Study and Analysis of Self-Tuning Sort with Traditional Sorting Algorithms

Abstract:

The aim of project is to compare the processing time of the existing sorting algorithm with the proposed self-tuning sorting technique. As part of the self-tuning sort, multiple random arrays of sizes ranging from 10 to 10,000 have been generated and sorted with few of the traditional algorithms along with the proposed self-tuning sort. To understand the behavior of the proposed algorithm, increasing arrays (i.e., sorted array generated after sorting random array) and decreasing arrays (reverse of the sorted array) were also sorted and the processing time was also recorded. And, to know whether the statistical distributions – Poisson Distribution, Normal Distribution have any impact on the processing time, sorting on a random input size with various statistical distributed parameters have been also analyzed.

Introduction:

All the traditional sorting algorithms sorts the array either by sequentially comparing the neighboring elements, dividing the array based on the mid value, or checking with the element which is after the given gap interval. While understanding and analyzing the sorting algorithms for identifying a new sorting approach, we came across an idea of performing pre-analysis on the input data before directly starting the sorting. Using the results that are identified in the analysis, our proposed algorithm sorts the given input array based on the observations.

Problem Definition:

If an input array has a minimum of 2 elements, we can always define whether the given array is an increasing array, decreasing array, or an equal array. From this idea, we came up with the approach of splitting the array into a group of sub-arrays such as an increasing subarray, decreasing subarray, and a random/equal subarray. And finally, the proposed algorithm sorts the array while grouping every individual subarrays.
Also, statistical analysis has to be carried out to analyze the impact of time efficiency and behavior of the algorithm when the distribution of the array is getting changed (for a constant input size).

Solution:

Self-Tuning Sort algorithm linearly compares the current element with the previous element, and based on the comparison condition, algorithm either splits the array or appends the current value to the existing sub array if the condition is matching. Finally, after generating multiple sub arrays, using merge results approach of Merge Sort's algorithm, algorithm groups and sorts the given input array.

For example, let us consider the array [39, 45, 51, 47, 43, 3, 9]
Self-Tuning Sort's split array functionality will start its comparison from the 1st index. 1st index value will be compared with the 0th index value, and as there is no flag and as the current value is greater than previous value, algorithm flags the comparison condition as the Increasing Array condition. And, adds the 1st index to a temporary list.

Current temporary list (Increasing array) = [39, 45]
Global List = []

2nd index value will be compared with the 1st index value, and as the current flag condition is Increasing Array and as the 2nd index value is greater than 1st index value, adds the 2nd index to the current temporary list.

Current temporary list (Increasing Array) = [39, 45, 51]
Global List = []
3rd index value will be compared with 2nd index value, and as the current flag condition is Increasing Array and as the 3rd index value is not greater than 1st index value, the existing temporary list is added to the global list and the current temporary list will be first cleared and the current element will be added to the current temporary list

Current temporary list (Increasing Array) = [47]
Global List = [[39, 45, 51]]

4th and 5th index values are lesser than 3rd and 4th index and are also matching to the flag condition, so the both the values will be added to the temporary list. While adding the value to the current list, every time current value will be added at the 0th index of the array.

Current temporary list (Reversed Decreasing Array) = [3, 43, 47]
Global List = [[39, 45, 51]]

Now 6th index value is compared with 5th index value, and as the current flag condition is Decreasing Array and as the comparison condition is not matching, existing temporary list will be added to the Global List and the current value will be added to current temporary list.

Current temporary list (Increasing Array) = [9]
Global List = [[39, 45, 51], [3, 43, 47]]

As we have reached the last index, and as there is a value in the temporary list, remaining values in the temporary list also will be added to Global List and the same will be returned.

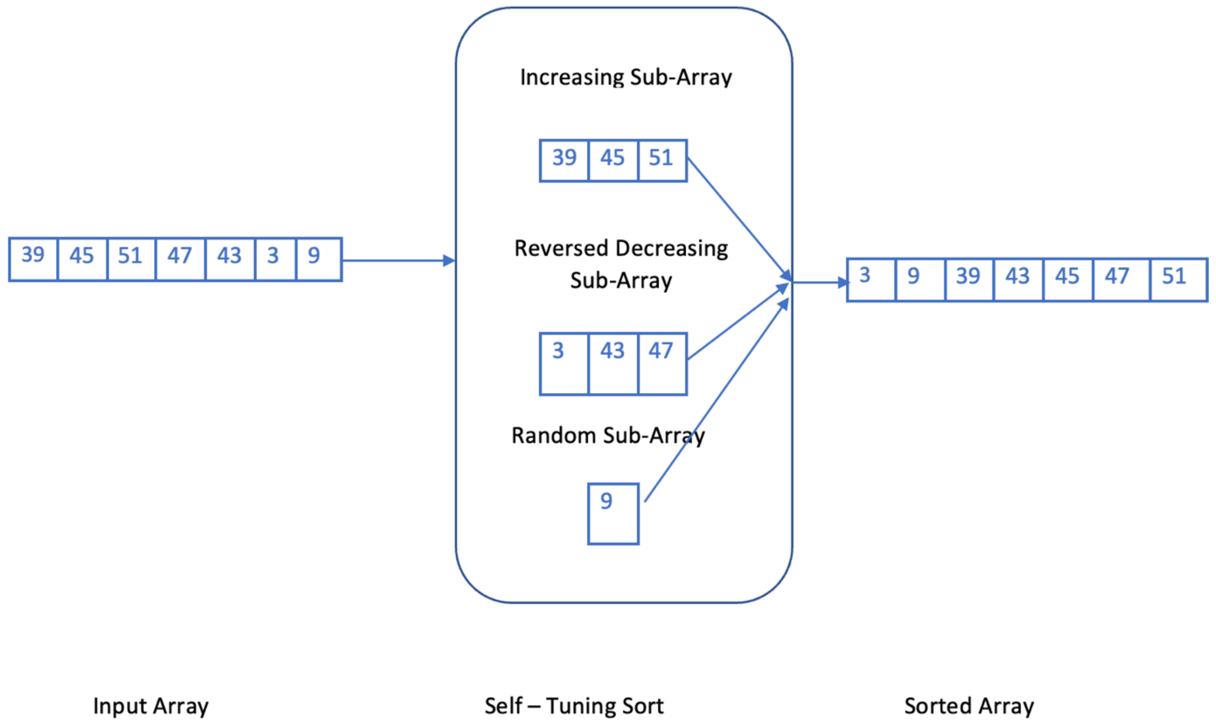
Global List = [[39, 45, 51], [3, 43, 47], [9]]

Total time complexity for splitting the array is equal to O(n)

Then the sub arrays will be further grouped and sorted using Merge functionality of Merge Sort's algorithm.

Total time complexity of merging sub arrays is equal to O(log m) where m is equal to the total number of sub arrays

Below figure displays the approach of self-tuning sort functionality for the above example

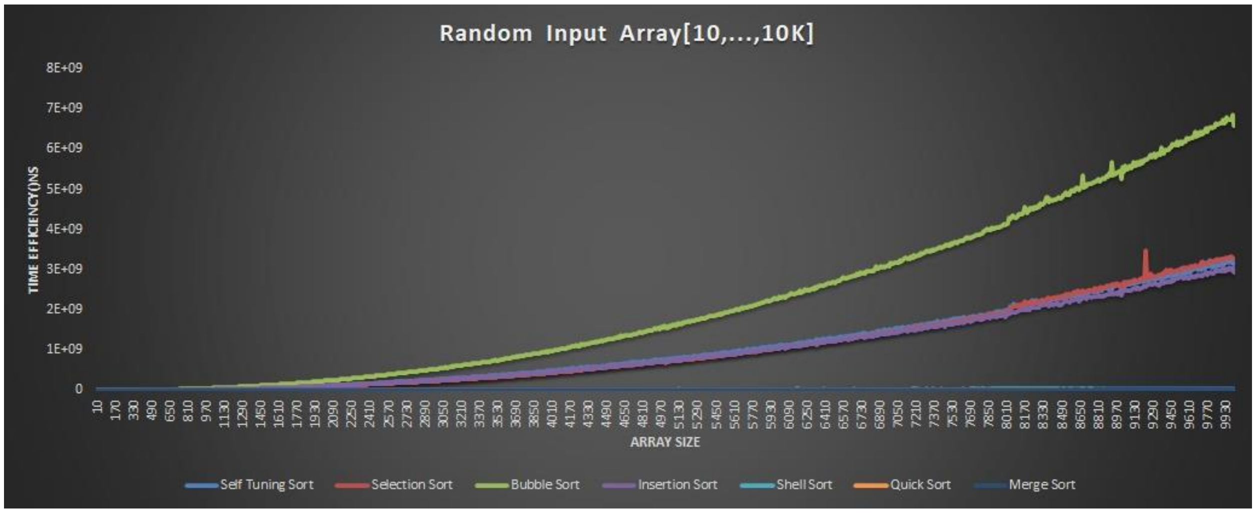


Performance Evaluation:
To evaluate the performance and analyze the time efficiency of the proposed solution, we performed the sorting experiment by generating random array with sizes from 10 to 10000 with an increment of 10 every time after 10 iterations. Sorted array that is generated after sorting the random array, i.e., increasing array and reverse of the sorted array, i.e., decreasing array also will be sorted using self- tuning and traditional sorting algorithms.

Time Efficiency Units: Nanoseconds
Data Considered for performing analysis: Average Time

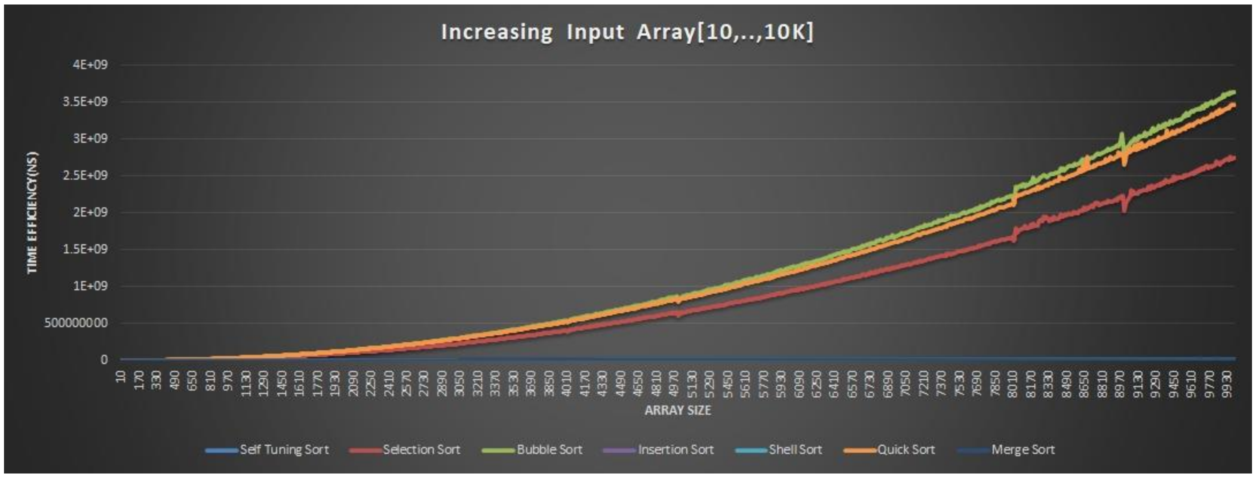
<u>Comparison of Processing Time of Self -Tuning sort with Traditional Algorithms:</u>

Random Array Input:



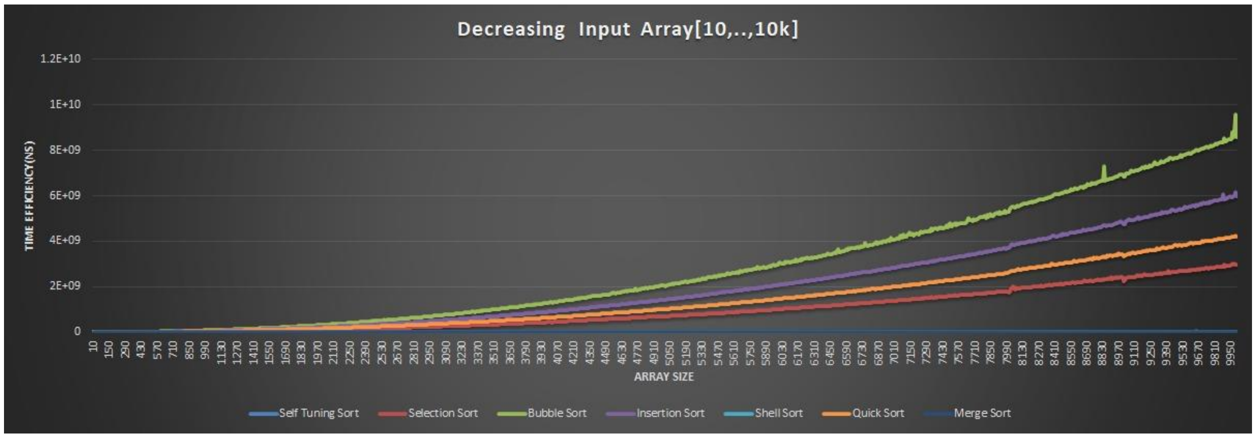
Analysis: Self – Tuning sort is performing better than Bubble Sort and is almost near to Selection Sort and Insertion Sort.

Increasing Array Input:



Analysis: Self tuning Sort performs better than other traditional algorithms.

Decreasing Array Input:



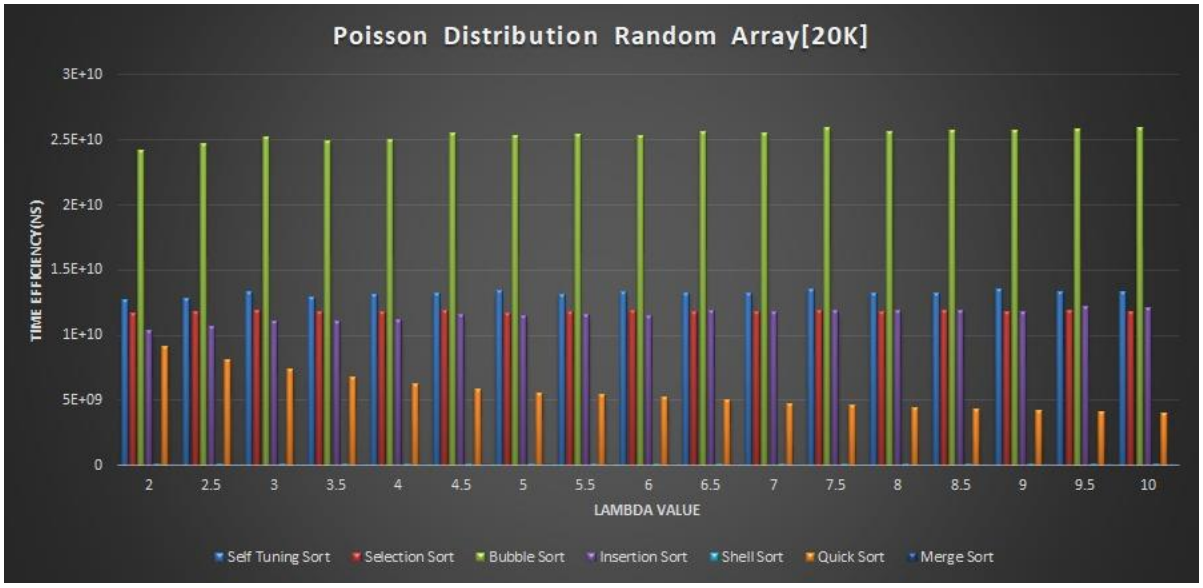
Analysis: Self tuning Sort performs better than other traditional algorithms.

<u>Comparison of Processing Time of Self – Tuning sort with Traditional Algorithms using Statistical methods/Approach:</u>

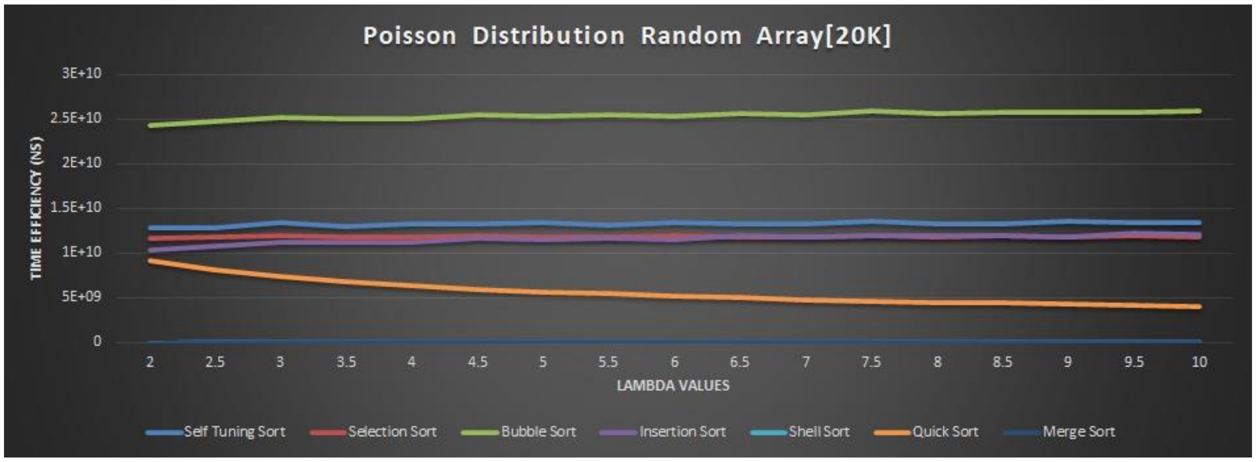
Poisson Distribution:
The Poisson distribution depends on the parameter ⅄ (lambda). In this distribution we varied ⅄ (lambda) from 2 to 10 in the interval of 0.5 and the fixed array size to 20000.

Random Array Input:

Bar Graph:



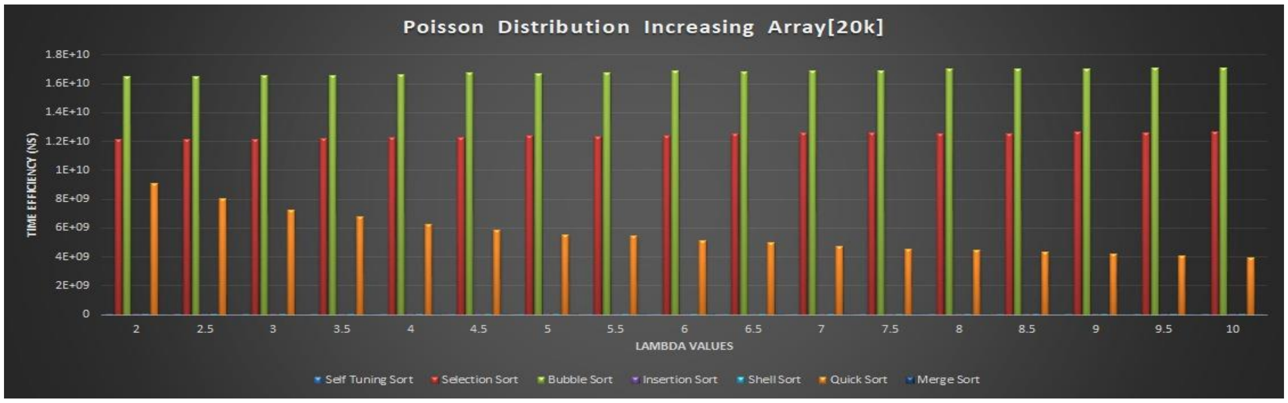
Line Graph:



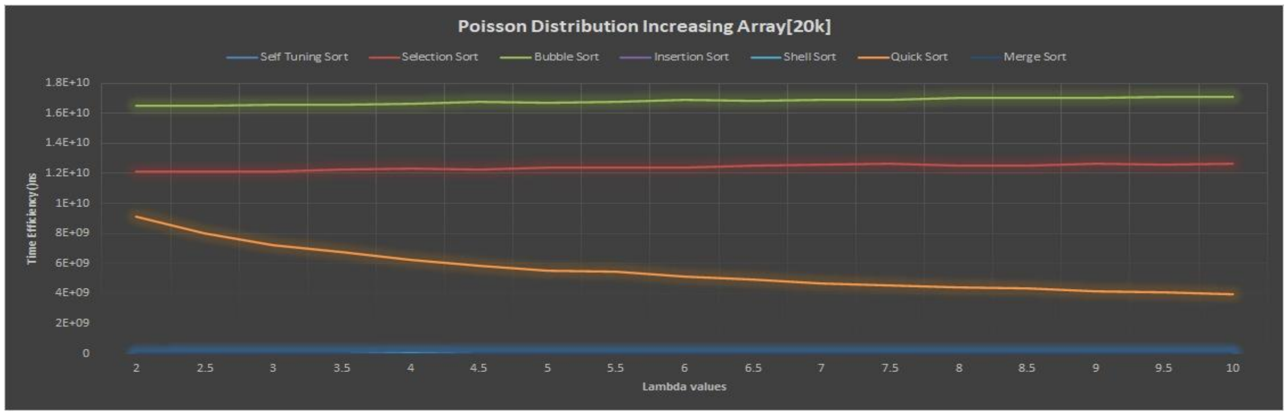
Analysis: Self – Tuning sort is performing better than Bubble Sort and is almost near to Selection Sort and Insertion Sort.

Increasing Array Input:

Bar Graph:



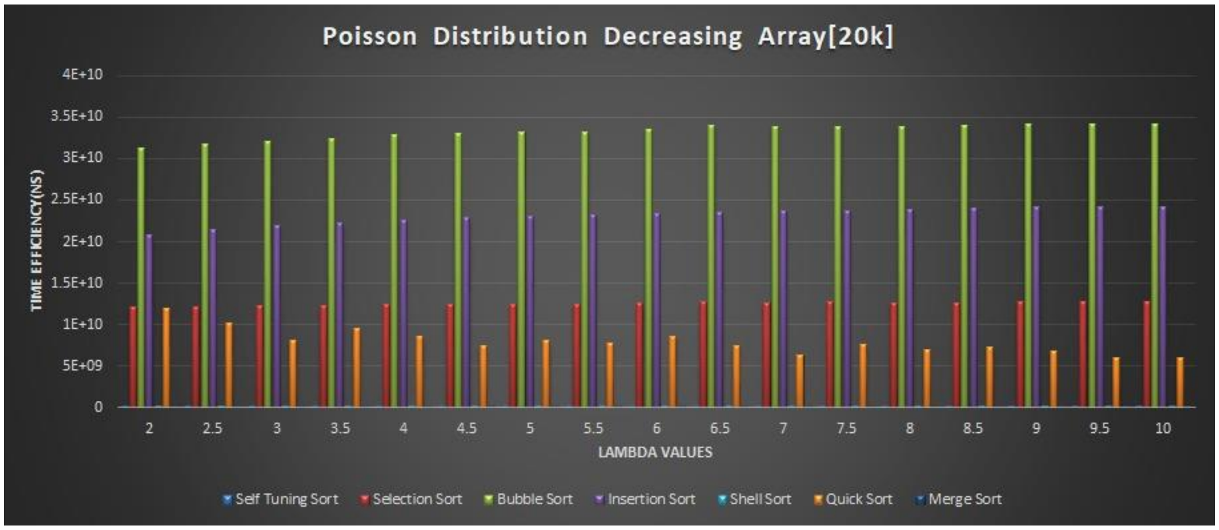
Line Graph:



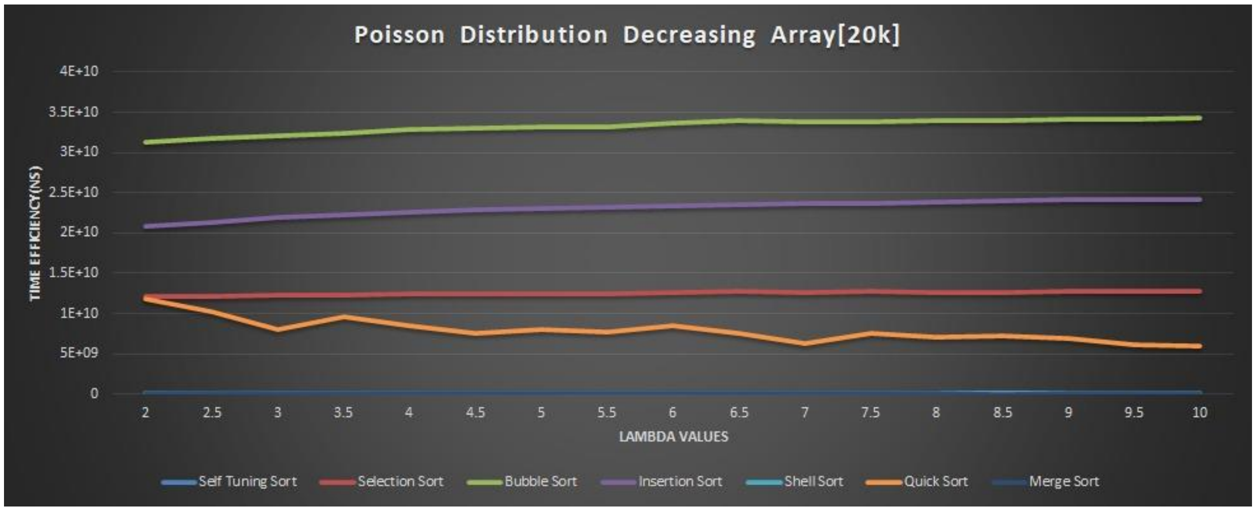
Analysis: Self tuning Sort performs better than other traditional algorithms.

Decreasing Array Input:

Bar Graph:



Line Graph:



Analysis: Self tuning Sort performs better than other traditional algorithms.

Normal Distribution:

This distribution depends on two parameters mean and variance or standard deviation (σ) i.e., (μ, σ^2). In this distribution we have performed two experiments.

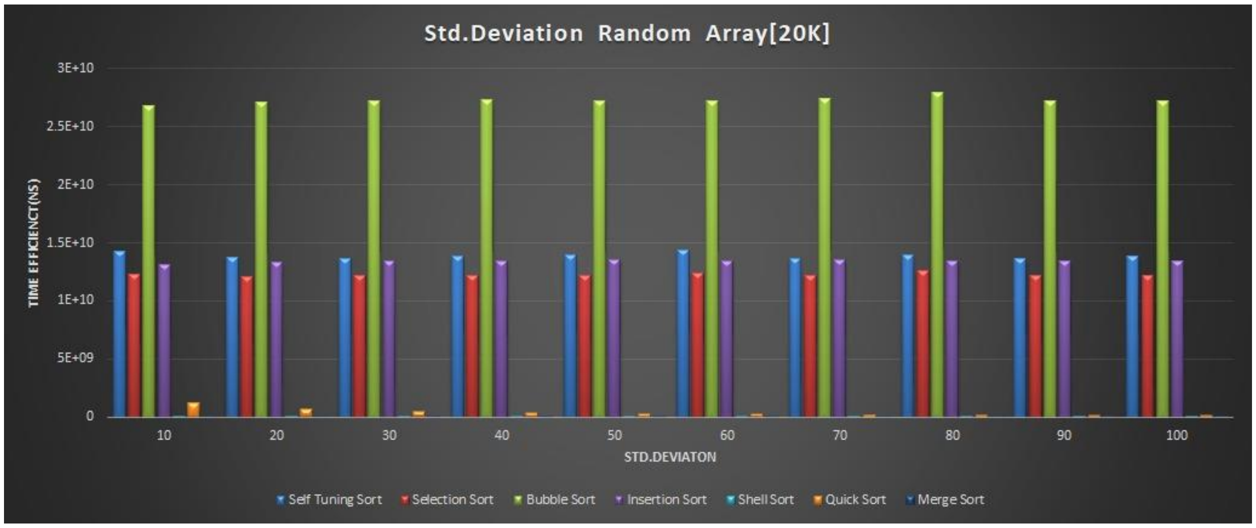
1. Mean (μ) constant and varied variance.
2. Varied mean and constant variance or standard deviation.

Case1:

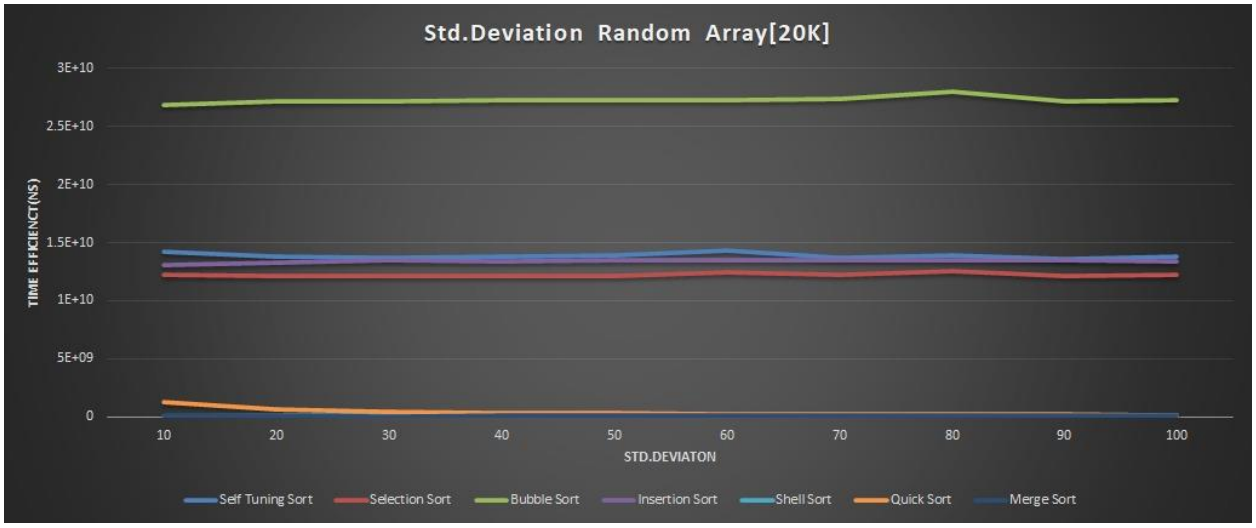
We kept mean as constant (50) and varied variance/standard deviation from 5 to 50 in the interval of 10 and fixed the size as 20000.

Random Array Input:

Bar Graph:



Line Graph:

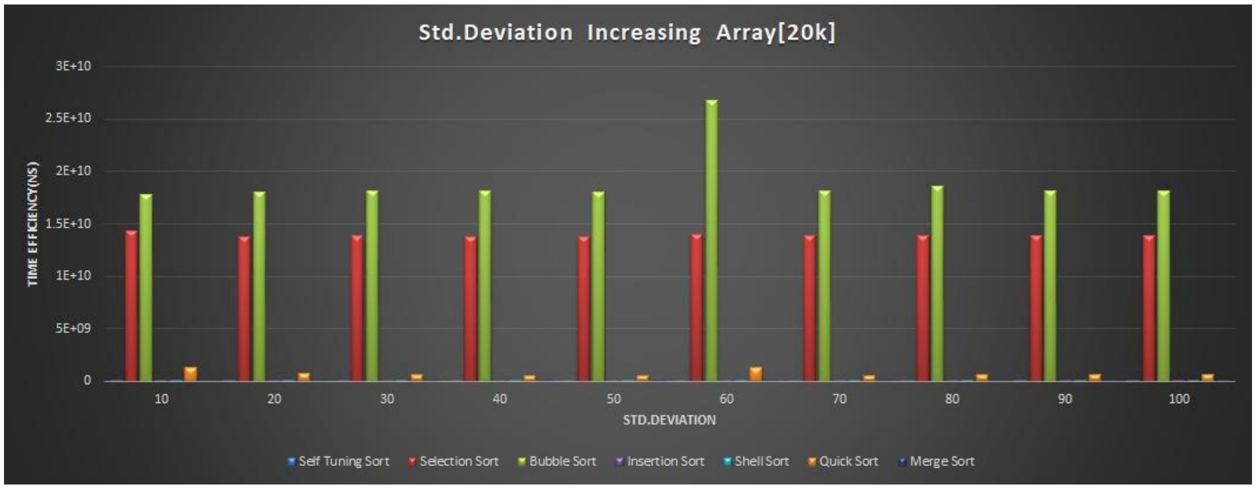


Analysis:

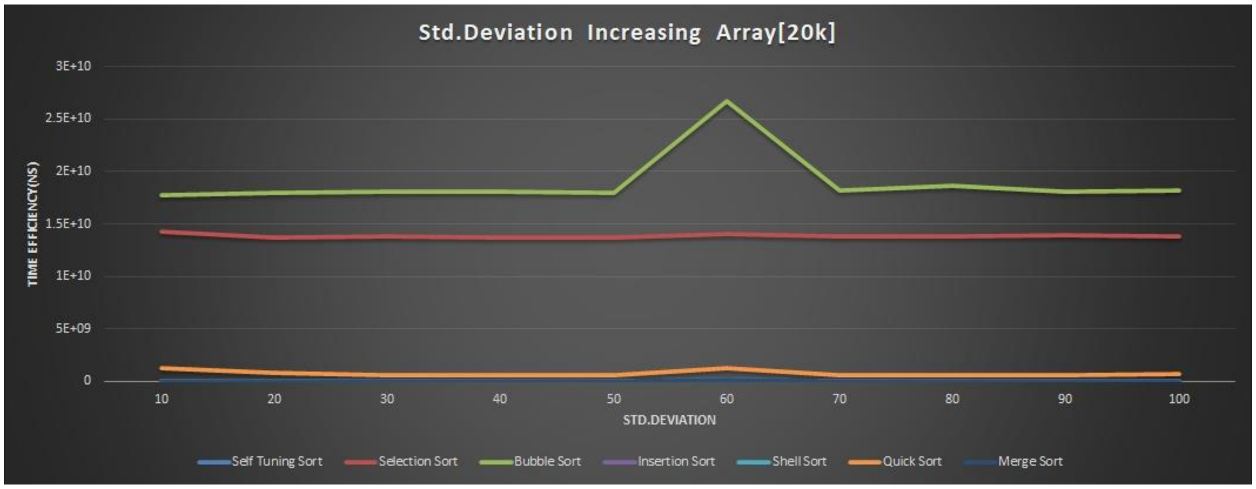
Self – Tuning sort is performing better than Bubble Sort and is almost near to Selection Sort and Insertion Sort.

Increasing Array Input:

Bar Graph:



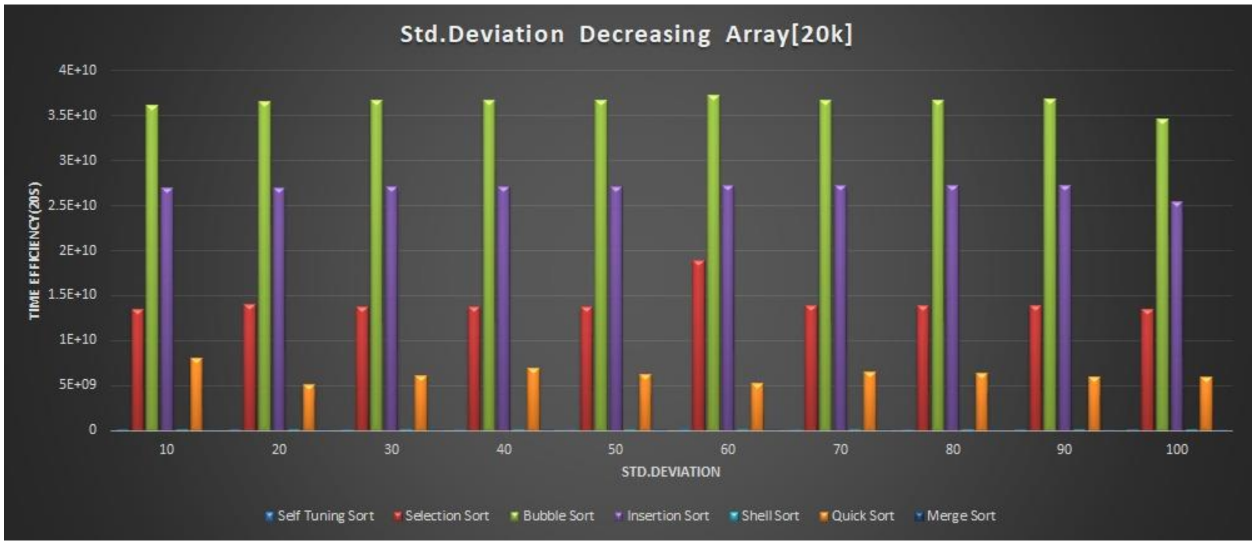
Line Graph:



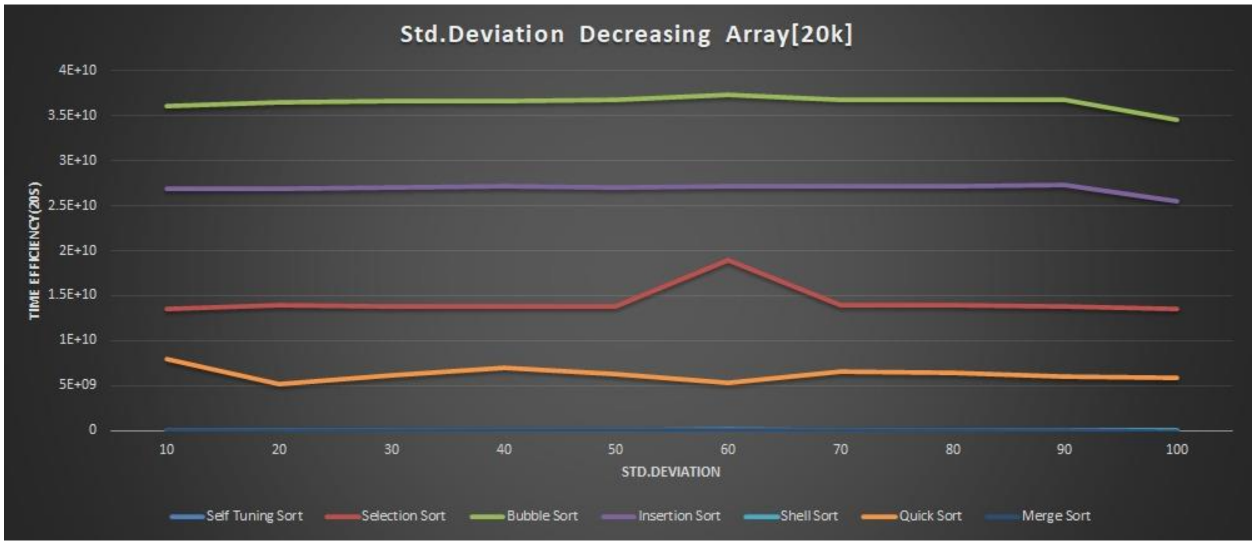
Analysis: Self tuning Sort performs better than other traditional algorithms.

Decreasing Array Input:

Bar Graph:



Line Graph:

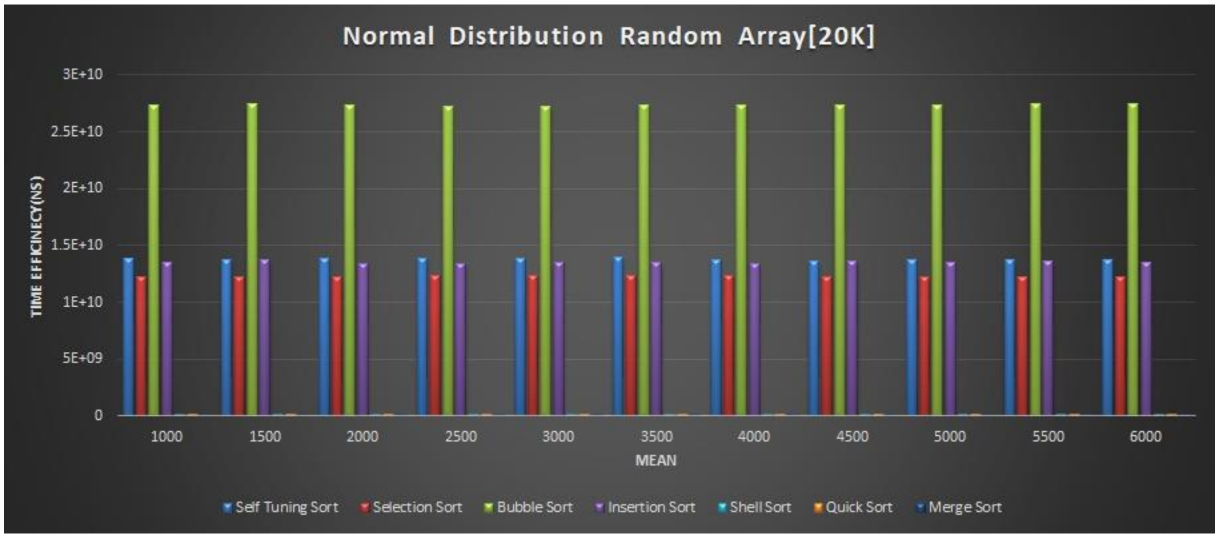


Analysis: Self-tuning Sort performs better than other traditional algorithms.

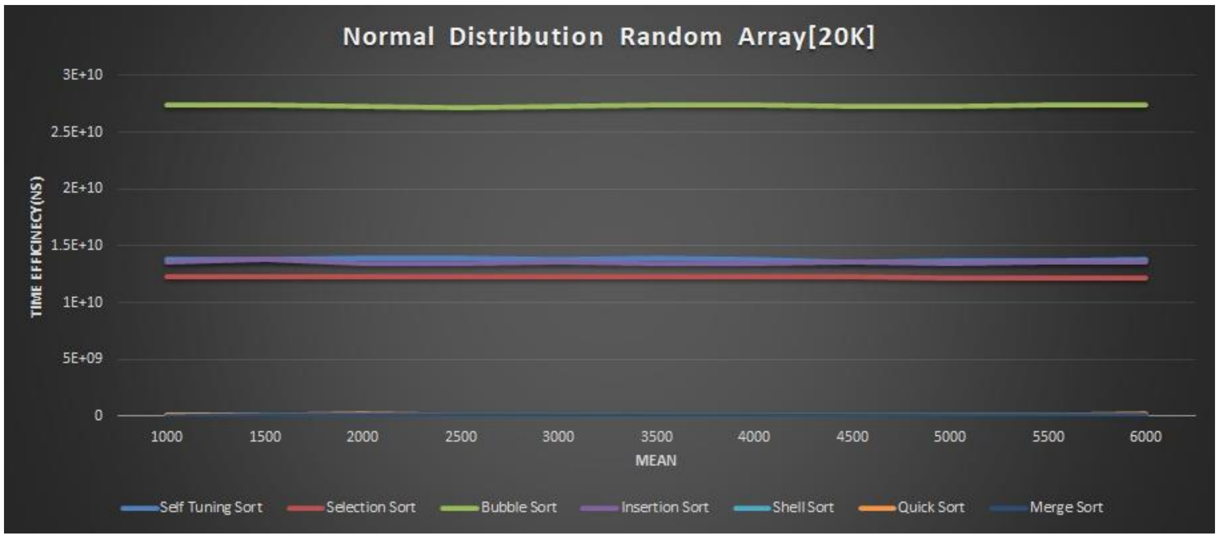
Case2:
We kept varied mean from 1000 to 6000 in the interval 500 and variance as constant (5) and fixed the size as 20000.

Random Array Input:

Bar Graph:



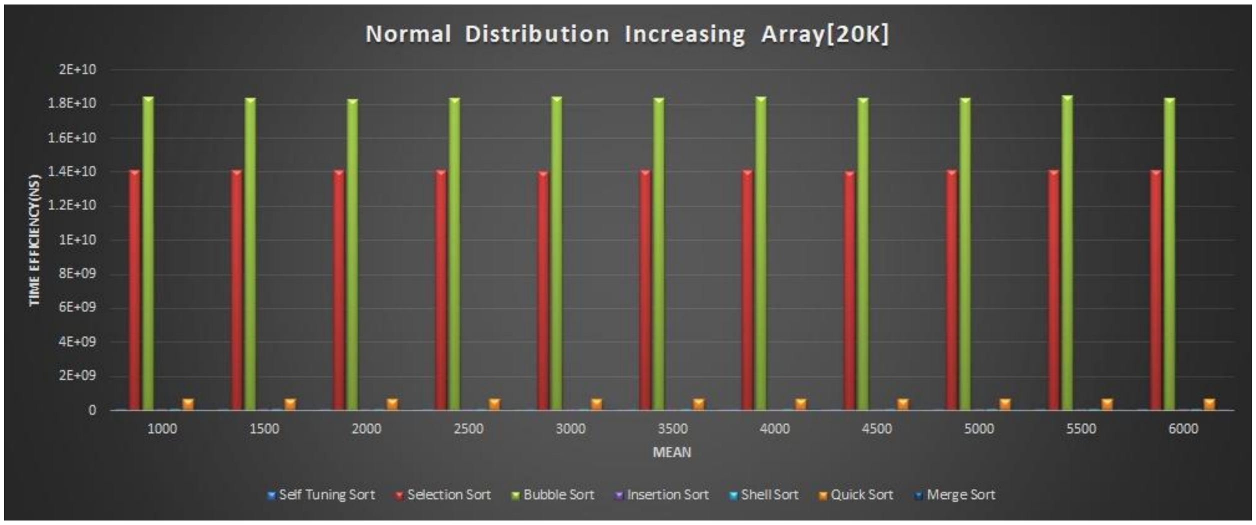
Line Graph:



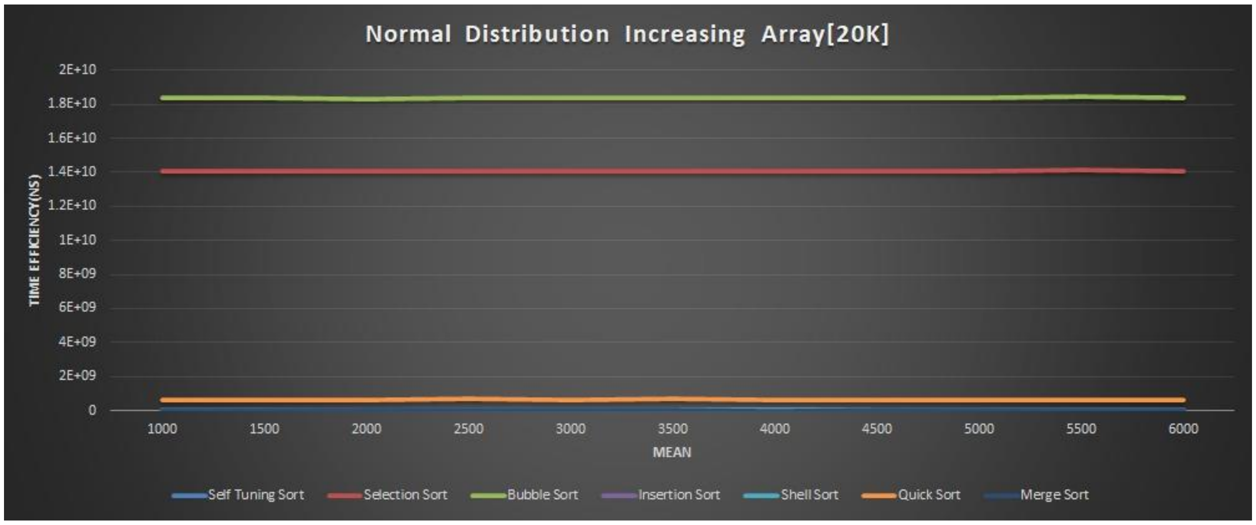
Analysis: Self – Tuning sort is performing better than Bubble Sort and is almost near to Selection Sort and Insertion Sort.

Increasing Array Input:

Bar Graph:



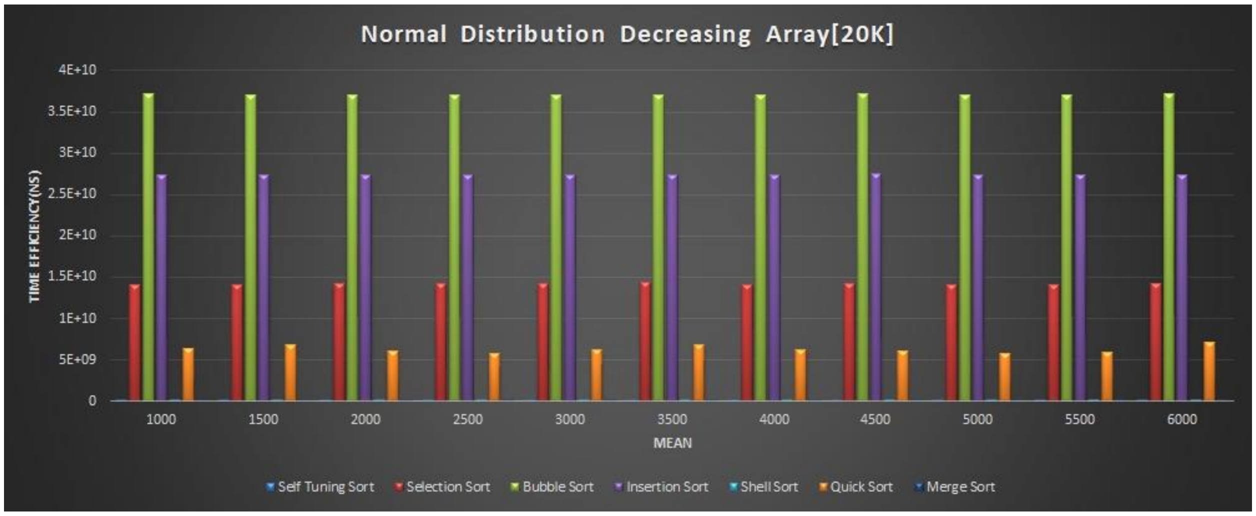
Line Graph:



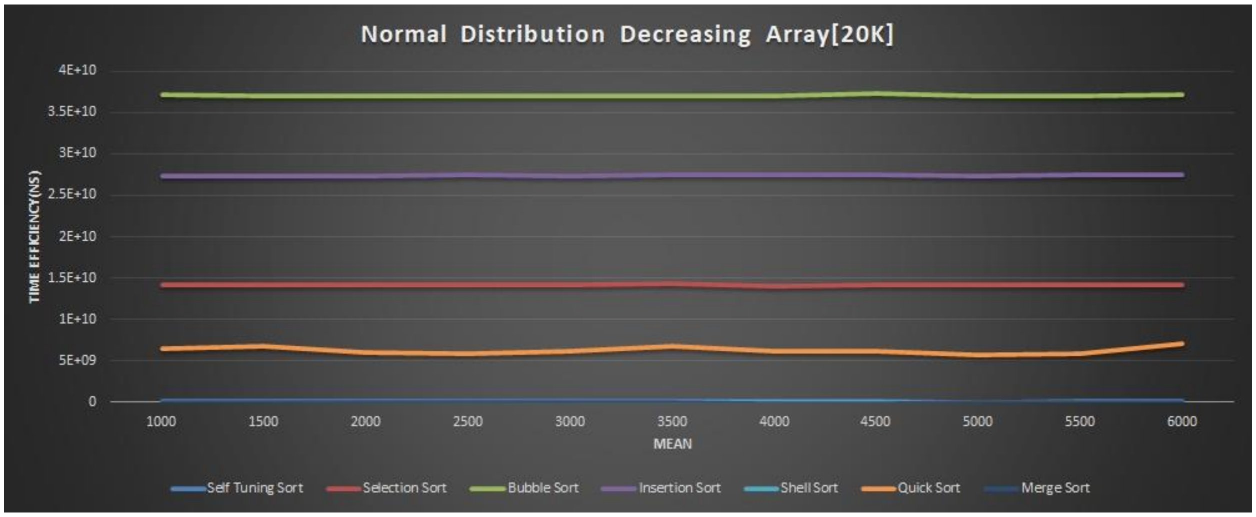
Analysis: Self-Tuning Sort performs better than other traditional algorithms.

Decreasing Input Array:

Bar Graph:



Line Graph:



Analysis: Self-Tuning Sort performs better than other traditional algorithms.

Conclusion:
According to the proposed idea of Self-Tuning Sort, we divided the array into increasing, decreasing, and random arrays. We then sorted the array using Merge Sort's merge sub-arrays algorithm and compared the processing time of Self-Tuning Sort with other traditional algorithms.

For random arrays, Self-Tuning Sort outperforms Bubble Sort for array sizes ranging from 10 to 10,000. Additionally, it performs closely to Insertion Sort and Selection Sort.

In the case of increasing and decreasing arrays, Self-Tuning Sort demonstrates superior performance compared to other existing algorithms for array sizes ranging from 10 to 10,000.

Future Work:

- Analysis of reducing the processing time of Self-Tuning Sort by implementing advanced programming concepts.
- Performing analysis using more statistical approaches/methods to check the impact on the time efficiencies.
- Usage of Machine Learning algorithms to identify the sort algorithm.
- Performing analysis for increased input array sizes.

References:

[1] Xiaoming Li, Maria Jesus Garzaran, David Padua. Optimizing sorting with genetic algorithms. In Proceedings of the international symposium on Code generation and optimization, CGO ’05, pages 99–110, Washington, DC, USA, 2005. IEEE Computer Society.

[2] Xiaoming Li, Mar´ıa Jes´us Garzar´an, David Padua. A dynamically tuned sorting library. In Proceedings of the international symposium on Code generation and optimization: feedback-directed and runtime optimization, CGO ’04, pages 111–, Washington, DC, USA, 2004. IEEE Computer Society.