Homework Assignment 1

**Abstract**

The analysis of an algorithm is the process of finding the computational complexity like time, space, and other resources that the algorithm requires. Resources such as memory, communication bandwidth, or computer hardware are of primary concern, but most often it is computational time that we want to measure to analyze the algorithm. The main goal of this report is to analyze the running time of insertion sort (naïve and improved) and merge sort for different input sizes ranging from 1000 to 2500000 and different vector dimensions. We’ll also be using differently ordered input vectors to check the performance of the algorithms and we’ll be running a test at least 10 times to compute the average running time of the algorithm for different input, input size, and vector dimensions.

**Introduction:**

The running time of an algorithm can be described as the function of the size of the input. The size of the input is usually considered as the number of items in the input (array size n for sorting).

The time complexity of the insertion sort depends on the size of the input. The time taken by insertion sort grows with the size of the input. Moreover, it takes different amounts of time to sort two same-size input sequences depending on how nearly sorted they already are.

Merge sort uses the divide and conquer strategy to sort a sequence. The n-element array to be sorted is divided into two sub-arrays of n/2 elements each and the two sub-arrays are sorted recursively using merge sort. We then merge the two sorted sub-arrays to produce the sorted answer.

We have three implementations of the sorting algorithms: naïve insertion sort, modified insertion sort, and merge sort. We have used different testing vectors of size m = 1000, 2500, 5000, 10000, 25000, 50000, 100000, 250000 and dimension n = 10, 25, 50. We have used random, sorted and inverse sorted vectors to further check the performance of these algorithms for different combinations of size, dimension, and order.

We’ll present and analyze the results of the three sorting algorithms in the next section and check how well they perform and whether they exhibit the normal standard behavior for that sort. We’ll also compare the running time of the algorithms with the best-case and worst-case time complexities to see how much their results deviate from the standard time complexities.

**Results and Evaluation:**

The average runtime of naïve insertion sort for different parameters is described in the table below.

Insertion Sort Runtime in ms (Average Summary)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 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 |
| 1000 | 11.5 | 0.1 | 23 | 28.8 | 0.2 | 57.5 | 57.5 | 0.3 | 119.875 |
| 2500 | 73.2 | 0.3 | 151.1 | 186.2 | 0.4 | 365.6 | 362.6 | 0.8 | 725.25 |
| 5000 | 297.8 | 0.6 | 585 | 727.1 | 0.7 | 1443.4 | 1443.8 | 1.3 | 2880.6 |
| 10000 | 1173.9 | 0.8 | 2326.33 | 2882 | 1.3 | 5756 | 5754.4 | 2.3 | 11476 |
| 25000 | 7334.9 | 1.1 | 14468.6 | 17918.66 | 2.8 | 35989 | 36325 | 5.9 | 72063 |
| 50000 | 29259.2 | 2.2 | 57849.66 | 72961 | 6 | 145150 | 147557 | 11.6 | 287556.5 |
| 100000 | 117590 | 5 | 233716 | 305902 | 11.7 | - | - | 23.8 | - |
| 250000 | 818848 | 12.2 | - | - | 30.1 | - | - | 63.3 | - |

As we can see, the running time of the insertion sort increases as the size of the input as well as the dimension of the vector array increases. The time complexity for the insertion sort is the best case when we pass a sorted vector as the input and the worst-case time complexity is when we pass a reverse sorted vector as input.

Below is the Input Size vs. Runtime comparison for different vector dimensions along with the best-case and worst-case time complexity.

From the above results, the insertion sort does not perform well as the size of the input increases. We’re not able to compute the running time for the sorting algorithm when the input size becomes greater than 100000 for n=10. For n=25, 50, the running time cannot be computed for input size greater than 50000.

For sorted input vector, the insertion sort performs well which is the best-case scenario. The running time of the sorted vector input is equivalent to best case time complexity O(n). For the random and inverse vector sort, the running time is almost doubled as we increase either the input size n or the vector dimension n and is comparable to the worst-time complexity O(n2) for insertion sort.

The average runtime of improved insertion sort for different parameters is described in the table below.

Improved Insertion Sort Runtime in ms (Average Summary)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 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 |
| 1000 | 0.8 | 0.1 | 1.6 | 1 | 0.1 | 1.4 | 0.8 | 0.1 | 1.2 |
| 2500 | 4 | 0.1 | 8.2 | 4.6 | 0.4 | 8.6 | 4.6 | 0.4 | 8 |
| 5000 | 16 | 0.2 | 32.6 | 16.66 | 0.6 | 32.6 | 17 | 0.6 | 32.8 |
| 10000 | 64.6 | 0.4 | 136.2 | 66 | 0.9 | 136.8 | 67 | 1.6 | 136.6 |
| 25000 | 416.33 | 0.9 | 837.6 | 418.66 | 1.6 | 828.8 | 421.4 | 3 | 838.8 |
| 50000 | 1664 | 1.3 | 3344.2 | 1667.33 | 3.4 | 3347 | 1680.33 | 6.2 | 3350 |
| 100000 | 6664 | 3 | 13354 | 6681.33 | 6.6 | 13351.33 | 6680 | 12 | 13352.5 |
| 250000 | 41640 | 7.4 | 84117.5 | 41983 | 15.8 | 84182.5 | 41732.5 | 32.2 | 83682.5 |

The improved insertion sort performs better than the naïve insertion sort, but the behavior of the improved insertion sort is similar in terms of the size of the input. As the input size increases the running time of the algorithm increases as well. The dimension of the input vector does not impact the running time of the insertion sort as much. The sorted input vector is still the best-case scenario and the inverse sorted vector is still the worst-case scenario and equivalent to best-case and worst-case time complexities.

Below is the Input Size vs. Runtime comparison for different vector dimensions along with the best-case and worst-case time complexity for insertion sort.

The above results show that the improved insertion sort has a better running time than the naïve insertion sort and exhibits the standard insertion sort behavior. We’re successfully able to compute the running time of the insertion sort for all the given input sizes and dimensions. The vector dimension size does not have much of an impact on the running time of the algorithm for higher values of input size since we pre-compute the length of the vector before sorting the elements. However, as we increase the input size, the running time of the algorithm increases exponentially. The sorted input vector still has the best performance among all the order of inputs and is equivalent to the best-case time complexity and is the only scenario where the vector dimension and input size has only a slight impact on the running time.

The average runtime of merge sort for different parameters is described in the table below.

Merge Sort Runtime in ms (Average Summary)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 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 |
| 1000 | 0.1 | 0.1 | 0.2 | 0.2 | 0.4 | 0.1 | 0.4 | 0.2 | 0.4 |
| 2500 | 0.8 | 0.3 | 0.5 | 0.4 | 0.6 | 0.3 | 0.8 | 0.8 | 0.6 |
| 5000 | 0.9 | 0.8 | 0.9 | 1.4 | 0.8 | 0.8 | 1.6 | 1 | 1.2 |
| 10000 | 2 | 1.7 | 1.8 | 2.8 | 2 | 2 | 3 | 2.8 | 2.4 |
| 25000 | 5.1 | 4 | 4 | 6 | 4.8 | 5 | 7.2 | 6.4 | 6.6 |
| 50000 | 11.5 | 8.4 | 8.5 | 12.4 | 10.2 | 10 | 15.4 | 12.6 | 12.8 |
| 100000 | 21.3 | 17 | 17.2 | 25 | 20 | 20.4 | 31.8 | 26.8 | 27.2 |
| 250000 | 55.3 | 43.6 | 43.8 | 67.6 | 54 | 53.6 | 86.8 | 71.4 | 71.6 |

The merge sort performs better than both naïve and improved insertion sort. As the input size increases, there is only a slight increase in the running time of the algorithm. The order of the input also does not have much impact on the running time of the merge sort.

Below is the Input Size vs. Runtime comparison for different vector dimensions along with the worst-case time complexities for merge sort.

Merge sort has the best running time among all the three sorting algorithms. The order of the input vector has little to no impact on the running time of the merge sort. The size of the vector dimension also does not have a huge impact on the running time of the algorithm. The biggest attributing factor for merge sort would be the input size. As the size of the input vector increases, the running time of the algorithm also increases. However, the increase in the running time is not as enormous for merge sort as that of insertion sort. This is mainly due to the divide and conquer strategy implemented by the merge sort and hence performs well even for high input sizes.

**Conclusion:**

We’re successfully able to test different sorting algorithms for different order, input, and dimension sizes and compare the efficiency of these algorithms. From the above results and analysis, we can conclude that the merge sort is the most efficient algorithm when the input size is large, and the dimension and order of the input vector do not impact the efficiency of the merge sort too much. For small input sizes, the insertion sort performs well but as the size of the input increases, the running time of the insertion increases rapidly. For the naïve insertion sort, the dimension of the input vector also has a large impact as we’re computing the length of each vector within the insertion sort algorithm. This is not the case for improved insertion sort as we’re precomputing the length of the input vector before sorting the input. That’s why the input dimension does not impact the running time of the improved insertion sort very much. The order of the input also impacts the running time of the insertion sort. The sorted input vector is the best-case scenario for insertion sort and hence performs well for the sorted input. However, the inverse sorted vector input is the worst-case scenario and has the worst-case time complexity for insertion sort.

The above test results agree with the time complexity for the insertion and merge sort and the behavior is as expected. We’re not able to compute the running time of the insertion sort for certain cases which is due to the machine constraints. We should be able to compute these cases as well if we run the algorithms on high performance computers. That might improve the running time for some other cases as well, but the general behavior of these sorting algorithms would still be the same.