







Key vocabulary	
Algorithm	A sequence of logical instructions for carrying out a task. In computing, algorithms are needed to design computer programs.
Computational Thinking	A problem-solving method using computer science techniques, where possible solutions are developed and presented in a way that can be understood by humans and computers.
Abstraction	The process of extracting or withdrawing something.
Decomposition	Breaking down a complex problem or system into smaller parts that are more manageable and easier to understand.
Pseudocode	Also written as pseudo-code. A method of writing up a set of instructions for a computer program using plain English. This is a good way of planning a program before coding.
Trace table	Trace tables are used to allow programmers to trace the value of variables as each line of code is executed. The values of the variables are displayed in a table and assist the programmer in identifying any potential errors.

Flowcharts			
	Line		Input/output
	Process		Decision
	Sub Program		Terminal

Searching Algorithms
<p style="text-align: center;">Linear Search</p> <p>A linear search is the simplest method of searching a data set. Starting at the beginning of the data set, each item of data is examined until a match is made. Once the item is found, the search ends.</p> <ol style="list-style-type: none"> 1. A way to describe a linear search would be: 2. Find out the length of the data set. 3. Set counter to 0. 4. Examine value held in the list at the counter position. 5. Check to see if the value at that position matches the value searched for. 6. If it matches, the value is found. End the search. 7. If not, increment the counter by 1 and go back to step 3 until there are no more items to search. <p>A linear search, although simple, can be quite inefficient. Suppose the data set contained 100 items of data, and the item searched for happens to be the last item in the set? All of the previous 99 items would have to be searched through first.</p> <p>However, linear searches have the advantage that they will work on any data set, whether it is ordered or unordered.</p>
<p style="text-align: center;">Binary Search</p> <p>A binary search is an efficient method of searching an ordered list. A binary search works like this:</p> <ol style="list-style-type: none"> 1. Start by setting the counter to the middle position in the list. 2. If the value held there is a match, the search ends. 3. If the value at the midpoint is less than the value to be found, the list is divided in half. The lower half of the list is ignored and the search keeps to the upper half of the list. 4. Otherwise, if the value at the midpoint is greater than the value to be found, the upper half of the list is ignored and the search keeps to the lower half of the list. 5. The search moves to the midpoint of the remaining items. Steps 2 through 4 continue until a match is made or there are no more items to be found.

Sorting Algorithms

Bubble Sort

Bubble sorts work like this:

Start at the beginning of the list. Compare the first value in the list with the next one up. If the first value is bigger, swap the positions of the two values. Move to the second value in the list. Again, compare this value with the next and swap if the value is bigger. Keep going until there are no more items to compare. Go back to the start of the list.

Each run through the list, from start to finish, is known as a pass. The bubble sort continues until a pass is made where no values have been swapped. At this point, the list is sorted.

Consider this unsorted list:

Position in list	0	1	2	3	4
Data value	9	4	2	6	5

The value at position 0 is 9, and the value at position 1 is 4. 9 is bigger than 4, so the two items would be swapped. The list would now be:

Position in list	0	1	2	3	4
Data value	4	9	2	6	5

We move up to the next position, position 1. The value at position 1 is 9, and the value at position 2 is 2. 9 is bigger than 2, so the two items would be swapped. The list would now be:

Position in list	0	1	2	3	4
Data value	4	2	9	6	5

We move up to the next position, position 2. The value at position 2 is 9, and the value at position 3 is 6. 9 is bigger than 6, so the two items would be swapped. The list would now be:

Position in list	0	1	2	3	4
Data value	4	2	6	9	5

We move up to the next position, position 3. The value at position 3 is 9, and the value at position 4 is 5. 9 is bigger than 5, so the two items would be swapped. The list would now be:

Position in list	0	1	2	3	4
Data value	4	2	6	5	9

The first pass is now complete. However, this list may still be unsorted, so another pass takes place.

Merge Sort

A merge sort is a more complex sort, but also a highly efficient one.

A merge sort uses a technique called divide and conquer. The list is repeatedly divided into two until all the elements are separated individually. Pairs of elements are then compared, placed into order and combined. The process is then repeated until the list is recompiled as a whole.

Consider this unsorted list:

7 11 10 5 12 4 18 15

The list is split into half:

7 11 10 5 12 4 18 15

The process repeats:

7 11 10 5 12 4 18 15

Until all elements are individually separated:

7 11 10 5 12 4 18 15

The algorithm looks at the individual elements and compares them as pairs. Each pair is sorted into order:

7 11 10 5 12 4 18 15

The process is repeated for the initial right hand division:

7 11 10 5 12 4 18 15

Eventually the list is recompiled

7 11 10 5 12 4 18 15

The list is now sorted into the correct order.

Insertion Sort

An insertion sort is less complex and efficient than a merge sort, but more efficient than a bubble sort. An insertion sort compares values in turn, starting with the second value in the list.

If this value is greater than the value to the left of it, no changes are made. Otherwise this value is repeatedly moved left until it meets a value that is less than it.

The sort process then starts again with the next value. This continues until the end of the list is reached.

Consider this unsorted list:

12 14 13 11 16 10 18 17

12 is the first value in the list. No other values are before it, so the sort moves on to the next value, 14. 14 is greater than the value to the left, 12, so the sort moves on again to the next value, 13. 13 is less than 14, so the two elements are swapped:

12 13 14 11 16 10 18 17

13 is greater than 12, so the sort moves onto the next value, 14. 14 is greater than 13, so again the sort moves on.

11 is less than 14, so the two values swap:

12 13 11 14 16 10 18 17

11 is less than 13, so the two values swap:

12 11 13 14 16 10 18 17

11 is less than 12, so the two values swap:

11 12 13 14 16 10 18 17

The process repeats:

11 12 13 14 10 16 18 17

11 12 13 10 14 16 18 17

11 12 10 13 14 16 18 17

11 10 12 13 14 16 18 17

10 11 12 13 14 16 18 17

10 11 12 13 14 16 17 18