Available Online at www.ijcsmc.com/journal

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IMPACT FACTOR: 7.056

IJCSMC, Vol. 11, Issue. 3, March 2022, pg.116 – 130

A Unified Vista and Juxtaposed Study on Sorting Algorithms

Anurag Dutta[1]; Veenu Chhabra[2]; Pijush Kanti Kumar[3]

[1]Undergraduate, Computer Science and Engineering, Government College of Engineering and Textile Technology, Serampore-712201, India

[2]Undergraduate, Information Technology, Government College of Engineering and Textile Technology, Serampore-712201, India

[3]Assistant Professor, Government College of Engineering and Textile Technology, Serampo-re-712201, India

[1] anuragdutta@gmail.com; [2] veenuchhabra2@gmail.com; [3] pijush752000@yahoo.com

DOI: https://doi.org/10.47760/ijcsmc.2022.v11i03.014

Abstract— Data is the new fuel. With the expansion of the global technology, the in-creasing standards of living and with modernization, data values have caught a great height. Now a days, nearly all top MNCs feed on data. Now, to store all this data is a prime concern for all of them, which is relieved by the Data Structures, the systematic way of storing data. Now, once these data are stored and charged in secure vaults, it's time to utilize them in the most efficient way. Now, there are a lot of operations that needs to be performed on these massive chunks of data, like searching, sorting, inserting, deleting, merging and so more. In this paper, we would be comparing all the major sorting algorithms that have prevailed till date. Further, work have been done and an inequality in dimension of time between the four Sorting algorithms, Bubble, Selection, Insertion, Merge, that have been discussed in the paper have been proposed.

Keywords— Sorting, Time Complexity, Recursion, Divide and Conquer.

I. INTRODUCTION

Sorting is one of the most important but basic operation that may be enacted on any data structure. It involves arranging the data in increasing order or decreasing order of its magnitude [1]. Sorting algorithms have got a lot to flaunt with, it's well balanced and cunning algorithm, its efficiency, and not only that, even some searching algorithms like binary search [2], interpolation search [3] [4] needs sorting algorithms to drop them in action. In this paper, we have compared all the popularly known and widely used sorting algorithms based on their dimensions of time. The merit of the paper also covers the proposition of a new inequation relating the time consumed by Bubble, Selection, Insertion, Merge Sort. To enact more on our inequation, both the Average and the Worst-Case Scenario have been considered.

II. Bubble Sort

It is the most basic sorting algorithm, erected upon the pillars of swap till doom, i.e., Swapping consecutives till we touch the core. Let us get the mathematical essence of this Sorting Algorithm.
Let we have been given a set of n unsorted elements in a list (data structure with language independency), such that

$$L(n) = \{\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}\}$$

Now, we will start by checking each set of consecutive two elements, (α_{i-1}, α_i), and if $\alpha_{i-1} > \alpha_i$, then swap both of them referentially. Once we have reached $i = n - 1$, the last element that will reach the end of the list will be the largest elements of the list, $max\big(L(n)\big)$, and intuitively, we can say that the last index is its correct place. Now, we will perform this checking consecutives for the newly forming lists for another $n - 2$ times, and each time, we will keep on decrementing the size of the subjected list.

The Pseudocode[5] for this algorithm will be

```
function bubble_sort(number_of_elements, list_of_elements)

    n ← number_of_elements
    for i = 0 to i = n - 1
       for j = 0 to j = n - 1 - i
          if list_of_elements[j] > list_of_elements[j + 1]
             swap(list_of_elements[j], list_of_elements[j + 1])
          end if
       end for
    end for
end
```

In this algorithm, the frequency count[6] of the inner loop will be

$$\sum_{j=0}^{n-1-i} (1) = \big((n-1-i) - (0) + 1\big) \times 1 = n - i$$

the frequency count of the outer loop will be

$$\sum_{i=0}^{n-1}\left(\sum_{j=0}^{n-1-i} (1)\right) = \sum_{i=0}^{n-1} (n-i) = n^2 - \left(\frac{n(n-1)}{2}\right) = \frac{n(n-1)}{2}$$

The Complexity in the dimensions of time for this Sorting Algorithm will be $O\left(\frac{n(n-1)}{2}\right) \approx O(n^2)$.

III. Selection Sort

It is another very basic sorting algorithm. In this algorithm, we simply keep on finding minimums in a selected list and work upon further.
Let we have been given a set of n unsorted elements in a list (data structure with language independency), such that

$$L(n) = \{\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}\}$$

Now we will start by finding minimum in a given list indexed between 0 and $n-1$ and will swap the element in initial index with the newly generated minimum referentially. In the next following steps, we will keep on incrementing the initial index until we have only one element left in the list.

The Pseudocode for this algorithm will be

```
function selection_sort(number_of_elements, list_of_elements)

    n ← number_of_elements
    for i = 0 to i = n - 1
       min_index ← i
       for j = i to j = n - 1
          if list_of_elements[min_index] > list_of_elements[j]
             min_index ← j
          end if
       end for
    swap(list_of_elements[min_index], list_of_elements[i])
    end for
end
```

In this algorithm, the frequency count of the inner loop will be

$$\sum_{j=i}^{n-1}(1) = \big((n-1)-(i)+1\big)\times 1 = n-i$$

the frequency count of the outer loop will be

$$\sum_{i=0}^{n-1}\left(\sum_{j=i}^{n-1}(1)\right) = \sum_{i=0}^{n-1}(n-i) = n^2 - \left(\frac{n(n-1)}{2}\right) = \frac{n(n-1)}{2}$$

The Complexity in the dimensions of time for this Sorting Algorithm will be $O\left(\frac{n(n-1)}{2}\right) \approx O(n^2)$.

IV. INSERTION SORT

This Sorting algorithm is quite similar in terms of its attire with the Selection Sort, but the face of this algorithm is holding a great difference concealed within it.

Let we have been given a set of n unsorted elements in a list (data structure with language independency), such that

$$L(n) = \{\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}\}$$

In this Sorting technique, we select a reference indexed element (say α_i) and start comparing that with the elements preceding it till we find the appropriate position for the reference indexed element. The affirmative comparison can be defined mathematically as $\alpha_j > \alpha_i, \forall j < i$ i.e., we keep on going back in index until we find an element to be smaller than the reference indexed elements, α_i. Meanwhile, in this process, we store the reference indexed element in secure vault and will shift each element to the right which follows the affirmation condition. The main merit of this algorithm is that the elements in left indices of the referential index are properly sorted.

The Pseudocode for this algorithm will be

```
function insertion_sort(number_of_elements, list_of_elements)

    n ← number_of_elements
    for i = 1 to i = n
        reference ← list_of_elements[i]
        j ← i - 1
        while list_of_elements[j] > reference and j ≥ 0
            list_of_elements[j + 1] = list_of_elements[j]
            decrement j
        end while
        list_of_elements[j + 1] = reference
    end for
end
```

In this algorithm, the frequency count of the inner loop will be i
the frequency count of the outer loop will be

$$\sum_{i=1}^{n-1}(i) = \frac{n(n-1)}{2}$$

The Complexity in the dimensions of time for this Sorting Algorithm will be $O\left(\frac{n(n-1)}{2}\right) \approx O(n^2)$.

V. MERGE SORT

Merge sort is a very good sorting technique as it follows the divide and conquer[7] algorithm.
Let we have been given a set of n unsorted elements in a list (data structure with language independency), such that

$$L(n) = \{\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{n-1}\}$$

Under this algorithm the list is divided into equally sized sub parts and merged step by step in a recursive[8] manner to bring it to sorted format. It is often referred to as the best sorting technique when we are required to sort a linked list.

The Pseudocode for this algorithm will be

```
function merge(list_of_elements, low_index, mid_index, high_index)

    size_vault1 ← mid_index - low_index + 1
```

```
size_vault2 ← high_index - (mid_index + 1) + 1
vault1[size_vault1]
vault2[size_vault2]
for i = 0 to i = size_vault1 - 1
    vault1[i] = list_of_elements[low_index + i]
end for
for i = 0 to i = size_vault2 - 1
    vault2[i] = list_of_elements[mid_index + 1 + i];
end for
i, j ← 0, 0
k ← low_index
while i < size_vault1 and j < size_vault2
    if vault1[i] > vault2[j]
        list_of_elements[k] = vault2[j]
        increment j, k
    else
        list_of_elements[k] = vault1[i];
        increment i, k
    end if
end while
while j < size_vault2
    list_of_elements[k] = vault2[j];
    increment j, k
end while
while i < size_vault1
    list_of_elements[k] = vault1[i];
    increment i, k
end while
```

```
end

function merge_sort(list_of_elements, low_index, high_index)

    if low_index < high_index

        mid_index ← low_index + (high_index - low_index) / 2

        merge_sort(list_of_elements, low_index, mid_index)

        merge_sort(list_of_elements, mid_index + 1, high_index)

        merge (list_of_elements, low_index, mid_index, high_index)

    end if

end
```

In this algorithm, we will try to find the recurrence relation, that is

$$\psi(n) = 2\psi\left(\frac{n}{2}\right) + \theta(n)$$

This can be solved using Master's Theorem of Divide and Conquer [9]

$$\psi(n) = n\,log_2\,n$$

The Complexity in the dimensions of time for this Sorting Algorithm will be $O(n\,log_2\,n)$.

VI. Average Case Scenario

Average Case is a case in which the time taken is totally dependent on the data elements chosen, it can surge high up to Worst Case Scenario, while it can go as low as Best Case Scenario. In this section we will be discussing in deeply about the absurd data sets used and the time taken to process them accordingly using Bubble, Selection, Insertion, and Merge Sort respectively.

A. *Random Elements*

In each and every programming language there exists randomizing functions that are capable of generating numbers (rather say data) randomly. These functions are developed in such a way that they generate numbers at random that doesn't repeat, to be precise, they repeat in such an interval that is too high to notice by human observation. Though there have been certain alternatives to this, but the most popular amongst all is the Mersenne Twister Algorithm[1]. In pseudo random generation, one most common variant of this Algorithm the 19337 Variant, which produces 32-bit pseudo random numbers. The Interesting fact about this Variant is that it repeats it's data after a period of $2^{19337} - 1$. We have plotted pie of pie chart for 5 sets of data, with 5 trials for each set, involving 4 types of Sorting, Bubble, Selection, Insertion, Merge, taking the time taken in terms of $10^{-9} seconds$ or nanoseconds by them as a whole of 100%.

TABLE I
TIME TAKEN IN NANO - SECONDS (AVERAGED FOR 5 TESTS [2]) FOR BISM SORT.

Number of Elements	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
1000	3008000	1381400	1020000	0
5000	62223200	32402800	17169600	993000
10000	268748200	130362400	67906000	1994800

[1] This algorithm was developed in the late 20s, by 松本 眞, and 西村 拓士 [10]. This algorithm can generate random numbers in the range $[0, 2^n - 1]$, where n is the word bit length. MT algorithm is based on the pillars of matrix linear recursion, recursed over a binary field.

[2] The data set for each test can be accessed from (here).

50000	8595182600	3240922600	1679175200	10003600
100000	34777961800	12538517400	6765823800	21556400

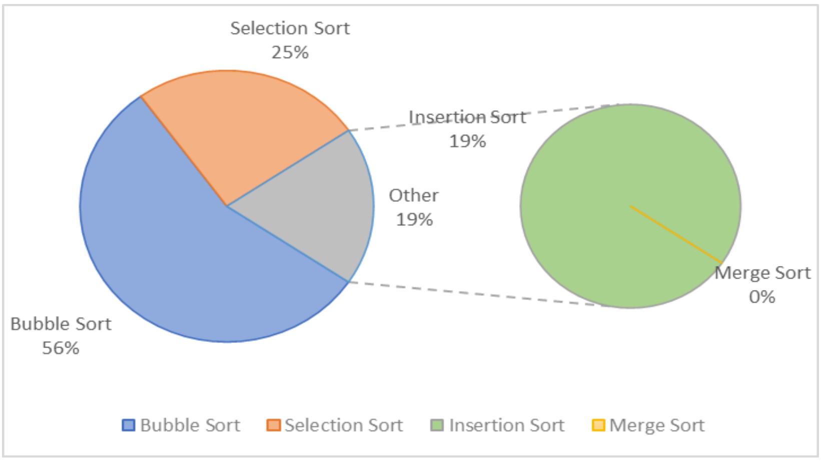

Fig. 1. For a Data set of 1000 entries

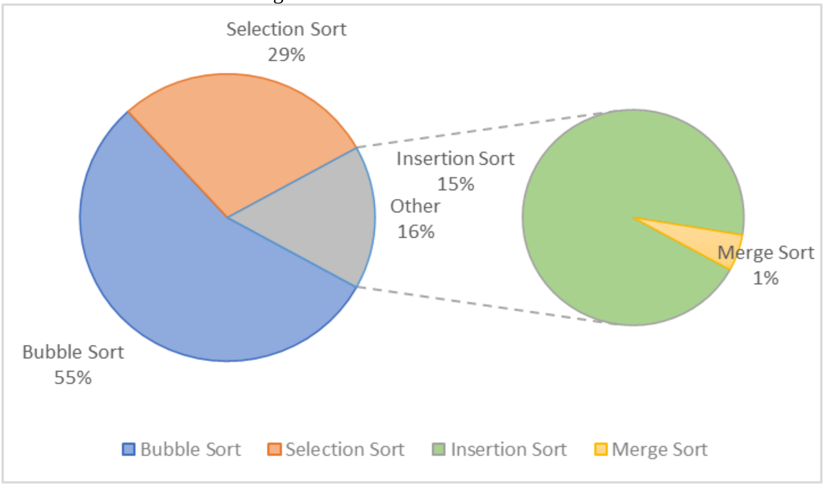

Fig. 2. For a Data set of 5000 entries

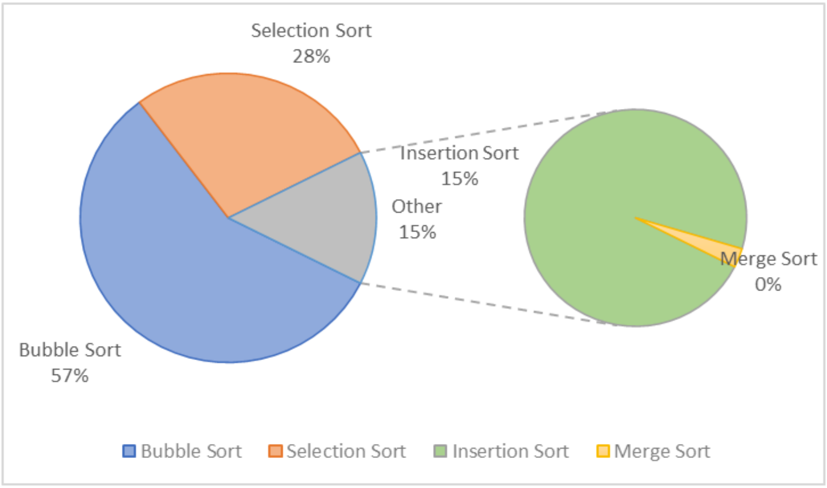

Fig. 3. For a Data set of 10000 entries

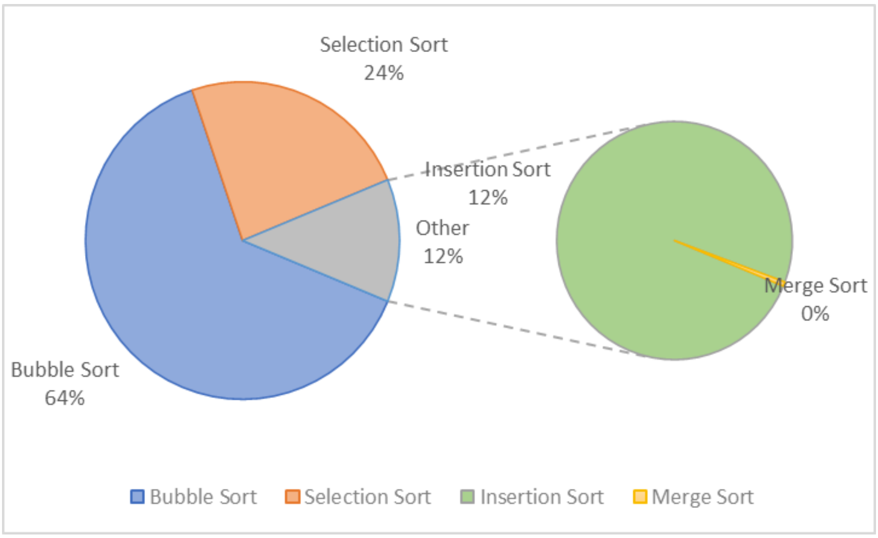

Fig. 4. For a Data set of 50000 entries

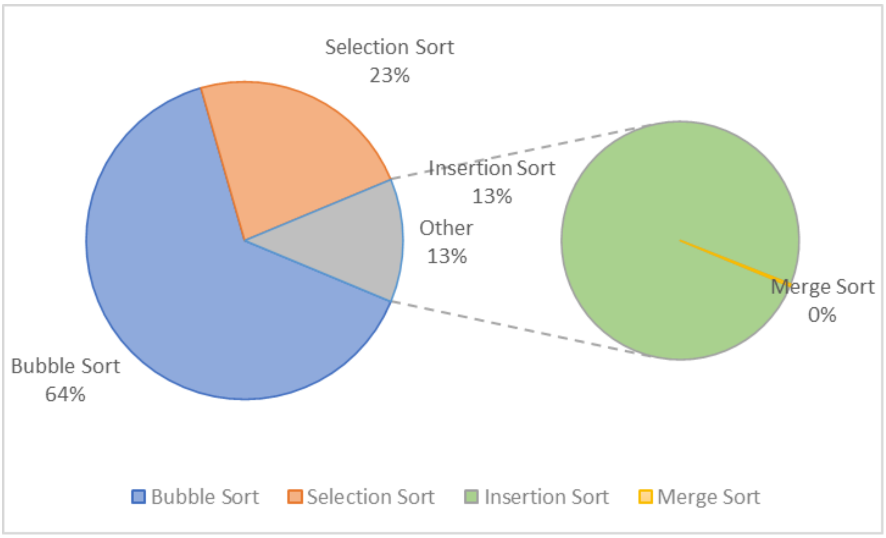

Fig. 5. For a Data set of 100000 entries

B. *Data Set involving Additive Inverse*

In this case, we had taken the consecutive elements of the list to be in the form $i, -i$. The series taken is

$$\{1, -1, 2, -2, 3, -3, \dots, {}^{n}/_{2}, -{}^{n}/_{2}\}$$

This case was designed to cater the average case scenario. Further we have plotted pie of pie chart for 5 sets of data, with 5 trials for each set, involving 4 types of Sorting, Bubble, Selection, Insertion, Merge, taking the time taken in terms of $10^{-9} seconds$ or nanoseconds by them as a whole of 100%.

TABLE III
TIME TAKEN IN NANO - SECONDS (AVERAGED FOR 5 TESTS [3]) FOR BISM SORT.

Number of Elements	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
1000	3987200	1595800	1795000	0
5000	118574600	41968000	36112600	1006600
10000	452954600	162894400	138829800	2989400
50000	11296089400	4100822600	3478913600	11175000
100000	45692997400	16390991600	13983360200	16132400

[3] The data set for each test can be accessed from (here).

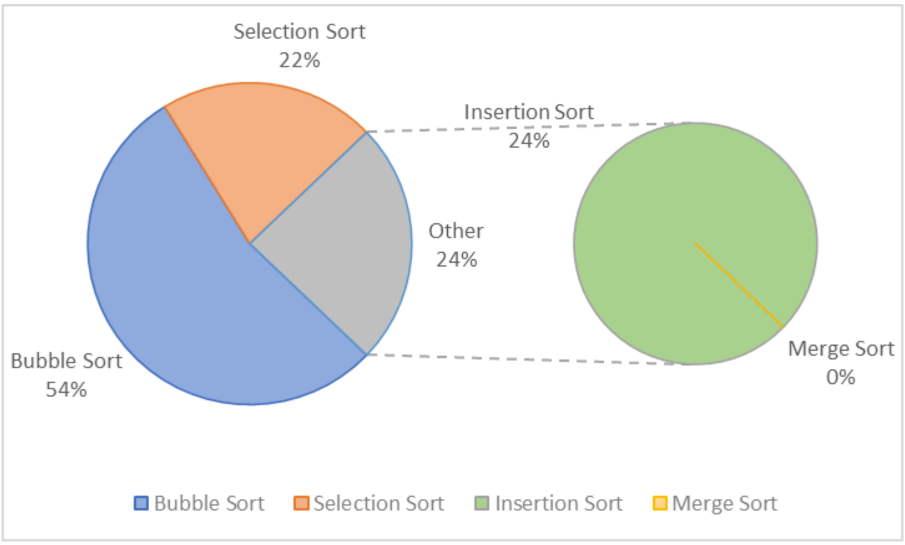

Fig. 6. For a Data set of 1000 entries

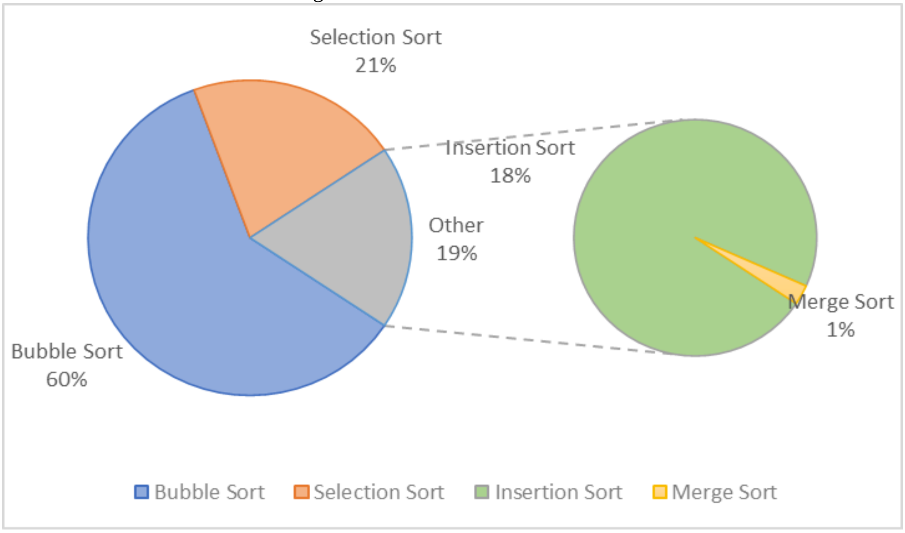

Fig. 7. For a Data set of 5000 entries

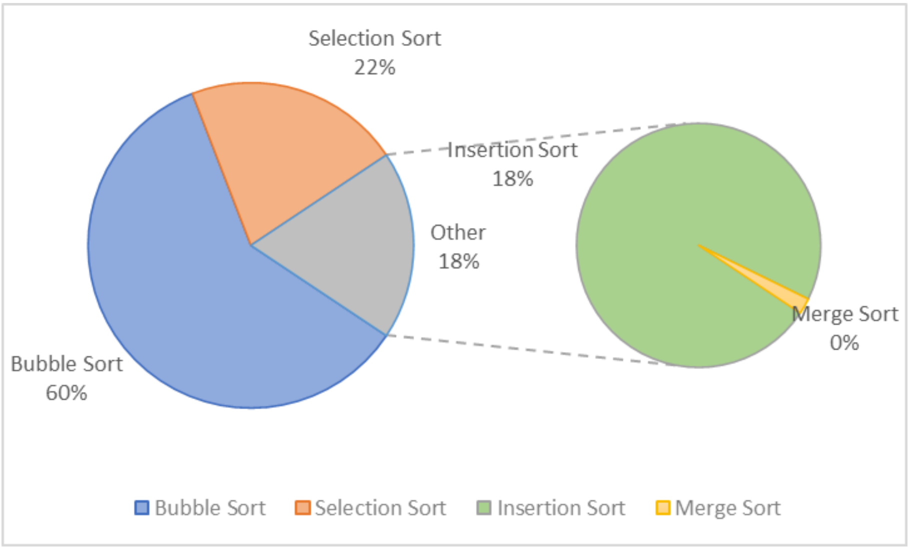

Fig. 8. For a Data set of 10000 entries

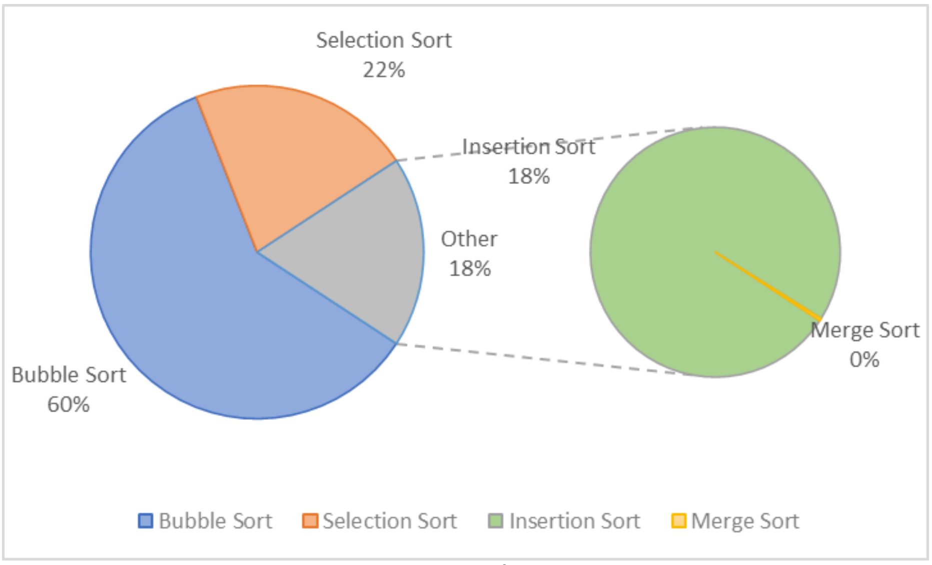

Fig. 9. For a Data set of 50000 entries

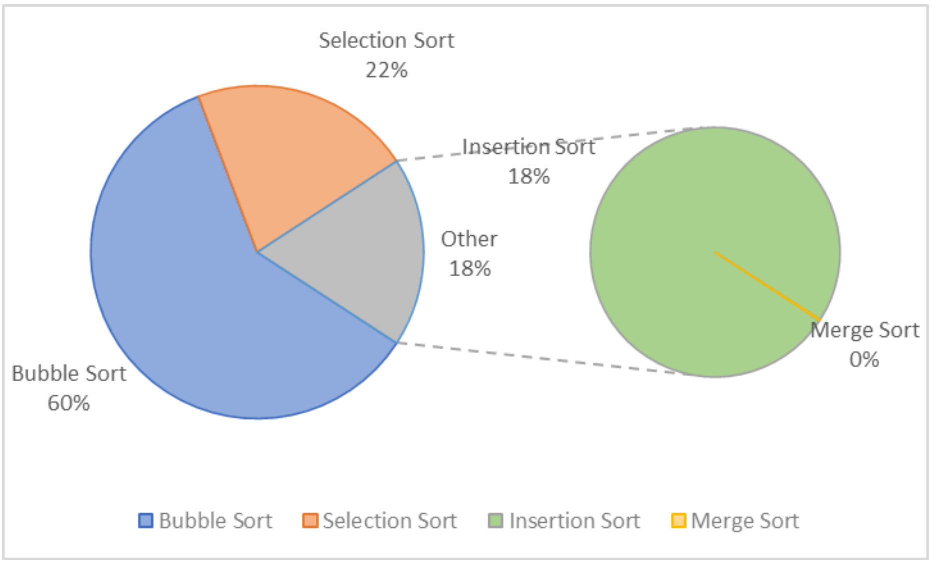

Fig. 10. For a Data set of 100000 entries

C. Data Set involving Multiplicative Inverse

In this case, we had taken the consecutive elements of the list to be in the form i, i^{-1}. The series taken is

$$\left\{1, \left({}^{1}/_{1}\right), 2, \left({}^{1}/_{2}\right), 3, \left({}^{1}/_{3}\right), \ldots, {}^{n}/_{2}, \left({}^{1}/_{\left({}^{n}/_{2}\right)}\right)\right\}$$

This case was designed to cater the average case scenario. Further we have plotted pie of pie chart for 5 sets of data, with 5 trials for each set, involving 4 types of Sorting, Bubble, Selection, Insertion, Merge, taking the time taken in terms of 10^{-9} *seconds* or nanoseconds by them as a whole of 100%.

TABLE IIIII
TIME TAKEN IN NANO - SECONDS (AVERAGED FOR 5 TESTS [4]) FOR BISM SORT.

Number of Elements	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
1000	3236583	1362185	1138929	197138.4
5000	74522670	33462720	37741309	1192544
10000	257476393	142642749	130494584	2456712.4
50000	7776991531	3255963837	2948694283	14732481.4
100000	26521438698	12117737273	11769060634	28317483.4

[4] The data set for each test can be accessed from (here).

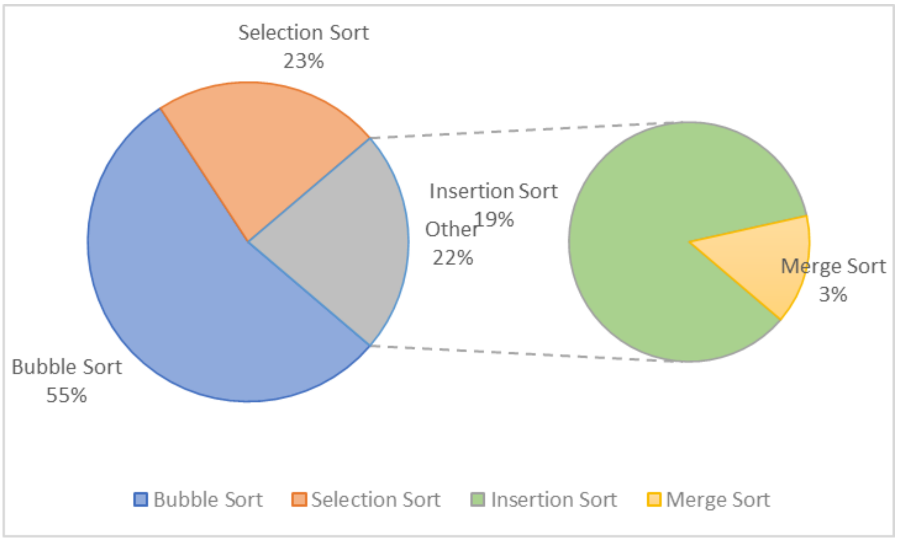

Fig. 11. For a Data set of 1000 entries

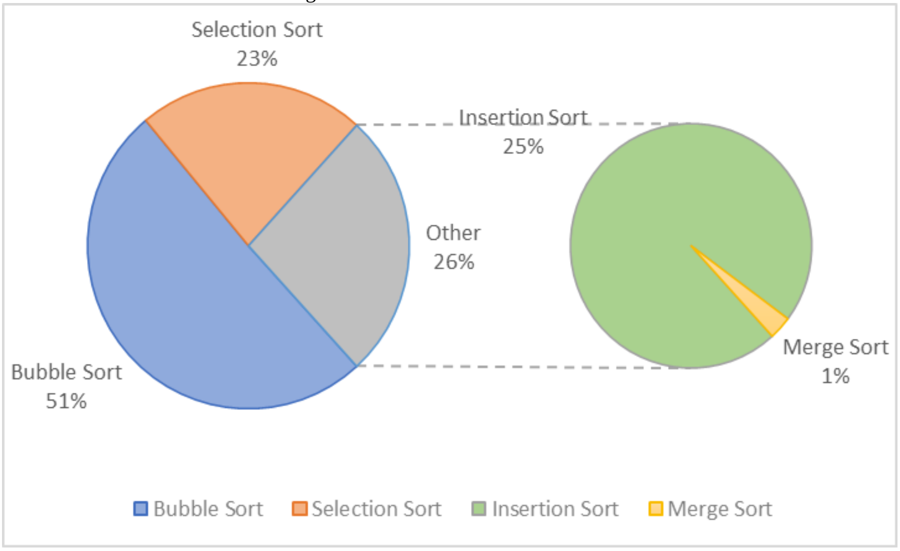

Fig. 12. For a Data set of 5000 entries

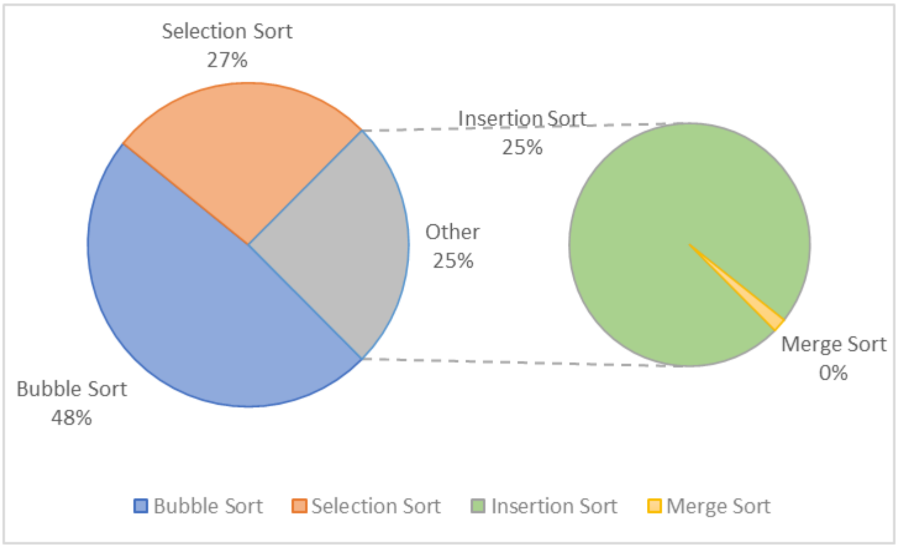

Fig. 13. For a Data set of 10000 entries

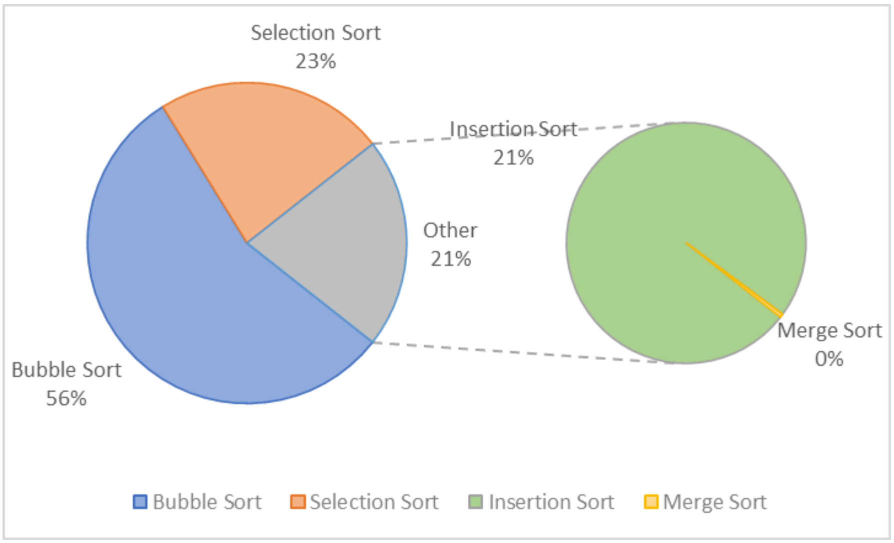

Fig. 14. For a Data set of 50000 entries

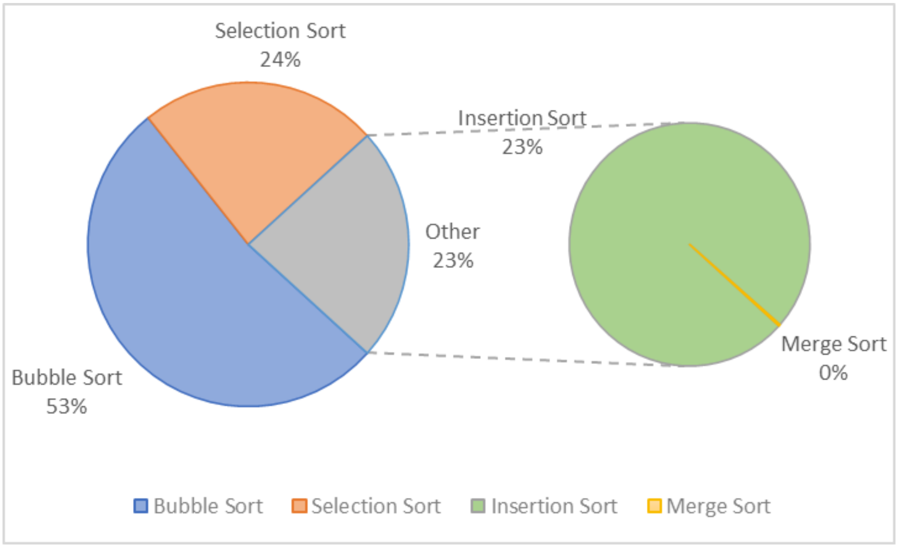

Fig. 15. For a Data set of 100000 entries

VII. WORST CASE SCENARIO

Every program, perhaps once in it's run time, faces the difficulty to go the hurdles and difficulties which stick to their maximum at that point of time, such a case is termed as Worst Case. We have plotted pie of pie chart for 5 sets of data, with 5 trials for each set, involving 4 types of Sorting, Bubble, Selection, Insertion, Merge, taking the time taken in terms of $10^{-9} seconds$ or nanoseconds by them as a whole of 100% and the result was quite promising. The data set used was arranged in descending order in terms of it's magnitude, and the algorithms we designed to arrange them in ascending order.

TABLE IV
TIME TAKEN IN NANO - SECONDS (AVERAGED FOR 5 TESTS [5]) FOR BISM SORT.

Number of Elements	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
1000	3401000	1593800	1792200	0
5000	76335200	36085600	36464800	1010400
10000	311938400	144146400	149144600	1614800
50000	7390573200	3642800600	3635167200	9175000
100000	29764243200	14739568400	14548738800	18749400

[5] The data set for each test can be accessed from (here).

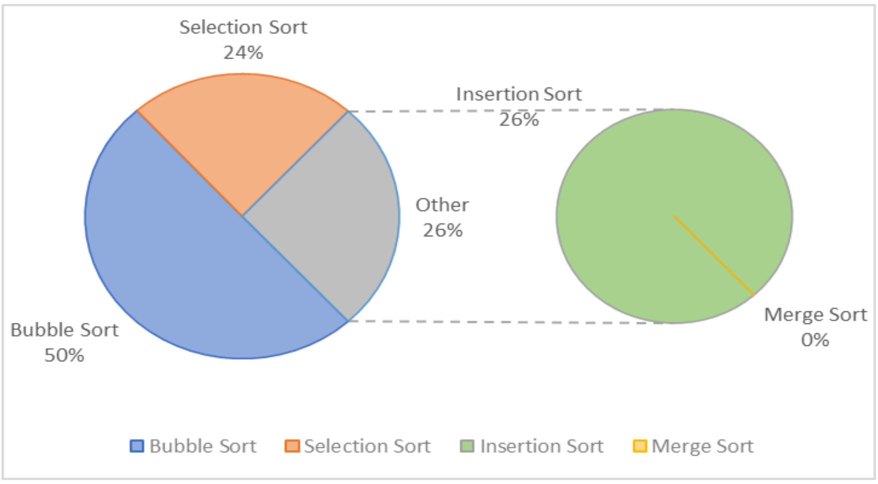

Fig. 16. For a Data set of 1000 entries

The Bar Chart representation[6] for this Data Set can be viewed from (here).

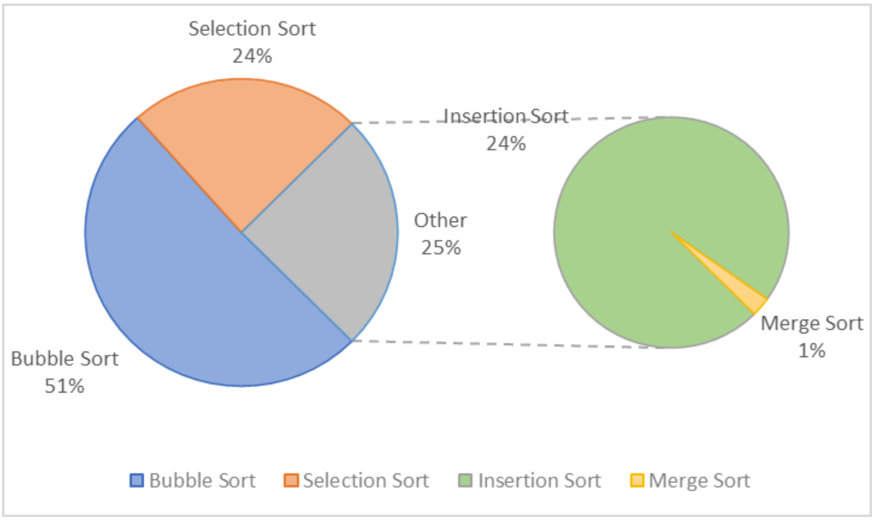

Fig. 17. For a Data set of 5000 entries

The Bar Chart representation for this Data Set can be viewed from (here).

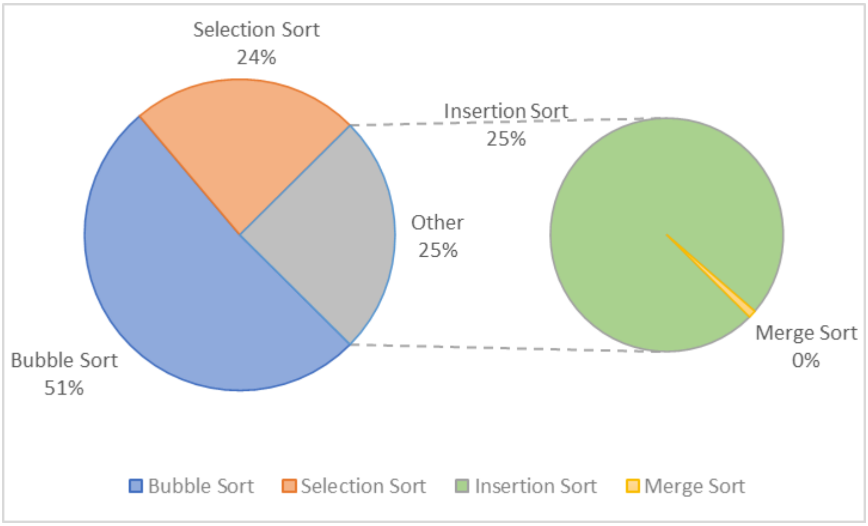

Fig. 18. For a Data set of 10000 entries

The Bar Chart representation for this Data Set can be viewed from (here).

[6] As in case of Worst-Case Scenario, the anomality is very less, so we have plotted Bar Graph as well to remove any confusion of any kind.

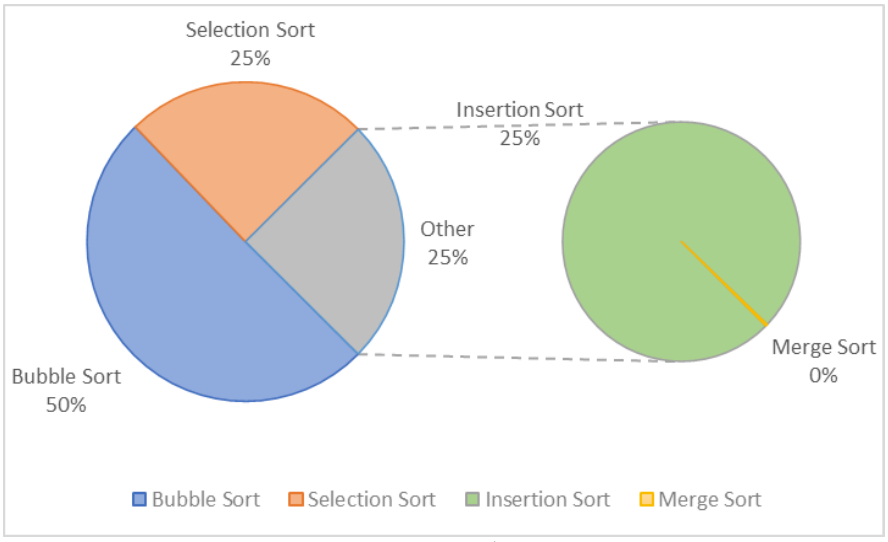

Fig. 19. For a Data set of 50000 entries

The Bar Chart representation for this Data Set can be viewed from (here).

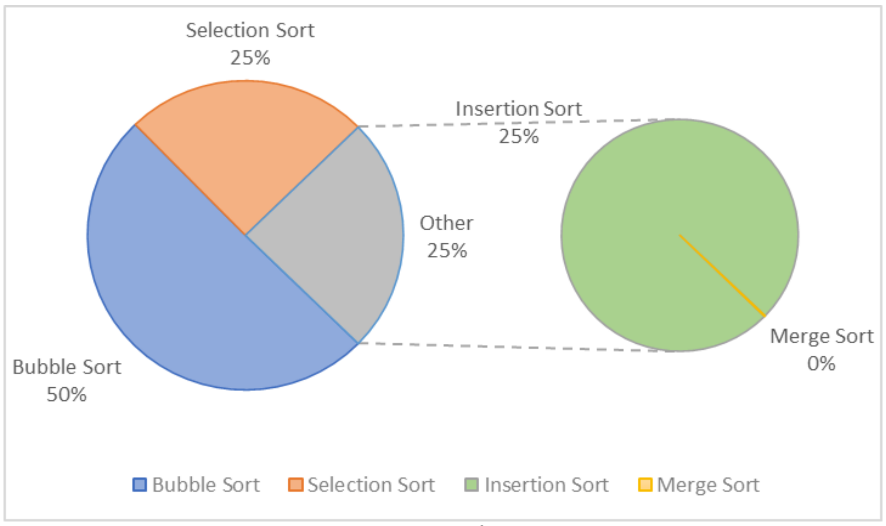

Fig. 20. For a Data set of 100000 entries

The Bar Chart representation for this Data Set can be viewed from (here).

VIII. Conclusion

From the study, we have conducted in this paper, we can conclude that, in each of trials, irrespective of Architecture and Specification of the computational device.

$$T(B) \geq T(S) + T(I) + T(M)$$

$$\ni B = Bubble\ Sort, I = Insertion\ Sort, S = Selection\ Sort, M = Merge\ Sort$$

where,
$T(\xi)$ is the time taken in terms of nanoseconds by that specific sorting algorithm
$\forall \xi \in$ Comparison Sorting Techniques and

$0\% \leq \left(\frac{\delta\varepsilon}{\varepsilon}\right) \times 100 \leq 5\%$, ε, being the tolerance limit.

This error / tolerance limit is due to the variation in processing speed and time due to various architectural aspects and physical aspects. Some of the aspects that may result in increasing the tolerance limit are:

- System Temperature (θ_s): If $\theta_s > \theta_0$, overheating of computation system occurs, that results in decrease of processing speed, and hence claims more time.

$$\frac{\delta\varepsilon}{\varepsilon} \propto \theta_s - \theta_0$$

where, $\theta_0 = Threshold\ Temperature$

- Network Stability: This issue is more taken in account if the computation is done online. The instability in network may induce more time being claimed.
- Power Stability: If the machine is undergoing a sudden surge or decline in power, the performance may be hindered, resulting in claim of more time.

The data sets that we have obtained in this study are generated by system working on Harvard Architecture[7], though the inequation developed is irrespective of the architecture.

REFERENCES

[1]. Thomas H. Cormen, Charles E Leiserson, Ronald L Rivest, Clifford Stein, “Introduction To Algorithms”, MIT Press, Cambridge, Massachusetts London, England, 2001.

[2]. Williams, Jr., Louis F. (1976), “A modification to the half-interval search (binary search) method”, Proceedings of the 14th ACM Southeast Conference. ACM, pp 95 – 101.

[3]. Gonnet G., Rogers L. and George J. (1980), “An algorithmic and complexity analysis of interpolation search”, Acta Informatica, Springer, 13, pp 39 – 52.

[4]. Reingold E. and Perl Y. (1977), “Understanding the complexity of interpolation search”, Information Processing Letters, Vol. 6, Issue 6, pp 219 – 222.

[5]. Nicholas Bennett, “Introduction to Algorithms and Pseudocode”, 2015.

[6]. David Harel, “Algorithmics: The Spirit of Computing”, Springer, 1987.

[7]. Jon Kleinberg, Éva Tardos, “Algorithm Design”, Pearson, 2005

[8]. Herbert Wilf, “Algorithms and Complexity”, Tailor and Francis, 2002.

[9]. Bentley J., Haken D., Saxe J. (1980), “A General Method for Solving Divide-and-Conquer Recurrences”, ACM SIGACT News, Vol. 12, Issue 3, pp 36 – 44.

[10]. Matsumoto, M, Nishimura, T, (1998), “Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator”, ACM Transactions on Modeling and Computer Simulation, Vol. 9, Issue 1, pp 3 – 30.

[7] The Harvard architecture is a computer architecture with separate storage and signal pathways for instructions and data. It contrasts with the von Neumann architecture, where program instructions and data share the same memory and pathways.