# Introduction

Sorting algorithms are used to put data, that is normally in an array or a list, in order. This data may be integers, strings, records and or objects. There are many sorting algorithms, but the focus is on these four, which are; bubble sort, insertion sort, merge sort and quick sort. We can evaluate the algorithms in terms of how long they take to execute and how much memory they use. Often speed can be increased at the expense of using more memory. However, the speed and efficiency of each algorithm also depends on the size of dataset. At the end of this report, there will be an evaluation based on which algorithm is the most efficient.

# Specification

## The Problem

The main problem is to find out which sorting algorithm out of the four is the most efficient at its given data set. The dataset will include several random integers which are to be sorted. There needs to be a program/software which can figure out which algorithm is the most efficient.

## The Solution

The aim of the project is to create a sort performance analysis toolkit. This will be a program which will analyse and compare the four different sort algorithms. The way it will be compared is by the time it takes for the array to be sorted. The time will be in microseconds as this is a sensible value to be measured by the program. The array will be generated with random values. The time it takes for each algorithm to sort that array will be stored in a .CSV file. The generated .CSV file will be opened through Microsoft Excel, through which a line graph will be created. On the graph we will have time taken in microseconds and the size of array in integers. This will help clearly identify which algorithm takes longer to sort and which algorithm takes the fastest. Thus, gives an analysis for the algorithm.

it is important to consider the best and most useful way to compare these four algorithms. This is through complexity. Complexity doesn't just show how fast an algorithm takes to perform, but it shows how well it can scale when given larger sets of data to act upon. For example, bubble sort algorithm would display that it works well with small data sets. However, when the data set is of a bigger size bubble sort would take a long time for it to run. This program that will be created will help prove that statement for the bubble sort algorithm. It will also show

In the end, there will be a clear evaluation based on the analysis showing which sort algorithm is the most efficient.

## Analysis of Requirements

This program will be created in Visual Studio. It will be coded in C++, this is because C++ allows you to have a lot of control as to how computer resources are to be used. It is a high-level language so its speed and ability to cheaply use resources makes it better than other languages. To successfully code this project, it is important to use as many programming skills as I can.

## Hardware Requirements

The proposed solution is to create a working program that works out the time taken for the array to be sorted by each algorithm. As there are four different algorithms there will be four copies of the exact same array that will be sorted by each individual algorithm. Below, a list of hardware requirements. It is crucial to have these components to successfully make the program.

* **Computer**, it is important to have a computer to download and run C++.
* **Keyboard**, this is important for user input.
* **Mice**, will be used to do further analysis on Excel.
* **Monitor**, gives a display for the graph to be presented.

## Software Requirements

* This program will be created in Visual Studio. It will be coded in C++, this is because C++ allows you to have a lot of control as to how computer resources are to be used. It is a high-level language so its speed and ability to cheaply use resources makes it better than other languages. To successfully code this project, it is important to use as many programming skills as I can.
* Microsoft Excel and Microsoft Word to create graphs and to make the documentation.

## Additional Requirement:

* Internet is needed to download Visual Studio and Microsoft Office if it’s not already installed.

## Success Criteria

* To create a program that works out the time each algorithm takes to sort the array.
* To successfully build the program in Visual C++.
* To successfully run the program.
* After the program has finished running it must create a .CSV file with sensible data.
* From the sensible data a line graph must be created.

# Design

## Pseudocode – Bubble Sort

Set swapMade to true

While swapMade is true

Set swapMade to false.

Start at position 0

For position = 0 to listlength-2

Compare the item at current position with an item one ahead.

IF they are out of order THEN

Swap items and set swapMade to true

END IF

NEXT position

END WHILE

Bubble sort is a simple sorting algorithm that uses an incremental approach. It compares each pairs of items in an array which are next to each other and if they are in the wrong place then it swaps them. For example, if an array goes, 4,2,3,6,7, then four and two will be swapped as they are in the wrong order. This is then repeated until the array is in the correct order.

## Pseudocode – Insertion Sort

Make the first item the sorted list, the remaining items are the unsorted list.

WHILE there are items in the unsorted list

Take the first item of the unsorted list

WHILE there is an item to the left of it which is smaller than itself

Swap with that item

END WHILE

The sorted list is now one item bigger

END WHILE

Insertion sort, this is also a simple algorithm. It uses an incremental approach. There will be a list with random values and each item in the list Is inserted into the correct order of the organised

## Pseudocode – Merge Sort

WHILE list1 is not empty and list2 is not empty

IF the first item in list1<list2 THEN

Remove the first item from list1 and add it to newlist

ELSE

Remove the first item from list2 and add it to newlist.

ENDIF

ENDWHILE

IF list1 is empty THEN

Add the remainder of list2 to newlist

ELSE

Add the remainder of list1 to newlist

ENDIF

Merge sort is quite like quick sort. It begins by placing each element into its own individual list. Then, each pair of adjacent lists is combined into one sorted list. This then continues until there is a final sorted list.

## Pseudocode – Quick Sort

Take the first item in the list, make it a list one item big and call it the pivot

Split the remainder of the list into two sub-lists: those less than or equal to the pivot and those greater than the pivot

Recursively apply step 2 until all sub-lists are pivots

The pivots can now be combined to form a sorted list.

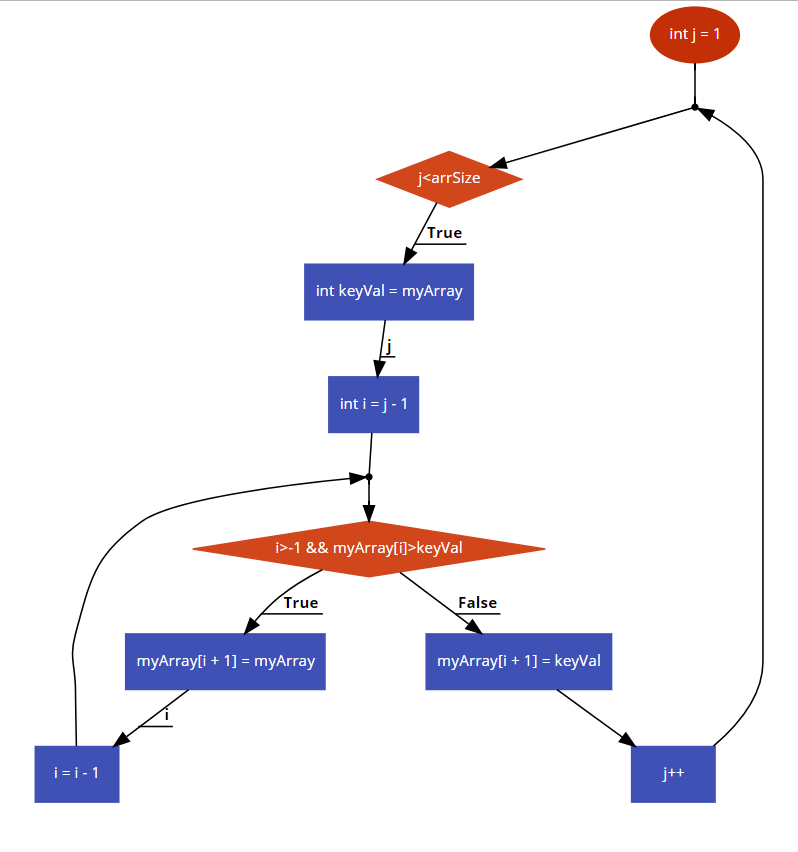
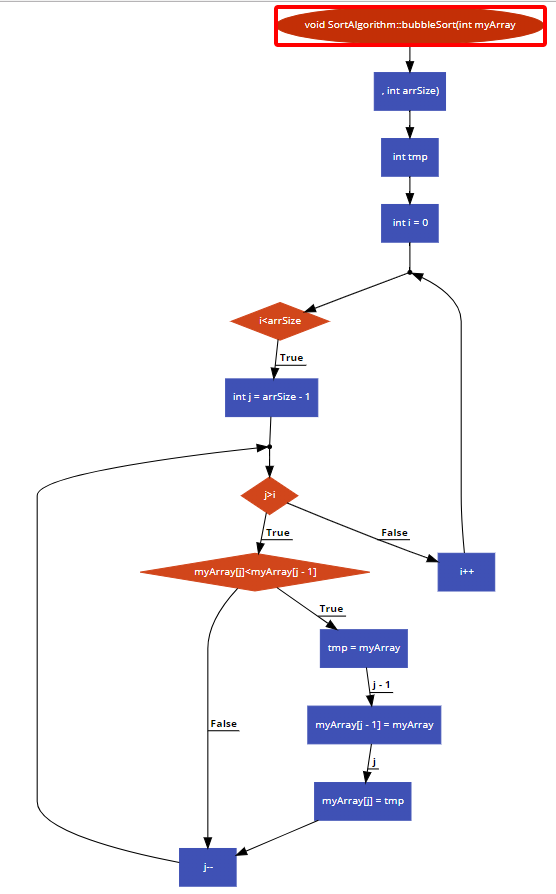
Quick sort, it breaks the original list into two partitions based on the value of a pivot value. One partition will eventually contain all the elements with values greater than the pivot value. The other will eventually contain all the elements with values less than or equal to the pivot value. The process is then repeated on each partition until the whole array is sorted. It is also a recursive algorithm as the algorithm calls itself.

## Class Diagram

These are the methods which will be implemented in the main program. What each of these methods do, is shown in the explanation after the pseudocode above. All the methods shown for the class, will be used to create the program and makes it possible for the algorithms to run.

For the program to run smoothly, classes will be used with functions. The class, SortingAlgorithm, must contains different methods which make each sort algorithm work. However, with this program it is possible to do it without classes as only functions can be used to build the program.

|  |
| --- |
| SortingAlgorithm |
|  |
| bubbleSort ()  insertionSort ()  mergeSort ()  mergePart1 ()  mergePart2 ()  quickSort ()  partition ()  quickSortPart2 () |

Flow chart for – Bubble Sort Flow chart for – Insertion Sort

## 

## Key Variables and Data Structures

### Bubble Sort

Key Variables and Data Structures for the Bubble sort algorithm:

* int myArray[] – the array that will be used for the program
* int arrSize – Uses the size of the whole array to be used in the program, helps find out when its on the last element and the bubbleSort algorithm has finished sorting.
* tmp – which stores the array getting sorted temporarily, and once its finished it stores the sorted array into this variable.

### Merge Sort

Key Variables and Data Structures for the Merge sort algorithm:

* int myArray[] – the array that will be used for the program
* int leftMostVal – this in other words is the first element of the array
* int middleVal – this is the middle value of the array, which helps with the division
* int rightMostVal – this is the last element of the array
* int n1, n2 – these are the lengths of the two sub lists. We find this so we know how long our two arrays need to be.
* leftList[] – From those two arrays you get a leftList[] when it is divided.
* rightList[] – From the two arrays you get a rightList[] when it is divided.
* largestVal – Create a large value for example, infinity, which is bigger than the largest element in the array. This will help the code identify that the array has been sorted and all integers in the array are sorted.

### Insertion Sort

Key Variables and Data Structures for the Insertion sort algorithm:

* int myArray[] – the array that will be used for the program
* int arrSize – Uses the size of the whole array to be used in the program, helps find out when its on the last element and the insertionSort algorithm has finished sorting.

### Quick Sort

Key Variables and Data Structures for the Quick Sort algorithm:

* int myArray[] – the array that will be used for the program
* int leftMostVal – this in other words is the first element of the array
* int rightMostVal – this is the last element of the array
* int middleVal – this is the middle value of the array, which helps with the division
* int tmpExchange – Stores the sorted array temporarily until all of it is sorted.
* int pivotVal - Quicksort works by partitioning the array based on a 'pivotVal' element, everything to the right of it are greater than or equal to the pivot, everything smaller than the pivot are moved to the left. Then this is repeated until the array is sorted.
* int indexVal – Used for comparison. There is an index value which will be used to store the sorted array temporarily.

### Main Function

This function will use the class and its different functions to workout the timing of the four different sorting algorithms. It should record the timing at the start of each item in an array to be sorted and once it has finished being sorted it should show the time. This should happen for each item of each sort algorithms array. It is important that the same array is printed four times so that it can be used individually for each algorithm. The array will be designed so that it creates the index and item in the array automatically given max elements in array, the number of points the array needs to reset as well as a value (100) which it will be divided by to give an average. This can be done by taking several elements from a list. Which is then used to sort the random arrays. The array size will be averaged as told earlier over one hundred instances. The times for an individual run for each sorting algorithm will be declared as a variable. The times are then added to temporary timers for each sorting algorithms. These times will then be added over the one hundred instances and then the average is found by dividing this number by hundred. It is then added to the time lists for each algorithm. Afterwards time taken will be found in microseconds for each algorithm. In the end it should be able to generate a unique file each time the program is run. This will be done by adding date and time to the file name. Therefore, every time the program is run there is a new name for each new file. A .CSV file should be created containing the sort algorithm names and their times for each item in the array. Once the .CSV file is made, a graph can be made through excel by selecting the data.

# Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Action | Purpose | Predicted Outcome | Actual Outcome | PASS/FAIL |
| 1 | Build the program in C++ | To make sure it successfully builds the program. | The program will successfully build without any errors |  | PASS |
| 2 | Running the program through the Local Windows Debugger in Visual C++ | Once the program has successfully been built. It needs to be run. This will help find out if it runs. | The program will run, and a console window will open. |  | PASS |
| 3 | To check whether a .CSV file is generated after running the program. | After the program has finished running. It is supposed to create a .CSV file. From which you can further analyse the data. | A .CSV file will be created. It should have the same name as shown in the console window. |  | PASS |
| 4 | To check if data has been written in the .CSV file | Data is supposed to be written in the .CSV file. This will help check if that has happened. | The .CSV file should have sensible data written inside it. |  | PASS |
| 5 | To check if the file runs on a new computer | Build the file. See if you get any error on other computers. | The file should successfully build. | Got a CRT\_SECURE\_NO\_WARNINGS, I researched and there were a couple of ways to resolve this. In order to resolve it I added this to my code.  #ifdef \_MSC\_VER  #define \_CRT\_SECURE\_NO\_WARNINGS  #endif  This resolved the situation. And ran the file without any problem. | FAIL |
| 6 | To check if the file runs on a new computer | Build the file. See if you get any error on other computers. | The file should successfully build. |  | FAIL |

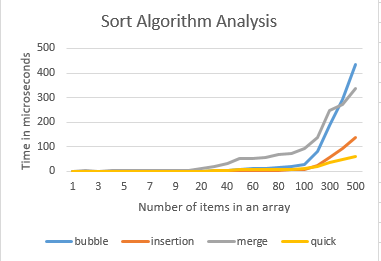
How to resolve the failed file:

1. Mark all the desired projects in solution explorer.  
   2. Press Alt-F7 or right click in solution explorer and select "Properties"  
   3.Configurations: All Configurations  
   4.Click on the Pre-processor Definitions line to invoke its editor  
   5.Choose edit...  
   6.Copy "\_CRT\_SECURE\_NO\_WARNINGS" into the Pre-processor Definitions white box on the top.

How to open the .CSV file:

It should be in the Sort Analysis Toolkit folder.

Overall, I have successfully met the success criteria for the program. However there were many things which could have been done differently. One of those things are the ways I have implemented the array. An array could have been exported from a text file and then been used in the main program. Moreover, a user input could have been created for the size of the array and then the elements in the array could have been done randomly in C++. However, I chose not to do this, due to people’s understanding of Sort Algorithms. If they were to select the size of the array, they would not see the potential results or the analysis of the sort algorithms. This is because each algorithm works differently with time and size of the array. From the .CSV file created from the actual code, you can clearly see how each algorithm works. With an array of under 500 elements, it is sensible to use bubble sort and insertion sort. However, once it’s over 500 elements, it is visible from the results that these two algorithms take really long to work. In my code, I have tried both ways and have commented out the code which tested bubble and insertion sort over 500 elements. From that it was evident that, a lot of memory was used. As it reached memory space nearly to 20mb. This is not efficient, which is why I stopped the function for bubble and insertion sort once over 500 elements are tested. This has also been explained through the comments in the source code.

By looking at the success criteria shown earlier, I can see that, I have successfully created a program that works out time each algorithm takes to sort the array. I have successfully built the program in Visual C++, this is tested in the testing section. The program has successfully ran in C++ which can also be seen in the testing. A .CSV file was also created showing a table of all the time taken for each element in the array. In that .CSV file I was successfully able to create a line graph from which you can easily analyse the data.

From running the program several times and making the graph a few times. From the screenshot, it is visible that bubble sort takes the longest to sort a 500 elements array. After that comes merge sort, this sort would work faster when the array is nearly sorted. It is also efficient to use the array when it’s over 500 items. It is also visible that quick and insertion sort are the fastest, when it comes to the array being 500 items.