VOL. 03 NO. 9,
FEBRUARY, 2024
E-ISSN 3026-958X
P-ISSN 3027-1169

African Scholar Publications & Research International
www.africanscholarpub.com

Journal of Science Innovation & Technology Research (JSITR)

COMPARATIVE COMPLEXITY STUDY OF BUBBLE SORT AND INSERTION SORT USING JAVA PROGRAMMING LANGUAGE: A REVIEW

IBRAHIM MUHAMMAD AL-AMIN, A.E OKEYINKA, ABDULLAHI IBRAHIM
Department of Computer Science, Ibrahim Badamasi Babangida University
Lapai Niger State, Nigeria.

Abstract

This research paper presents the review of the computational complexity of bubble sort and insertion sort; the run-time performances of bubble and insertion sort were evaluated, and the comparisons of the time complexity of both sorts were also analyzed. These two algorithms was the area of focus for this research and the result practically proved that bubble sort is not better compare to insertion sort and therefore agreed that the best performing algorithm in terms of the best case, average case, and worse case is insertion sort. The research also provides a drawback to the bubble sorting algorithm because it keeps sorting the input data even if they are arranged at their normal positions.

Keywords: Algorithm, Bubble Sort, Insertion Sort, Complexity, Efficiency, Performance.

INTRODUCTION

The complexity of an algorithm is analyzed based on speed and efficiency. It calculates how slowly or fast a certain algorithm can perform. The computational complexity performs the function by classifying algorithms to their best performance group (Anand, 2015). In the time complexity of an algorithm, the total time required by the program to run to completion is identified and represented using asymptotic notation (Yash, 2017). The functionality of

asymptotic notation is to identify the upper bound and lower bound of the algorithms. Analysis of an algorithm raises a problem that emanates enormous curiosity in the computational environment which requires techniques or methods for analyzing and computing its complexity. But the best and most accurate solution is by computing its run-time analysis. This study is to compare the complexity analysis of the insertion sort algorithm and bubble sort. The analysis of time complexity in any sorting algorithm defines the number of times that the program/instruction set is executed and not the complete time used when executing. The time complexity of an algorithm brought about the issue that stimulates curiosity because the total execution time works based on some physiological factors like the use of a compiler, the input size, and the speed of the processor. It creates uncertainty within the computational environment because the user can't decide which algorithm is best and when to be used for a given data set.

This paper compares the time complexity of bubble sort and insertion sort and suggests which one can be best for a given data. Also, the amounts of computational time for both algorithms were calculated using Java programming Language. The research also compares the foundation to various parameters like method and complexity.

The primary aim of this study is to compare the analysis of time complexity in Insertion sort algorithm bubble sort algorithms and the objectives are:

a. To review the computational complexity of bubble sort and insertion sort
b. To evaluate the run-time performance of bubble sort and insertion sort.

LITERATURE REVIEW

In the research of (Mishra and Ashutosh, 2013), some sorting algorithms like merge, bubble, selection, insertion, and index sort were compared on the data input of 10, 100, and 1000 10000 and found out that the best performing sorting algorithms in term of their complexity is index sort. Index sort uses lesser C.P.U time than the aforementioned algorithms in the cases of big or larger data.

Hammad Jehad (2015) finds out in his review paper on the comparison between heap sort, quick sort, bubble sort, gnome sort, min sort, insertion sort, max sort, merge sort, linear search, binary search, and towers sort algorithms on 5000, 10000, 20000, and 30000 data inputs values and finds out that the best performing sorting algorithms is a gnome. He also finds out that selection sort is quicker among gnome sort and bubble sort when considering random data. Their research also provides a drawback to the selection sorting algorithm because it keeps sorting the input data even if they are arranged at their normal positions, whereas the bubble sort and gnome sorting algorithm replaces any required data.

Ahmed and Zira (2013) researched the comparison between quick sort and insertion sort based on the characters and integer arrays. The research finds out that the

complexity of insertion sort is better compared to that of quick sort. The two algorithms are efficient in analyzing small data inputs. The C.P.U consumption time when working with character arrays is high compared to integer arrays. (Nidhi et al., 2013) conducted their research on the comparative analysis of insertion sort, heap sort, and quick sort based on their time complexity and the performance parameters. They used random numbers of the range 10000-30000 as the input variables. The research finds out that the insertion sort is very slow compared to quick and heap sort. The three algorithms have the worst case of time complexity of N*N. The best performing algorithm among the three is quick sort because as the input data continues to increase, the insertion and heap sort work on showing exponential growth and perform worse.

Yash (2020) researched the comparisons between different sorting algorithms using time complexity as the scope. He finds out the running time of bubble, selection, insertion, merge, and quick sort as a mathematical entity and analyzed their abstract point of view. He concluded that in every sorting algorithm, there exist merits and demerits which create room for the programmer to choose the best algorithm to work with. Shweta (2014) researched the time complexity comparative analysis between three (3) sorting algorithms. The research provides a comparative analysis between bubble, insertion, and selection sort. He concluded that insertion sort is better compare to bubble and selection sort algorithms.

Souradh (2015) conducted research from the University of Jammu on the run-time analysis of some sorting algorithms using C# language. The researcher uses C# to calculate the time complexity of the algorithms using random data chosen in the range of 10000, 20000, 30000, 40000, and 50000. The calculation and results for the average waiting time were presented in a graph. It was clear that quick sort performs the best and it's recommended to be used for larger data sets among all the algorithms because of its efficiency.

RESEARCH METHODOLOGY

For the course of this research, bubble and insertion methods were used to conduct this research. They are techniques used to sort data set. The running time of both insertion sort and bubble sort algorithms is to be calculated using Java programming languages.

A. The Bubble Technique

In the bubble method, the sequences of elements in the table are compared and, eventually, if the keys of the two (2) elements are not detected in proper order, they will be interchanged. The sorting starts from the initial of the table and continues to the last element in the table. The result of this will pushed the element with the largest key to the last element position. Then the pass two (2) is made. The second pass works exactly like the pass one in exclusion of the

last element. After the pass two, the following largest element is then pushed down to occupy the next to last position. Pass two is then followed by pass three and continuously till (N-1) passes or there exists no interchange between the passes.

Algorithm for the Bubble Technique

1. Set N = Length of Array
2. Array[j] > Array[j + 1] (Decision) if no - continue with the step 4
3. Swap Array[j] and Array[j+1]
4. Increment j (j++)
5. j < N - i - 1 (Decision) if yes - continue with the step 2
6. Clear j (j = 0)
7. Increment i (i++)
8. i < N - 1 (Decision) if yes continue with the step 2
9. Ordered Array

B. Insertion Technique

An insertion method iterates and moves the element from its original position and fixes it to the suitable position in the sorting list. Insertion sort works the same as the way we sort cards in a game. We assume the first card in our hand is already sorted; we then select an unsorted one. If the unsorted one is greater than the one in hand, it is placed on the right otherwise, to the left. The same way other unsorted cards are followed and put on the right place. Similar technique is used by insertion sort.

Algorithm for the Insertion Technique

1. Set N = Length of Array
2. Set I = 1
3. Set Value = Array[I]
4. Set J = I - 1
5. J >= 0 AND Array[J] > Value
6. Set Array [J + 1] = Array [J]
7. Decrement J
8. J--
9. Array[J + 1] = Value
10. Increment I (I++)
11. I < N

Conceptual Frame Work

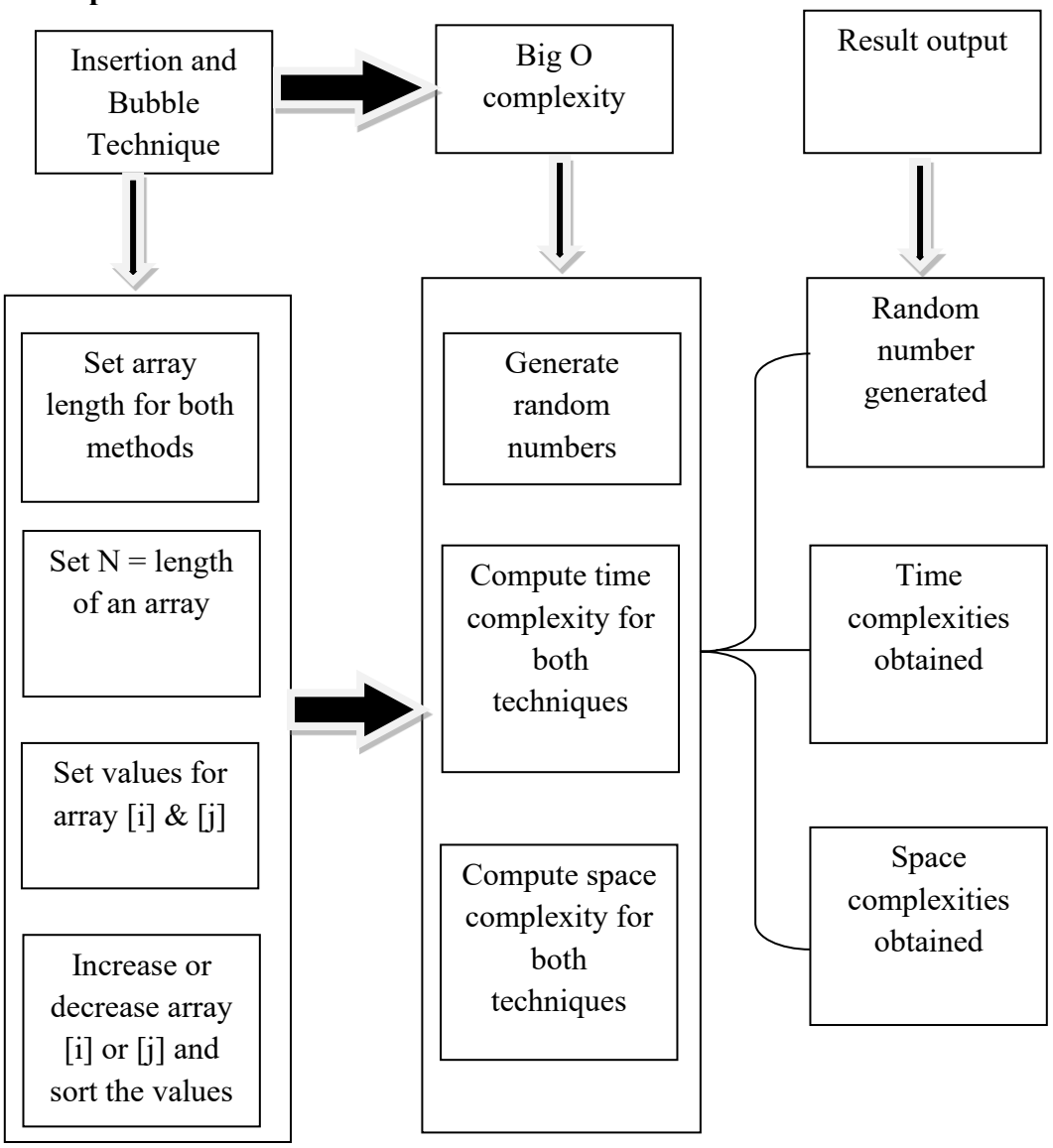


Fig 1: Conceptual Frame Work

RESULT AND DISCUSSION

The designed system was based on the dynamic user perspective. The user accepts the values to be generated and the console application sort the values based on bubble sort and insertion sort techniques. The overall procedure is presented below:

Data Items	Bubble Sort Time Complexity (Seconds)	Insertion Sort Time Complexity (Seconds)
Small Data Items (20)	43.11	19.71
Medium Data Items (50)	168.13	60.97
Large Data Items (100)	581.99	144.11

Table 1: Time Complexity for Bubble and Insertion Sort

For the data input (20), the computational complexities for both bubble and insertion sort are 43.11 seconds and 19.71 seconds respectively. The computational complexities for the (50) data input are 168.13 seconds and 60.97 seconds for bubble and insertion sort respectively. And for the (100) data input, the computational complexities were recorded as 581.99 seconds and 144.11 seconds for bubble and insertion sort respectively. Therefore from the three scenarios above, it is clearly stated that insertion sort performs better than bubble sort because, the running time for the insertion sort is less compare to bubble sort algorithm.

CONCLUSION

From the findings, bubble sort performs poorly compared to the insertion sort on the three cases (small, average and large data items). The performances in terms of time required to complete the sorting process for the insertion sort is faster and better compared to bubble sort algorithm. These demonstrations were presented in the first, second and third scenarios above. The superiority of insertion sort is over that of bubble sort and if there is therefore the need to choose between bubble sort and insertion sort for sorting data item, insertion sort is recommended.

RECOMMENDATIONS

From this study, bubble sort and insertion sort were compared and the following recommendations were made:

1. Insertion sort should be use if there is need to choose between bubble sort and insertion sort for any data item.
2. There should be need to compare insertion sort with other sorting algorithms like quick sort, selection sort, heap sort and merge sort to determine their efficient and know which would be best for a given sorting items.
3. Researchers and computer scientists should focus on analyzing the complexities of sorting algorithms as it is the basic of all computer fields.

FUTURE RESEARCH

In our future research, we will analyze the comparisons between different sorting algorithms using different programming languages to investigate the run-time complexity and the utilization of memory space. The researchers shall also analyze the complexity of different sorting algorithms using different operating systems and build the graphical user interface for their comparisons.

REFERENCES

A. M. Alnaser, O. AlHeyasat, A.A.-K. Abu-Ein, H. Moh'd, S. Hatamleh, A. A. M. Sharadqeh (2019), " Time Comparing between Java and {C}++ Software", vol. 5, no. 8, pp. 630-633, 2019.

D. Y. Pawade, M. Metha, K. Shah, and J. Rathod (2022), " Python Based Software Complexity Calculator using Halstead Metrics", Int. J. Adv. Found. Res. Comput. 2, no. Special Issue (NCRTIT 2022), pp. 390-394, 2022.

Christopher Kormanyos (2021), "Real-time C++: efficient object-oriented and template microcontroller programming", Springer, 2021.

Karthick, S. (2016, September-October). Odd Even based Binary Search. International Journal of Computer Engineering & Technology (IJCET), 7(5), 40-55. Retrieved October 2, 2018, from http://www.iaeme.com/ijcet/issues.asp?JT ype=JJCET&VType=7&IType=5

Kavitha, S., Vijay, V., & Sakeh, A. (2016). Matrix Sort- A Parallelizable Sorting Algorithm. Internatioal Journal of Computer Applications, 143(9), 1-6.

Kazim, A. (2017). A Comparative Study of Well Known Sorting Algorithms. International Journal of Advanced Research in Computer Science, 8(1), 277- 280.