**CS 590 - Assignment 1**

**Submitted By: Shweta Madhale**

**CWID: 20015921**

For the given problem statement, implementation of the three sorting algorithms viz. naïve insertion sort, improved insertion sort, and merge sort was done. Along with implementation, the runtime for each of the algorithm for different input sizes, and direction was observed.

The naïve insertion sort algorithm had a time complexity of O(n2), which does not fair well on most cases. As an improvement, the vector lengths were precomputed which reduced the complexity of always computing vector length for comparison in the loop. This in turn, reduced the runtime for the insertion sort approach. This precomputing vector lengths idea was implemented for merge sort algorithm as well.

Each algorithm was tested for n = 10, 25, 50, m = 10000, 25000, 50000, 100000, 250000, 500000, 1000000, 2500000, against sorted vectors(best case), random vectors(average case), and inversely sorted vectors(worst case). The runtime for these were noted and plotted in a scatterplot. Also using time complexity equations, the theoretical runtimes were also plotted for best fit.

**Theoretical runtimes:**

1. Insertion Sort:

|  |  |  |
| --- | --- | --- |
| Best Case (Sorted) | Average Case (Unsorted) | Worst Case (Inversely Sorted) |
| O(n) | O(n2) | O(n2) |

1. Merge Sort:

|  |  |  |
| --- | --- | --- |
| Best Case (Sorted) | Average Case (Unsorted) | Worst Case (Inversely Sorted) |
| O(n\*log(n)) | O(n\*log(n)) | O(n\*log(n)) |

**Observations:**

The runtime for each mentioned case was observed and is displayed in the following. Also, the scatterplots of these runtimes against theoretical ones are displayed.

1. **Naïve Insertion Sort**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Naïve Insertion Sort | | | | | | | | | | |
| m | n = 10 | | | n = 25 | | | n = 50 | | | |
| Random Vectors | Sorted Vectors | Inverse Sorted Vectors | Random Vectors | Sorted Vectors | Inverse Sorted Vectors | Random Vectors | Sorted Vectors | Inverse Sorted Vectors |
| 10000 | 1235.2 | 0.6 | 2280.9 | 2924.6 | 1.5 | 6889.8 | 6728.5 | 2.7 | 13193.9 |
| 25000 | 7247.5 | 1.2 | 13851 | 18577 | 3.4 | 41403.1 | 42444.5 | 7.2 | 74564.2 |
| 50000 | 33065.4 | 2.3 | 53971 | 81468.7 | 6.6 | 141079.2 | 183570.7 | 13.6 | 299571.5 |
| 100000 | 131953 | 4.8 | 222178.6 | 395642.1 | 14.3 | - | - | 29.4 | - |
| 250000 | 1293497.1 | 13.1 | - | - | 35.4 | - | - | 66.8 | - |
| 500000 | - | 25.4 | - | - | 73.2 | - | - | 129.2 | - |
| 1000000 | - | 46.3 | - | - | 151.2 | - | - | 274.2 | - |
| 2500000 | - | 116.8 | - | - | 337.5 | - | - | 568.7 | - |

1. **Improved Insertion Sort**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Improved Insertion Sort | | | | | | | | | |
| m | n = 10 | | | n = 25 | | | n = 50 | | |
| Random Vectors | Sorted Vectors | Inverse Sorted Vectors | Random Vectors | Sorted Vectors | Inverse Sorted Vectors | Random Vectors | Sorted Vectors | Inverse Sorted Vectors |
| 10000 | 87.2 | 0.5 | 157.6 | 83.8 | 1.2 | 150.3 | 130.3 | 2.5 | 147.6 |
| 25000 | 422.8 | 1.2 | 848.1 | 422.8 | 2.9 | 865.2 | 459.6 | 4.2 | 838.5 |
| 50000 | 1681.8 | 2.9 | 3529.3 | 1660.4 | 4.9 | 3485.3 | 1821.2 | 8.4 | 3310.2 |
| 100000 | 7205.2 | 5.3 | 14848.4 | 46020 | 9.5 | 14852.8 | 7719.7 | 13.2 | 14103.7 |
| 250000 | 48049.1 | 9.3 | 92689.1 | 46122.4 | 21.9 | 94749.8 | 55070 | 32.3 | 91955.1 |
| 500000 | 215913.3 | 15.8 | 384345.6 | 187059.5 | 34.4 | 414731.8 | 212860 | 60.7 | 382791.9 |
| 1000000 | - | 26.1 | - | - | 69.5 | - | - | 120.8 | - |
| 2500000 | - | 68.2 | - | - | 166.9 | - | - | 302.6 | - |

1. **Merge Sort**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Improved Insertion Sort | | | | | | | | | |
| m | n = 10 | | | n = 25 | | | n = 50 | | |
| Random Vectors | Sorted Vectors | Inverse Sorted Vectors | Random Vectors | Sorted Vectors | Inverse Sorted Vectors | Random Vectors | Sorted Vectors | Inverse Sorted Vectors |
| 10000 | 11.6 | 18 | 12.3 | 19.2 | 20.8 | 17.2 | 22.5 | 29.4 | 30.8 |
| 25000 | 27.7 | 34.9 | 33.7 | 36.6 | 37.5 | 40.9 | 46.1 | 48.7 | 50.9 |
| 50000 | 47.2 | 53 | 55 | 67.2 | 64.7 | 68.4 | 85.7 | 84.6 | 85 |
| 100000 | 89.4 | 91.6 | 90.6 | 128.5 | 124.2 | 119.2 | 169.9 | 164.1 | 162.1 |
| 250000 | 255.9 | 213.1 | 220.9 | 321.6 | 288.1 | 287.8 | 432 | 405.6 | 401.7 |
| 500000 | 459.8 | 429.7 | 428.7 | 643.7 | 593 | 590.2 | 884 | 833.2 | 838.8 |
| 1000000 | 995.9 | 913.3 | 874.9 | 1312.5 | 1230.3 | 1217.1 | 1826.7 | 1723.9 | 1724.9 |
| 2500000 | 2534.5 | 2364.1 | 2436.7 | 3587.7 | 3357.3 | 3258.5 | 6993.2 | 5997.1 | 6131.3 |

**Threshold:**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | n = 10 | | | n = 25 | | | n = 50 | | |
| Random Vectors | Sorted Vectors | Inverse Sorted Vectors | Random Vectors | Sorted Vectors | Inverse Sorted Vectors | Random Vectors | Sorted Vectors | Inverse Sorted Vectors |
| Naïve Insertion Sort | 250,000 | - | 100,000 | 100,000 | - | 50,000 | 50,000 | - | 50,000 |
| Insertion Sort | 500,000 | - | 500,000 | 500,000 | - | 500,000 | 500,000 | - | 500,000 |
| Merge Sort | - | - | - | - | - | - | - | - | - |

**Plots:**

|  |  |  |
| --- | --- | --- |
| Naïve Insertion Sort | | |
| n = 10 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |
| n = 25 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |
| n = 50 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |

|  |  |  |
| --- | --- | --- |
| Improved Insertion Sort | | |
| n = 10 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |
| n = 25 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |
| n = 50 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |

|  |  |  |
| --- | --- | --- |
| Merge Sort | | |
| n = 10 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |
| n = 25 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |
| n = 50 | Random vector |  |
| Sorted vector |  |
| Inverse Sorted vector |  |

**Analysis:**

1. Average time complexity of improved insertion is O(n2) but precomputing the vector lengths made a significant difference in the runtime performance of insertion sort algorithm.
2. For naïve insertion sort, observed performance is worse than theoretical.
3. For improved insertion sort, observed performance fared better than theoretical.
4. Merge sort’s observed performance is much better than the theoretical.
5. The Naïve Insertion sort performs well for the best case but is significantly slower for the average and worse case.
6. Naïve Insertion Sort reaches threshold significantly early on.
7. The input constraints heavily affect the performance for both insertion sort problems.
8. Improved insertion sort has way better performance than the naïve one. It does extremely well for best case. Also, it works efficiently for lower input constraints but drastically worsens for higher input constraints.
9. Improved insertion sort reaches threshold slower than naïve insertion sort but much faster than merge sort.
10. Merge sort has persistent O(n\*log(n)) time complexity for all the three cases.
11. Merge sort performs very well for average and worse cases but is significantly slower for best cases.
12. Merge sort shows gradual increase in runtime with increase in input constraints.

**Deductions:**

1. Precomputing vector lengths makes significant improvements in the runtime performance of both insertion sort and merge sort algorithms.
2. Naïve insertion sort performs the worst out of the three for the average and worse cases, but performs better than merge sort for best case. It is the least favorable algorithm when considering runtime performance.
3. Improved insertion sort is a well-performing algorithm but is limited to input constraints. It is the ideal choice for nearly sorted or sorted vectors. It fares well for lower input but has drastic change in performance with increase in input.
4. Merge sort performed the best for average and worst cases. It has the same performance for all three cases so is not an ideal selection for sorted arrays. It does not have a drastic change in performance with increase in input so is a great choice when working with higher input constraints.