**Experimental Analysis on Specific Sorting Algorithms**

As per the title name, this report talks about the analysis of specific sorting algorithms namely, Merge Sort Algorithm, Three Way Merge Sort Algorithm, Heap Sort Algorithm and Insertion Sort Algorithm in terms of number of comparisons and Time complexity each algorithm takes.

**Importance:** Let me begin with why is it important to do this analysis. In real world, we deal with lot of data and so many times we would get situations that sorting a data in specific manner is important. Sorting can be done based on numbers, characters, sizes, dates and many more. Storing the images based on the date or name in our mobile is one such example which makes our life easier to find and experience the memories. So, it is very important to know what type of algorithm is best at its own way based on the circumstances. Let us proceed with theoretical analysis.

**Theoretical Analysis:**

Many scientists had invested their time in finding lot of sorting algorithms and techniques and handed over to us the beautiful concepts. Each algorithm will have their impact based on the type of data being sorted.

The time complexity of an algorithm is represented by the asymptotic behavior of the growth of time functions of a particular algorithm. It is always a good idea to notate the growth with tight boundaries which is (Θ).

Text book says that the time complexity of the above-mentioned algorithms are as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Algorithm | Best Case | Average Case | Worst Case |
| Merge Sort | Θ(n log2n) | Θ(n log2n) | Θ(n log2n) |
| Three Way Merge Sort | Θ(n log3n) | Θ(n log3n) | Θ(n log3n) |
| Heap Sort | Θ(n log2n) | Θ(n log2n) | Θ(n log2n) |
| Insertion Sort | Θ(n) | Θ(n2) | Θ(n2) |

* *Insertion sort* is a simple algorithm and it is a in place algorithm which means that it doesn’t require any additional memory in sorting process.
* *Merge sort* uses the divide and conquer technique to sort the data and it requires additional space for computation.
* *Three-way Merge Sort* is as similar as normal merge sort but the array will be split into three parts which results in base 3 of log while time growth calculation whereas Merge sort takes base 2 of log.
* *Heap Sort* holds two properties. One is the data structure it follows is a complete binary tree or almost complete tree. And another one is either Max or Min heap property which says that root element will be wither maximum element or minimum element respectively.

Now it is our time to implement and test the algorithms to find some exciting things which will be helpful.

**Practical Performance of Algorithms:**

Python programming language has been used to implement the algorithms for the analysis purpose.

*Configurations and Tools used*: RAM – 16Gb, OS: Windows 11 Pro, Processer: 11th Gen i7. Python 3.10 IDLE

Below are the tabulated results of each algorithm in terms of Input Type, Input Size, Number of comparisons made and Execution time in Milli Seconds.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Type of Input | Input Size | Number of Comparisons | Execution Time in Milli Seconds |
| **Merge Sort** | Descending Ordered | 20000 | 287232 | 63 |
| 40000 | 614464 | 62 |
| 80000 | 1308928 | 63 |
| 160000 | 2777856 | 78 |
| 320000 | 5875712 | 125 |
| Identical | 40000 | 287232 | 62 |
| 40000 | 614464 | 141 |
| 80000 | 1308928 | 297 |
| 160000 | 2777856 | 563 |
| 320000 | 5875712 | 1187 |
| Random | 20000 | 287232 | 63 |
| 40000 | 614464 | 140 |
| 80000 | 1308928 | 282 |
| 160000 | 2777856 | 594 |
| 320000 | 5875712 | 1297 |
| Sorted | 20000 | 287232 | 78 |
| 40000 | 614464 | 125 |
| 80000 | 1308928 | 281 |
| 160000 | 2777856 | 578 |
| 320000 | 5875712 | 1187 |
| **Three Way Merge Sort Output** | Descending Ordered | 20000 | 370560 | 79 |
| 40000 | 791060 | 140 |
| 80000 | 1724472 | 312 |
| 160000 | 3512496 | 563 |
| 320000 | 7618048 | 1344 |
| Identical | 20000 | 370560 | 62 |
| 40000 | 791060 | 157 |
| 80000 | 1724472 | 297 |
| 160000 | 3512496 | 578 |
| 320000 | 7618048 | 1312 |
| Random | 20000 | 370560 | 78 |
| 40000 | 791060 | 156 |
| 80000 | 1724472 | 344 |
| 160000 | 3512496 | 594 |
| 320000 | 7618048 | 1407 |
| Sorted | 20000 | 370560 | 63 |
| 40000 | 791060 | 125 |
| 80000 | 1724472 | 312 |
|  |  | 160000 | 3512496 | 609 |
| 320000 | 7618048 | 1343 |
| **Heap Sort** | Descending Ordered | 20000 | 493298 | 94 |
| 40000 | 1067725 | 203 |
| 80000 | 2292812 | 422 |
| 160000 | 4906721 | 937 |
| 320000 | 10450529 | 2000 |
| Identical | 20000 | 59994 | 16 |
| 40000 | 119994 | 16 |
| 80000 | 239994 | 31 |
| 160000 | 479994 | 78 |
| 320000 | 959994 | 156 |
| Random | 20000 | 510475 | 94 |
| 40000 | 1101627 | 218 |
| 80000 | 2363214 | 469 |
| 160000 | 5045759 | 1032 |
| 320000 | 10732009 | 2188 |
| Sorted | 20000 | 529078 | 109 |
| 40000 | 1138257 | 219 |
| 80000 | 2437130 | 469 |
| 160000 | 5197154 | 984 |
| 320000 | 11040499 | 2062 |
| **Insertion Sort** | Descending Ordered | 20000 | 199990000 | 16188 |
| 40000 | 799979992 | 64375 |
| 80000 | 3199959914 | 259047 |
| 160000 | 12799919284 | 1049391 |
| 320000 | 51199834965 | 4210922 |
| Identical | 20000 | 19999 | 15 |
| 40000 | 39999 | 16 |
| 80000 | 79999 | 15 |
| 160000 | 159999 | 31 |
| 320000 | 319999 | 46 |
| Random | 20000 | 100550401 | 8187 |
| 40000 | 399780117 | 32718 |
| 80000 | 1592870996 | 135063 |
| 160000 | 6411201713 | 562625 |
| 320000 | 25630496119 | 2301969 |
| Sorted | 20000 | 19999 | 16 |
| 40000 | 39999 | 16 |
| 80000 | 79999 | 15 |
| 160000 | 159999 | 16 |
| 320000 | 319999 | 47 |

*Based on the above data below graphs have been plotted comparing the algorithms in terms of input size and time taken in milli seconds.*

Performance of Algorithms plotted for Identical Data.

Performance of Algorithms plotted for Descending Ordered Data.

Please note that Insertion Sort data is not included in the graph to represent other sorting algorithms better.

Performance of Algorithms plotted for Random Data.

Performance of Algorithms plotted for Ascending Ordered Data.

Based on the above table data and Graphs plotted, below points have been observed.

**Analysis and Observations:**

* *Insertion sort’s* performance is best when the input data is sorted or almost sorted and Identical compared to other algorithms. Number of comparisons made will be n-1 when data is already sorted or identical. But I am vexed with the time taken by this algorithm when the input data is in descending order or random and huge data.

Here we can think of some algorithm which is better in terms of time when data is huge, i.e., *Merge Sort.*

* *Merge sort* works better when the data is in large quantities. The number of comparisons made for same input size but different types of data remain same. But the drawback of the merge algorithm is it takes additional memory to perform the sorting. If space is a concern, then this algorithm will not justice.
* Although *Three-way Merge Sort Algorithm*’s time complexity looks better in theoretical analysis, The results surprised me that it is taking a little bit more time to compute the sorting. This because the number of comparisons made will be more in number compared with merge sort. This algorithm holds the same drawbacks as Merge sort.

And hence we need a more efficient algorithm which runs faster and doesn’t take more space at a time, here comes the *Heap Sort*.

* *Heap Sort* algorithm slightly takes more time than merge sort but it doesn’t consume additional memory. We can go with heap sort if we have space constraint and need to deal with bulk data.

**Conclusion:**

*There are some more factors that might affect the performance of algorithm like running these algorithms using other programming languages (Python is a little bit slower in performance) and the configurations of machine going to use. But these are not valid arguments when we measure the growth of time functions of each algorithm in terms of input size. Hence, I would say that I am convinced with the theoretical analysis and actual performance.*