Assignment 1: Sorting Running Time

# **Abstract:**

I have implemented and analyzed the time-complexity of Naïve Insertion Sort, Improved Insertion Sort and Merge Sort. By comparing the run-time of the random vector, sorted vector and reverse sorted vector, I have documented the results and plotted the graphs of runtime for a better understanding of these algorithm's running times.

# **Result:**

1. Naïve Insertion Sort



\* Hyphen (-) indicates that the running time is more than 300000ms.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Graph 1.1 Graph 1.2

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Graph 1.3 Graph 1.4

1. Improved Insertion Sort



Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Graph 2.1 Graph 2.2

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Graph 2.3 Graph 2.4

1. Merge Sort



Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

Graph 3.1 Graph 3.2

Chart, line chart, scatter chart

Description automatically generatedChart, line chart

Description automatically generated

Graph 3.3 Graph 3.4

# **Discussion:**

1. **Naïve Insertion Sort:**

* In this algorithm, whenever we swap vectors for sorting, we calculated the ‘ivector\_length’ every time and hence it is not much efficient.
* The given naïve insertion sort algorithm has a time-complexity of O(m2 x n). The time taken for sorting is much higher and not suitable for large data and can be inferred from Table 1.
* Running time is directly proportional to the input size, running time increases as the size of the input increases and can be seen from Graphs 1.1 – 1.4.
* There is an exponential growth in the running time of Random sorted vector and Reverse Sorted vector. For high values of ‘m’ and ‘n’, running time is extremely high which is one of the biggest limitations of this algorithm.
* For n = 25, 50, 75 and m = 100000, 500000 execution time is nearly or more than 300000 ms for Reverse Sorted Vector and Random Sorted Vector. Also, for m = 500000 and n = 10 the running time is more than 30000 ms hence, are the threshold values of m and n.
* ‘m’ has more impact on the running time as compared to ‘n’ and can be deducted from Table 1 and Graphs 1.1 – 1.4.
* We can improve the time-complexity of this algorithm from O(m2 x n) to O(m2).

1. **Improved Insertion Sort**

* In this algorithm, instead of calculating the ‘ivector\_length’ every time, I created an integer vector named ‘sums’ in which I store the ‘ivector\_length’ of all the vectors. Each time I swap the vectors for sorting, I also swap the corresponding sum in the vector ‘*sums’*.
* Time-complexity of creating vector ‘*sums*’ is O(m x n).
* Improved insertion sort executes faster than Naïve insertion sort and has a time-complexity of O(m2) and can be inferred from Table 2.
* Running time is directly proportional to the input size, running time increases as the size of the input increases and can be seen from Graphs 2.1 – 2.4.
* It is not possible to reduce the time-complexity anymore.
* ‘m’ has more impact on the running time as compared to ‘n’.

1. **Merge Sort**

* Merge Sort is a recursive algorithm; hence every time call we function *merge\_sort* ‘ivector\_length’ is calculated for every vector and so it is also not much efficient.
* In this algorithm, I created an integer vector ‘sums’ to store the ‘ivector\_length’ of all the vectors and declared it as a *global variable*.
* Time-complexity of creating vector ‘*sums*’ is O(m x n).
* I changed the header file ***sort.h*** and added the variable and method there. Also changed the same in ***main.cpp***
* Merge Sort executes extremely faster than Naïve or Improved insertion sort and has a time-complexity of O(m x log(m)) and can be inferred from Table 3.
* Running time is directly proportional to the input size, running time increases as the size of the input increases and can be seen from Graphs 3.1 – 3.4.
* ‘m’ has more impact on the running time as compared to ‘n’.
* My Merge Sort algorithm gives “Segmentation Fault” for m >= 700000 and n = 10 and are the threshold values for ‘m’ and ‘n’.

# **Conclusion:**

From the above results I conclude that Merge Sort is the fastest algorithm among the given 3 algorithms. Insertion sort works better when we have smaller input size to sort, or the input vector is already sorted. Improved Insertion sort reduces the running time to much extent but still it is not feasible for larger input size. For Merge sort there is minor difference between running time of Sorted Vector and Inverse Sorted vector. Also, it is preferred for larger input size.