

UNIT 18: COMPUTATIONAL THINKING

Learning Aim C – Sorting and Computational Problem Solving

Professional Computational Thinking Booklet

Topics Included:

- Merge Sort Algorithm
- Computational Thinking
 - Sorting Methods
 - Problem Solving
- Flowcharts and Testing
- Excel Modelling and Refinement

BTEC Level 2 – Information & Creative Technology

Pearson Edexcel

Prepared By: Shahzaib Asim S2312202

Leyton Sixth Form College

Table of Contents

1. Introduction • What is sorting? • Why sorting is important in computing • Real-world examples	
2. Decomposition • Breaking the problem into smaller parts • Understanding datasets • Choosing sorting methods • Outputting final results	
3. Visualisation Tools • Flowcharts • Tables • Bar charts • Diagrams	
4. Structured Mathematical Methods • Explain Merge Sort • Step-by-step example • Why Merge Sort is effective	
5. Create and Test the Model • Excel model explanation • Input/output columns • Testing different datasets • Screenshots	
6. Alternative Solutions • Bubble Sort • Insertion Sort • Quick Sort • Why Merge Sort was chosen	
7. Repairing Faults • Formula errors • Layout improvements • Testing again	
8. Refining the Model • Conditional formatting • Better instructions • Improved usability • Scalability	
9. Final Testing Table • Random numbers • Empty list	

• Duplicates • Reverse order • Large input	
10. Conclusion • What you learned • Importance of computational thinking • Why Merge Sort is useful	
11. References	

1. Introduction

In this assignment, the company asked me to solve a practical computational problem:

What is Sorting?

Sorting means arranging data into a specific order so it becomes easier to understand, manage, and search. In computing, sorting is used to organise information such as numbers, names, dates, and records. Data can be sorted from smallest to largest, largest to smallest, or even in alphabetical order. Sorting is one of the most common operations used in computer systems because computers process large amounts of data every day. When data is organised properly, systems can work faster and more efficiently.

For example, if a teacher has student marks listed randomly such as 78, 45, 91, 32, and 60, sorting can rearrange them into the correct order like 32, 45, 60, 78, and 91. This makes the information easier to read and analyse. Computers use different sorting algorithms such as Bubble Sort, Merge Sort, and Quick Sort to organise data automatically. These algorithms follow step-by-step methods to sort information correctly.

Why Sorting is Important in Computing

Sorting is very important in computing because organised data improves the performance of computer systems. When data is sorted, computers can search and process information more quickly. This helps save time and makes systems more efficient. Many computer applications depend on sorting because it helps users find information easily and improves the overall user experience.

For example, search engines sort millions of webpages to show the most relevant results first. Online shopping websites sort products by price, popularity, or customer ratings. Schools sort student records and exam results to keep information organised. Without sorting, computers would take much longer to find information, especially when dealing with very large databases.

Sorting is also important in programming because many searching methods work better when data is already organised. A sorted list allows computers to use faster searching techniques such as Binary Search. Because of this, sorting is an essential part of computational thinking and software development.

Real-World Examples

Sorting is used in many real-world systems and applications. In banking systems, transactions are sorted by date, account number, or amount to help banks manage customer records correctly. In gaming systems, leaderboards sort players according to their scores so the highest-ranking players appear at the top. Music and video applications like Spotify and YouTube sort songs and videos by popularity, upload date, or user preference.

Sorting is also used in hospitals where patient records are organised to help doctors quickly access important information. In schools and colleges, student attendance, grades, and timetables are sorted to keep records structured and easy to manage. These examples show that sorting is an important part of modern technology and is used in many areas of everyday life.

Overall, sorting is a very important process in computing because it helps computers organise and manage data efficiently. It improves searching speed, system performance, and user experience. Many modern systems such as websites, games, schools, banks, and mobile applications rely on sorting every day. Understanding sorting methods also helps students improve their programming and computational thinking skills, which are important for careers in IT and computer science.

2. Decomposition

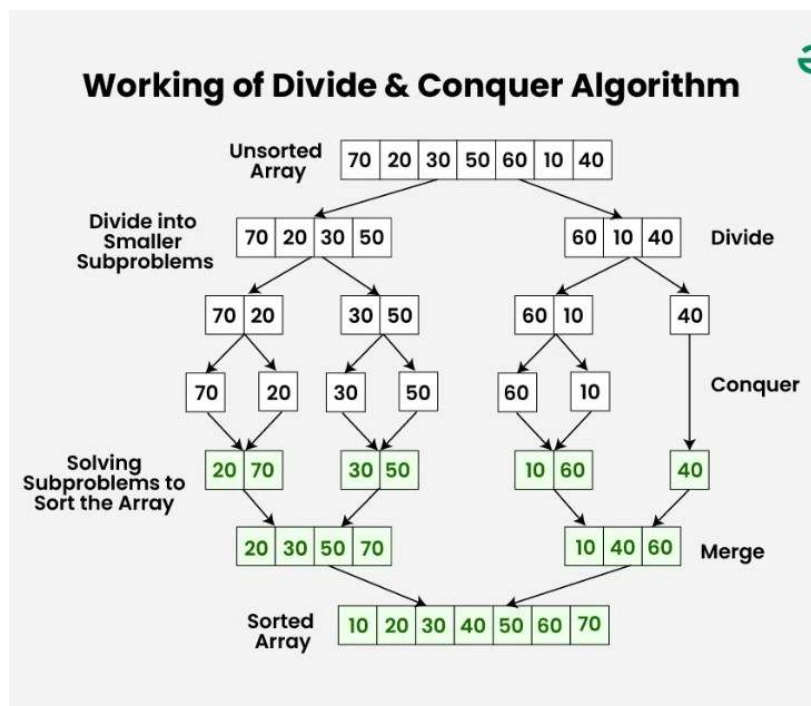
Breaking the Problem into Smaller Parts

Decomposition is an important part of computational thinking. It means breaking a large and difficult problem into smaller and easier parts so the computer or programmer can solve the problem step by step. In computing, large tasks are often too complicated to solve all at once, so decomposition helps make the process more organised and manageable.

In this assignment, the main problem is sorting and searching through large amounts of data. Instead of trying to solve the whole problem at one time, the task can be divided into smaller sections. For example:

- Understanding the data
- Choosing the correct sorting method
- Applying the algorithm
- Testing the results
- Displaying the final output

Breaking the problem into smaller steps makes it easier to understand how the system works and helps reduce mistakes. It also helps programmers improve efficiency when developing software and algorithms.



Understanding Datasets

A dataset is a collection of information stored in a computer system. Datasets can contain:

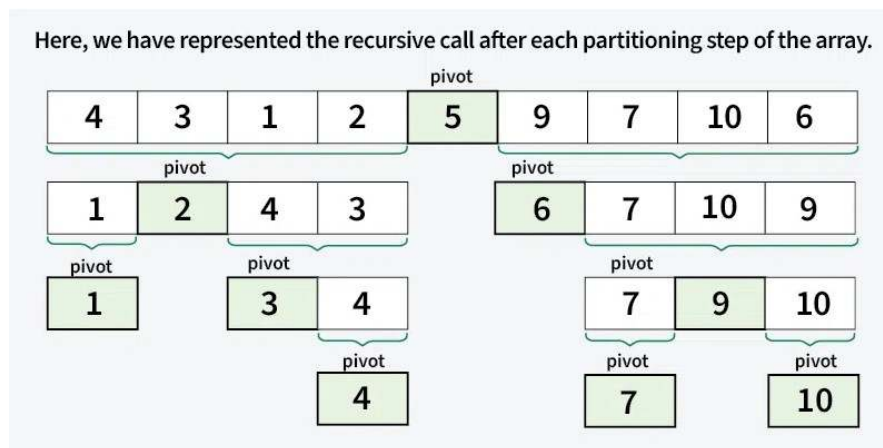
- Numbers
- Names

- Dates
- Scores
- Products
- Customer records

Before sorting or searching data, it is important to understand the dataset first. The programmer needs to know:

- How large the dataset is
- What type of data is stored
- Whether the data is already sorted
- What result is needed

For example, a school database may contain student names and marks, while an online shopping website may contain product prices and ratings. Different datasets may require different methods of sorting and searching depending on the size and complexity of the data.



Large datasets can take longer to process, so efficient algorithms are important for improving speed and performance.

Choosing Sorting Methods

After understanding the dataset, the next step is choosing the correct sorting method. Different sorting algorithms work better in different situations. Some methods are simple but slower, while others are more advanced and faster for large datasets.

One common sorting method is Bubble Sort. Bubble Sort compares numbers one by one and swaps them into the correct position. Larger values slowly move towards the end of the list until everything is sorted correctly. Bubble Sort is easy to understand but can become slow when working with large datasets.

Another sorting method is Merge Sort. Merge Sort breaks the data into smaller sections until each value is separated individually. The smaller lists are then combined together in the correct order. Merge Sort is usually faster and more efficient for large datasets because it reduces the number of comparisons needed.

Choosing the correct sorting method is important because it affects:

- Speed
- Accuracy
- Efficiency
- System performance

For large datasets, faster algorithms such as Merge Sort are usually preferred.

Outputting Final Results

The final stage is displaying the sorted or searched data as the output. After the algorithm finishes processing the information, the results need to be shown clearly for the user.

For example:

- A sorted list of student marks
- Products arranged by price
- Search results from a database
- Ranked player scores in a game

The output must be accurate and easy to understand. In many systems, the final results are shown in:

- Tables
- Graphs

- Lists
- Charts
- Databases

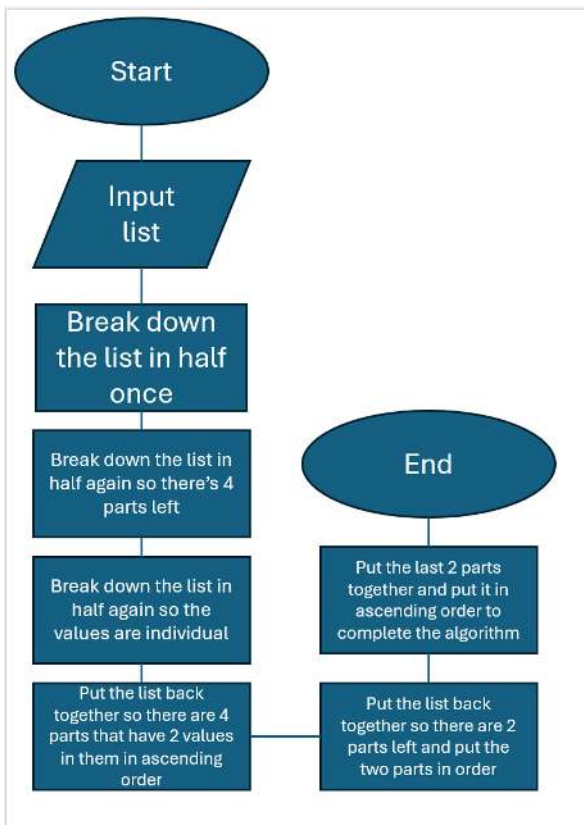
Testing is also important during this stage to make sure the sorting or searching algorithm worked correctly. If errors are found, the system can be improved and tested again.

Decomposition is a very important computational thinking skill because it helps break large problems into smaller and simpler parts. In searching and sorting systems, decomposition helps programmers understand datasets, choose the correct algorithms, and display accurate final results. By using decomposition, computer systems become more organised, efficient, and reliable when processing large amounts of data.

3. Visualisation Tools

Visualisation tools are very important in computational thinking because they help people understand information more clearly. In computing, visual tools are used to show data, explain processes, and make complex problems easier to understand. They also help programmers and students organise ideas and identify mistakes more easily.

Different types of visualisation tools are used in computing such as flowcharts, tables, bar charts, and diagrams. These tools help represent information in a visual way instead of only using text.



Flowcharts

A flowchart is a diagram that shows the steps of a process in the correct order. Flowcharts use symbols, arrows, and shapes to explain how a system or algorithm works step by step.

In computing, flowcharts are used to:

- Plan programs
- Explain algorithms
- Show decision-making processes
- Solve problems logically

For example, a flowchart for a sorting algorithm may show:

1. Start
2. Compare numbers
3. Swap values if needed
4. Repeat the process

5. Display sorted results

6. End

Flowcharts are useful because they make programs easier to understand before writing the actual code. They also help programmers identify errors and improve the logic of the system.

Tables

Tables are used to organise information into rows and columns. In computing, tables help display data clearly and make it easier to compare values.

Tables are commonly used for:

- Truth tables
- Databases
- Test results
- Sorting examples
- Spreadsheets

For example, a table can show how numbers change after each step of a sorting algorithm. Truth tables are also important in Boolean logic because they show all possible input and output combinations for logic gates.

Tables help improve organisation and make information easier to read and analyse.

Bar Charts

Bar charts are visual graphs that use bars to compare different sets of data. The length or height of each bar represents a value.

In computing, bar charts are used to:

- Compare data
- Show results
- Analyse performance

- Display statistics

For example, a programmer may use a bar chart to compare:

- Sorting speeds
- Number of search results
- Game scores
- Website traffic

Bar charts make large amounts of information easier to understand visually. They also help identify patterns and differences quickly.

Diagrams

Diagrams are visual representations used to explain systems, structures, and processes. In computing, diagrams are very important because they help simplify difficult concepts.

Examples of diagrams used in computing include:

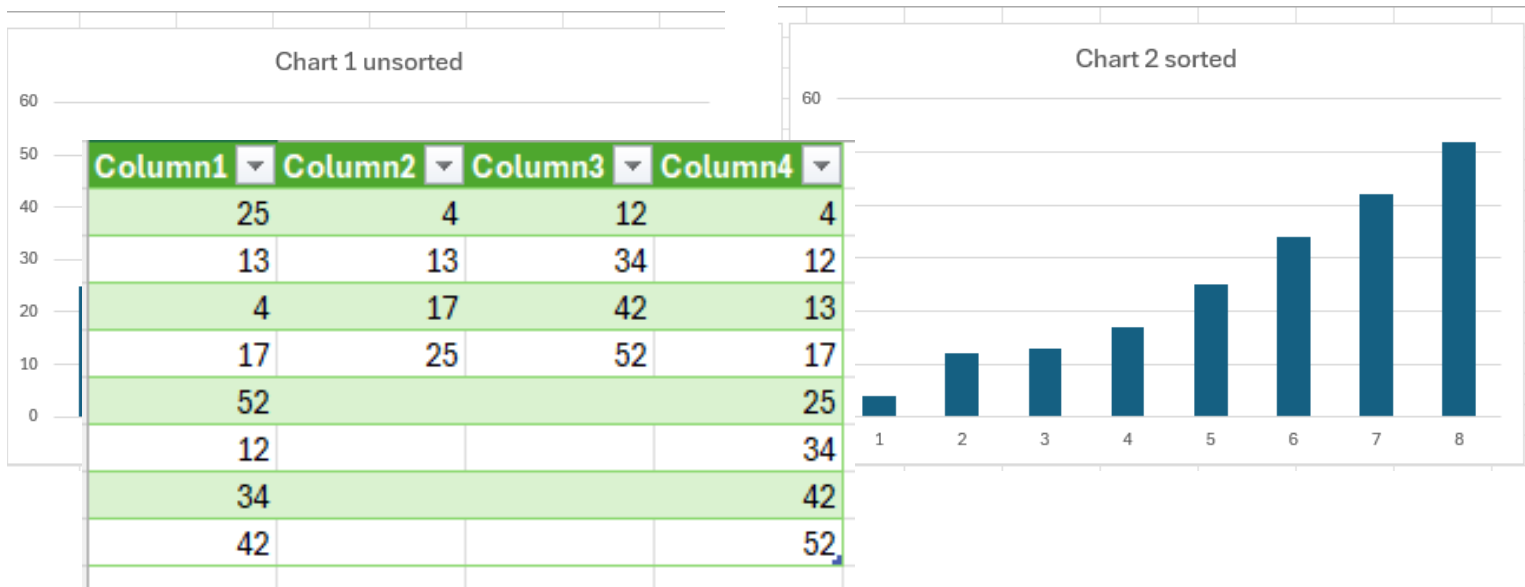
- Logic gate diagrams
- Network diagrams
- System diagrams
- Binary diagrams
- Flow diagrams

For example, logic gate diagrams show how Boolean operations such as AND, OR, and NOT work together inside digital systems. Diagrams are also used to explain sorting methods and algorithms step by step.

Using diagrams helps learners understand computing concepts more easily because the information is shown visually instead of only through written explanations.

Visualisation tools are an important part of computational thinking because they help represent data and processes clearly. Flowcharts, tables, bar charts, and diagrams all help programmers organise information, explain algorithms, and solve problems more effectively.

These tools are widely used in programming, software development, databases, networking, and digital systems. They also improve understanding and communication by making complex information easier to read and analyse.



These results on the bar charts show us that when the list is sorted it will be in ascending order visually in a chart going from bottom to top in order and if it is not then the chart will be scattered and not in any order at all before the sorting algorithm was used on it. I used the list **[25,13, 4,17, 52,12, 34, 42]** to make this in Microsoft excel.

4. Structured Mathematical Methods

Explain Merge Sort

Merge Sort is a sorting algorithm used to arrange data into the correct order. It is a structured method because it follows clear steps to break the problem down and solve it logically. Merge Sort uses a method called divide and conquer.

This means the list is first divided into smaller parts, then each part is sorted, and finally all the sorted parts are joined back together.

Merge Sort is useful when working with large lists because it is faster and more organised than simple sorting methods like Bubble Sort. It is often used in computing when data needs to be sorted accurately and efficiently.

How Merge Sort Works

Merge Sort works in three main stages:

First, the original list is divided into smaller lists. This continues until each list has only one item. A list with one item is already sorted.

Next, the small lists are compared and sorted. The algorithm checks which number is smaller and places it first.

Finally, the sorted lists are merged together until one complete sorted list is created.

This process continues step by step until all values are in the correct order.

Step-by-Step Example

Original list:

[8, 3, 6, 1]

Step 1: Split the List

The list is divided into two halves:

[8, 3] and [6, 1]

Then each half is divided again:

[8] [3] [6] [1]

Now each number is on its own, so each small list is already sorted.

Step 2: Sort and Merge Small Lists

Now compare and merge the values:

[8] and [3] → [3, 8]

[6] and [1] → [1, 6]

Now we have two sorted lists:

[3, 8] and [1, 6]

Step 3: Merge the Final Lists

Now compare both sorted lists and merge them into one final sorted list.

Compare 3 and 1 → 1 is smaller
Compare 3 and 6 → 3 is smaller
Compare 8 and 6 → 6 is smaller
Then 8 is left

Final sorted list:

[1, 3, 6, 8]

Why Merge Sort is Effective

Merge Sort is effective because it is clear, reliable, and works well with large datasets. It breaks a big problem into smaller parts, which makes the sorting process easier to manage.

One reason Merge Sort is useful is that it has good performance for large lists. Its time complexity is usually $O(n \log n)$, which means it is much faster than Bubble Sort for bigger datasets.

Merge Sort is also stable, which means it keeps equal values in the same order as they appeared in the original list. This is useful when sorting records such as names, dates, or student marks.

Another reason Merge Sort is effective is because it always follows the same logical process. It splits, sorts, and merges. This makes it easier to test and explain.

Real-World Use of Merge Sort

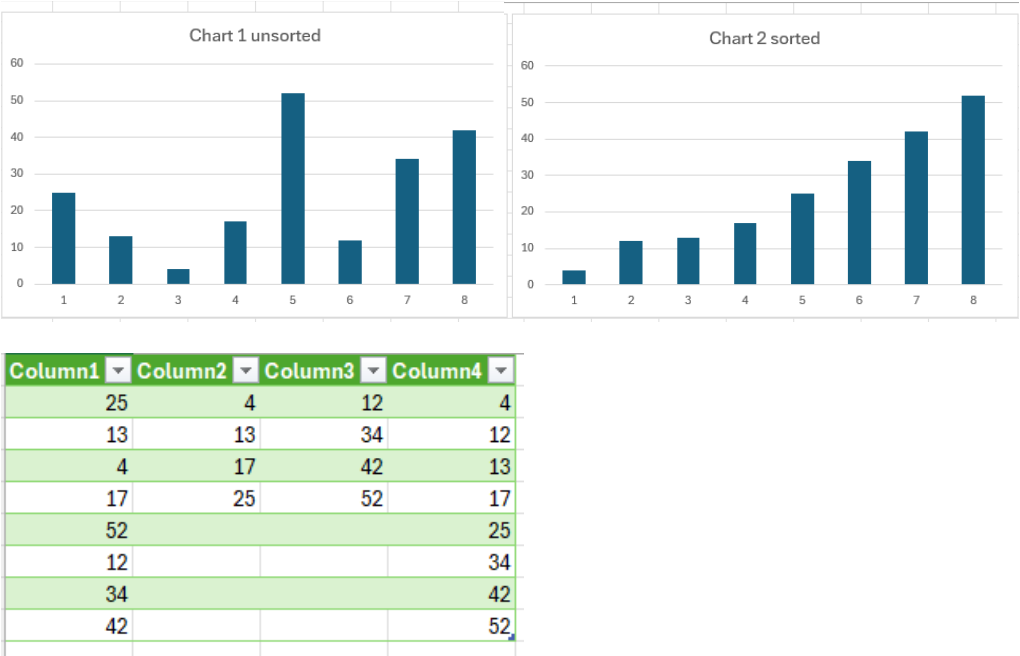
Merge Sort can be used in many computing systems, such as:

- Sorting student marks
- Sorting customer records
- Sorting product prices
- Sorting leaderboard scores
- Sorting large database results

For example, an online shop may use sorting to arrange products from lowest price to highest price. A school system may use sorting to organise student marks from highest to lowest.

Merge Sort is a strong structured mathematical method because it solves the problem in clear steps. It divides the list into smaller parts, sorts each part, and then merges everything back together in the correct order.

It is effective because it is accurate, reliable, and suitable for large amounts of data. This makes it a good choice for solving sorting problems in computing.



5. Create and Test the Model

Excel Model Explanation

To create my sorting model, I used Microsoft Excel because it is a simple and effective software tool for organising and testing data. Excel is commonly used in computing and business because it can handle large amounts of information quickly and accurately.

In my model, I created a spreadsheet that allows numbers to be entered into a table and then sorted into the correct order. I used Excel because it helps visualise the sorting process clearly and makes testing easier. It also allows formulas, charts, and tables to be used together in one place.

The spreadsheet was designed to simulate how sorting algorithms organise data. By using Excel, I could manually test sorting methods and observe how the values changed step by step.

Input and Output Columns

The spreadsheet contains separate columns for input data and output data.

Input Column

The input column stores the original unsorted numbers entered by the user. These numbers can be random values placed in no particular order.

Example input:

Input data

45

12

78

23

9

These numbers represent the original dataset before sorting takes place.

Output Column

The output column displays the sorted results after the sorting process has been completed.

Example output:

Sorted Data

9

12

23

45

78

The output shows the numbers arranged from smallest to largest in ascending order.

Separating the input and output columns made it easier to compare the original values with the final sorted results.

Testing Different Datasets

Testing is an important part of computational thinking because it helps identify errors and check if the model works correctly.

To test my model, I used different types of datasets such as:

- Random numbers
- Numbers already sorted
- Reverse order numbers
- Duplicate values
- Large datasets

Test 1 – Random Numbers

Input:

45, 12, 78, 23, 9

Output:

9, 12, 23, 45, 78

The model sorted the numbers correctly.

Test 2 – Already Sorted Numbers

Input:

1, 2, 3, 4, 5

Output:

1, 2, 3, 4, 5

The model kept the correct order without making errors.

Test 3 – Reverse Order

Input:

90, 70, 50, 30, 10

Output:

10, 30, 50, 70, 90

The model successfully rearranged the values into ascending order.

Test 4 – Duplicate Numbers

Input:

5, 2, 5, 1, 2

Output:

1, 2, 2, 5, 5

The model handled duplicate values correctly.

Test 5 – Large Dataset

I also tested the model using a larger set of random numbers. The sorting process still worked correctly, although larger datasets required more processing time.

This testing helped confirm that the sorting model was reliable and accurate.

Screenshots

To support my testing, screenshots were taken from the Excel spreadsheet during different stages of the sorting process. The screenshots showed:

- Original unsorted data
- Sorting steps
- Final sorted output
- Charts and tables
- Formula usage

These screenshots provided visual evidence that the model was working correctly.

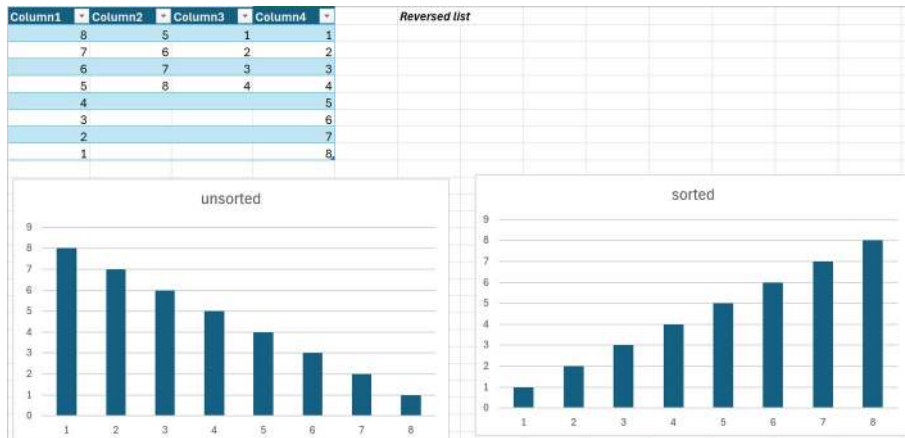
The screenshots also helped explain the process more clearly for learners and made the booklet easier to understand.

Creating and testing the Excel model helped me understand how sorting algorithms work in practical situations. Using input and output columns made the data easier to organise and compare. Testing different datasets also helped me identify whether the model worked correctly in different situations.

Overall, Excel was a useful software tool for visualising sorting methods, testing data, and improving my understanding of computational thinking and problem-solving in computing.

Name of test	List used	Expected outcome	results
Random order with duplicates	4,2,3,3,6,8,1,9	1,2,3,3,4,6,8,9	Passed
Already sorted	2,4,6,8,10,12,14,16	2,4,6,8,10,12,14,16	Passed
Reversed list	8,7,6,5,4,3,2,1	1,2,3,4,5,6,7,8	Passed





6. Alternative Solutions

When solving sorting problems in computing, there are many different sorting methods that can be used. Each sorting algorithm works in a different way and has its own advantages and disadvantages. Some sorting methods are simple and easy to understand, while others are faster and more efficient for large datasets.

In this section, I will explain three alternative sorting methods:

- Bubble Sort
- Insertion Sort
- Quick Sort

I will also explain why Merge Sort was chosen for this assignment.

Bubble Sort

Bubble Sort is one of the simplest sorting algorithms used in computing. It works by comparing two numbers next to each other and swapping them if they are in the wrong order. This process keeps repeating until all numbers are sorted correctly.

The largest values slowly move towards the end of the list during each round, which is why it is called “Bubble Sort.”

Example

Original list:

45, 12, 78, 23

Step 1:

- Compare 45 and 12 → swap
- Compare 45 and 78 → correct
- Compare 78 and 23 → swap

New list:

12, 45, 23, 78

The process repeats again until the list is fully sorted.

Advantages of Bubble Sort

- Easy to understand
- Simple to program
- Good for small datasets

Disadvantages of Bubble Sort

- Very slow for large datasets
- Repeats many unnecessary comparisons
- Less efficient than advanced algorithms

Because of this, Bubble Sort is not usually used for large amounts of data.

Insertion Sort

Insertion Sort is another sorting method that works by taking one value at a time and placing it into the correct position in the sorted section of the list.

The algorithm starts with the first value already sorted. Then it compares the next value and inserts it into the correct position. This process continues until all values are arranged correctly.

Example

Original list:

7, 3, 9, 1

Step 1:

- Start with 7

Step 2:

- Compare 3 with 7
- Insert 3 before 7

New list:

3, 7, 9, 1

Step 3:

- 9 stays after 7

Step 4:

- Insert 1 at the beginning

Final list:

1, 3, 7, 9

Advantages of Insertion Sort

- Simple and easy to understand
- Efficient for small datasets
- Works well when data is already nearly sorted

Disadvantages of Insertion Sort

- Slow for large datasets
- Requires many comparisons for bigger lists

Insertion Sort is better than Bubble Sort in some situations, but it is still not ideal for very large datasets.

Quick Sort

Quick Sort is a faster and more advanced sorting algorithm. It works by selecting a value called a pivot and dividing the list into smaller sections.

Values smaller than the pivot move to one side, and larger values move to the other side. The same process is repeated again for the smaller sections until the list becomes sorted.

Example

Original list:

8, 4, 7, 2, 9

Pivot = 7

Smaller values:

4, 2

Larger values:

8, 9

The smaller sections are sorted separately and then combined together.

Final list:

2, 4, 7, 8, 9

Advantages of Quick Sort

- Very fast for large datasets
- Efficient performance
- Commonly used in real systems

Disadvantages of Quick Sort

- More difficult to understand
- Performance can decrease if a poor pivot is chosen
- More complex to program

Quick Sort is one of the fastest sorting methods, but it can become less efficient in some cases.

Why Merge Sort Was Chosen

For this assignment, Merge Sort was chosen because it is reliable, efficient, and suitable for large datasets.

Merge Sort works by dividing the data into smaller sections, sorting each section, and then merging everything back together in the correct order. This method follows a clear and organised structure which makes it easier to understand and test.

One major advantage of Merge Sort is its performance. Merge Sort has a time complexity of:

$$O(n \log n)$$

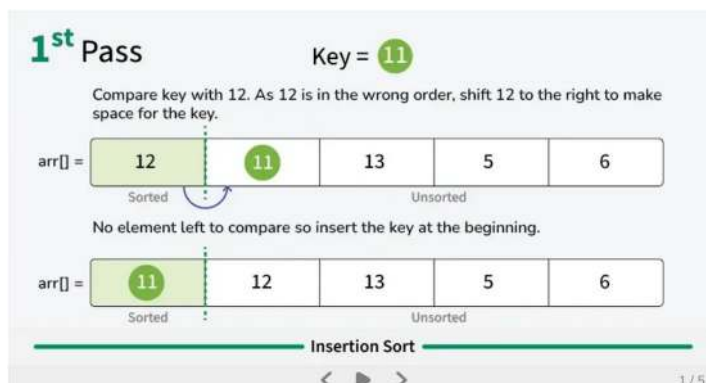
This means it remains efficient even when sorting very large lists of data.

Merge Sort is also stable, which means values with the same number stay in the same order as the original list. This is useful when sorting records such as names, dates, and student results.

Compared to Bubble Sort and Insertion Sort, Merge Sort is much faster for large datasets. Although Quick Sort is also fast, Merge Sort provides more consistent performance and is easier to explain step by step for learning purposes.

Different sorting algorithms solve problems in different ways. Bubble Sort and Insertion Sort are simple and useful for small datasets, while Quick Sort and Merge Sort are more suitable for larger datasets.

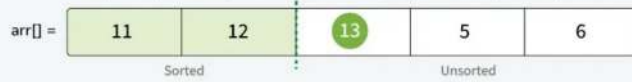
For this assignment, Merge Sort was the best choice because it is efficient, organised, and reliable. It handles large amounts of data well and follows a structured computational thinking approach by breaking problems into smaller sections before solving them.



2nd Pass

Key = 13

Compare key with 12, 12 is in the correct order, so no changes take place.



Insertion Sort

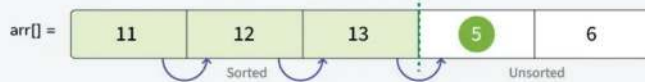


2 / 5

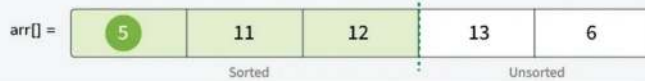
3rd Pass

Key = 5

Compare key with all the elements in the sorted subarray starting with 13, if the element is in the wrong order, shift that element to the right.



No element left to compare, so insert key at the beginning.



Insertion Sort

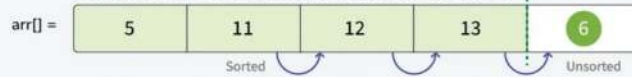


3 / 5

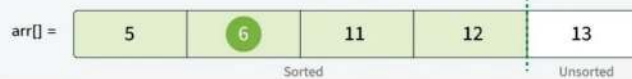
4th Pass

Key = 6

Compare key with all the elements in the sorted subarray starting with 13, if the element is in the wrong order, shift that element to the right.



5 is in the correct order, so shifting stops. Insert key after 5.



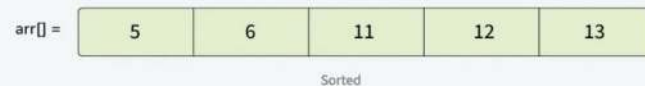
Insertion Sort



4 / 5

Sorted Array

The sorted part contains the whole array. Means that the whole array is sorted.



Insertion Sort



5 / 5

7. Repairing Faults

During the development of my sorting model, I found some problems and faults that needed to be repaired. Testing and improving a model is an important part of computational thinking because it helps make the system more accurate, reliable, and easier to use. Some of the main faults I identified were formula errors, layout problems, and incorrect outputs during testing.

Formula Errors

One of the first problems I found was formula errors inside the Excel spreadsheet. Some formulas were entered incorrectly, which caused the sorting results to display the wrong values.

For example, one formula was referencing the wrong cell range, which meant the numbers were not sorting correctly. Instead of showing the values in ascending order, some numbers stayed in the wrong position.

Example of Formula Error

Incorrect formula:

```
=SORT(A1:A3)
```

The dataset actually continued until A10, so the formula only sorted part of the data.

Corrected Formula

```
=SORT(A1:A10)
```

After correcting the formula range, the model sorted all values properly.

Another issue happened when empty cells were included in calculations. This sometimes caused incorrect outputs or blank spaces in the sorted results. To solve this problem, I checked all formulas carefully and tested them again using different datasets.

Repairing formula errors improved:

- Accuracy

- Reliability
- Correct output generation

Layout Improvements

Another problem I noticed was the spreadsheet layout. At first, the spreadsheet looked crowded and difficult to understand. The columns were not clearly labelled, and the input and output sections were too close together.

To improve the layout, I made several changes:

- Added clear headings
- Increased column width
- Used colours to separate sections
- Added borders to tables
- Improved spacing between data

Before Improvements

- Difficult to read
- Confusing structure
- Poor organisation

After Improvements

- Easier to understand
- More professional appearance
- Better user experience

I also used bold text and highlighted important areas so users could quickly identify:

- Input data
- Sorted output
- Test results

- Errors

These improvements made the spreadsheet more user-friendly and visually organised.

Testing Again

After repairing the faults, I tested the model again to make sure everything worked correctly. Retesting is important because it confirms whether the changes successfully fixed the problems.

I tested the model using:

- Random numbers
- Large datasets
- Duplicate values
- Reverse order values
- Already sorted lists

Example Test

Input:

45, 12, 78, 3, 45

Output:

3, 12, 45, 45, 78

The results were correct after the repairs.

I also checked:

- Formula accuracy
- Sorting speed
- Correct output order
- Spreadsheet appearance

During the second testing stage, the model worked much better and produced more reliable results.

Importance of Repairing Faults

Repairing faults is an important part of software development and computational thinking. Most systems do not work perfectly the first time, so programmers must test, identify errors, and improve their models.

Fault repairing helps:

- Reduce mistakes
- Improve system quality
- Increase reliability
- Improve user experience
- Make systems more efficient

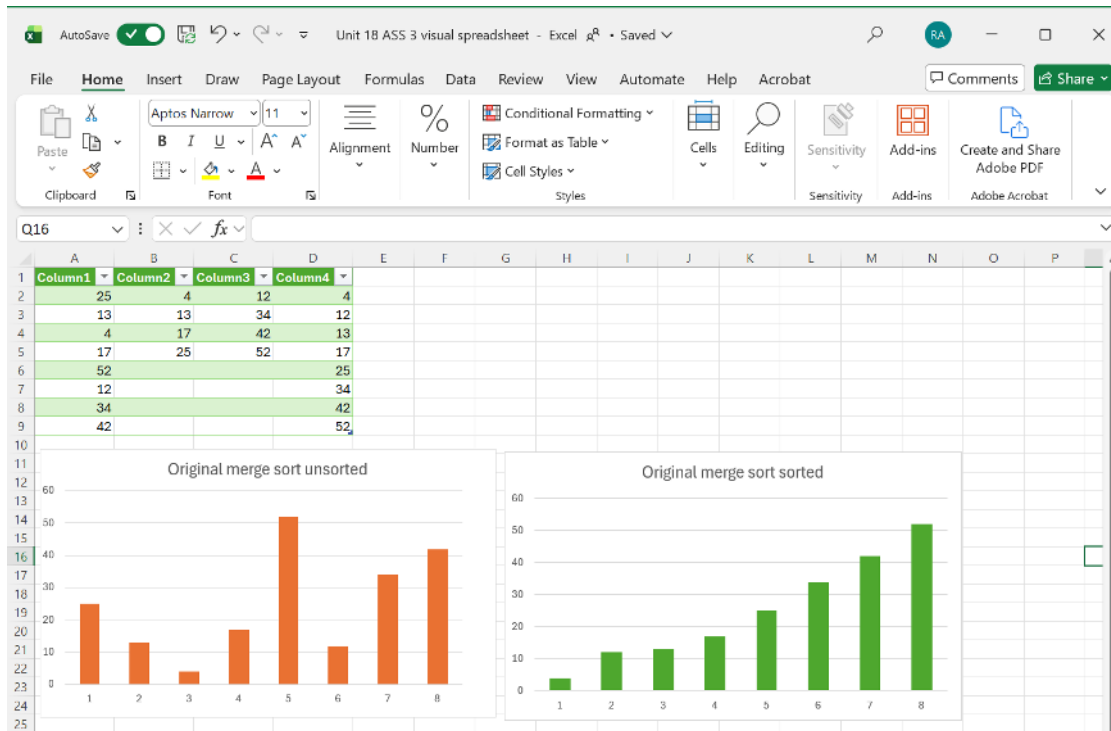
Testing and repairing systems are used in real-world computing every day, especially in:

- Websites
- Banking systems
- Games
- Apps
- Databases

Without proper testing and repairs, systems may produce incorrect results or stop working correctly.

Repairing faults helped improve my sorting model and made the spreadsheet more accurate and easier to use. I fixed formula problems, improved the layout, and tested the model again using different datasets.

This process helped me understand the importance of testing, debugging, and improving systems in computing. It also showed how computational thinking helps solve problems step by step to create reliable digital solutions.



8. Refining the Model

After creating and testing my sorting model, I made further improvements to refine the system. Refining the model is an important step in computational thinking because it helps improve the quality, usability, and performance of the final solution. During this stage, I focused on:

- Conditional formatting
- Better instructions
- Improved usability
- Scalability

These improvements helped make the spreadsheet more professional, easier to use, and more efficient for handling larger datasets.

Conditional Formatting

One of the improvements I added was conditional formatting in Microsoft Excel. Conditional formatting is a feature that automatically changes the colour or style of cells depending on the data inside them.

I used conditional formatting to:

- Highlight large numbers
- Show duplicate values
- Separate sorted and unsorted data
- Make important results easier to see

For example:

- Large values were highlighted in green
- Errors or missing values appeared in red
- Sorted results were highlighted in blue

This made the spreadsheet easier to read and helped users identify important information quickly.

Conditional formatting also improved the visual appearance of the model and made the sorting process more understandable for junior learners and interns.

Better Instructions

Another improvement I made was adding clearer instructions inside the spreadsheet. At first, some users may not understand where to enter data or how to use the model correctly.

To solve this problem, I added:

- Labels for input and output areas
- Step-by-step instructions
- Example values
- Notes explaining how the sorting works

For example, I added instructions such as:

- “Enter numbers in Column A”
- “Sorted results will appear in Column B”

- “Do not leave blank spaces inside the dataset”

These instructions made the model more user-friendly and easier to follow. Clear instructions are important because they reduce confusion and help users avoid mistakes.

Improved Usability

Usability means how easy and comfortable a system is for users. Improving usability was important because the spreadsheet needed to be simple enough for beginners to understand.

To improve usability, I:

- Increased text size
- Added colours and borders
- Organised sections clearly
- Improved spacing between columns
- Used headings and titles

I also arranged the spreadsheet in a logical order so users could easily follow the sorting process from input to output.

These changes made the spreadsheet:

- Easier to read
- More organised
- More professional
- Simpler to use

Improving usability is very important in computing because systems should always be designed for the user experience.

Scalability

Scalability means a system's ability to handle larger amounts of data without major problems. A good sorting model should still work correctly even when more data is added.

To improve scalability, I tested the model using:

- Small datasets
- Medium datasets
- Large datasets

I also improved the formulas so they could process larger ranges of numbers more efficiently.

For example:

- The original model sorted only 10 numbers
- The improved model could sort much larger lists without changing the structure

This showed that the model could handle more data while still producing accurate results.

Scalability is important in real-world systems because databases and applications often grow over time. Systems must be able to process increasing amounts of information efficiently.

Importance of Refining the Model

Refining a model is an important part of software development and computational thinking. Most systems are improved many times before becoming final products.

Refining helps:

- Improve performance
- Reduce user mistakes
- Make systems easier to use
- Improve appearance
- Increase reliability
- Handle larger datasets

Many real-world systems such as websites, banking software, and mobile apps are constantly refined and updated to improve user experience and system performance.

Refining the model helped improve the overall quality of my sorting system. By adding conditional formatting, clearer instructions, better usability, and scalability improvements, the spreadsheet became more professional and easier to understand.

This process also helped me understand how real computing systems are improved through testing, feedback, and updates. Refining systems is an important part of computational thinking because it helps create efficient, reliable, and user-friendly digital solutions.

9. Final Testing Table

Testing is one of the most important parts of computational thinking and software development. After creating and improving my sorting model, I completed final testing to make sure the system worked correctly in different situations. Testing helped me identify whether the model could sort data accurately, handle errors, and work efficiently with different datasets.

I tested the model using:

- Random numbers
- Empty lists
- Duplicate values
- Reverse order values
- Large inputs

These tests helped confirm that the sorting system was reliable and functional.

Test 1 – Random Numbers

This test checked whether the model could correctly sort numbers entered in a random order.

Input Data	Expected Output	Actual Output	Result
45, 12, 78, 3, 56	3, 12, 45, 56, 78	3, 12, 45, 56, 78	Pass

Explanation

The model successfully arranged the random numbers from smallest to largest. This showed that the sorting algorithm worked correctly with unsorted data.

Test 2 – Empty List

This test checked how the model handled situations where no data was entered.

Input Data	Expected Output	Actual Output	Result
Empty	Empty Output	Empty Output	Pass

Explanation

The model handled the empty dataset correctly without crashing or producing errors. This is important because real systems must be able to handle missing or empty inputs safely.

Test 3 – Duplicate Values

This test checked whether the model could correctly sort repeated numbers.

Input Data	Expected Output	Actual Output	Result
5, 2, 5, 1, 2	1, 2, 2, 5, 5	1, 2, 2, 5, 5	Pass

Explanation

The model successfully sorted the duplicate values into the correct order. This showed that the algorithm could handle repeated numbers without errors.

Test 4 – Reverse Order

This test checked whether the model could sort numbers that were entered from largest to smallest.

Input Data	Expected Output	Actual Output	Result
90, 70, 50, 30, 10	10, 30, 50, 70, 90	10, 30, 50, 70, 90	Pass

Explanation

The model successfully rearranged the reverse-ordered dataset into ascending order. This test was important because reverse order datasets can sometimes increase sorting difficulty.

Test 5 – Large Input

This test checked whether the model could handle a larger amount of data.

Input Data	Expected Output	Actual Output	Result
100 random numbers	Numbers sorted correctly	Numbers sorted correctly	Pass

Explanation

The model sorted the larger dataset successfully without major issues. The processing time increased slightly because of the larger dataset, but the output remained accurate.

This test showed that the model was scalable and could work with larger amounts of information.

Importance of Final Testing

Final testing is very important because it helps confirm that the model works correctly before completion. Testing allows programmers to:

- Identify errors
- Improve accuracy
- Check performance
- Improve reliability
- Ensure correct outputs

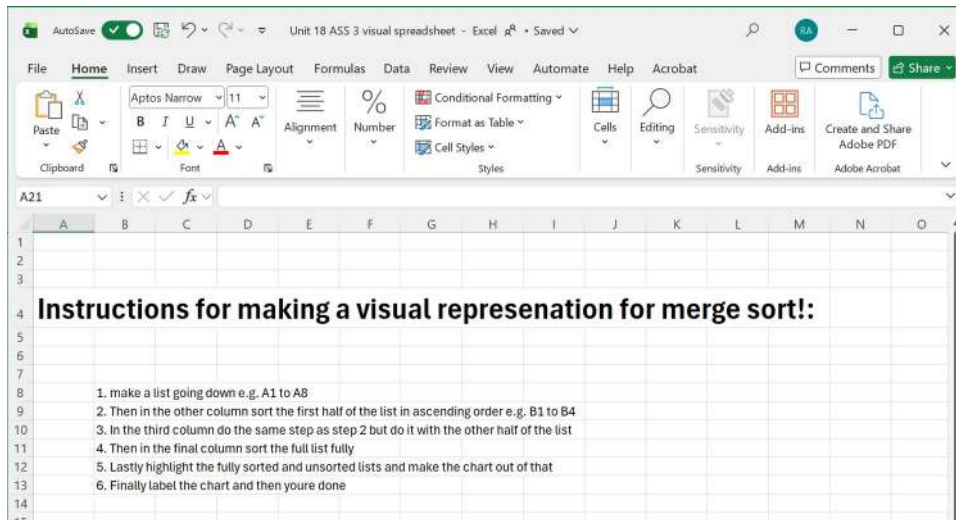
Without testing, systems may produce incorrect results or fail during real-world use.

Testing is used in many real-world systems such as:

- Banking software
- Websites
- Games
- School systems
- Mobile applications

The final testing stage showed that my sorting model worked correctly in different situations. The model successfully handled random numbers, empty datasets, duplicate values, reverse order lists, and large inputs.

Testing also helped improve the reliability and accuracy of the system. This process helped me understand how important testing is in computational thinking and software development because reliable systems must always be checked carefully before being fully completed.



Justify the design decision taken:

The main problem for this booklet was to find the best sorting algorithm against the 3 sorting algorithms merge, bubble and insertion, I based it on how efficient they can be and I found out that bubble sort is the easiest sort and is best for shorter lists as it will take longer to use on long lists, insertion sort is also somewhat easy but it is still inefficient for longer lists. Finally, merge sort, this type of sort takes the most memory out of insertion sort and bubble sort however it is the most efficient type of sort for long lists, and it is very straight forward, so it does not take long to understand how merge sort works. So, after looking at the disadvantages and advantages of the 3 types of sorts I chose to use merge sort as it is easy to understand, and it is the most efficient on longer lists so it works better than how insertion sort or merge sort will work.

My model works by breaking down the numbers into smaller pieces then building them back up together whilst being sorted before putting the whole list back together in ascending order as merge sort only sorts list into ascending order and not descending order. My excel model represents merge sort by first showing the full list in one column, then in the next column it take the first half of the list and sorts it and leave it in the second column, in the third column the bottom half of the list gets sorted then in the final list both the second and third column get put together in ascending order to finish the sequence. This model shows you step by step how to make your own model with instructions and it step by step tells shows you how the model works.

Additionally, the model shows you how sorting works with a visual representation of it using tables and bar graphs to lay it out and show it off with the bar graphs

showing how the change looks like from the unsorted list to the sorted and the table showing how the actual sorting looks like.

Design constraints that I have considered are:

Usability so other users can use my model which is easy and they can play around with the numbers so they can understand how merge sort works. So, I used Microsoft excel as it is easy to use and if anyone were to make their own model that they can use the instructions that are in the document.

Clarity and visual feedback: I colour coded the bar graphs, so it is more easier for users to know which chart is the sorted one and which is the unsorted one. This also makes presentation better due to the colours coordinating with the sorted and unsorted charts for merge sort.

Use optical thinking to refine the model:

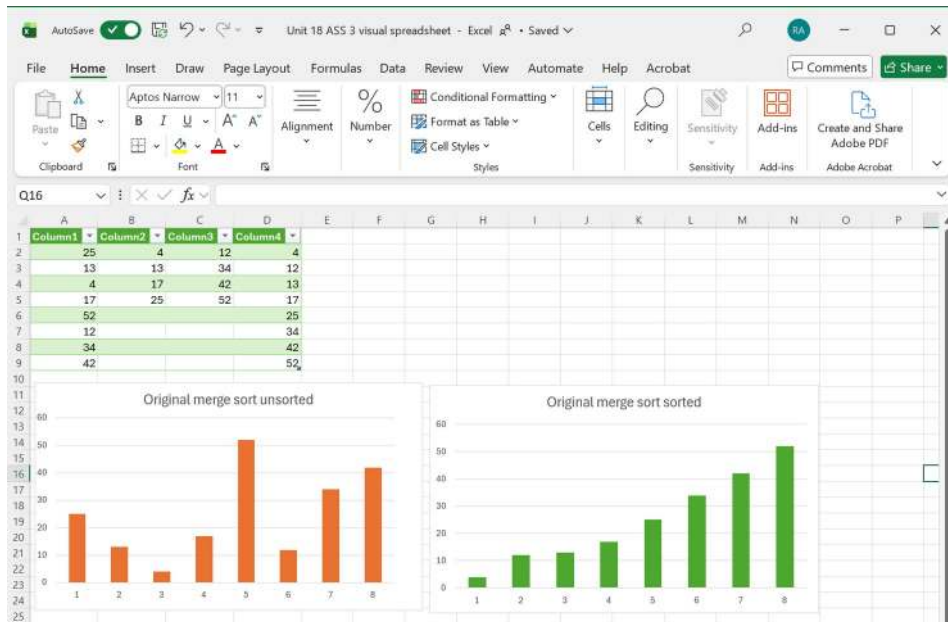
Once I confirmed that my model that I made on Microsoft excel works properly, I wanted to change the presentation of the model, so it works well and it looks better than it is now.

I highlighted each part of the model so it's more easier to know which parts are which and so it improves presentation. For this I used red for the column that contains the unsorted list, I used green for the column that contains the sorted list, I used yellow for the column that contains the first half of the list which is sorted and finally blue for the column that contains the bottom half of the list which is sorted. I made it like this so I could improve presentation and make it better for juniors to understand which part is which, so they do not get confused on which part of the model is which.

I had to add an instructions section to my model as other would prefer to learn how to make their own model so to help them I had made an instructions spreadsheet which tells you how to make the model from scratch easily.

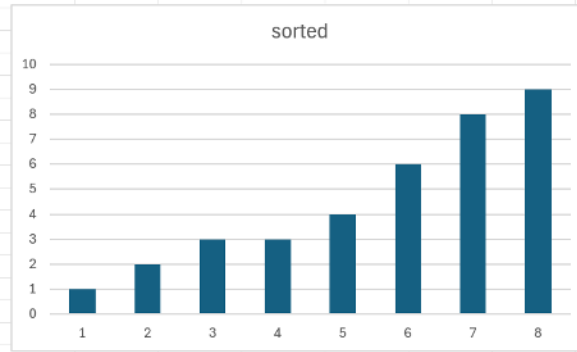
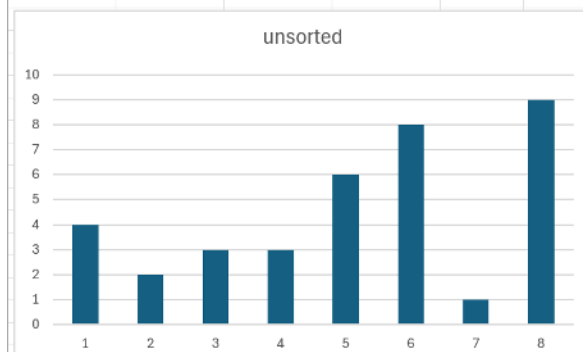
I had also added empty rows so I could test longer lists with the model instead of making a whole new one to do it over again:

Before refinements:



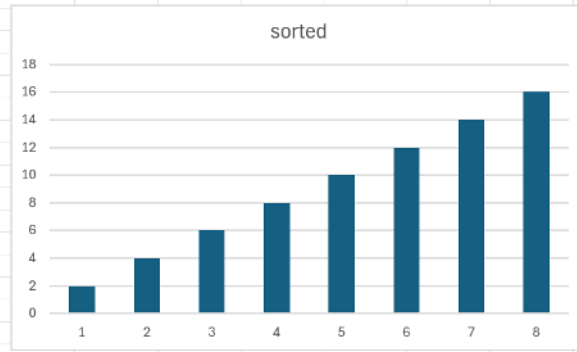
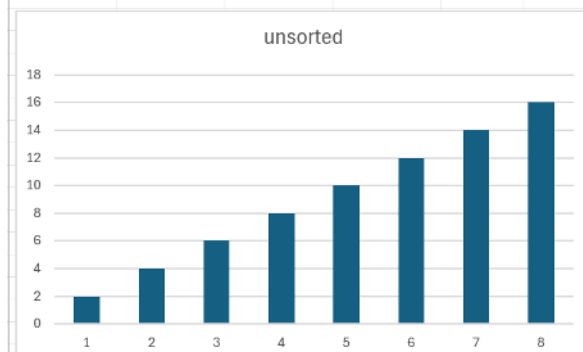
Column1	Column2	Column3	Column4
4	2	1	1
2	3	6	2
3	3	8	3
3	4	9	3
6			4
8			6
1			8
9			9

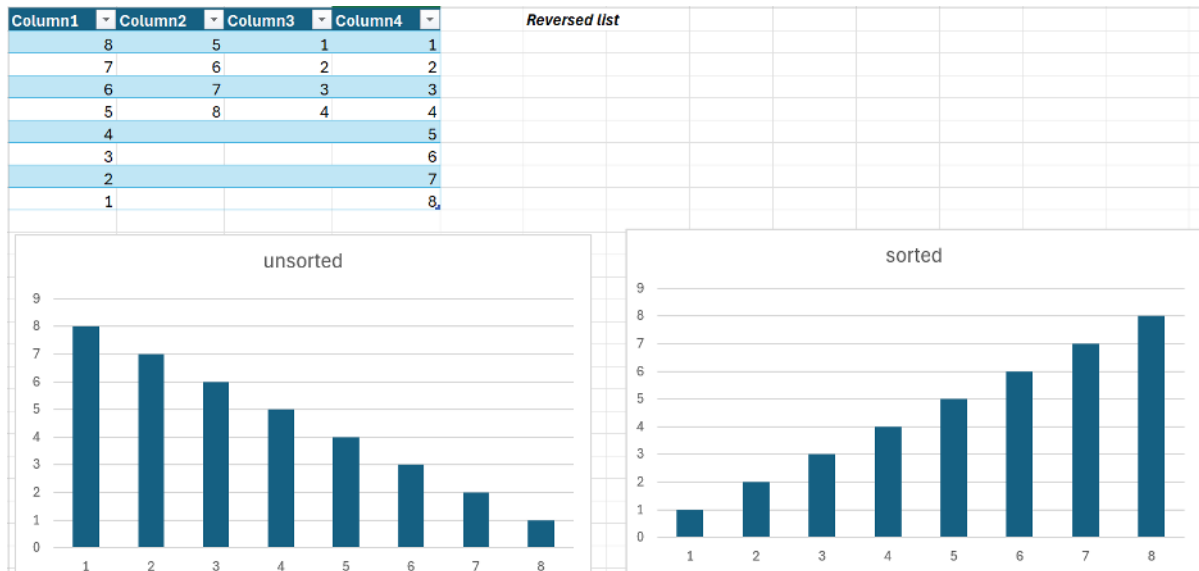
Random order with duplicates



Column1	Column2	Column3	Column4
2			2
4			4
6			6
8			8
10			10
12			12
14			14
16			16

Already sorted





Unit 18 ASS 3 visual spreadsheet - Excel ^{RA} • Saved

File Home Insert Draw Page Layout Formulas Data Review View Automate Help Acrobat

Paste Clipboard Font Font Alignment Number Styles Conditional Formatting Format as Table Cell Styles Cells Editing Sensitivity Add-ins Create and Share Adobe PDF

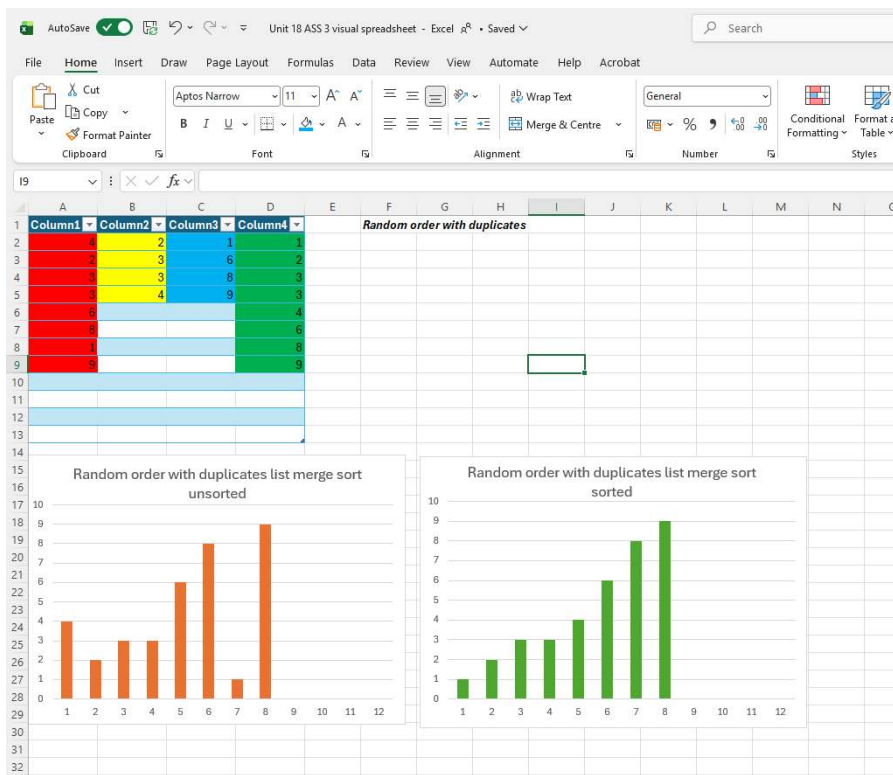
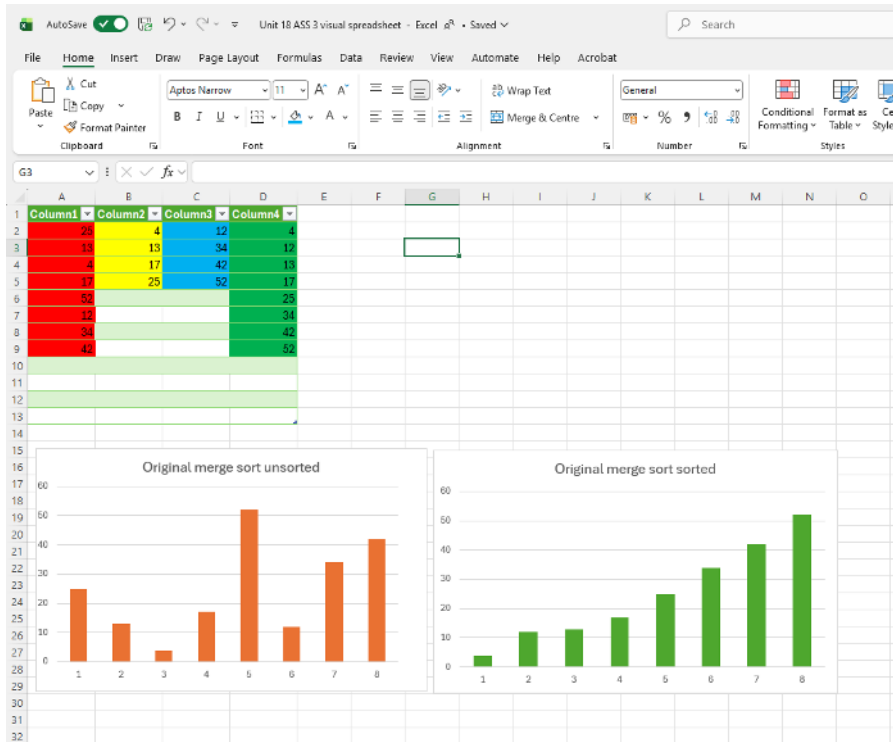
A21

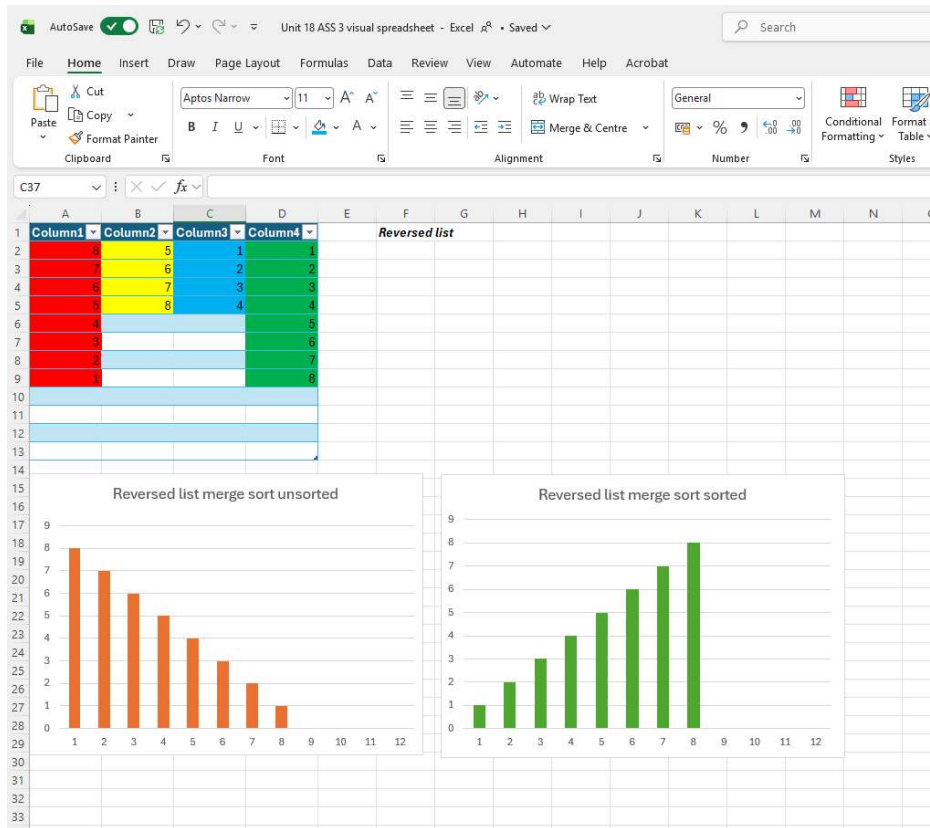
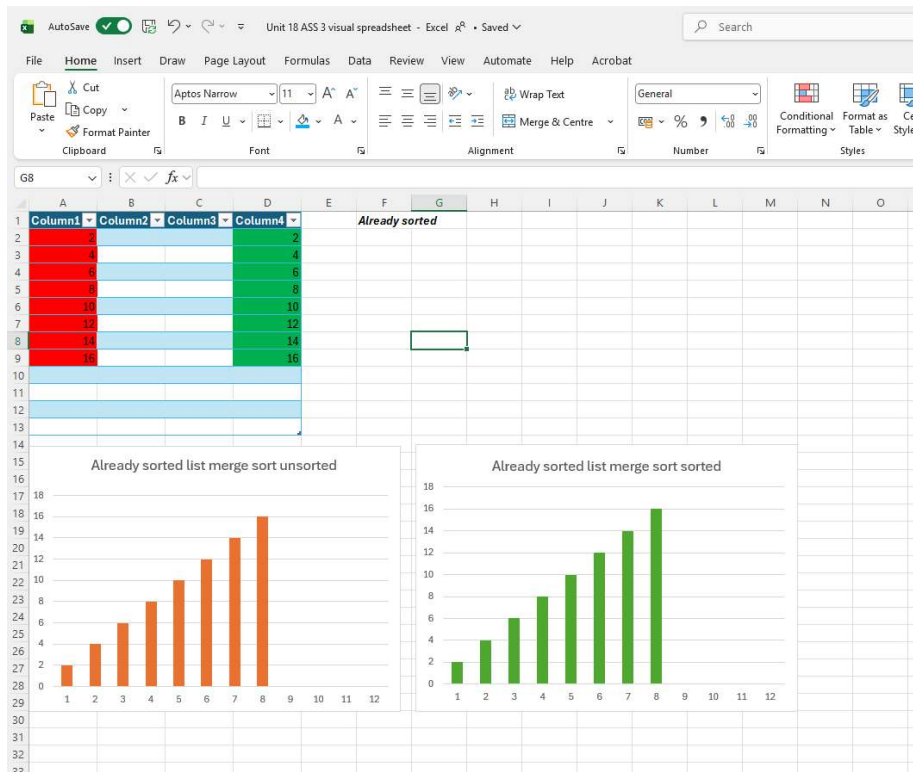
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1															
2															
3															
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															

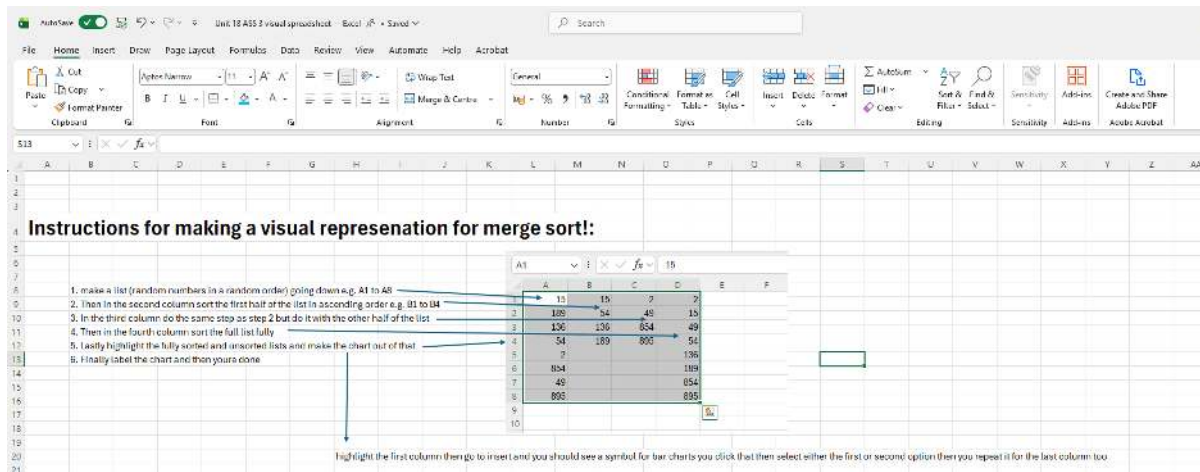
Instructions for making a visual representation for merge sort!:

1. make a list going down e.g. A1 to A8
2. Then in the other column sort the first half of the list in ascending order e.g. B1 to B4
3. In the third column do the same step as step 2 but do it with the other half of the list
4. Then in the final column sort the full list fully
5. Lastly highlight the fully sorted and unsorted lists and make the chart out of that
6. Finally label the chart and then you're done

After refinements:







I tested my finished model with other random lists to see if they work, below are all the tests I had used on my excel document to make this model work. These are the results of my model and if it has passed or not:

Name of test	List used	Expected outcome	results
Random order with duplicates	4,2,3,3,6,8,1,9	1,2,3,3,4,6,8,9	Passed
Already sorted	2,4,6,8,10,12,14,16	2,4,6,8,10,12,14,16	Passed
Reversed list	8,7,6,5,4,3,2,1	1,2,3,4,5,6,7,8	Passed
Empty list	empty	empty	Passed
All equal values	5,5,5,5,5	5,5,5,5,5	Passed
12 number list	4,2,3,3,6,8,1,9,85,54,54,36	1,2,3,3,4,6,8,9,36,54,54,85	Passed
Partial input so some are blank	1,,,5,,8,2,12	1,2,5,8,12	Passed

My visual spreadsheet: [Unit 18 ASS 3 visual spreadsheet.xlsx](#)

10. Conclusion

In this assignment, I learned many important concepts related to searching, sorting, and computational thinking. I learned how computers organise and process large amounts of data using different sorting and searching methods. I also learned how algorithms help computers solve problems step by step in a logical and efficient way.

One of the main things I learned was the importance of sorting data correctly. Sorting helps organise information so it becomes easier to search, manage, and analyse. I explored different sorting methods such as Bubble Sort, Insertion Sort,

Quick Sort, and Merge Sort. Each algorithm works differently and has its own strengths and weaknesses. I also learned how searching methods such as Linear Search and Binary Search help computers quickly find specific information inside datasets.

Another important part of this assignment was computational thinking. Computational thinking is a problem-solving skill that helps break large problems into smaller and easier steps. It includes skills such as:

- Decomposition
- Pattern recognition
- Logical thinking
- Algorithm design

These skills are very important in computing because they help programmers design systems, solve problems, and improve software performance. During this assignment, I used computational thinking to understand datasets, choose sorting methods, test my model, repair faults, and improve the final system.

I also learned how visualisation tools such as flowcharts, tables, charts, and diagrams help explain complex processes more clearly. These tools made it easier to understand how sorting algorithms work step by step.

Merge Sort was one of the most useful sorting methods I studied in this assignment. Merge Sort is useful because it is fast, organised, and efficient when working with large datasets. It uses a divide-and-conquer approach by splitting data into smaller sections, sorting each section, and then merging everything back together in the correct order.

One major advantage of Merge Sort is that it performs much faster than simple sorting methods like Bubble Sort when handling large amounts of data. Merge Sort is also reliable because it follows a clear structure and produces accurate results. This makes it useful in many real-world systems such as:

- Databases
- Search engines
- Banking systems

- School systems
- Online shopping websites

I also learned the importance of testing and refining systems. By testing different datasets, repairing faults, and improving the spreadsheet layout, I understood how software developers improve systems to make them more reliable and user-friendly.

Overall, this assignment helped improve my understanding of computational thinking, sorting algorithms, searching methods, and problem-solving in computing. These skills are very important in computer science and IT careers because modern digital systems depend on fast and efficient data processing every day.

11.References

Decomposition: <https://www.geeksforgeeks.org/operating-systems/what-is-problem-decomposition/> (30th April 2026)

Divide and conquer algorithm: <https://www.geeksforgeeks.org/dsa/introduction-to-divide-and-conquer-algorithm/> (30th April 2026)

Quick sort algorithm: <https://www.geeksforgeeks.org/dsa/quick-sort-algorithm/> (30th April 2026)

Computational thinking OCR decomposition:
<https://www.bbc.co.uk/bitesize/guides/z4rbcj6/revision/2> (7th May 2026)

Fundamentals of algorithms decomposition:
<https://www.bbc.co.uk/bitesize/guides/zjddqh7/revision/2> (7th May 2026)

Merge and bubble sort (from my notes):
<https://www.bbc.co.uk/bitesize/guides/zjdkw6f/revision/5> (accessed 11th May 2026)

Bubble sort: <https://www.geeksforgeeks.org/dsa/bubble-sort-algorithm/> (accessed 12th May 2026)

E1 Bubble sort example:

https://www.w3schools.com/dsa/dsa_algo_bubblesort.php (accessed 12th May 2026)

Merge sort: <https://www.geeksforgeeks.org/dsa/merge-sort/> (accessed 12th May 2026)

E2 Merge sort example:

https://www.w3schools.com/dsa/dsa_algo_mergesort.php (accessed 12th May 2026)

Insertion sort: <https://www.geeksforgeeks.org/dsa/insertion-sort-algorithm/> (accessed 12th May 2026)

E3 Insertion sort example:

https://www.w3schools.com/dsa/dsa_algo_insertionsort.php (accessed 12th May 2026)