Q1. Identify and explain the workings of TWO sorting algorithms and discuss and compare their performance/efficiency (i.e. Big O) (309 words)

Insertion Sorting – Insertion sorting algorithms are when a sorted array is built up one element at a time. So, with each iteration an insertion will remove an element from the data that has been already inputted. The algorithm will find the location that which is belongs to within the sorted array and then proceeds to insert the new element and will repeat this step over and over until no more elements remain. Insertion iterates the array, whilst growing the list behind itself. At each position, the element will be checked against the largest value within the sorted list and will shift larger values up to provide space.

Quick Sorting – Quick sorting, as the name suggests, is faster than insertion due to its divide and conquer technique. The algorithm will firstly divide the array into smaller arrays that is known by 'low elements' and 'high elements'. For this to happen, quick sorting will find a pivot element to help partition the array with elements less than the pivot being 'low' and higher elements being 'high' whilst equal elements can go either way. Then the algorithm will repeat this over and over with the partitioned arrays till no more elements remain.

Quick sorting is faster for most uses in the real world. As data can be extremely large, quick sort has the advantage of memory hierarchy by taking advantage of caches and virtual memory. The implementation of real-world data is more suited to modern computer types. However, when it comes to smaller data, insertion would work quicker in niche environments. When (Ci * n^2) < (Cq * (n log n)) insertion is faster. This is due to the asymptotic behavior of Big-O that when n is large. Meaning, insertion will work faster for small 'n' due to quick sorting having a repeating function that adds to the overhead of the algorithm.

Q2 - Identify and explain the workings of TWO search algorithms and discuss and compare their performance/efficiency (i.e. Big O) (326 words)

Linear – A linear search algorithm works by checking each element one by one in order until the entire list is searched and a match is found. Linear searches make the most comparisons when it comes to 'n', where 'n' is considered the length, but equally takes the longest time. So, while each element of 'n' is being searched equally, a linear search has an average of 'n over 2' comparisons. Another way of defining linear searches is that it will transverse an array in order to find the value it has been given to search for.

Binary – A binary search algorithm is a lot more efficient compared to linear when it comes to searches. Reason being is that it compares the value given to search to the middle element of the array and if the value matches it will return the position. But, if the value is not found, and is less than the element the search will half itself again within the first half until the value is found. And vice versa if the given value is greater. This makes binary more efficient due to half of the array not needed to be searched. This step is repeated until the given value is found or if the search is exhausted. The area that is being searched is continuously being halved or reduced and at most the number of comparisons is log[N + 1].

Both, linear and binary, are useful dependent on what the application needs to be done. If the array is a data structure with the elements arranged in order, binary would be better to use due to how quick the search is. But, it is required that that the elements of the array are sorted for a binary search to be effective, because binary searches need a middle element to work of off. In the case that the array isn't sorted sequentially, then a linear search will be used as the search algorithm.