***Abstract:***

The purpose of this assignment is to analyze the execution time complexities of three sorting algorithms i.e., Insertion Sort, Improved Insertion Sort, and Merge Sort by making them sort a vector array of dimensions m\*n, the sorting is done by calculating the vector length of each array using the formula:

The algorithms are tested using values of ***m*** being ***10000, 25000, 50000, 100000, 25000, 1000000, 2500000*** and ***n*** being ***10, 25, 50***. Each value of ***“m”*** was tested with all ***“n”*** values.

***Results:***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **n=10** | | | **n=25** | | | **n=50** | | |
| **M** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** |
| **10000** | 1.355 | .0009 | 2.897 | 3.934 | .0024 | 7.596 | 9.721 | .0053 | 25.324 |
| **25000** | 9.043 | .0022 | 16.668 | 24.136 | .0054 | 47.630 | 68.098 | .007 | 112.036 |
| **50000** | 35.690 | .0028 | 85.177 | 103.787 | .0076 | 176.507 | 326.977 | .0164 | 518.691 |
| **100000** | 229.481 | .0076 | 338.211 | 611.614 | .019 | 942.226 | 1283.955 | .0478 | 1813.611 |
| **250000** | 1850.282 | .019 | 2274.159 | 3670.171 | 0.314 | 4449.691 | 6727.927 | .1047 | 7798.529 |
| **1000000** | 5184.312 | .056 | 12234.977 | 20479.554 | 0.0619 | 22203.961 | 35254.339 | .2251 | 31584.0425 |
| **2500000** | - | - | - | - | - | - | - | - | - |

*Table 1: For Naïve Insertion Sort*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **n=10** | | | **n=25** | | | **n=50** | | |
| **M** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** |
| **10000** | .0808 | .0002 | .1164 | .0896 | .001 | .1662 | .0872 | .0018 | .1658 |
| **25000** | .5122 | .0008 | .988 | .5198 | .0018 | .9862 | .4936 | .004 | .9796 |
| **50000** | 3.277 | .002 | 3.922 | 1.953 | .0036 | 3.867 | 1.950 | .0072 | 3.898 |
| **100000** | 7.770 | .0036 | 22.796 | 7.745 | .0078 | 15.514 | 7.760 | .0148 | 15.556 |
| **250000** | 64.865 | .0072 | 235.011 | 50.266 | .0168 | 206.645 | 69.613 | .054 | 139.56 |
| **1000000** | 1142.809 | .0262 | 2159.555 | 1465.113 | .0922 | 2898.640 | 1199.414 | .1992 | 2184.803 |
| **2500000** | - | - | - | - | - | - | - | - | - |

*Table 2: For Improved Insertion Sort*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **n=10** | | | **n=25** | | | **n=50** | | |
| **M** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** | **Random Vector** | **Sorted Vector** | **Inverse Sorted Vector** |
| **10000** | .014 | .0124 | .0132 | .0172 | .0152 | .0152 | .0216 | .0204 | .0242 |
| **25000** | .0364 | .0382 | .0348 | .047 | .0424 | 0.042 | .061 | .0566 | .0562 |
| **50000** | .075 | .073 | .0742 | .0958 | .0918 | .0888 | .1168 | .1178 | .0918 |
| **100000** | .1536 | .147 | .1534 | 1.94 | .191 | .1826 | .2418 | .2356 | .2458 |
| **250000** | .3954 | .3742 | .3754 | .4822 | .471 | .4702 | .6404 | .6198 | .6152 |
| **1000000** | 1.731 | 1.650 | 1.652 | 2.184 | 2.092 | 2.083 | 2.721 | 2.705 | 2.714 |
| **2500000** | 4.434 | 4.229 | 3.855 | 7.204 | 6.163 | 6.552 | 15.084 | 14.418 | 14.769 |

*Table 3: For Merge Sort*

***Graphical Visualization:***

A graph with numbers and lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

A graph of a function

Description automatically generated with medium confidence

A graph with numbers and lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

A graph with numbers and lines

Description automatically generated

***Evaluation of Results:***

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Case** | **Evaluation** |
| **Naive Insertion Sort** | **Average Case (Random Sorted Array)** | The time is increasing exponentially and as indicated by R value, both the attributes are highly dependent as well, since for every increase in value of m, the time for computation also increased by the factor of (n\*m) |
| **Best Case (Sorted Array)** | Execution time is only dependent on the number of elements into the array as there is only a single comparison being made. The graph also shows a linear trend |
| **Worst Case (Inverse Sorted Array)** | The time complexity is the worst as it keeps on growing by the factor of n, as in each iteration, every element is going to be swapped. The time complexity is highly dependent on the value of n, as it can be seen in graph as an exponential trend |
| **Insertion Sort Improved** | **Average Case (Random Sorted Array)** | The time is increasing exponentially and as indicated by R value, both the attributes are highly dependent as well, since for every increase in value of m, the time for computation also increased by the factor of (n\*m) |
| **Best Case (Sorted Array)** | Execution time is only dependent on the number of elements into the array as there is only a single comparison being made. The graph also shows a linear trend |
| **Worst Case (Inverse Sorted Array)** | The time complexity is the worst as it keeps on growing by the factor of n, as in each iteration, every element is going to be swapped. The time complexity is highly dependent on the value of n, as it can be seen in graph as an exponential trend |
| **Merge Sort** | **Average Case (Random Sorted Array)** | The time complexity is independent of the type of case, as in each iteration, the problem is split into 2 sub problems which can be observed into the graph as a linear function. |
| **Best Case (Sorted Array)** |
| **Worst Case (Inverse Sorted Array)** |

***Conclusion:***

It can be concluded that each algorithm has its own positives and negatives, like if the input size is small than the effect of any algorithm is not clearly visible, and insertion sort would be a wise choice due to its simple nature as well as low memory impact. But as the input size grows, we can clearly observe in the execution time of algorithms that merge sort clearly outperforms insertion sort in terms of running term.