Introduction
General House Keeping

- No Fire Drills Planned! Assembly point is at the car park at the front of the building
- Each floor is laid out as a square, the toilet on this floor are located on the opposite side of the square
- Breaks:
  - 18:30 – 15 Minute Break
Over the Next 6 weeks ...

- Week 1: Sorting
- Week 2: Algorithm Design Techniques
- Week 3: Data Representation
- Week 4: Cryptography
  
  *** EASTER BREAK ***

- Week 5: Algorithm Development
- Week 6: Resource Creation and Pedagogical Techniques
Sorting
Why Sorting?

Ok so it is not strictly specified in the current GCSE syllabus so why?

- ‘Classic’ examples of algorithms within computer science – which make great examples!
- Programming
  - 1-D Arrays / Lists
  - Reading from Files
Let’s Give Sorting a Try ...

Within your groups you have a stack of un-ordered cards labelled from 1 to 30.

Firstly: Sort the cards in ascending order.

Secondly: Can you write down how you went about sorting these cards so someone else can follow it?
Sorting Algorithms
Bubble Sort

One of the simplest sorting algorithms to describe is something called a Bubble Sort.

It is named after the fact that the higher values ‘bubble’ to the end of the list.
Bubble Sort

A Bubble sort works like this:

• Compare the first and second element of the list. If they are in the wrong order then swap them.
• Continue to compare every pair of elements until you reach the end of the list.
• If you have made any swaps then repeat, if not the list is sorted.
Bubble Sort

1) Create a flow-diagram which represents a Bubble sort on a list.

2) Write a program in Pseudocode which represents a Bubble sort on a list.
pos1 = first
pos2 = second

Is the
element at
pos1 > pos2?

Yes
Swap elements at
pos1 and pos2

No
pos1 = pos1 + 1
pos2 = pos2 +1

Has a
swap
taken
place?

Yes
Loop until pos2
is last element

No
Done
Bubble Sort - Pseudocode

1. Let Swapped = True
2. WHILE Swapped
3. Let Position1 = 0
4. Let Position2 = 1
5. Let Swapped = False
6. WHILE Position2 < size of List
7. IF element at Position1 > element at Position1
8. Let Temp = element at Position1
9. Let Position1 of the list = element at Position2
10. Let Position2 of the list = Temp
11. Let Swapped = True
12. END IF
13. Let Position1 = Position1 + 1
14. Let Position2 = Position2 + 1
15. END WHILE
16. END WHILE
Insertion Sort

An Insertion sort is another type of sort which is common due to its simplistic nature.

It is named after the fact that each element is inserted into a new list in the correct order.
An Insertion sort works like this:

- Take the first element of the list and insert it into a new list.
- For every element left in the original list insert it into the new list in the correct order by comparing it to the elements within the list.
Insertion Sort

1) Create a flow-diagram which represents an Insertion sort on a list.

2) Write a program in Pseudocode which represents a Insertion sort on a list.
Complexity Of Algorithms

When we look at different sorting algorithms it is natural to try and decide which is faster.

When we consider complexity we are referring to how many times each line of code needs to be run.

So for the two sorts we have discussed so far they run in the worst case for ...

Can we do better???
Merge Sort

To complete this type of sort we are all going to follow the same instructions:

- If you have 1 card – pass it back to the person who passed it to you.
- If you have more than 1 card, split the cards in half and pass each pack to two different people.
- When you get the two piles back they will be sorted so merge (in order) the packs and pass them back to the person who passed them to you.
Merge Sort

1) Create a flow-diagram which represents a Merge Sort on a list.

2) Write a program in Pseudocode which represents a Merge Sort on a list.
Merge Sort - Pseudocode

1. Let Result be a new list
2. IF there is one item in the list
3.   return list
4. IF there are more than one items in the list
5.   Split the list into two lists approximately half the size of the original
6.   mergesort the first-half
7.   mergesort the second-half
8. WHILE there are elements left in both the first-half and second-half
9.   IF the first element on the first-half < the first element of the second-half
10.  append the first element of the first-half to Result
11.   remove first element from first-half
12. ELSE
13.   append the first element of the second-half to Result
14.   remove first element from second-half
15. append remaining elements of first-half to Result
16. append remaining elements of second-half to Result
17. return Result
Sorting at Speed? What else effects Sorting?

There are many different things which effect the speed at which things are sorted:

- The way in which the numbers are arranged to start with (i.e. partially sorted)
- How the numbers are stored

http://www.sorting-algorithms.com
Testing Sorting Algorithms: Python

I have coded three sorting algorithms (bubble sort, insertion sort and merge sort) within Python.

You can generate a list of random numbers using GenerateNumberList.py by specifying the quantity you want to generate.

The programs BubbleSort.py, InsertionSort.py and MergeSort.py will read in these numbers and print them sorted.
A little bit of Python Code ...
Conclusion

Over the session you have seen a range of sorting techniques.

Next week we will be looking at Algorithm Design Techniques – We will be exploring the different methods of solving a wide selection of problems.