**Introduction**

An algorithm can be described as a precise, systematic method for solving a class of problems [7]. At the heart of computer science is Data Structure and sorting is an important issue in Data Structure that generates the sequence of the list of items. Just as sorting is important in everyday life is the same reason that it is important in programming- it is much easier to locate items that are sorted than unsorted [1].

The proliferation of data particularly over the last few years, according to Forbes- 90% of all the data in the world has been generated in the last two years alone [9]. Developing sorting algorithms through improved performance and decreasing complexity has attracted a considerable amount of research and effort [10].

Numerous computations and tasks become simple by properly sorting data in advance. It is worth nothing that an estimated 25% of the total active time of computers is devoted to sorting procedures [2], this makes it an area of high importance for further study and effort to improve the performance of the many electronic devices we now rely on so heavily. Much focus was dedicated on sorting collections of data that were too large for the computers of the day to store in memory [3]. In their article Paira, Chandra and Alam state that the effectiveness of sorting algorithms is to optimise the importance of other sorting algorithms and the optimality of these algorithms is judged when calculating their time and space complexities [4].

Time Complexity

Simplicity is an important characteristic of a good algorithm according to Chang,SK. A clear and well documented program is simple to read and easy to explain and maintenance can be made easily long after its conception either by the author or other programmers. Achieving a high efficiency requires some cost in the use of resources that the algorithm needs to reach such efficiency [5]. Time complexity of an algorithm does not equal to the actual time required to execute some code, but the number of times that a statement executes [6].

When considering an algorithm it is essential to identify the worst, average and best case scenarios of the algorithm. At a very high level, the worst case complexity of an algorithm is the greatest number of operations necessary to solve the problem, the best-case complexity of an algorithm is the least number of operations and the average-case complexity of an algorithm is the average number of operations required to solve the problem over all possible inputs of size *n.* In order to compare the efficiencies of competing algorithms for a given problem, it is a requirement to consider the number of operations performed by each algorithm and this is done by classifying and comparing the growth rates of each algorithm’s complexity function[7].

When deciding which algorithm to implement for the given problem, several factors must be considered. Firstly, the time complexity, this determines the amount of time that can be taken by and algorithm to run. Time complexity is usually written in form big O(n) notation. Big O notation measures how quickly a function grows or declines, and the value n represents the number of operations performed by the algorithm. The next consideration is stability, meaning that the algorithm keeps elements with equal values in the same relative order in the output as they were in the input. Some algorithms are stable by its nature ad an example of these would be; insertion sort, bubble sort or merge sort while others such as quick sort are not stable. Memory space is the final factor to consider. Recursive algorithms need more copies of sorting data to maintain memory and improve efficiency.

Deciding which sorting algorithm to use [8]:

|  |  |
| --- | --- |
| **Criteria** | **Sorting Algorithm** |
| Small number of items | Insertion Sort |
| Items relatively pre-sorted | Insertion Sort |
| Concerned about worst-case scenarios | Heap Sort |
| Interested in good average-case result | Quicksort |
| Write as little code as possible | Insertion Sort |

Desirable properties:

* Stability
* Time complexity- Good run-time- best, worst and average case time complexity
* In-place sorting- if memory is a concern
* Suitability- the properties of the sorting algorithm are well matched to the class of input instances which are expected- i.e. consider specific strengths and weaknesses when choosing a sorting algorithm

An important aspect when selecting a sorting algorithm, the best-case, worst-case and average-case time complexity