Sorting Algorithm Survey

Eman Asem, Nada Badawy, Rinal Hamdy

THE AMERICAN UNIVERSITY IN CAIRO

I. Abstract

There are various problems encountered in many related fields such as computer sciences, database applications, networks, and artificial intelligence. In solving these problems, there are many intermediate steps that must be taken to reach for the final output at the end. One very basic operation that is needed as an intermediate step in many problems and found in the much fundamental application including real-time systems, and operating systems is the sorting operation[1]. Sorting is basically the process of rearranging a set of elements in a particular list. And because researchers are obsessed about the algorithms' optimization and effective problem-solving techniques, they spent a good deal of time improving sorting techniques in order to enhance the overall performance of the fundamental applications as well as the sorting technique itself. One very principal factor that researchers focused on to enhance the performance of execution is the time complexity. In addition, there are many other factors that affect the execution such as memory usage, mode of implementation, stability, adaptability, parallelism, and comparison mode. The sorting techniques that will be addressed in this paper are selection sort, insertion sort, heap sort, quicksort, and bubble sort. This research paper provides an analytical comparison for the sorting techniques mentioned above.

II. Introduction

One of the most important parts in managing data is sorting algorithms. Sorting is the process of arranging items in a correct order either in ascending or descending order. The sorted data items are more efficient than handling random data as it maintains the information and eases the retrieval of it. Therefore, the research in this topic is very important in computer science studies, in early 1956 bubble sort was analyzed[1], whereas many have considered that the problem is solved; however, many researches shed the light on sorting techniques to cope up with the rapidly information growth which leads to many interesting developments in the sorting algorithms. There are various number of algorithms developed to sort data, each one of them has its own technique and procedures in executing. Furthermore, each problem can be solved by more than one sorting technique. For instance, to sort a list of 10 elements, the person can use a variety of techniques and methods while the difference will not be noticed. On the other hand, if the person wants to sort a list of 100000000000 elements it would dramatically differ from one algorithm to the other. Thus, to compare between the sorting algorithms that will be presented in details in this paper, a set of criteria will be used to assess each one of the techniques in order to reach a final conclusion regarding their general method of implementation beside the nature and size of the input datasets. Each method of implementation has many applicable sorting algorithms. This study will consider examples from each group and make a comparison between them to come up with the best solution for a specific situation. However, finding the best and the most efficient sorting algorithm for each situation demands investigating specific criteria to handle the debate. There are many other dimensions to test the efficiency of the algorithm such as complexity, memory usage, recursion, stability, work with comparison operator or not, parallelism and adaptability. Each one of these criteria is so important but here the paper will focus mainly on complexity as the main comparative criteria, and how it differs dramatically from one algorithm to another and how to deal with its limitation to effectively compare between heap, quick, bubble, insertion and selection sort.

[1] K. Al-Kharabsheh, I. AlTurani, A. AlTurani and N. Zanoon, "Review on Sorting Algorithms A Comparative Study", International Journal of Computer Science and Security, vol. 7, no. 3, pp. 120-126, 2013

III. Background

Sorting in computer science is a combination of instructions that is applied to a series of numbers so that they become organized into a specific order; for instance, from highest to lowest and vice versa. Sorting can be done by algorithms which are some right ordered procedures conducted to solve a specific problem. There can be several solutions for the same problem. However, there are algorithms which are more efficient than others and some are faster and there are short cut ways to reach the same solution. Thus, in the paper our concern is to find the best algorithm to reach a required solution. This algorithm usually takes an array of numbers as an input, compare between the elements, operate some instructions on it, and it gives an ordered array as an output. There are many types of sorting methods like bubble sort and heap sort...etc. Each one of the sorting methods has specific instructions which can be classified as efficient or not based on some specifications; for instance, complexity, memory usage, parallelism, stability and adapting.[2]

Complexity: It is the time taken by the algorithm to perform an operation and deliver an output. Often is calculated by the Big O notation which indicates the performance of an algorithm according to the growth rate of the function and represented by a graph of time versus input size (n). O (n), O (n log n) and O (n^2) are the common complexity cases which indicate respectively: linear time complexity which is fair, logarithmic time complexity which is efficient, and quadratic time complexity which is not efficient.

Memory usage: In computer science, the memory is referred to be the RAM which hold the temporary data while completing the operations. In any algorithm, memory usage can be either in place or out of the place. First, “in place” means that the algorithm can sort the data structure within itself without using any auxiliary memory or any help from other data structures, but it can use some extra variables; however, it does not need much memory. On the other hand, “out of the place” refers to the algorithms which needs an extra memory storage and can sort the data by using another data structure (usually arrays). Thus, in place algorithms occupies less memory as it does not create another data structure, but it can increase the size of the existing one. Not all cases the in-place sort are the most efficient way, for example, Merge sort can be done by both algorithms, but the situation of “out-of- place” is more efficient as it decreases the time complexity for it.

Recursion: It is a way of solving problems depending on dividing the problem into smaller subproblems that can be handled easily. The idea is the function calls itself more than one time.

Parallelism: It defines the ability of the algorithm to execute many processes or make calculations simultaneously and effectively.

Stability: stable sorting algorithm keeps the relative order of the elements if they are equal.

Adaptability: The algorithm is adaptable if it changes its behavior and its time complexity value at the time run according to the input data.

IV. Related work

This paper presents an analysis of the performance of different sorting algorithms. Various sorting techniques have been developed over the years, so determining which sorting method is better in a specific scenario depends on multiple effectiveness considerations that differ from a problem to another. That’s why the most important ones of these considerations are picked to represent the criteria of assessment for all sorting techniques for them to be evaluated on a common ground. The criteria that is going to be used in order to assess each sorting technique are computational complexity, memory usage, recursion, stability, parallelism, and adaptability. The algorithms that will be addressed in this paper are selection sort, insertion sort, heap sort, quicksort, and bubble sort. The time and space limitations of these methods are presented based on the worst-case scenario, although with proper precautions it rarely happens.

A**.) Selection sort**: selection sort is an unstable, comparison based algorithm that is considered to be the simplest among all the sorting techniques. However, it was found that it perfectly works for only small elements’ lists. The general method of its implementation is selection. This algorithm works by scanning to find the smallest element in the list, swapping, and interchanging, this is done to every element in the list, so its computational complexity is O(n^2) which makes it very inefficient in large amounts of data because of doing many comparisons. On the basis of memory usage, selection sort algorithm is implemented internally, or in place, in the memory. Regarding its implementation, it can be implemented either iteratively or recursively [3].[3]

B.) Insertion sort: insertion sort is a stable, comparison based algorithm that perfectly work with small data sets. The general method of its implementation is insertion. This algorithm mainly inserts the elements in the list one by one in a proper position in a newly created list, so its computational complexity is O(n^2) which leads it to be very time

[2] Mishra.D.A, 2009. Selection of Best Sorting Algorithm for a Particular Problem. Computer Science And Engineering Department Thapar University.

[3,4,6] M. Gul, N. Amin and M. Suliman, "An Analytical Comparison of Different Sorting Algorithms in Data Structure", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 5, no. 5, pp. 1289-1298, 2015.

consuming as it scans the list every time before inserting a new element in the newly established list. On the basis of memory usage, insertion sort algorithm is implemented internally, or in place, in the memory. Regarding its implementation, it can also be implemented either iteratively or recursively.[4]

C.) Heap sort: heap sort is an unstable, comparison based algorithm that is very fast. The general method of its implementation is selection. This algorithm works by building the heap according to the input data then the maximum element is stored at the root of its imaginary tree, then replaced by the last item in the heap while decrementing the size of the heap by one, so it basically works by visualizing the data elements of the array as binary data structure tree called heap. The visualization of the elements can be considering as a parent (node) and each parent has two children (lift and right). If the index of any element in the array is i, then the element of 2i+1 will be the left child and the element in 2i+2 will become the right child. Also, the parent of any element at index i can be found in index (i-1)/2. That's why its computational complexity in all cases is O(n log(n)) which leads it to be very fast and widely used. On the basis of memory usage, heap algorithm is implemented internally, or in place, in the memory. Regarding its implementation, it can also be implemented either iteratively or recursively although the iterative version was found to be unnecessary[4].[5]

D.) Quicksort: quick sort is an unstable, comparison based algorithm that is considered to be one of the fastest algorithms on average as well as an option for getting a perfect locality for cache behavior and reference in arrays. The general method of its implementation is Partitioning. This algorithm mainly works by following a divide and conquer strategy to end up by having a sorted array from two sub-lists that are created from the very original list. What it does is picking a pivot element and then recursively sort the array based on whether that other elements are greater or less than that pivot element, so its computational complexity is O(n^2) which leads it to have a very slow performance. Also, poor choices of the pivot element leads to very slow performance. On the basis of memory usage, quicksort sort algorithm is implemented internally, or in place, in the memory. Regarding its implementation, it must be implemented recursively[5].[6]

E.) Bubble sort: bubble sort is a stable, comparison based algorithm that known for its ease of implementation. However, it is very inefficient for large data sets. The general method of its implementation is exchanging. This algorithm mainly works by comparing the adjacent elements in the required array and swapping the elements based on the larger element in the pair, so its computational complexity is O(n^2) which leads it to be very time consuming as it scans the list every time before inserting a new element in the newly established list. On the basis of memory usage, insertion sort algorithm is implemented internally, or in place, in the memory. Regarding its implementation, it can also be implemented either iteratively or recursively.[7]

V. Discussion:

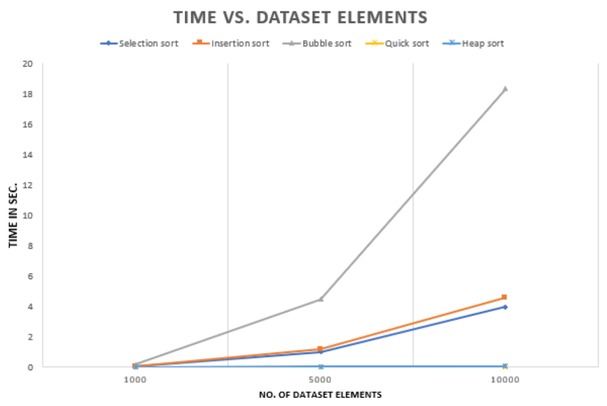
This paper provides an analysis to the comparison and non-comparison-based sorting algorithms merely based on algorithms' computational complexities to reach the most preformat and efficient one. By comparing between complexities of selection, insertion, bubble, quick and heap sort, heap sort algorithm was recursively implemented using a logarithmic-time priority queue instead of using the regular linear search so that it overcomes the rest of the algorithms in the worst-case runtime scenarios. As it has big O notation of O (n log n) while all the others have O (n^2). The runtime analysis of the heap sort mainly depends on implementing the happify recursive function as it is the place where all the sets of data comparisons are to be executed. The basic heap operation in each heapify operation takes O (log n). adding to that, we need to sort n elements so the total extraction time for the maximum number of elements is O (n log n) [9][8]. After showing how is the heap sort has the least growth rate among the others sorting techniques as the big O is only O (n log n). Now we should discuss that the time complexity is not the only criteria which is used to assess the efficiency of the algorithm as it has many limitations. For example, in practice quick sort with O (n^2) is faster than heap sort with O (n log n). The big O notation does not tell about how fast an algorithm is, it tells about the scale it grows with. This scale tells how the algorithm behave for large data input and can decide which algorithm will be faster for this high range of inputs. These limitations affect mostly in choosing the efficient sorting algorithm. To narrow the research, the paper will consider the size of the data items input and discuss how can different algorithms behave according to these inputs. To validate our thoughts, we

[4,5] R. Verma1 and J. Singh, "A Comparative Analysis of Deterministic Sorting Algorithms based on Runtime and Count of Various Operations", International Journal of Advanced Computer Research, vol. 5, no. 21, pp. 380-385, 2015.

[8] Mount, Dave. "CMSC 251: Algorithms1 Spring 1998." (1998).

took a tested data from a survey [8][9]. We used these data to come up with a visual representation of them in graph (1), (2) and (3). Now let us choose the best sorting algorithm based on the context of the input data.

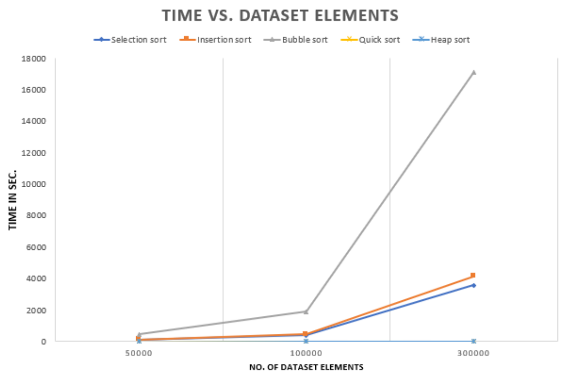
A. Small Dataset:



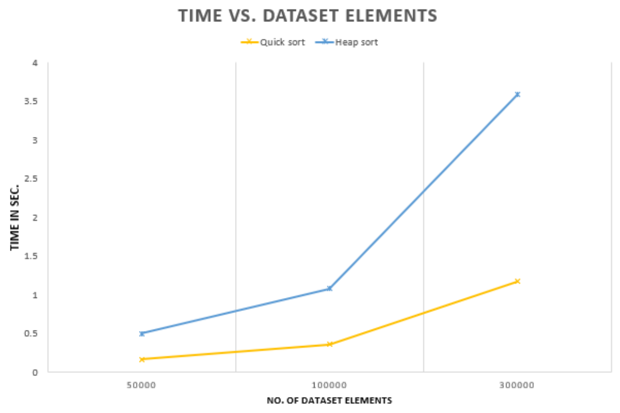
Graph (1): Time vs. Small dataset with respect to all sorting algorithms.

In a problem, if n is too small, then the big O notation is a wrong model for choosing the most appropriate algorithm. As shown in the graph for the small data items all algorithms near have the same deviation by slightly differences between them. Thus, our discussion here will not asses the algorithm according to the time complexity. For a small data quick and heap are not the beast choice as in quick sort it will be challenging with choosing a good pivot to catalyze the process, furthermore, it needs modest space usage. Heap also are not practical and need more complex implementation. For this case, bubble, insertion and selection sort are more suitable choices as they sort the algorithm in place and need limited auxiliary memory, O (1). Moreover, they are practical and have simple implementation process. However, insertion sort is better than the selection sort when the data is closed to be sorted as the selection sort must scan all the remaining elements to locate the next element in the list, while, the insertion scans only the elements that it needs (does not go for whole array). Thus, for the small datasets the paper goes with insertion sort.

B. Large dataset:



Graph (2): Time vs. Large dataset with respect to all sorting algorithms.



Graph (3): Time vs. Large dataset with respect to Heap and Quick sort.

Selection sort, insertion sort and bubble sort were proven to be effective and fast with small datasets but when it comes to large datasets, these methods become inefficient as they reach O (n^2) which will take long time to deliver an output[3]. Therefore, the most effective methods to be used with large datasets are quicksort and heap sort according to graph(2). According to graph(3), quicksort is faster and more effective than heapsort. The comparison between quicksort and heap sort is depending on the practical more than the hypothetical results. Although heap sort is using O (n log n) with big datasets even in worst case in while quicksort is O (n^2), quick sort is faster than heap sort for average amount of data. That is because quicksort is randomized algorithm so the probability of sorting the list in less time depends on the position of the pivot element. Thus, the worst case is very hard to hit. Furthermore, quick sort is easily optimized by simple cache strategies. furthermore, quicksort does not require additional storage as it uses in place memory. Thus, for the large datasets the paper goes with quicksort.

[9] Karunanithi, Ashok Kumar. "A survey, discussion and comparison of sorting algorithms." Department of Computing Science, Umea University (2014)

VI. Conclusion:

This paper held a comparison between some sorting algorithms; heap, quick, insertion, selection and bubble sort. First, it analyzed the performance of these techniques according to the time complexity and found that heap sort has the least and the most efficient big O notation among them. However, having the most efficient time complexity does not qualify heap sort to be the most efficient algorithm between them. Thus, after another comparison in the paper, it was found out that for the small input performance of all the five algorithms are almost suitable but the most efficient one was insertion sort. On the other hand, for the large datasets quicksort was the chosen one.

VII. Literature cited:

[1]M. Gul, N. Amin and M. Suliman, "An Analytical Comparison of Different Sorting Algorithms in Data Structure", International Journal of Advanced Research in Computer Science and Software Engineering, vol. 5, no. 5, pp. 1289-1298, 2015.

[2]R. Verma1 and J. Singh, "A Comparative Analysis of Deterministic Sorting Algorithms based on Runtime and Count of Various Operations", International Journal of Advanced Computer Research, vol. 5, no. 21, pp. 380-385, 2015.

[3]K. Al-Kharabsheh, I. AlTurani, A. AlTurani and N. Zanoon, "Review on Sorting Algorithms A Comparative Study", International Journal of Computer Science and Security, vol. 7, no. 3, pp. 120-126, 2013.

[4]G. Kocher and N. Agrawal, "Analysis and Review of Sorting Algorithms", International Journal of Scientific Engineering and Research (IJSER), vol. 2, no. 3, pp. 81-84, 2014.

[5]Mishra.D.A, 2009. Selection of Best Sorting Algorithm for a Particular Problem. Computer Science And Engineering Department Thapar University.

[6]J. Hammad, "A Comparative Study between Various Sorting Algorithms", IJCSNS International Journal of Computer Science and Network Security, vol. 15, no. 3, pp. 11-15, 2015.

[7]R. Mehra, "Research Paper on Sorting Algorithms", 2014. [Online]. Available: https://www.cse.iitk.ac.in/users/cs300/2014/home/~rahume/cs300A/techpaper-review/5A.pdf. [Accessed: 19- Sep- 2018].

[8] Karunanithi, Ashok Kumar. "A survey, discussion and comparison of sorting algorithms." Department of Computing Science, Umea University (2014)

[9] Mount, Dave. "CMSC 251: Algorithms1 Spring 1998." (1998).

[10] M. Alam and A. Chugh, "Sorting Algorithm: An Empirical Analysis", *International Journal of Engineering Science and Innovative Technology (IJESIT)*, vol. 3, no. 2, pp. 118-126, 2014.