Comparative Analysis of Comparison and Non Comparison based Sorting Algorithms

Adolf Fenyi
Sefwi Wiawso College of Education
P. O. Box 94, Sefwi Wiawso, Ghana

Michael Fosu
Cape Coast Technical University
Cape Coast
P. O. Box DL 50, Cape Coast, Ghana

Bright Appiah
Presbyterian College of Education
P. O.Box 27, Akropong Akuapem, Ghana

ABSTRACT
Sorting is one of the most important task in many computer applications. Efficiency becomes a big problem when the sorting involves a large amounts of data. There are a lot of sorting algorithms with different implementations. Some of them sort data by comparison while others don't. The main aim of this thesis is to evaluate the comparison and non-comparison based algorithms in terms of execution time and memory consumption. Five main algorithms were selected for evaluation. Out of these five, three were comparison based algorithms (quick, bubble and merge) while the remaining two were non-comparison based (radix and counting). After conducting an experiment using array of different data sizes (ranging from 1000 to 35000), it was realized that the comparison based algorithms were less efficient than the non-comparison ones. Among the comparison algorithms, bubble sort had the highest time complexity due to the swapping nature of the algorithm. It never stops execution until the largest element is bubbled to the right of the array in every iteration. Despite this disadvantage, it was realized that it is memory efficient since it does not create new memory in every iteration. It relies on a single memory for the swapping array operation. The quick sort algorithm uses a reasonable amount of time to execute, but has a poor memory utilization due to the creation of numerous sub arrays to complete the sorting process. Among the comparison based algorithms, merge sort was far better than both quick and bubble. On the average, merge sort utilized 32.261 seconds to sort all the arrays used in the experiment while quick and bubble utilized 41.05 and 165.11 seconds respectively. The merge algorithm recorded an average memory consumption of 5.5MB for all the experiment while quick and bubble recorded 650.792MB and 4.54MB respectively. Even though the merge sort is better than both quick and bubble, it cannot be compared to the non-comparison based algorithms since they perform far better than the comparison based ones. When the two groups were evaluated against execution time, the comparison based algorithms recorded an average score of 476.757 seconds while the non-comparison obtained 17.849 seconds. With respect to the memory utilization, the non-comparison based algorithms obtained 27.12MB while the comparison ones obtained 1321.681MB. This clearly reveals the efficiency of the non-comparison based algorithms over the comparison ones in terms of execution time and memory utilization.

General Terms
Algorithms, sorting, comparison, arrays and data structures

Keywords
Bubble, Quick, Merge, Counting, Radix and Time Complexity

1. INTRODUCTION
In computer Science, sorting is very important in many applications as far as locating and searching for a an important data is concerned. Sorting is therefore the process of rearranging the data in a list into a specific order. This order could be lexicographical or numeric. It arranges integers into either ascending or descending order and strings into alphabetical order. Searching a sorted list or array takes less time as compared to unsorted list [1]. Several researchers have analyze the complexity and propose a number of good sorting algorithms. These research have solved a lot of problems in computer science[2]. However, every sorting algorithm comes with its advantages and disadvantages. For example, Quick sort is better for sorting a large amount of data while Bubble sort is better for small data size. The performance of every sorting algorithm depends on the computer (main memory and hard drive) and the size of the data being sorted[3]. Therefore to evaluate the performance of sorting algorithms, there is the need to consider the execution time and space complexity of the algorithms[4]. Because sorting algorithms are popular in Computer Science, some of it contexts are well known in algorithm concepts. Examples include divide and conquer, randomized algorithms and data structures. The efficiency of most of the sorting algorithms is either O(nlogn) or O(n^2)

1.1 Objectives
The objective of this paper is to evaluate the various types of sorting algorithms(comparison and non-comparison) based on their time and space complexities.

1.2 Limitations
This research could not analyze all the various types of sorting algorithms due to their number. It therefore considered only the major and well known algorithms. The researcher could not also conduct test on arrays (large arrays) with sizes greater than 40000 bytes, due to insufficient memory on the computer.

2. LITERATURE REVIEW
This section describes the various types of sorting algorithms. Each algorithm is defined and explained with their advantages and disadvantages. It present two main types which are comparison and non-comparison based algorithms. The comparison based algorithm works by comparing the element in the array with one another while the non-comparison sorts the elements by not comparing but by other approaches[5]. The comparison based algorithms that are considered in this study are quick, bubble and merge sorts, while the non-comparison are radix and counting.

2.1 Quick sort
Quick sort has been declared as the fastest sorting algorithm

as compared to other complicated ones. This algorithm is better than Merge since it does not need extra memory space for sorting. This makes it applicable in real time applications that require extremely large datasets. It utilizes divide and conquer approach for dealing with problems. It operates by fetching elements from unsorted array which is called pivot and divide the array into two sub arrays. It then reconstruct one of the arrays with the elements larger than the pivot while the other array is reconstructed with elements smaller than the pivot. This operation is recursively repeated for both sub arrays. The algorithm selects either the rightmost or leftmost element as the pivot. This selection was adopted in the early version of the algorithm and it creates worst case behavior for array that has already been sorted. However, the problem was solved by randomly selecting a pivot and computing the median of first, middle and last elements. It works efficiently in a virtual memory environment. Quick sort is fast and efficient algorithm for large data sets. It is however inefficient for array that has already been sorted and contains the same elements. It also has high space complexity since it uses additional space for recursive function calls[6].

2.2 Bubble Sort

This algorithm[2] is said to be the slowest but simple. It works by comparing the array elements with their neighbours and rearranges them if they are unsorted. The algorithm continues this rearrangement operation until it discovers that all the elements in the array are sorted. This process slows down the algorithm when the input size increases. It is said to be inefficient for large volume of data. The advantage of this algorithm is that it is simple and easy to implement.

2.3 Merge Sort

This algorithm adopts the divide and conquer approach to solve problems[7]. It operates by dividing a given array into many sub arrays until each one contains only one element. It then merges the sub arrays into a single sorted array. This algorithm is considered to be stable since it preserves order of elements with equal key. This algorithm requires more memory than other complicated sorting algorithms. Due to it recursive nature, it is not recommended for smaller arrays. It is also difficult to implement.

2.4 Radix sort

This works without comparing any element in an array, and is therefore considered as a linear sorting algorithm[8]. It operates by sorting array elements with keys. The keys are mostly denoted by integers. It works by rearranging each value in the input element. It can begin with the least significant digit until it get to the most significant digit. It is considered as a stable algorithm since it preserves the order of elements with equal keys. The two main implementation of radix sort are Least Significant Digit (LSD) and Most Significant Digit (MSD). The LSD method works by sorting the array based on the least significant value until the most significant value is reached while the MSD starts with the most significant value to the least value. The efficiency of Radix does not depend on the size and type of the input elements being sorted. It is very difficult to implement and also consumes a lot of memory.

2.5 Counting

This algorithm[9] is considered to have linear running time complexity since it is an integer sorting algorithm. It also operates based on keys that ranges from zero to the size of the input array. It operates by counting the occurrences of the array element and storing this information into another array (B). It then applies arithmetic operations on array B to determine the position of each value in the complete sorted array. It also preserves the order of array elements with equal keys. One of the advantages of this algorithm is that it can be used as a subroutine to another algorithm since it uses key values as indexes into an array. This algorithm is not good for strings and large arrays[10].

3. METHODOLOGY

In this section, the study will implement the various comparison and non-comparison based sorting algorithms in PHP programming language. A laptop machine with speed 2.7 GHz and memory of 8GB was used to evaluate the various algorithms with respect to time and memory complexities. The microtime function which accepts Boolean value was used to determine the execution time while memory_get_peak_usage() was used to determine the memory consumption of the various algorithms. The pseudo codes of Quick[11], Bubble[12], Merge[13], Radix[14] and counting[15] algorithms were studied and implemented in PHP programming language.

The results are shown in the next section of this paper.

4. RESULT ANALYSIS AND DISCUSSIONS

In this section, the study will evaluate the various algorithms based on time (seconds) and memory (MB) complexities. It will be bad to use any sorting algorithm without looking at it efficiency. Efficiency deals with memory, CPU, network and disk usage. The time complexity of an algorithm depends on the number of basic operations it performs. The study will evaluate the running time and memory consumption of an unsorted array whose size ranges from 1000 to 35000. The elements in this array will be sorted by the various algorithms. The study will also implement the big O notation which does not depend on any machine's architecture to determine the complexities. The three main cases that would be examined are best, average and worst cases. The best case is defined by the fastest amount of time required by the algorithm to solve a given problem. For example, the amount of time required to sort an array that has already been sorted. The average case is the average amount of time the algorithm needs to solve a problem. This could be done by running the algorithm several times over many input sizes and computing the average. The worst case is the maximum amount of time the algorithm needs to solve a problem. For example, this could be the time to sort an array that is sorted in a reverse order.

Table 1: Evaluating the running time of the various algorithms in seconds

No. of elements	Quick	Bubble	Merge	Radix	Counting
1000	0.0684	0.334	0.073	0.307	0.001
5000	1.448	8.517	1.318	1.606	0.005
10000	6.343	34.912	5.516	3.219	0.007
20000	32.029	144.865	28.239	7.084	0.009
30000	83.186	332.707	64.896	10.775	0.02
35000	122.96	469.331	93.527	12.630	0.034

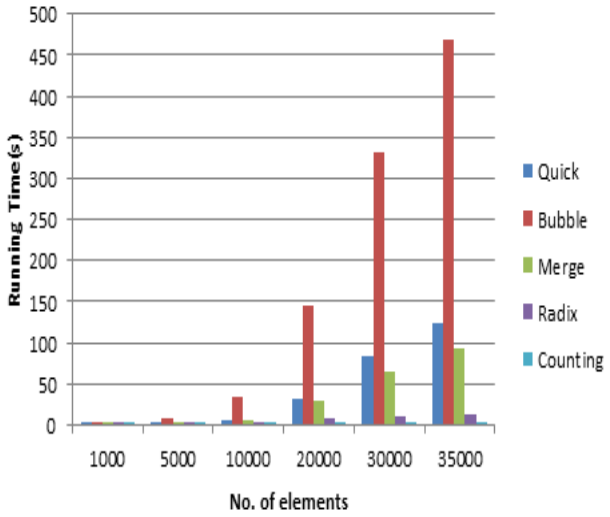

Fig. 1: Graphical representation of the running time of the algorithms against the data size

Table 2: Evaluating the time complexities using Big O notation

Algorithm	Best	Average	Worst
Quick	O(n log n)	O(n log n)	O(n2)
Bubble	O(n2)	O(n2)	O(n2)
Merge	O(n log n)	O(n log n)	O(n log n)
Radix	$O(n \cdot \frac{k}{s})$	$O(n \cdot \frac{k}{s})$	$O(n \cdot \frac{k}{s})$
Counting	$O(n + 2^k)$	$O(n + 2^k)$	$O(n + 2^k)$

Where n is the input size, k is the key size and s is the chunk size

Table 3: Evaluating the memory consumption of the various algorithms in megabytes

No. of elements	Quick	Bubble	Merge	Radix	Counting
1000	2.252	0.811	0.817	0.817	0.812
5000	39.209	1.843	1.880	1.849	1.843
10000	150.465	2.636	3.400	2.615	2.637
20000	590.155	6.343	6.467	6.349	6.343
30000	1323.736	7.139	9.329	7.012	7.141
35000	1798.934	8.492	11.135	8.330	8.492

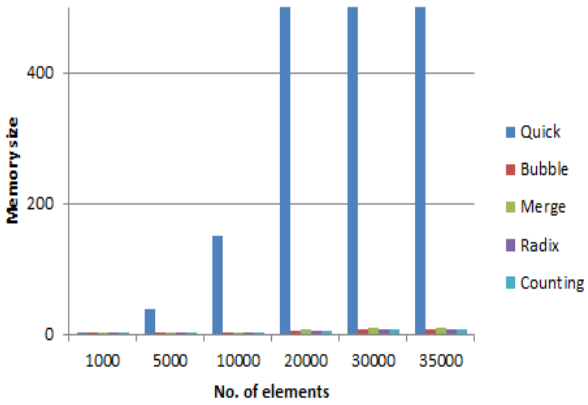

Fig. 2: Graphical representation of the memory consumption of the algorithms against the data size

5. CONCLUSION

In this research, several experiments were carried out to test the performance of the various sorting algorithms with respect to time and memory complexities. The experiments revealed that non-comparison based algorithms are more recommended than the comparison based ones in terms of running time. For example, an average running time of 17.849 seconds was obtained for the non-comparison based algorithms against 476.757 seconds for the comparison ones. The high time complexity recorded in the comparison based algorithms is due to the number of swapping of the individual elements in the array. As the size of the array grows, more swapping is needed to sort the elements, thereby increasing the execution time. For example, the bubble sort algorithm that recorded the highest execution time will never stop its iteration until the highest number in the array bubbles to the right. It was also realized from the experiment that for small array (1000 elements array), the performance of all the algorithms were almost the same. However, a significant change is seen in the comparison based algorithms when the array size becomes very large. With respect to the memory consumption of the algorithms, it was realized that the non-comparison based algorithms utilize less memory than the comparison based ones. For example, an average memory of 27.12 MB was used by the non-comparison based algorithms against 1321.681MB for the comparison ones. This clearly indicate the poor performance of the comparison based algorithms in terms of memory consumption. It was also realized from the experiment that quick sort algorithm which is part of the comparison based ones utilized the largest memory capacity. This problem is due to the recursive nature of the algorithm where pivot is generated for each sub array, before its elements are being swapped. This operation utilizes enormous amount of memory. Even though bubble sort had the highest time complexity from the experiment, it was also seen that the algorithm had a small memory consumption because the swapping operation occurs within a single array and does not require allocation of any new memory from the operating system.

It is therefore clear from the experiment that the non-comparison based algorithms are more efficient than the comparison based ones in terms of time and space complexities.

6. REFERENCES

[1] M.S. Garai Canaan.C and M. Daya. Popular sorting algorithms. *World Applied Programming*, 1:62{71, April 2011.

[2] A.D. Mishra and D. Garg. Selection of Best Sorting Algorithm. *International Journal of Intelligent Processing*, 2(2):363{368, July-December 2008.

[3] Pankaj Sareen. Comparison of sorting Algorithms (on the basis of average case). *IJARCSSE*, 3:522{532, March 2013.

[4] Donald E.Knuth. *The Art of Computer Programming Second Edition*, volume 3. ADDISON-WESLEY, 1998.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2009.

[6] Omar khan Durrani, Shreelakshmi V, Sushma Shetty, and Vinutha D C. Analysis and determination of asymptotic behavior range for popular sorting algorithms. *Special Issue of International Journal of Computer Science Informatics (IJCSI), ISSN(PRINT):2231-5292, Vol-2, Issue-1,2*.

[7] Coenrad Bron. Merge sort algorithm [m1] (algorithm 426). *Commununication , ACM*, 15(5):357, 1972.

[8] Harold H Seward. Information sorting in the application of electronic digital computers to business operations. Master's thesis, M.I.T, 1954.

[9] Clifford A. Shaffer. *A Practical Introduction to Data Structures and Algorithm Analysis Third Edition*. Prentice Hall, April 2009.

[10] D. L. Shell. A high-speed sorting procedure. *Commun. ACM*, 2(7):30{32, July 1959.

[11] B.L Teague, *The Quick sort implementation in PHP*, accessed 13th January 2020, https://medium.com/the-protean-journal/the-quicksort-algorithm-implemented-in-php

[12] J. M Shaffer, *Bubble Sort Algorithm in PHP*, accessed 20th July 2020, https://eresdev.com/bubble-sort-algorithm-in-php

[13] Codexpedia, *Merge sort Implementation in PHP*, accessed 7th June 2020, https://www.codexpedia.com/php/merge-sort-example-in-php/

[14] K.T Toida, *PHP Program - Radix Sort*, accessed 15th June 2020, https://www.alphacodingskills.com/php/pages/php-program-for-radix-sort.php.

[15] I. Wegener, *PHP Program - Counting Sort*, accessed 10th May 2020, https://www.alphacodingskills.com/php/pages/php-program-for-counting-sort.php