creates a loop with size of array
        randomArray[i] = randomInt (min, max) //sets the first number to a random integer between 1 and 100. This will repeat itself until the array is full of randomly generated numbers.
else //if the user chooses to enter their own integers
    strData = input (“Enter your integers with a space in between each one. Please enter a maximum of 100 that are up under or equal to 100 in size. If you don’t do this the program will not work.”) //gets the user to input their integers

    userArray = int[] //creates an integer array
    strSplit = data.split() //splits up the input into individual integers form the input string
    For i in split do //for each integer in the splitted list
        userArray[j] = i // sets the first number to the first number from the splitted list. This will repeat until all elements of the split are in the array
```

```
strContinue = input("Continue to sort or search or exit? Please enter continue or exit - case does not matter") //asks the user if they want to continue to the sort/search or exit the program
if strContinue == "continue" do
    continue()
else
    exit()
```

Bubble Sort

```
//Bubble sort

Start BubbleSort

IntTemp = 0 //creates a variable with an integer value of 0

strArrayChoice =  input ("Use random array? Y/N") //asks the user if they want to use the random array
if arrayChoice == "Y" then //if the user picks yes
    array = randomArray //sets the array used in sort to be a random one
else //if they didn't pick random
    array = userArray //uses a user made array instead

for item in array do //creates a loop with size of the array
    print(array[item]) //displays each element in the list

for item in array -1 do //creates a loop with size of the array - 1
    for item in array -1 do
        if array[item] > array[item+1] //if the first number is bigger than the second
            temp = array[item+1] //gives the temporary variable the value of the first number
            array[item] = array[item+1] //sets the first number to the second number
            array[item+1] = temp //sets the second number to the temporary variable - the larger number

//this swaps the numbers e.g. 7,2 will become 2,7

//this will repeat for each number in the list continuously until all numbers are sorted in ascending order

For item in array do
    print(array[item]) //displays all the numbers in the list. At this point they should be in order
```

```
end
```

Linear Search

```
//Linear search

Start LinearSearch

strArrayChoice =  input ("Use random array? Y/N") //asks the user if they want to use the random array
if strArrayChoice == "Y" then //if the user picks yes
    array = randomArray //sets the array used in sort to be a random one
else //if they didn't pick random
    array = userArray //uses a user made array instead

intFind = input ("Search for which number: ")  //asks the user to input a number to search for

For item in array do //creates a loop with size of the array
    If array[item] == find //if the current number is the number that is being searched
        print (find+" found at position"+item.position) //displays the number and its position
    else //if the number was not found
        Print (find+" not found in array") tells the user that it was not found in the array

end
```

Development of the coded solution

Plan for each iteration

Iteration 1

Iteration 1 should include a random array feature and at least bubble sort and linear search for some basic functionality. If possible, it should also have the ability to put in your own array. Some levels of validation should be present. User interface will be basic at this point and may not be the most usable.

Iteration 2

Iteration 2 should include bubble sort, insertion sort, merge sort, quick sort, linear search and binary search. It should have the option to have either a random array or user generated array implemented adequately. The interface should be easy to use and more validation should be present

Iteration 3

As the final iteration it should involve all sorts and searches mentioned in the analysis section, if that is not possible then all ones mentioned in the AQA A Level CS specification (those in Iteration 2). The interface should be easy to use and understand for anyone, and sufficient help should be provided by the program if users need it, along with pseudocode accompanying the sort/search process. Add the option to save steps/results to a text file.

All of this should have been achieved (by iteration 3) to meet the needs of users:

1. *To show a list of all searches and sorts available to the user for ease of use.*
    1.1. *Font size must be large enough so that all users can read it.*
    1.2. *Font must be clear so that all users can read. For example no joke fonts such as Wingdings.*
    1.3. *Font, background and bars must be in contrasting colours. For example a black background with white bars (grey when selected) and blue text*

2. *To allow users to enter their own data if they want to know how a certain data set is processed with each sort, or let it be randomised.*
    2.1. *data must be integers - strings cannot be sorted, floats cannot due to restraints of library being used.*
    2.2. *data must be positive - negative integers would not work, due to nature of how displaying bars will work.*
    2.3. *integers must not be higher than 100 - due to visual restraints.*
    2.4. *array must contain more than one item - otherwise it will always be sorted*
    2.5. *array must contain no more than 100 items - due to time restrictions, some algorithms can be slow*
    2.6. *Duplicate data is allowed*

3. *To provide a visual representation of the data, this makes it easier for those who don't understand how sorts/searches work.*
    3.1. *bars must be clear to see - can easily see where bars are and their relative size.*
    3.2. *animation must appear fluid - can't be too slow, may not be possible depending on technical specifications of the user's device.*
    3.3. *bars will be white before the sort begins, red when not sorted, green when fully sorted and grey when highlighted.*

4. *To provide pseudocode for each sort to allow users to replicate each sort/search in their preferred programming language if they wish to.*
    4.1. *text of pseudocode must appear alongside the visual representation - to know which sort goes with each code*
    4.2. *text must follow some of the same rules as the sort/search list: large and clear. - so users can read it*
    4.3. *colour will be orange - to stand out from other text.*

5. *To store results in a text file for sorts to allow users to look back on previous data.*
    5.1. *store steps of each sort in the file - see step by step afterwards what happened*
    5.2. *file will be .txt - this keeps file sizes to a minimum, and all devices can read .txt files.*

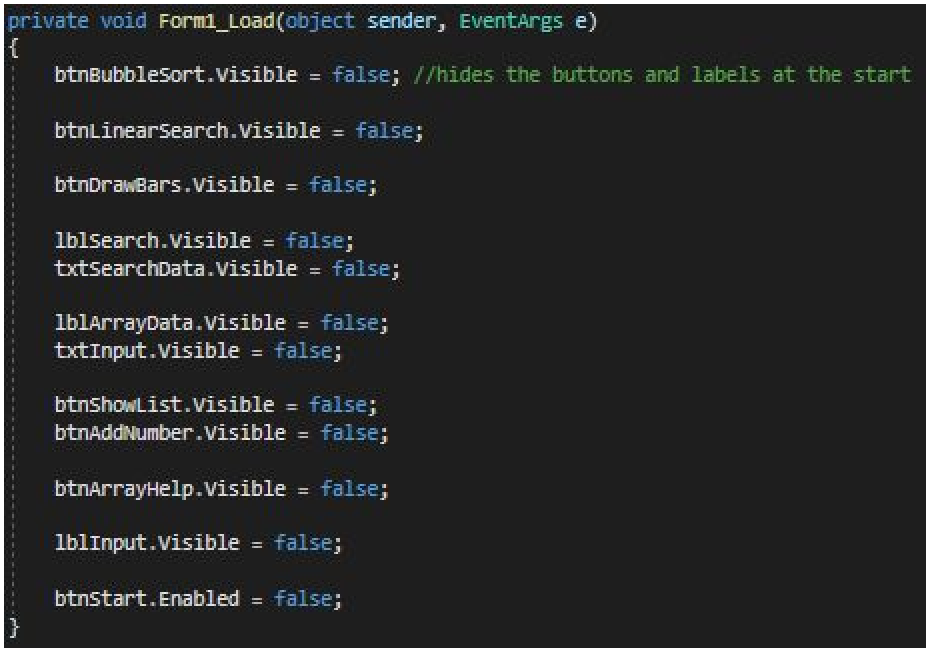
Main Menu

The purpose of the main menu is to allow the user to input their own array if they choose to or to have a random one generated for them. They must always supply a length of the array (as of iteration 1.1)

Implementation in iteration 1

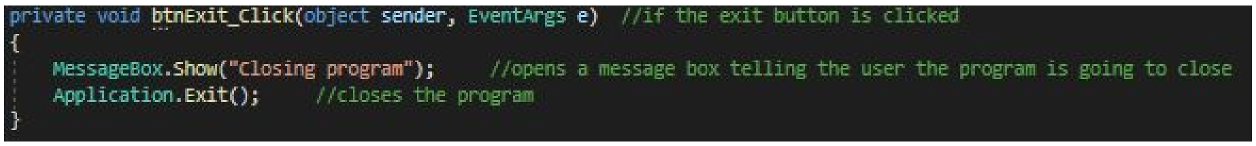
Includes code not currently implemented or functioning commented out

Form Initialisation



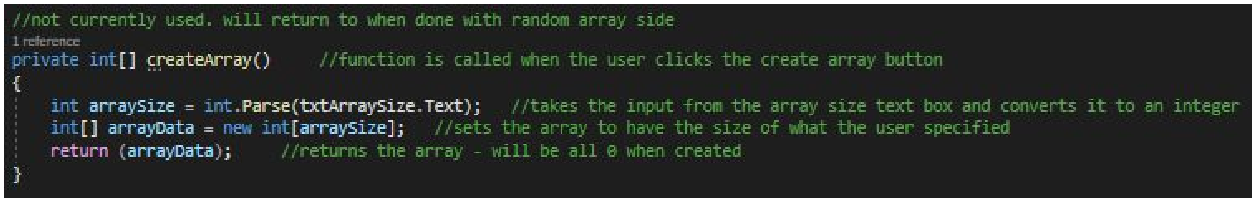
```
private void Form1_Load(object sender, EventArgs e)
{
    btnBubbleSort.Visible = false; //hides the buttons and labels at the start

    btnLinearSearch.Visible = false;

    btnDrawBars.Visible = false;

    lblSearch.Visible = false;
    txtSearchData.Visible = false;

    lblArrayData.Visible = false;
    txtInput.Visible = false;

    btnShowList.Visible = false;
    btnAddNumber.Visible = false;

    btnArrayHelp.Visible = false;

    lblInput.Visible = false;

    btnStart.Enabled = false;
}
```

Exit



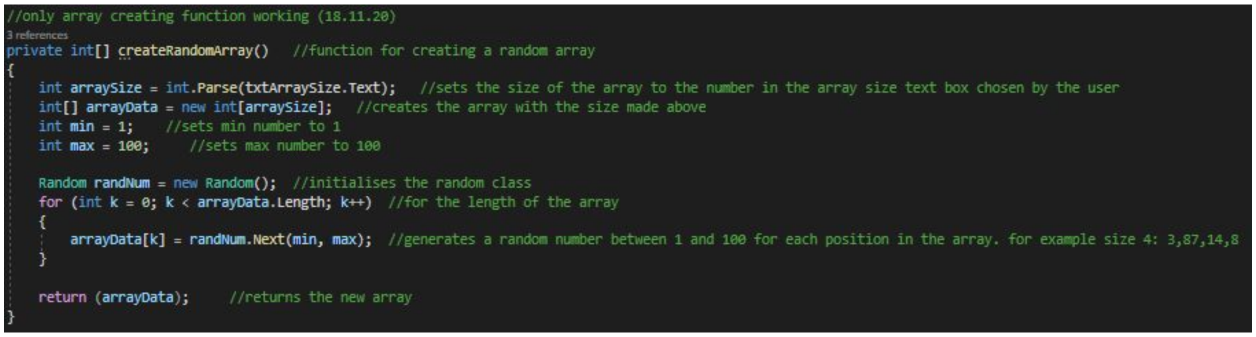
```
private void btnExit_Click(object sender, EventArgs e)  //if the exit button is clicked
{
    MessageBox.Show("Closing program");     //opens a message box telling the user the program is going to close
    Application.Exit();     //closes the program
}
```

Creating user array



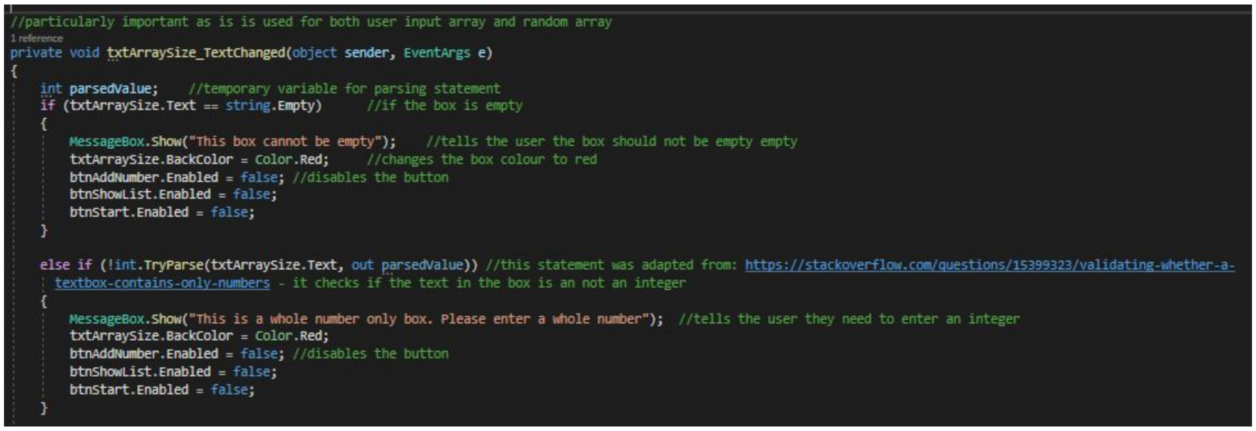
```
//not currently used. will return to when done with random array side
1 reference
private int[] createArray()     //function is called when the user clicks the create array button
{
    int arraySize = int.Parse(txtArraySize.Text);   //takes the input from the array size text box and converts it to an integer
    int[] arrayData = new int[arraySize];   //sets the array to have the size of what the user specified
    return (arrayData);     //returns the array - will be all 0 when created
}
```

Creating random array

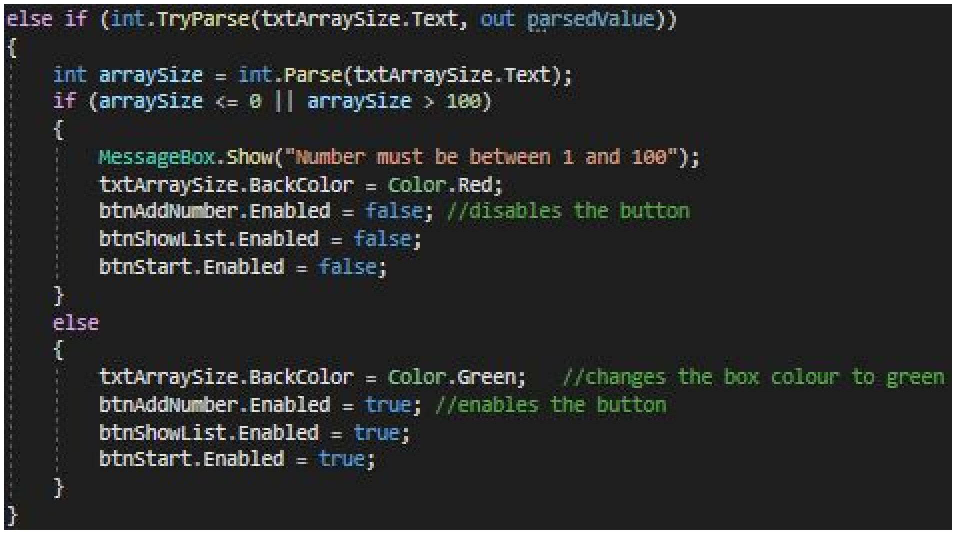


```
//only array creating function working (18.11.20)
3 references
private int[] createRandomArray()   //function for creating a random array
{
    int arraySize = int.Parse(txtArraySize.Text);   //sets the size of the array to the number in the array size text box chosen by the user
    int[] arrayData = new int[arraySize];   //creates the array with the size made above
    int min = 1;    //sets min number to 1
    int max = 100;     //sets max number to 100

    Random randNum = new Random();  //initialises the random class
    for (int k = 0; k < arrayData.Length; k++)  //for the length of the array
    {
        arrayData[k] = randNum.Next(min, max);  //generates a random number between 1 and 100 for each position in the array. for example size 4: 3,87,14,8
    }

    return (arrayData);     //returns the new array
}
```

Array size text box validation

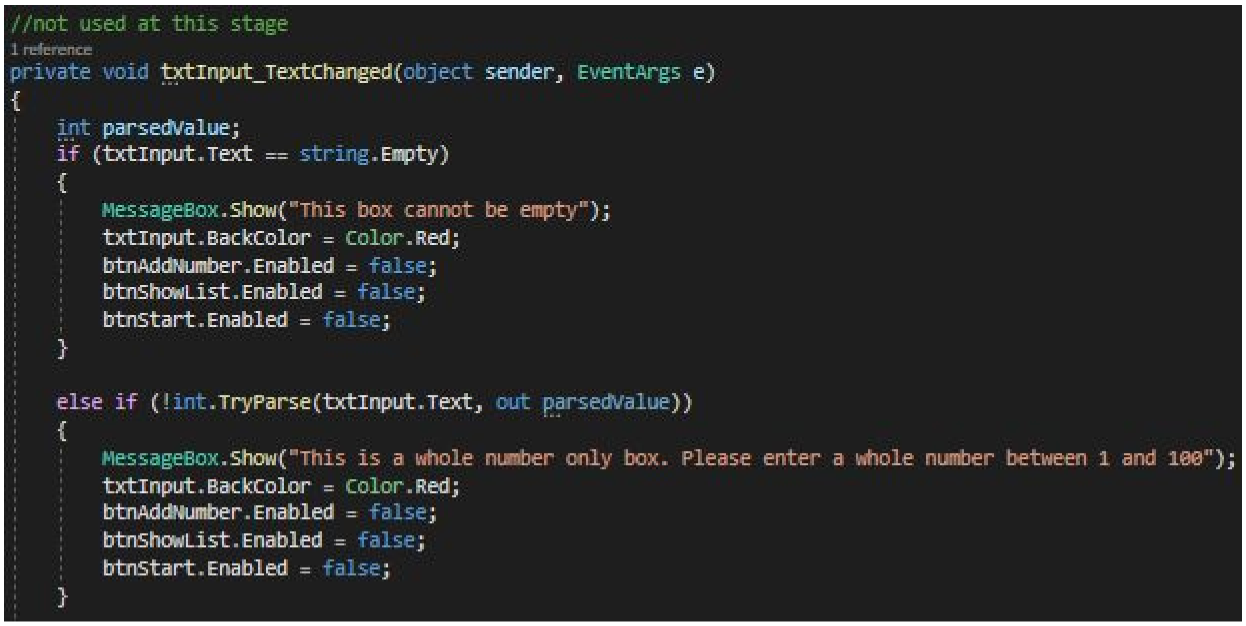


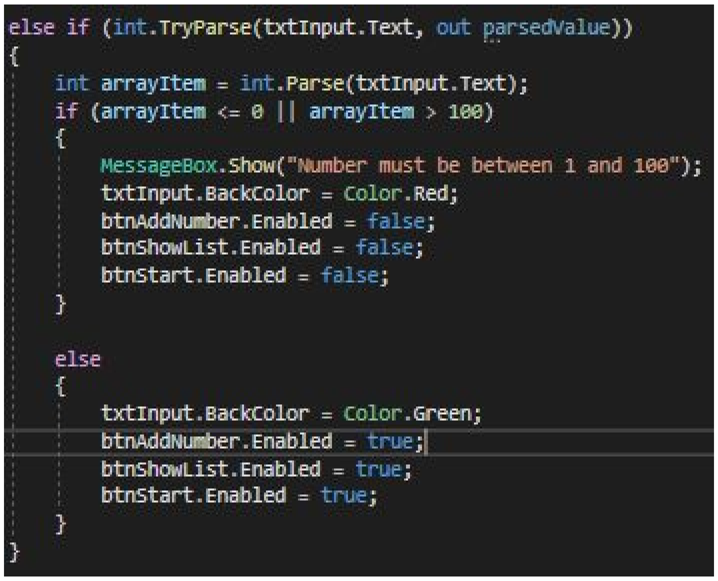
```
//particularly important as is is used for both user input array and random array
1 reference
private void txtArraySize_TextChanged(object sender, EventArgs e)
{
    int parsedValue;    //temporary variable for parsing statement
    if (txtArraySize.Text == string.Empty)      //if the box is empty
    {
        MessageBox.Show("This box cannot be empty");    //tells the user the box should not be empty empty
        txtArraySize.BackColor = Color.Red;     //changes the box colour to red
        btnAddNumber.Enabled = false; //disables the button
        btnShowList.Enabled = false;
        btnStart.Enabled = false;
    }

    else if (!int.TryParse(txtArraySize.Text, out parsedValue)) //this statement was adapted from: https://stackoverflow.com/questions/15399323/validating-whether-a-
    textbox-contains-only-numbers - it checks if the text in the box is an not an integer
    {
        MessageBox.Show("This is a whole number only box. Please enter a whole number");  //tells the user they need to enter an integer
        txtArraySize.BackColor = Color.Red;
        btnAddNumber.Enabled = false; //disables the button
        btnShowList.Enabled = false;
        btnStart.Enabled = false;
    }
```



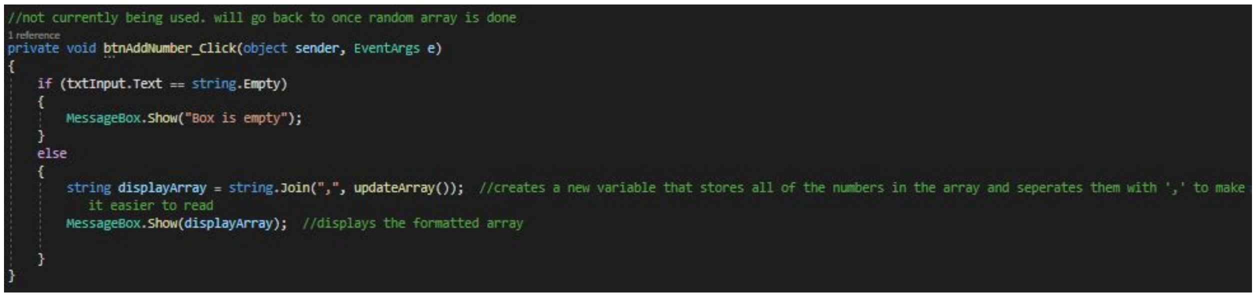
```
else if (int.TryParse(txtArraySize.Text, out parsedValue))
{
    int arraySize = int.Parse(txtArraySize.Text);
    if (arraySize <= 0 || arraySize > 100)
    {
        MessageBox.Show("Number must be between 1 and 100");
        txtArraySize.BackColor = Color.Red;
        btnAddNumber.Enabled = false; //disables the button
        btnShowList.Enabled = false;
        btnStart.Enabled = false;
    }
    else
    {
        txtArraySize.BackColor = Color.Green;   //changes the box colour to green
        btnAddNumber.Enabled = true; //enables the button
        btnShowList.Enabled = true;
        btnStart.Enabled = true;
    }
}
```

Array data text box validation


```
//not used at this stage
1 reference
private void txtInput_TextChanged(object sender, EventArgs e)
{
    int parsedValue;
    if (txtInput.Text == string.Empty)
    {
        MessageBox.Show("This box cannot be empty");
        txtInput.BackColor = Color.Red;
        btnAddNumber.Enabled = false;
        btnShowList.Enabled = false;
        btnStart.Enabled = false;
    }

    else if (!int.TryParse(txtInput.Text, out parsedValue))
    {
        MessageBox.Show("This is a whole number only box. Please enter a whole number between 1 and 100");
        txtInput.BackColor = Color.Red;
        btnAddNumber.Enabled = false;
        btnShowList.Enabled = false;
        btnStart.Enabled = false;
    }
```

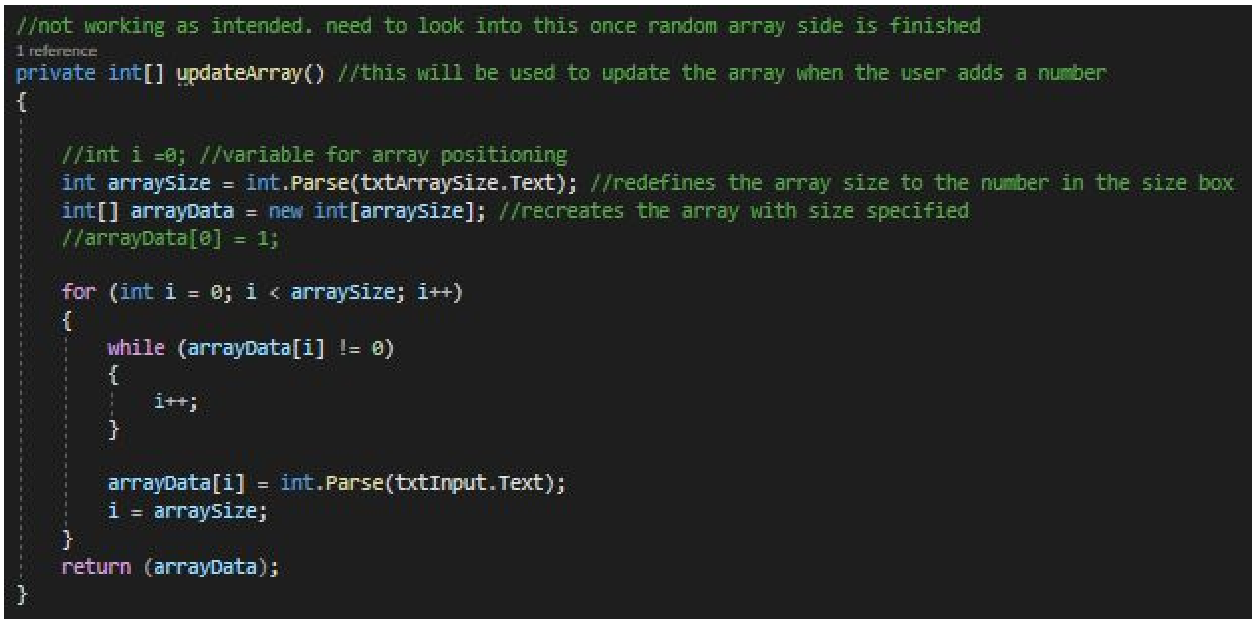

```
else if (int.TryParse(txtInput.Text, out parsedValue))
{
    int arrayItem = int.Parse(txtInput.Text);
    if (arrayItem <= 0 || arrayItem > 100)
    {
        MessageBox.Show("Number must be between 1 and 100");
        txtInput.BackColor = Color.Red;
        btnAddNumber.Enabled = false;
        btnShowList.Enabled = false;
        btnStart.Enabled = false;
    }

    else
    {
        txtInput.BackColor = Color.Green;
        btnAddNumber.Enabled = true;
        btnShowList.Enabled = true;
        btnStart.Enabled = true;
    }
}
```

Add number button with validation



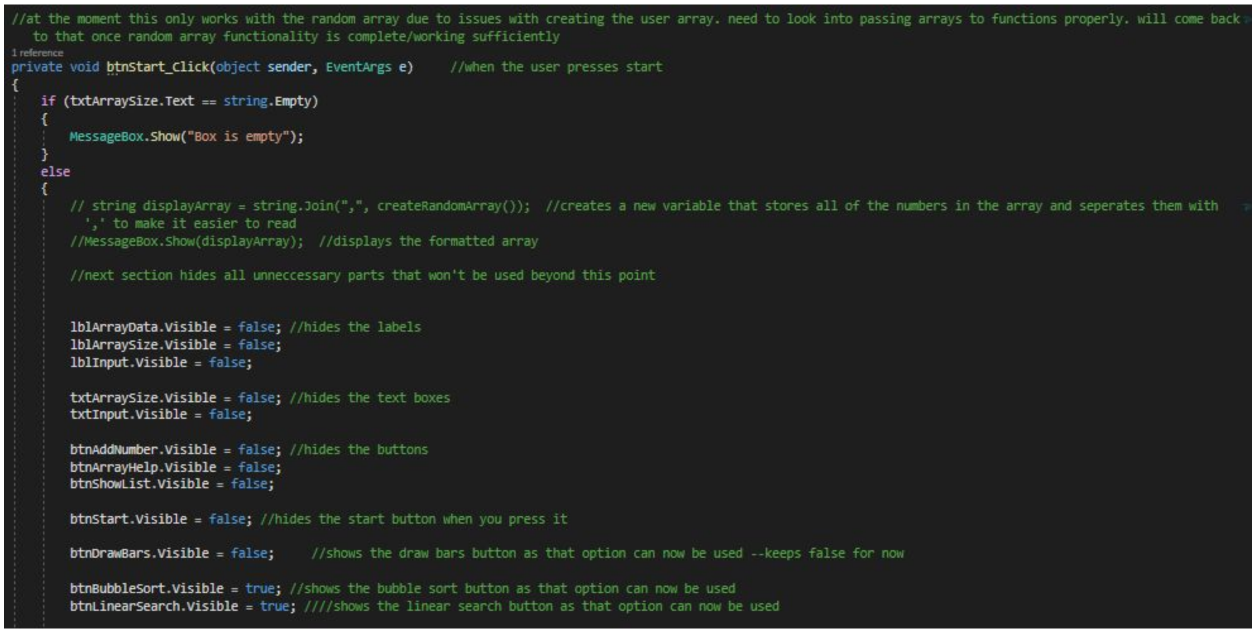
```
//not currently being used. will go back to once random array is done
1 reference
private void btnAddNumber_Click(object sender, EventArgs e)
{
    if (txtInput.Text == string.Empty)
    {
        MessageBox.Show("Box is empty");
    }
    else
    {
        string displayArray = string.Join(",", updateArray());  //creates a new variable that stores all of the numbers in the array and seperates them with ',' to make
           it easier to read
        MessageBox.Show(displayArray);  //displays the formatted array

    }
}
```

Update array button

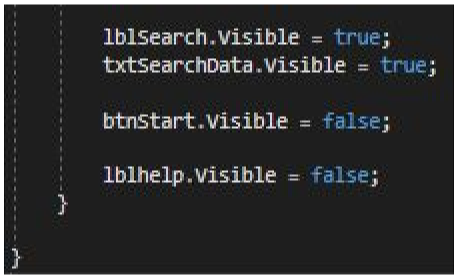


```
//not working as intended. need to look into this once random array side is finished
1 reference
private int[] updateArray() //this will be used to update the array when the user adds a number
{

    //int i =0; //variable for array positioning
    int arraySize = int.Parse(txtArraySize.Text); //redefines the array size to the number in the size box
    int[] arrayData = new int[arraySize]; //recreates the array with size specified
    //arrayData[0] = 1;

    for (int i = 0; i < arraySize; i++)
    {
        while (arrayData[i] != 0)
        {
            i++;
        }

        arrayData[i] = int.Parse(txtInput.Text);
        i = arraySize;
    }
    return (arrayData);
}
```

Start button with validation

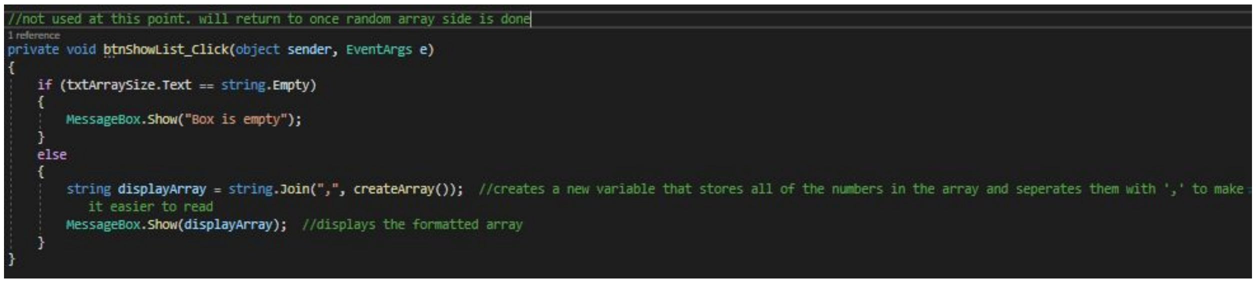


```
//at the moment this only works with the random array due to issues with creating the user array. need to look into passing arrays to functions properly. will come back
   to that once random array functionality is complete/working sufficiently
1 reference
private void btnStart_Click(object sender, EventArgs e)     //when the user presses start
{
    if (txtArraySize.Text == string.Empty)
    {
        MessageBox.Show("Box is empty");
    }
    else
    {
        // string displayArray = string.Join(",", createRandomArray());  //creates a new variable that stores all of the numbers in the array and seperates them with
          ',' to make it easier to read
        //MessageBox.Show(displayArray);  //displays the formatted array

        //next section hides all unneccessary parts that won't be used beyond this point


        lblArrayData.Visible = false; //hides the labels
        lblArraySize.Visible = false;
        lblInput.Visible = false;

        txtArraySize.Visible = false; //hides the text boxes
        txtInput.Visible = false;

        btnAddNumber.Visible = false; //hides the buttons
        btnArrayHelp.Visible = false;
        btnShowList.Visible = false;

        btnStart.Visible = false; //hides the start button when you press it

        btnDrawBars.Visible = false;     //shows the draw bars button as that option can now be used --keeps false for now

        btnBubbleSort.Visible = true; //shows the bubble sort button as that option can now be used
        btnLinearSearch.Visible = true; ////shows the linear search button as that option can now be used
```



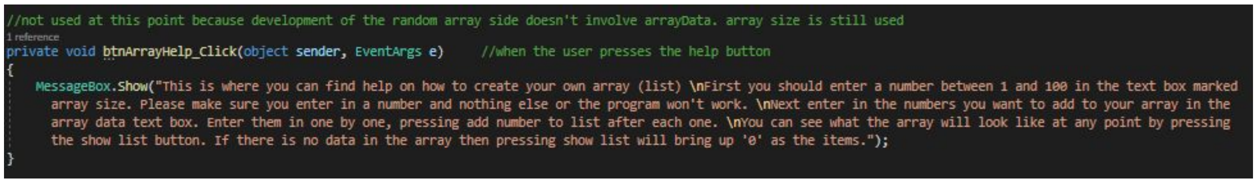
```
        lblSearch.Visible = true;
        txtSearchData.Visible = true;

        btnStart.Visible = false;

        lblhelp.Visible = false;
    }

}
```

Show list button with validation



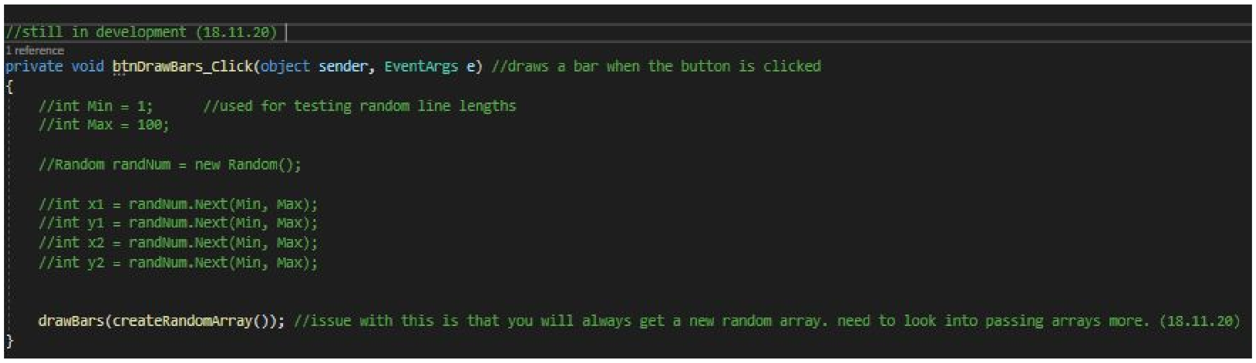
```
//not used at this point. will return to once random array side is done
1 reference
private void btnShowList_Click(object sender, EventArgs e)
{
    if (txtArraySize.Text == string.Empty)
    {
        MessageBox.Show("Box is empty");
    }
    else
    {
        string displayArray = string.Join(",", createArray());  //creates a new variable that stores all of the numbers in the array and seperates them with ',' to make
          it easier to read
        MessageBox.Show(displayArray);  //displays the formatted array
    }
}
```

Array creation help button



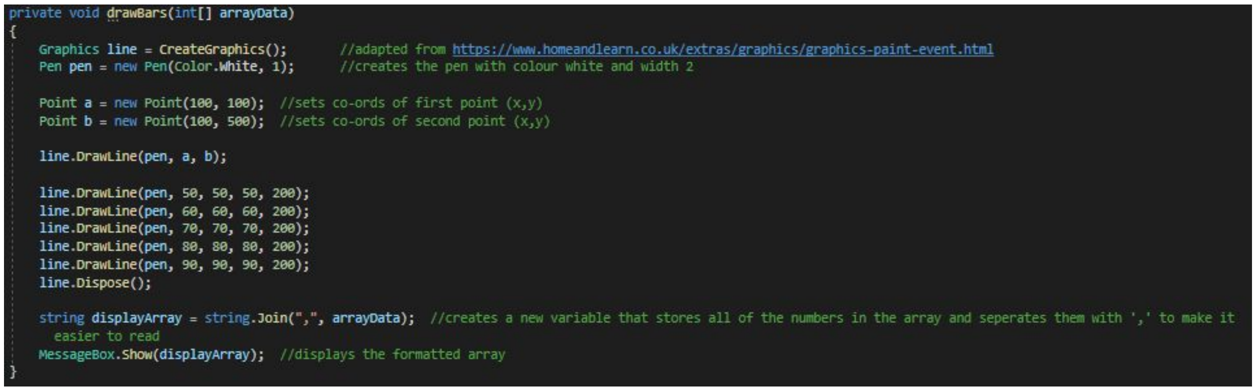
```
//not used at this point because development of the random array side doesn't involve arrayData. array size is still used
1 reference
private void btnArrayHelp_Click(object sender, EventArgs e)     //when the user presses the help button
{
    MessageBox.Show("This is where you can find help on how to create your own array (list) \nFirst you should enter a number between 1 and 100 in the text box marked
      array size. Please make sure you enter in a number and nothing else or the program won't work. \nNext enter in the numbers you want to add to your array in the
      array data text box. Enter them in one by one, pressing add number to list after each one. \nYou can see what the array will look like at any point by pressing
      the show list button. If there is no data in the array then pressing show list will bring up '0' as the items.");
}
```

Draw bars button



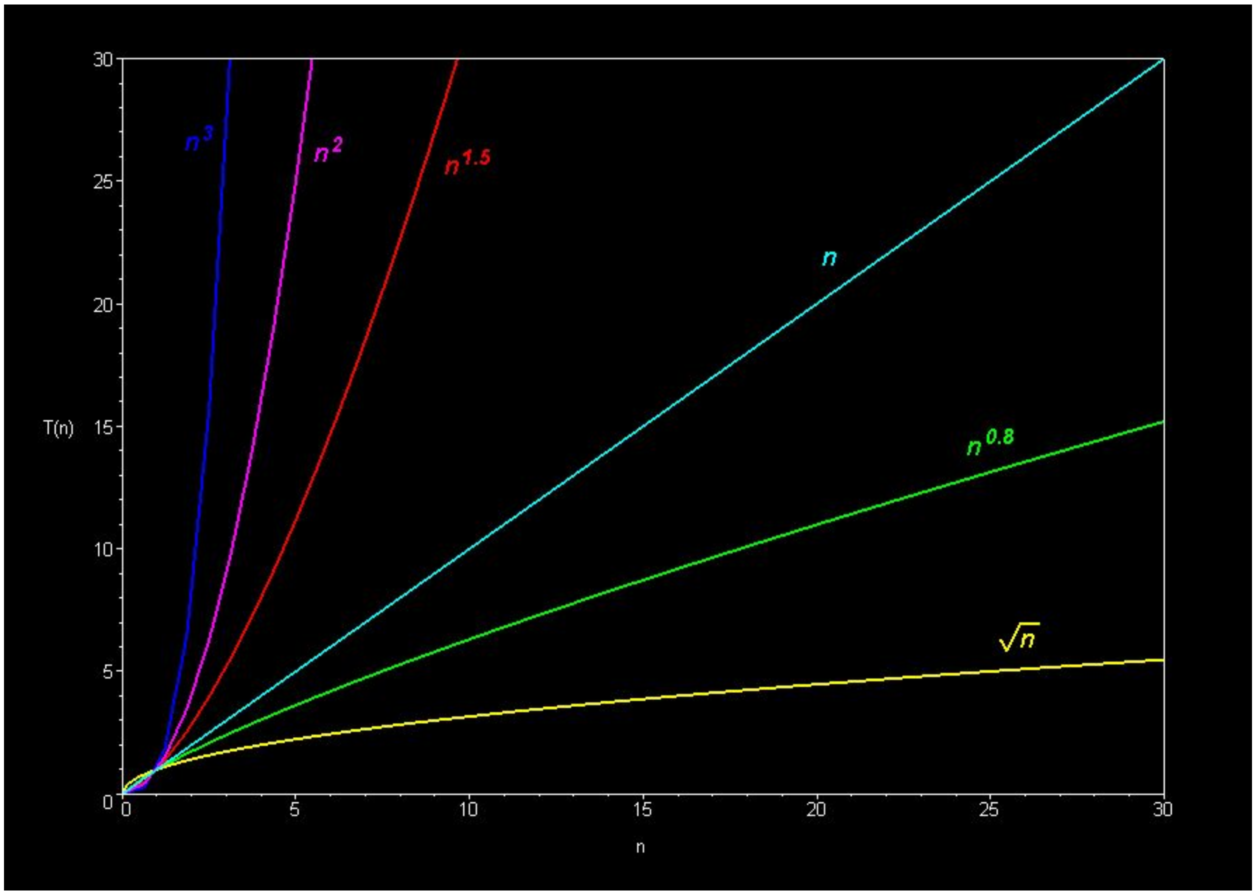
```
//still in development (18.11.20)
1 reference
private void btnDrawBars_Click(object sender, EventArgs e) //draws a bar when the button is clicked
{
    //int Min = 1;      //used for testing random line lengths
    //int Max = 100;

    //Random randNum = new Random();

    //int x1 = randNum.Next(Min, Max);
    //int y1 = randNum.Next(Min, Max);
    //int x2 = randNum.Next(Min, Max);
    //int y2 = randNum.Next(Min, Max);


    drawBars(createRandomArray()); //issue with this is that you will always get a new random array. need to look into passing arrays more. (18.11.20)
}
```

Draw bars function

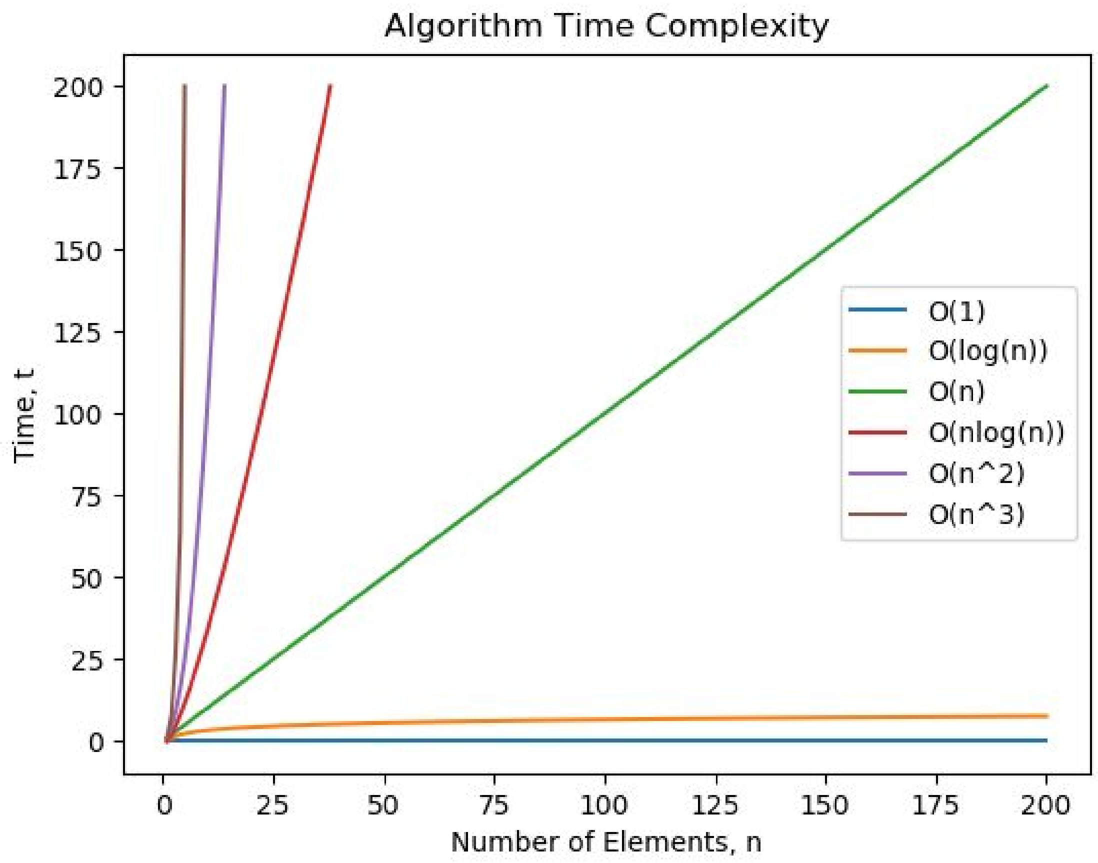


```
private void drawBars(int[] arrayData)
{
    Graphics line = CreateGraphics();       //adapted from https://www.homeandlearn.co.uk/extras/graphics/graphics-paint-event.html
    Pen pen = new Pen(Color.White, 1);      //creates the pen with colour white and width 2

    Point a = new Point(100, 100);  //sets co-ords of first point (x,y)
    Point b = new Point(100, 500);  //sets co-ords of second point (x,y)

    line.DrawLine(pen, a, b);

    line.DrawLine(pen, 50, 50, 50, 200);
    line.DrawLine(pen, 60, 60, 60, 200);
    line.DrawLine(pen, 70, 70, 70, 200);
    line.DrawLine(pen, 80, 80, 80, 200);
    line.DrawLine(pen, 90, 90, 90, 200);
    line.Dispose();

    string displayArray = string.Join(",", arrayData);  //creates a new variable that stores all of the numbers in the array and seperates them with ',' to make it
      easier to read
    MessageBox.Show(displayArray);  //displays the formatted array
}
```

Run-time graphs



- http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/index.html



- https://blog.mbedded.ninja/programming/algorithms-and-data-structures/algorithm-time-complexity/

Sorts

Bubble Sort

Information

The Bubble sort is an extremely simple comparison quadratic sorting algorithm which works by comparing two adjacent values [n] and [n+1] and swaps them if they are in the wrong order. This is repeated until every item in the array is in its correct position.
Its simplicity makes it very useful for educating students as an introduction to sorting algorithms but that is where its use ends. Bubble sort is very slow when working with large data sets.
This is due to its worst case and average time complexity (expressed with Big O notation) of O(n^2), where n = the number of items being sorted. This means the time it takes on average to complete the algorithm goes up with the square of the amount of items in the data set.
For example the time it would take to perform a bubble sort on 10 items would be 10^2 so 100. Whereas with 50 items it would be 50^2 so 2500, 25 times longer. The difference in time

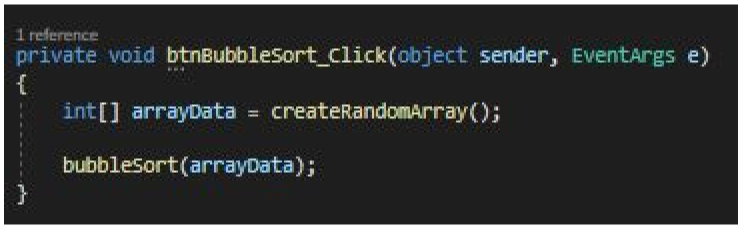
becomes massive the larger your data set is, the Bubble sort is already quite inefficient so each step takes longer to begin with.
Luckily (in the case of this program) the difference in time will not be too noticeable. This is due to the restrictions placed on the size of the array. However users will still be able to see the inefficiencies in the sort itself.
~~Sorting 1,000,000 integers would take 15,625x more time than sorting 8,000. Sorting 8,000 takes about one second on average (on my laptop, with an average over 1000 runs) so 1,000,000 would take 15,625 seconds to sort. That's equivalent to 4.34 hours if workings out are correct. Which would take 4,340 hours (about 25 weeks) to get 1000 runs to work out its average. Luckily we already have its Big O notation so we wouldn't have to do all of this manually now.~~

Link to time complexity research sheet here

Implementation in Iteration 1

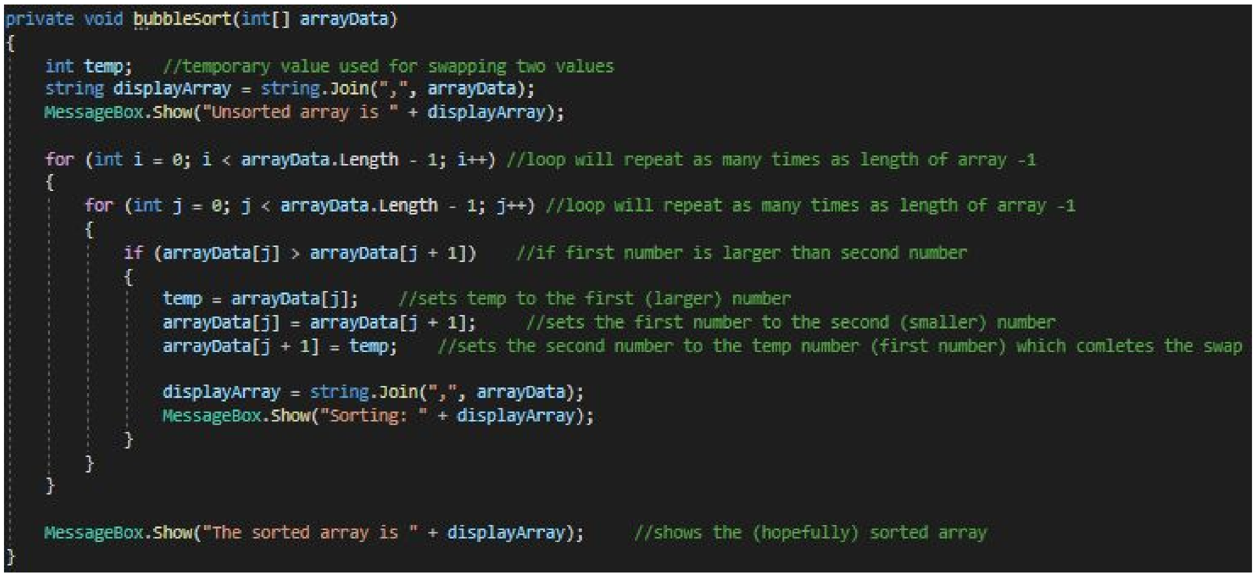
Bubble sort was the easiest sort to implement due to its simplicity. No issues.

Button



```
1 reference
private void btnBubbleSort_Click(object sender, EventArgs e)
{
    int[] arrayData = createRandomArray();

    bubbleSort(arrayData);
}
```

Function



```
private void bubbleSort(int[] arrayData)
{
    int temp;   //temporary value used for swapping two values
    string displayArray = string.Join(",", arrayData);
    MessageBox.Show("Unsorted array is " + displayArray);

    for (int i = 0; i < arrayData.Length - 1; i++) //loop will repeat as many times as length of array -1
    {
        for (int j = 0; j < arrayData.Length - 1; j++) //loop will repeat as many times as length of array -1
        {
            if (arrayData[j] > arrayData[j + 1])    //if first number is larger than second number
            {
                temp = arrayData[j];    //sets temp to the first (larger) number
                arrayData[j] = arrayData[j + 1];     //sets the first number to the second (smaller) number
                arrayData[j + 1] = temp;    //sets the second number to the temp number (first number) which comletes the swap

                displayArray = string.Join(",", arrayData);
                MessageBox.Show("Sorting: " + displayArray);
            }
        }
    }

    MessageBox.Show("The sorted array is " + displayArray);     //shows the (hopefully) sorted array
}
```

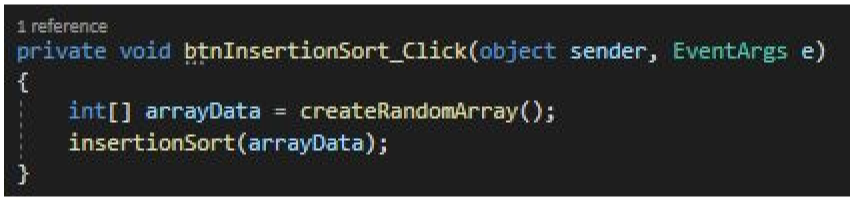
Insertion Sort

Information

Insertion sort is another simple comparison quadratic sorting algorithm. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.
It shares the same weaknesses and strengths of bubble sort, it is slow on large lists/arrays, with its worst case and average time complexity of O(n^2). It is still relatively quick on small arrays. It may share the same time complexity with bubble sort on average but it is a more efficient algorithm, therefore it completes its steps quicker, leading to it being faster than bubble sort but still heavily outclasses by sorts like merge sort and quicksort.
Variations of insertion sort include the shell sort which has variations of itself. These are generally more efficient but their time complexity depends on the gap used when comparing values.

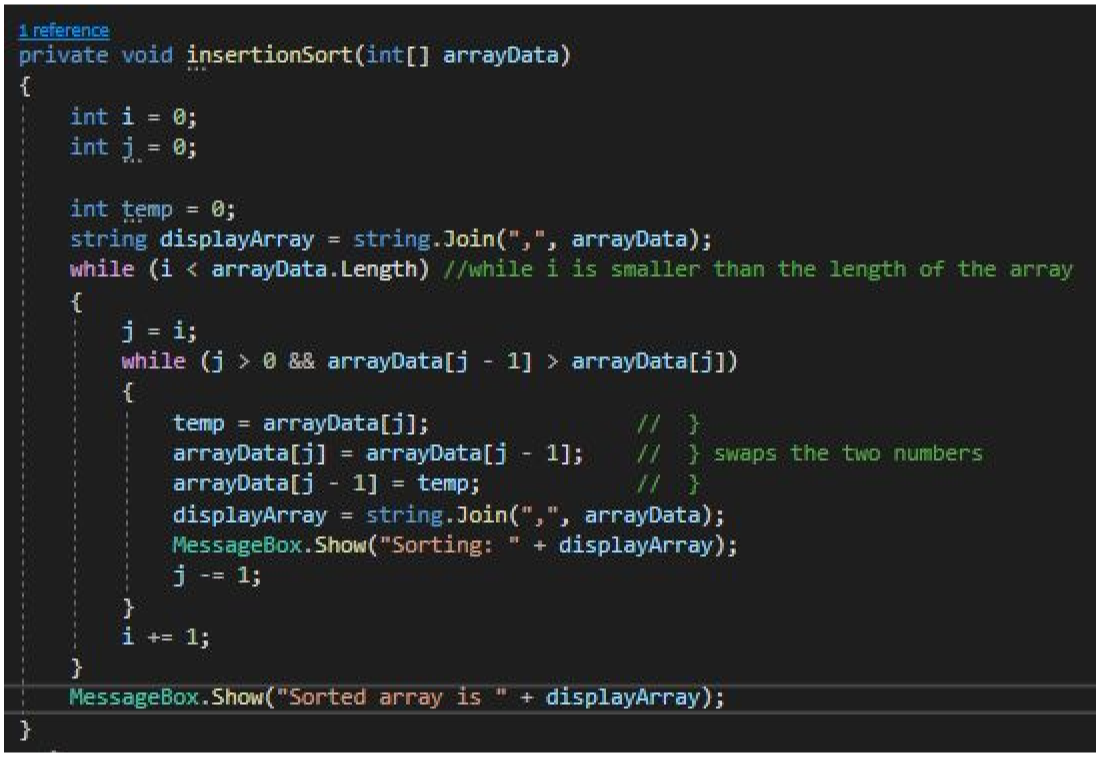
Implementation in iteration 2

Button



```
1 reference
private void btnInsertionSort_Click(object sender, EventArgs e)
{
    int[] arrayData = createRandomArray();
    insertionSort(arrayData);
}
```

Function



```
1 reference
private void insertionSort(int[] arrayData)
{
    int i = 0;
    int j = 0;

    int temp = 0;
    string displayArray = string.Join(",", arrayData);
    while (i < arrayData.Length) //while i is smaller than the length of the array
    {
        j = i;
        while (j > 0 && arrayData[j - 1] > arrayData[j])
        {
            temp = arrayData[j];                //  }
            arrayData[j] = arrayData[j - 1];    //  } swaps the two numbers
            arrayData[j - 1] = temp;            //  }
            displayArray = string.Join(",", arrayData);
            MessageBox.Show("Sorting: " + displayArray);
            j -= 1;
        }
        i += 1;
    }
    MessageBox.Show("Sorted array is " + displayArray);
}
```

Merge Sort

Information

Merge sort is an efficient comparison sorting algorithm that words by process of ‘divide and conquer’. It splits the array into n (number of items in array) sublists. So each sublist contains one item to begin with. It then repeatedly merges sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.
This scales well to very large lists/arrays, because its worst-case and average time complexity is O(n log n).

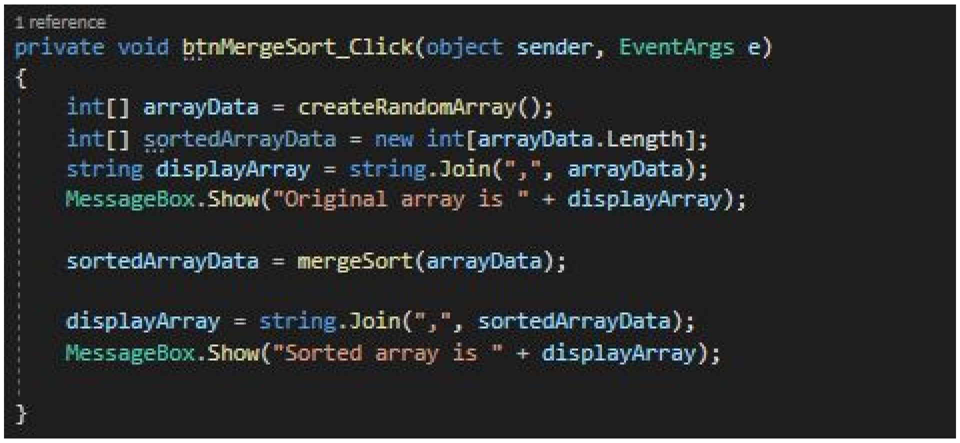
Implementation in iteration 2

Merge sort relies on using two functions:

1. Recursively splitting the array
2. Sort and merging the arrays into one

~~Because of this at the moment it is not possible to show the steps of the sort~~ (resolved 08/12/2020). So unlike the previous sorts, the output is displayed through the button rather than the sort function. Despite it being a more complex algorithm it was fairly simple to implement.

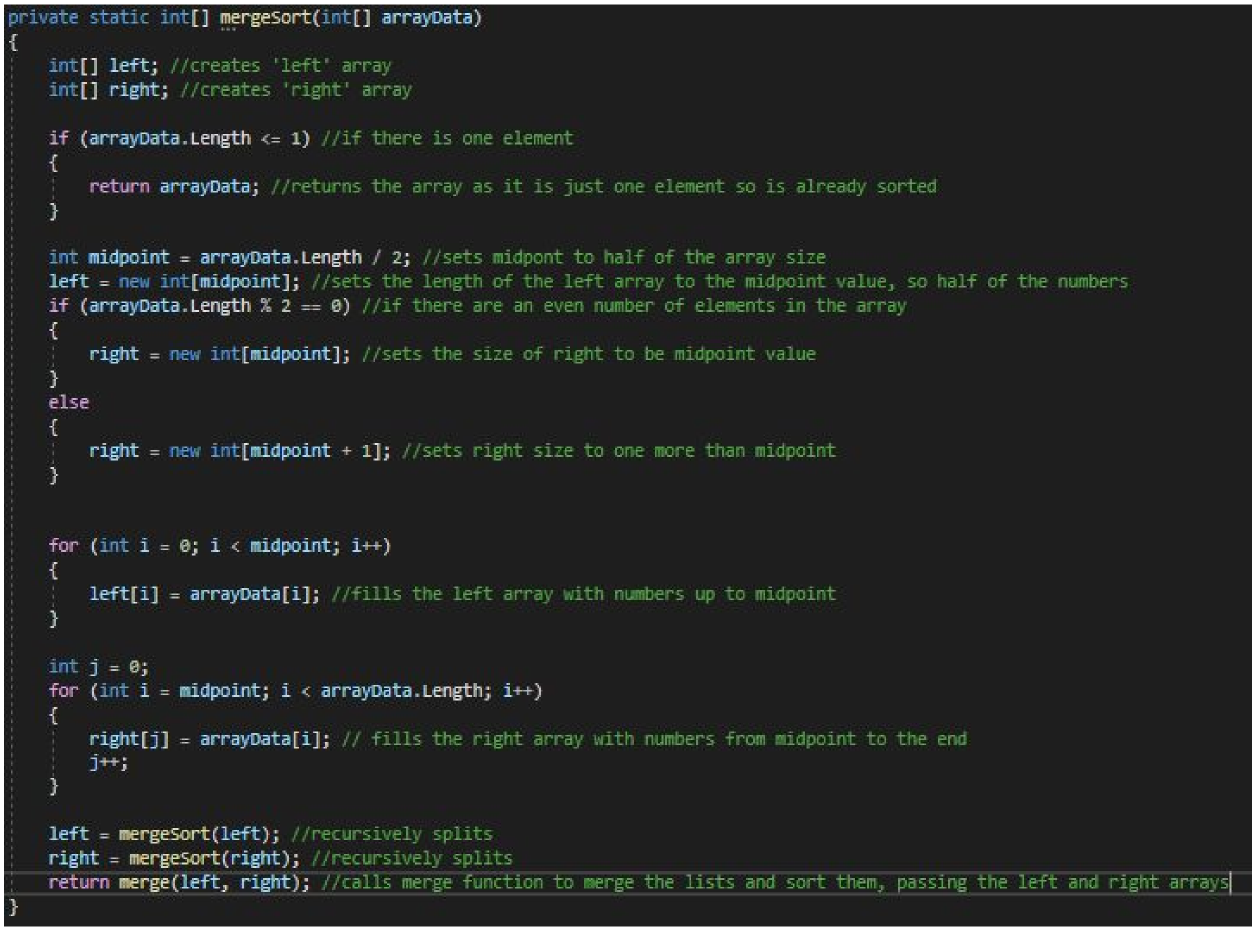
Button



```
1 reference
private void btnMergeSort_Click(object sender, EventArgs e)
{
    int[] arrayData = createRandomArray();
    int[] sortedArrayData = new int[arrayData.Length];
    string displayArray = string.Join(",", arrayData);
    MessageBox.Show("Original array is " + displayArray);

    sortedArrayData = mergeSort(arrayData);

    displayArray = string.Join(",", sortedArrayData);
    MessageBox.Show("Sorted array is " + displayArray);

}
```

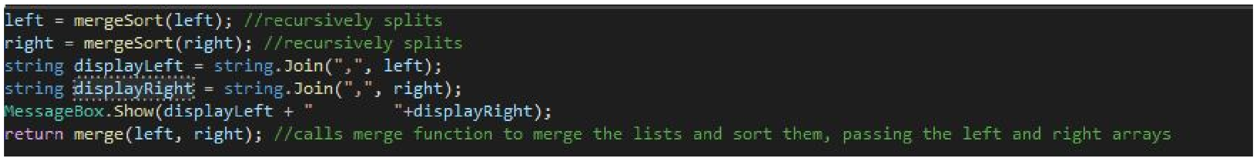
Functions

Splitting



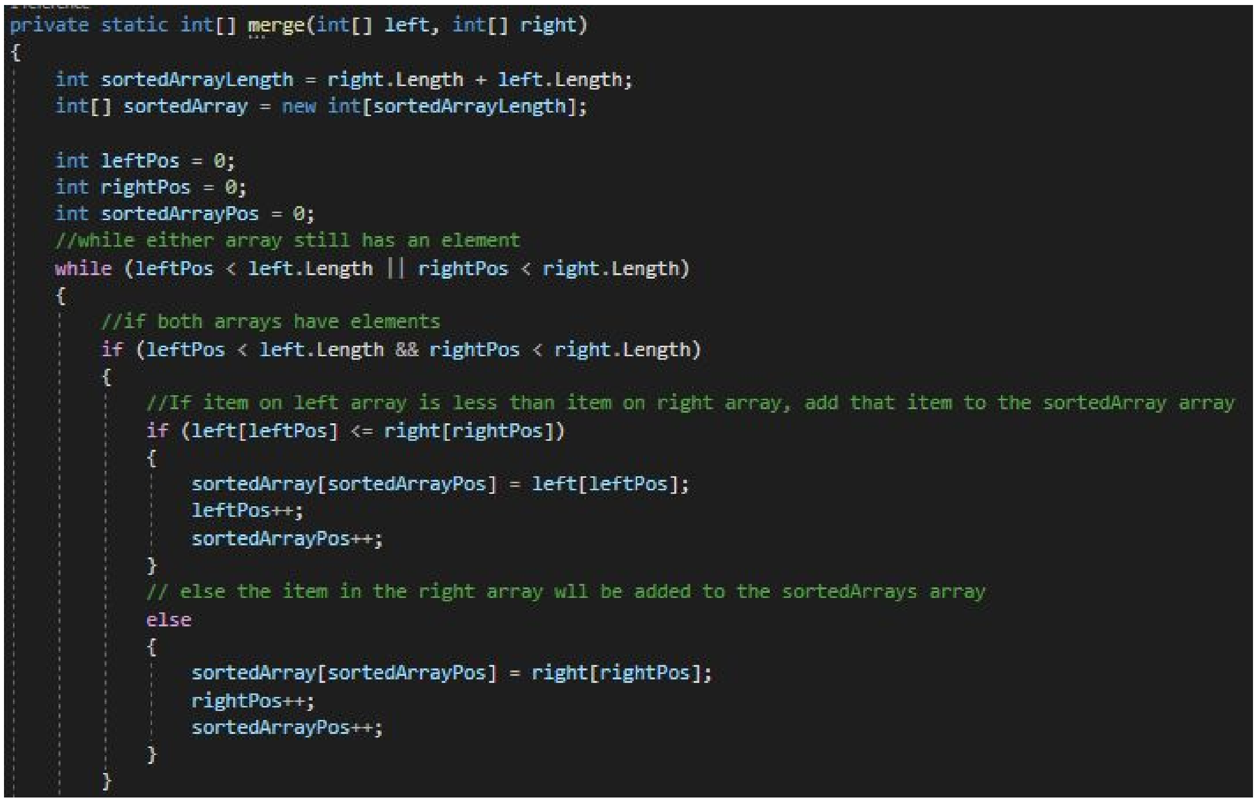
```
private static int[] mergeSort(int[] arrayData)
{
    int[] left; //creates 'left' array
    int[] right; //creates 'right' array

    if (arrayData.Length <= 1) //if there is one element
    {
        return arrayData; //returns the array as it is just one element so is already sorted
    }

    int midpoint = arrayData.Length / 2; //sets midpont to half of the array size
    left = new int[midpoint]; //sets the length of the left array to the midpoint value, so half of the numbers
    if (arrayData.Length % 2 == 0) //if there are an even number of elements in the array
    {
        right = new int[midpoint]; //sets the size of right to be midpoint value
    }
    else
    {
        right = new int[midpoint + 1]; //sets right size to one more than midpoint
    }


    for (int i = 0; i < midpoint; i++)
    {
        left[i] = arrayData[i]; //fills the left array with numbers up to midpoint
    }

    int j = 0;
    for (int i = midpoint; i < arrayData.Length; i++)
    {
        right[j] = arrayData[i]; // fills the right array with numbers from midpoint to the end
        j++;
    }

    left = mergeSort(left); //recursively splits
    right = mergeSort(right); //recursively splits
    return merge(left, right); //calls merge function to merge the lists and sort them, passing the left and right arrays
}
```

Splitting with showing steps - shows changed code

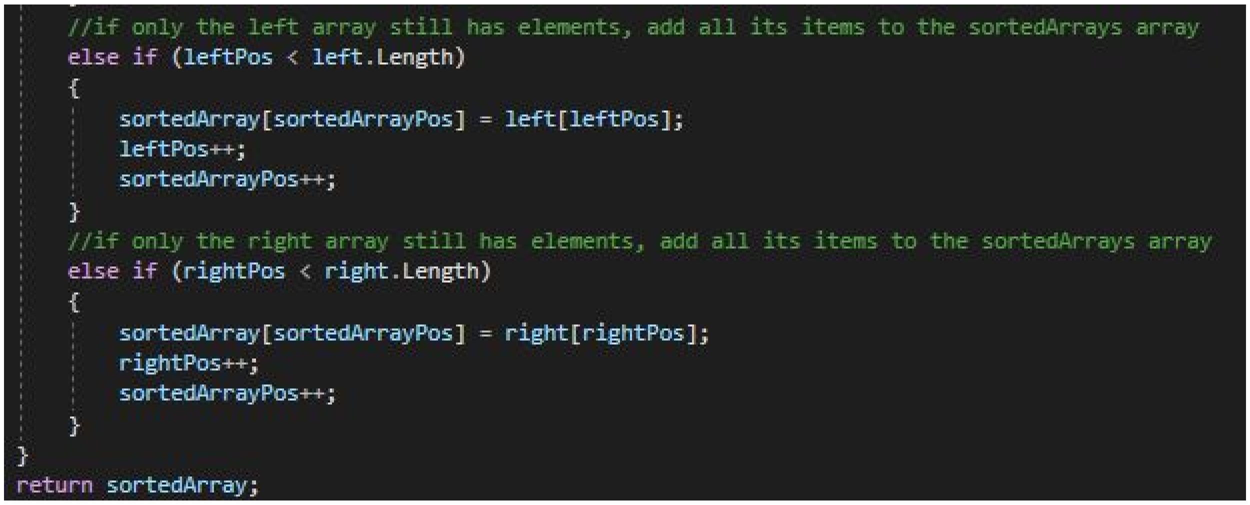


```
left = mergeSort(left); //recursively splits
right = mergeSort(right); //recursively splits
string displayLeft = string.Join(",", left);
string displayRight = string.Join(",", right);
MessageBox.Show(displayLeft + "        "+displayRight);
return merge(left, right); //calls merge function to merge the lists and sort them, passing the left and right arrays
```

Merging

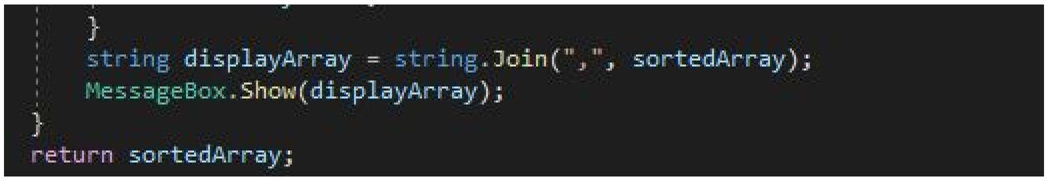


```
private static int[] merge(int[] left, int[] right)
{
    int sortedArrayLength = right.Length + left.Length;
    int[] sortedArray = new int[sortedArrayLength];

    int leftPos = 0;
    int rightPos = 0;
    int sortedArrayPos = 0;
    //while either array still has an element
    while (leftPos < left.Length || rightPos < right.Length)
    {
        //if both arrays have elements
        if (leftPos < left.Length && rightPos < right.Length)
        {
            //If item on left array is less than item on right array, add that item to the sortedArray array
            if (left[leftPos] <= right[rightPos])
            {
                sortedArray[sortedArrayPos] = left[leftPos];
                leftPos++;
                sortedArrayPos++;
            }
            // else the item in the right array wll be added to the sortedArrays array
            else
            {
                sortedArray[sortedArrayPos] = right[rightPos];
                rightPos++;
                sortedArrayPos++;
            }
        }
```



```
        //if only the left array still has elements, add all its items to the sortedArrays array
        else if (leftPos < left.Length)
        {
            sortedArray[sortedArrayPos] = left[leftPos];
            leftPos++;
            sortedArrayPos++;
        }
        //if only the right array still has elements, add all its items to the sortedArrays array
        else if (rightPos < right.Length)
        {
            sortedArray[sortedArrayPos] = right[rightPos];
            rightPos++;
            sortedArrayPos++;
        }
    }
    return sortedArray;
```

Merging with showing steps - shows changed code



```
    }
    string displayArray = string.Join(",", sortedArray);
    MessageBox.Show(displayArray);
}
return sortedArray;
```

Quick Sort

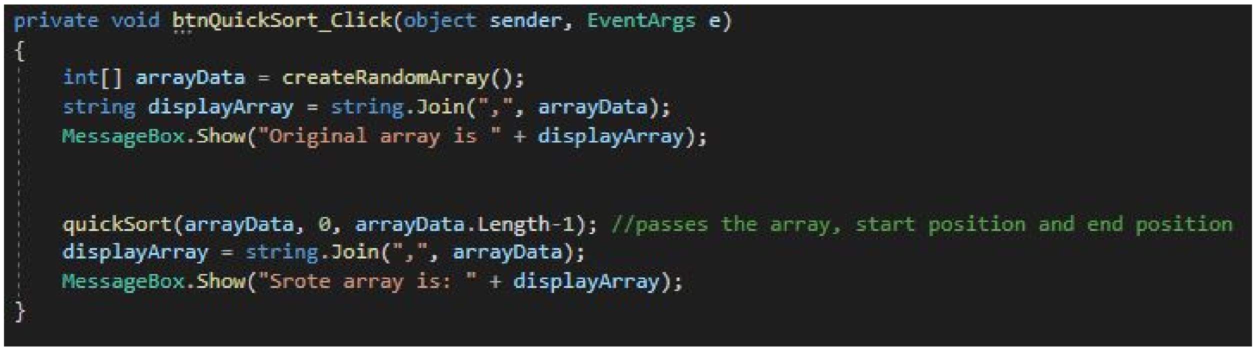
Information

Quicksort is another divide and conquer algorithm, however this one relies on a partition operation. To partition an array, an element called a pivot is selected. All elements smaller than the pivot are moved before it and all greater elements are moved after it. The lesser and greater sublists are then recursively sorted.
This yields an average time complexity of O(n log n) when a favourable partition is chosen. However its worst-case is O(n^2). This is usually when the partition is the end or start of the list, otherwise it is very rare. In practice choosing a random pivot almost certainly yields O(n log n) performance.
Efficient implementations of quicksort (with in-place partitioning) are typically unstable sorts and somewhat complex, but are among the fastest sorting algorithms in practice.
There are faster sorting algorithms that do not rely on comparison such as radix sort, these algorithms are quite complex ~~- quite outside my range of understanding~~. They are also not required to be learnt for the AQA CS A-level or GCSE specification. Because of this they will not be covered in this program because users (if they are in high school or college) will not need to know them. They will only be included if there is sufficient time before project submission which is unlikely.

Implementation in iteration 2

Many implementations of quick sort use a pivot of the first or last element of the array. Whilst this is simple and does function, it will likely lead to the time complexity of O(n^2). This is the worst case scenario, as such a random pivot will be chosen to hopefully reach the average time complexity of O(n log n) - a much more favourable outcome as it will scale better with larger lists. This will use Lomuto's partition Scheme (probably).

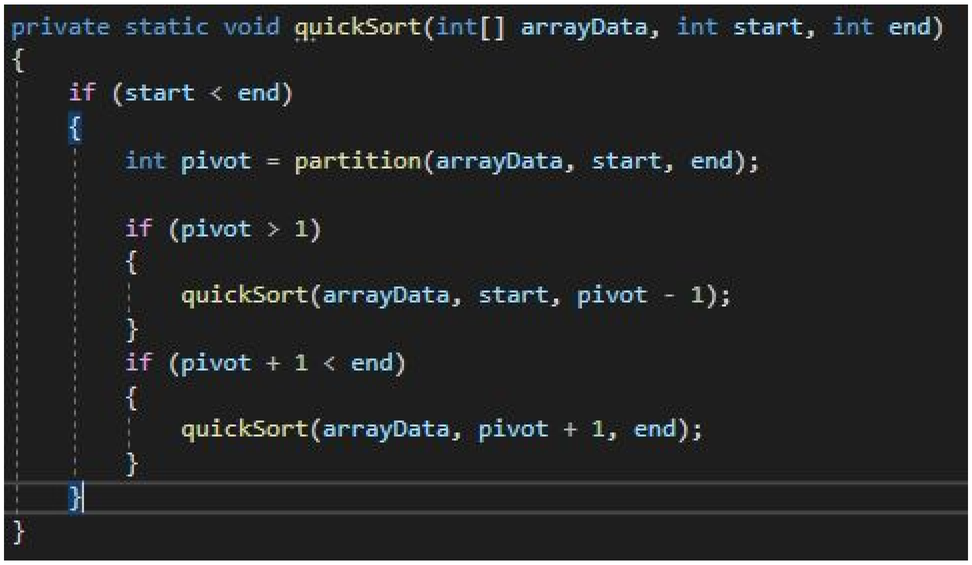
Button



```
private void btnQuickSort_Click(object sender, EventArgs e)
{
    int[] arrayData = createRandomArray();
    string displayArray = string.Join(",", arrayData);
    MessageBox.Show("Original array is " + displayArray);


    quickSort(arrayData, 0, arrayData.Length-1); //passes the array, start position and end position
    displayArray = string.Join(",", arrayData);
    MessageBox.Show("Srote array is: " + displayArray);
}
```

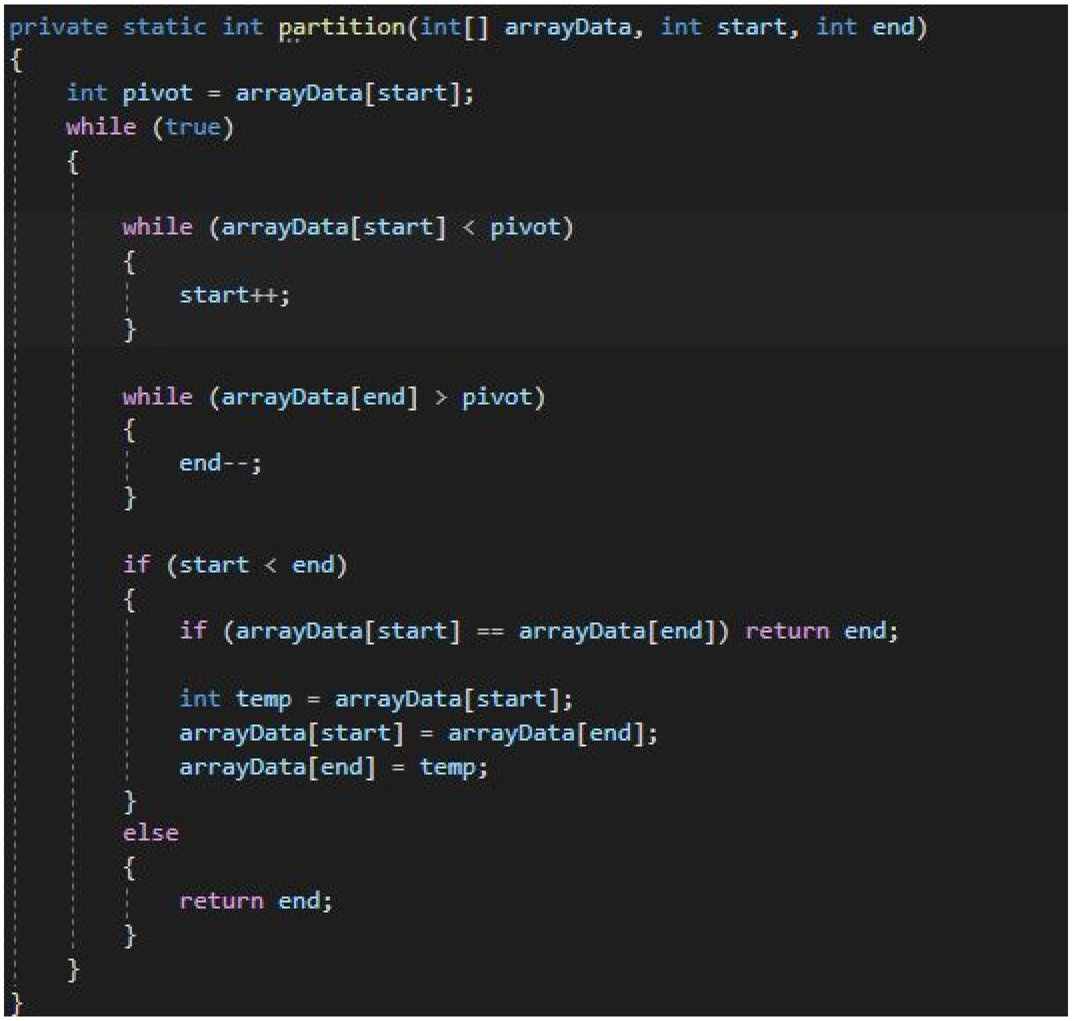
Functions

Quick Sort



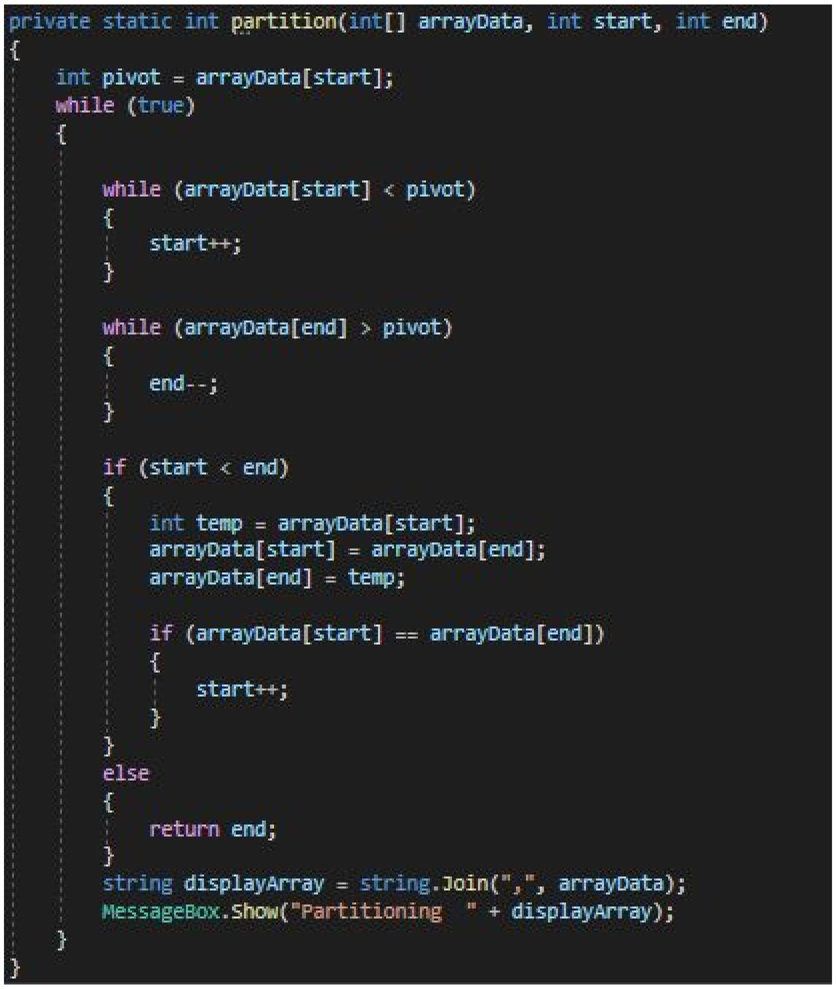
```
private static void quickSort(int[] arrayData, int start, int end)
{
    if (start < end)
    {
        int pivot = partition(arrayData, start, end);

        if (pivot > 1)
        {
            quickSort(arrayData, start, pivot - 1);
        }
        if (pivot + 1 < end)
        {
            quickSort(arrayData, pivot + 1, end);
        }
    }
}
```

Partition



```
private static int partition(int[] arrayData, int start, int end)
{
    int pivot = arrayData[start];
    while (true)
    {

        while (arrayData[start] < pivot)
        {
            start++;
        }

        while (arrayData[end] > pivot)
        {
            end--;
        }

        if (start < end)
        {
            if (arrayData[start] == arrayData[end]) return end;

            int temp = arrayData[start];
            arrayData[start] = arrayData[end];
            arrayData[end] = temp;
        }
        else
        {
            return end;
        }
    }
}
```

Partition allowing for duplicate numbers



```
private static int partition(int[] arrayData, int start, int end)
{
    int pivot = arrayData[start];
    while (true)
    {

        while (arrayData[start] < pivot)
        {
            start++;
        }

        while (arrayData[end] > pivot)
        {
            end--;
        }

        if (start < end)
        {
            int temp = arrayData[start];
            arrayData[start] = arrayData[end];
            arrayData[end] = temp;

            if (arrayData[start] == arrayData[end])
            {
                start++;
            }
        }
        else
        {
            return end;
        }
        string displayArray = string.Join(",", arrayData);
        MessageBox.Show("Partitioning  " + displayArray);
    }
}
```

Searches

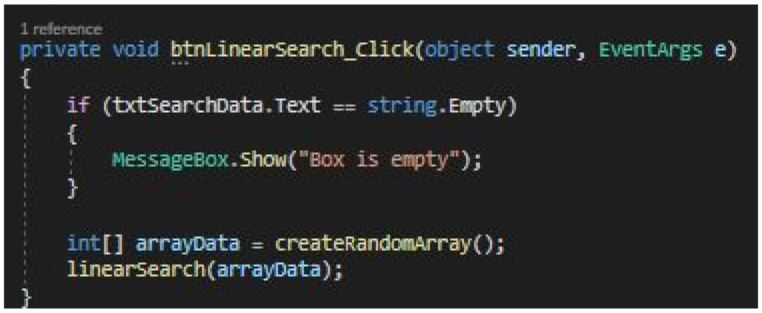
Linear Search

Information

Linear search is a simple search algorithm that sequentially checks each element in a list until a match is found. It is effective on short lists but is much weaker when compared to binary search. Its average time complexity is O(n/2) but worst is O(n). A benefit of linear search is that, unlike binary search, the list does not have to be sorted beforehand

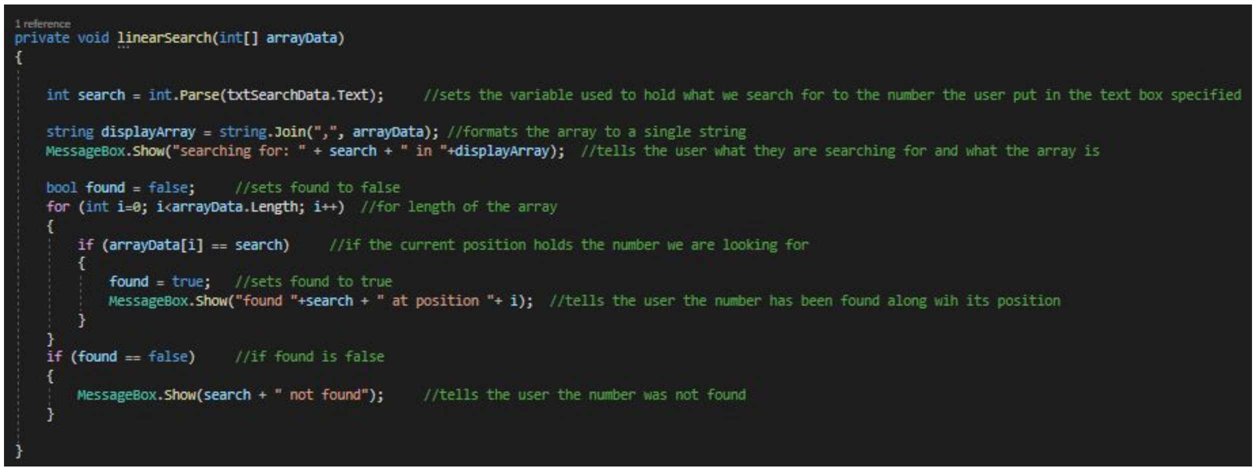
Implementation in iteration 1

Button



```
1 reference
private void btnLinearSearch_Click(object sender, EventArgs e)
{
    if (txtSearchData.Text == string.Empty)
    {
        MessageBox.Show("Box is empty");
    }

    int[] arrayData = createRandomArray();
    linearSearch(arrayData);
}
```

Function



```
1 reference
private void linearSearch(int[] arrayData)
{

    int search = int.Parse(txtSearchData.Text);     //sets the variable used to hold what we search for to the number the user put in the text box specified

    string displayArray = string.Join(",", arrayData); //formats the array to a single string
    MessageBox.Show("searching for: " + search + " in "+displayArray);  //tells the user what they are searching for and what the array is

    bool found = false;     //sets found to false
    for (int i=0; i<arrayData.Length; i++)  //for length of the array
    {
        if (arrayData[i] == search)     //if the current position holds the number we are looking for
        {
            found = true;   //sets found to true
            MessageBox.Show("found "+search + " at position "+ i);  //tells the user the number has been found along wih its position
        }
    }
    if (found == false)     //if found is false
    {
        MessageBox.Show(search + " not found");     //tells the user the number was not found
    }

}
```

Binary Search

Information

Binary search is an algorithm that works by splitting a list and comparing its middle element to the target value. If they are not equal, the half in which the target cannot lie is eliminated and the search continues on the remaining half, again taking the middle element to compare to the target value, and repeating this until the target value is found. Its average and worst case time complexity is O(log n). Binary search performs well with large lists and there are very few downsides to it. If you are searching through a very small list then linear search would be better and the list must be sorted. But with the use of an efficient sorting algorithm, the time spent is still minimal.

Implementation in iteration 2

-

Button

-

Function

-

Testing

Test key:

1.2

Where 1 is the test number and 2 is any **reiteration** of the test e.g. 1.3 is test 1, iteration 3 (not program iteration)

Testing Table

Testing table is here

Evaluation

Meeting the requirements

Results of each iteration

Iteration 1

Iteration 1 did not go entirely as planned. This was mostly due to issues with creating graphics and with passing the array into functions. Users could not fully create their own array at this stage. Because of this those options are hidden but remain in code until fixed. However a random array could successfully be produced. And a linear search or bubble sort can be shown via text on that array.The issues mentioned should be resolved by iteration 2.

Iteration 2

Iteration 2 included an overhaul of the validation system in the main menu. Rather than each text box having its own personalised procedure that it would follow, it simply called the validate function which utilised objects. Due to this code is now a lot more clean.
Merge sort was also added, along with its steps.

Iteration 3

Feedback from users

Iteration 1

The user interface had some issues with it which were mostly to do with the colour choices. “Blue on black is not easy on the eyes.” Because of this the general text colour will be yellow to add more of a contrast.

The text box background colour was grey, when trying to click in the box using the mouse, the cursor disappeared, so the background colour of text boxes will be changed to white. The background colour of buttons will remain grey as they do not suffer from the same issue.

A major issue with the program is that when sorting through large amounts of data, the program takes a while to complete and cannot be closed without the use of Task Manager. This is because the only available sort (currently) is bubble sort which is very slow when being used on a large data set. To resolve this there will be a button or checkbox to disable/enable showing steps to prevent this

Iteration 2

-

Iteration 3

-

Limitations

-

Improvements

-

Bibliography

Author Surname/Corporate name, Initial(s) Year (page revised/created), Title of page, viewed Day Month Year, <URL>.

Microsoft, 2019, Visual Studio 2019 System Requirements | Microsoft Docs, 04/05/2020, <https://docs.microsoft.com/en-us/visualstudio/releases/2019/system-requirements>

Physics & Maths Tutor, 2015, Advanced Notes - 2.2.2 Computational Methods - OCR Computer Science A-level, 30/11/2020, <https://pmt.physicsandmathstutor.com/download/Computer-Science/A-level/Notes/OCR/2.2-Problem-Solving-and-Programming/Advanced/2.2.2.%20Computational%20Methods.pdf>

Stack Overflow, 2019, c# - Validating whether a textbox contains only numbers, 28/10/2020, <https://stackoverflow.com/questions/15399323/validating-whether-a-textbox-contains-only-numbers>

Home and Learn, (no date), The .NET Paint Event, 18/11/2020, <https://www.homeandlearn.co.uk/extras/graphics/graphics-paint-event.html>

mbedded.ninja, 2019, Algorithm Time Complexity | mbedded.ninja, 25/11/20, <https://blog.mbedded.ninja/programming/algorithms-and-data-structures/algorithm-time-complexity/>

Wikipedia, 2020, Bubble sort, 01/11/2020, <https://en.wikipedia.org/wiki/Bubble_sort>

Wikipedia, 2020, Insertion sort, 01/11/2020, <https://en.wikipedia.org/wiki/Insertion_sort>

Wikipedia, 2020, Shellsort, 29/11/2020, <https://en.wikipedia.org/wiki/Shellsort>

Wikipedia, 2020, Merge sort, 29/11/2020, <https://en.wikipedia.org/wiki/Merge_sort>

Wikipedia, 2020, Quicksort, 30/11/2020, <https://en.wikipedia.org/wiki/Quicksort>

Wikipedia, 2020, Radix sort, 30/11/2020, <https://en.wikipedia.org/wiki/Radix_sort>

Wikipedia, 2020, Linear search, 01/11/2020, <https://en.wikipedia.org/wiki/Linear_search>

Wikipedia, 2020, Binary search, 30/11/2020, <https://en.wikipedia.org/wiki/Binary_search_algorithm>

Wikipedia, 2020, Big O Notation, 25/11/2020, <https://en.wikipedia.org/wiki/Big_O_notation>

Marshall, J., 2002, Running Time Graphs, 25/11/2020, <http://science.slc.edu/~jmarshall/courses/2002/spring/cs50/BigO/index.html>

Wikipedia, 2020, Best, worst and average case, 25/11/2020, <https://en.wikipedia.org/wiki/Best,_worst_and_average_case>

Wikipedia, 2020, Time complexity, 25/11/2020, <https://en.wikipedia.org/wiki/Time_complexity>

GeeksforGeeks, 2020, Hoare’s vs Lomuto partition scheme in QuickSort, 25/11/2020, <https://www.geeksforgeeks.org/hoares-vs-lomuto-partition-scheme-quicksort/>

w3resource, 2020, C# Sharp Exercises: Quick sort, 9/12/2020, <https://www.w3resource.com/csharp-exercises/searching-and-sorting-algorithm/searching-and-sorting-algorithm-exercise-9.php>

Links - to be removed before submission

Trello

Testing Sheet

Testing Screenshots

User Feedback

Time Complexity Sheet

Project Handbook

Project Mark Scheme