Chapter 5.3: Merge Sort
1 of 3

Introduction
You have seen two algorithms that can sort data in to an ordered list:

- **Bubble Sort**
- **Insertion Sort**

The list can be ordered:

- **Ascending**
- **Descending**

We have seen that just looking at these two algorithms they have advantages and disadvantages.

- **Refresh your memory on these and why it is the case!**

Objectives for this session

- **Understand and be able to explain how a Merge Sort works**
- **Be able to explain the advantages and disadvantages of a Merge Sort**
- **Understand and explain the term 'RECURSION'**

Merge sort

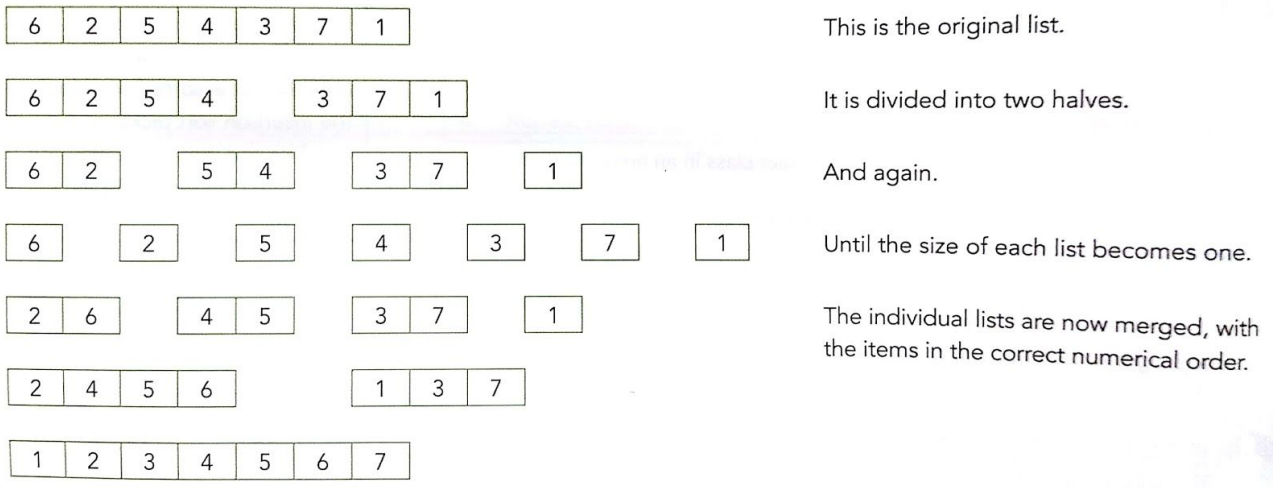
The merge sort algorithm is used to sort an unordered list by repeatedly (recursively) dividing a list into two smaller lists until the size of each list becomes one. This is why it is called a 'divide and conquer' algorithm. The individual lists are then merged. It works like this:

1. If there is only one item in the list then stop.
2. Divide the list into two parts.
3. Recursively divide these lists until the size of each becomes one.
4. Recursively merge the lists with the items in the correct numerical order.

WORKED EXAMPLE

Sort the following list into ascending order.

6 2 5 4 3 7 1



Task 1

Produce a table showing the swaps needed to sort the following numbers in ascending order using the merge sort. (the breakdown of the list and how it is built back up, as the example above). You can do it on paper VERY NEATLY! Scan it in using the printer just outside IT1 as an alternative method.

2, 8, 5, 14, 32, 23, 26, 12 Different colours represent different lists.
2, 8, 5, 14 32, 23, 26, 12
2, 8 5, 14 32, 23 26, 12
2 8 5 14 32 23 26 12
2, 8 5, 14 23, 32 12, 26
2, 5, 8, 14 12, 23, 26, 32
2, 5, 8, 12, 14, 23, 26, 32

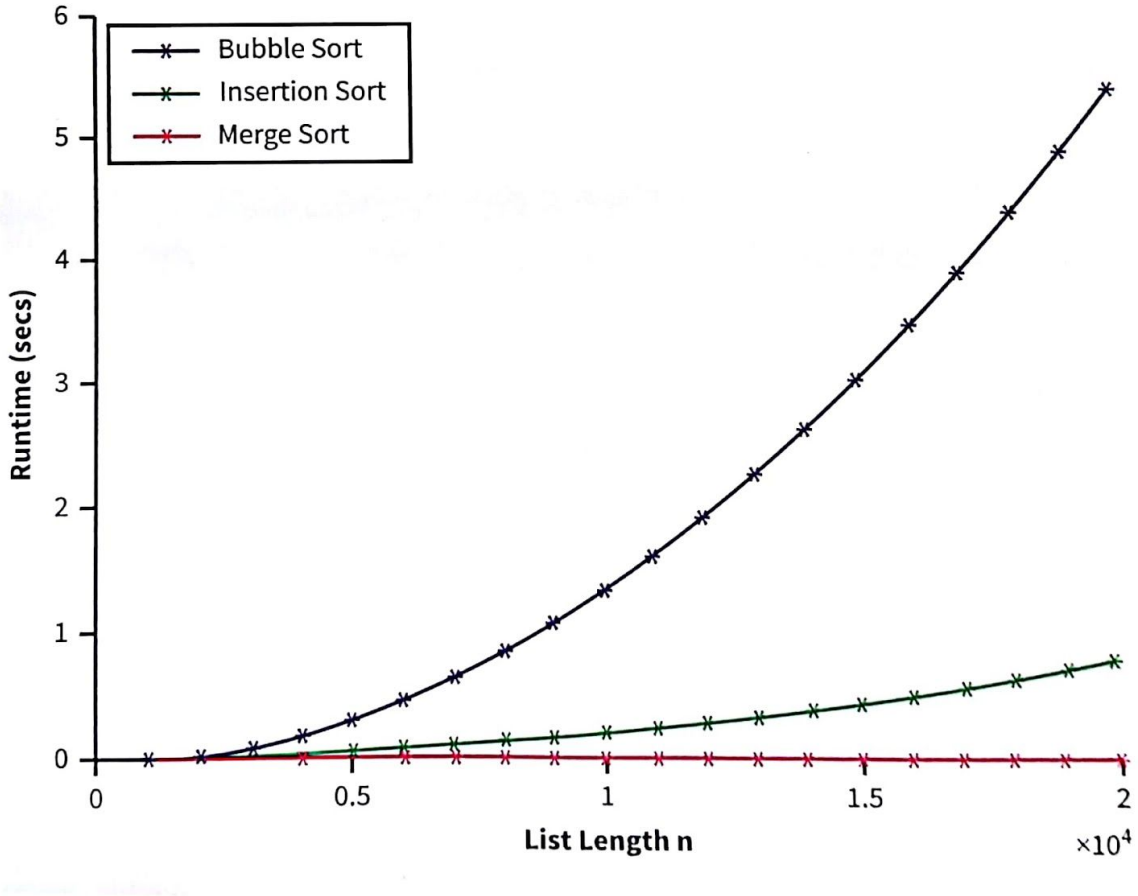
Which algorithm should I use?

Choosing which sorting algorithm depends on what you want to do.

	Advantages	Disadvantages
Bubble sort	Simplest and easiest to code	Slowest
Insertion sort	Simple and easy to code	Twice as fast as the bubble sort but slower than the merge sort
Merge sort	Fastest. Faster to sort the list than the other two	More difficult to code than the other two

So if you have a very large list (more than 1000 items) and speed of execution is important, it is worth spending the extra time coding the merge sort algorithm. But if you have a list of less than 1000 items, then the time saved in execution is so tiny that it is negligible and so you could use an insertion or bubble sort, which is easier to code.

The following graph shows the times taken for the algorithms to sort lists of different lengths. As you can see, with small lists the differences are less than one second.



Extension 1

In future lessons we will be moving on to look at Searching Algorithms. In the box below. Write down a list of steps to find a number in a unordered list.

myList = [1, 0, 3, 21]
If 3 in myList:
  print("True")
Else:
  print("False")
This actually worked

```
myList = [1, 0, 3, 12]
if 3 in myList:
  print("True")
else:
  print("False")
```

```
[GCC
True
```

Extension 2

In future lessons we will be moving on to look at Searching Algorithms. In the box below. Write down a list of steps to find a number in a ordered list.

I guess you could do the same and check if something is an element already in the list .

randomNumbers = [2532, 14124124, 12454, 75]