Global Scientific JOURNALS

GSJ: Volume 9, Issue 4, April 2021, Online: ISSN 2320-9186
www.globalscientificjournal.com

Comparison between Quicksort, MergeSort and Insertion Sort

Dinesh Bajracharya | **Devika Acharya**

dinacharya@gmail.com | joniacharya81@gmail.com

Abstract

Sorting plays an important role in computer programs and human lives. Sorting of data is a time intensive process. Several algorithms have been developed to sort list of data; each algorithm has its own merits and demerits. QuickSort, MergeSort and Insertion Sort are three well-known and used sorting algorithms. Quick sort and merge sort are based on the concept of divide and conquer and are considered as the most efficient sorting algorithms. Insertion sort uses concept of arranging cards. This research work compares these different sorting algorithms in terms of time complexity with the help of programs written in JAVA language using different sizes list with varying nature of data.

Keywords: Quicksort, MergeSort, Insertion Sort, Java, Big-Oh

Introduction

Sorting is a method of arranging data in ascending or descending order. Arranging a large list data in ascending order or descending order is time-intensive process. Several computer programs work on sorted data; searching will be fast in sort list of data. Several sorting algorithms have been developed to sort list (array) of data. These different algorithms have their own advantages and disadvantages. The performance of these algorithms depends of size of the list to be sorted, nature of data in the list and is measured in terms of time and space used by the algorithms while sorting the data. Some algorithms are very fast at sorting data, some are not; some algorithms use more memory space, some don't.

Among several purposed sorting algorithms: Quicksort, MergeSort and Insertion sorts are well-known. This research article compares Quicksort, MergeSort and Insertion sort in terms of the time taken to sort a list of data. The time taken by an algorithm to complete task is measured by counting the number of key operations of the algorithm and space required by the algorithm is calculated by counting amount of memory usemd by the algorithm during its execution.

The big-Oh notation is defined to specify the running time and space requirements of an algorithm in terms of some parameter 'n' [5]. Big-Oh notation for Quicksort and MergeSort is O(nlogn) and $O(n^2)$ for the Insertion sort, where n is the number of elements. Algorithms is O(nlogn) are every efficient compared to algorithms with $O(n^2)$.

Objective of This research work

The objective of this research is to find out how the considered sorting algorithms: QuickSort, MergeSort and Insertion Sort perform with the different datasets of different sizes and nature. The datasets consist of sorted unique data, unsorted unique data and repeated unsorted data.

Method

To compare considered sorting algorithms following tasks were performed.

- Three different Java Programs were written: one for quicksort, second for mergesort and last one for insertion sort.
- A JAVA program was developed to generate more than five hundred thousand random numbers using Math.random() method and saved in two different the files, one with the sorted numbers another with the unsorted numbers.
- Another set of data of size four hundred seventy-one thousand was created in MS Excel using randbetween() function, this file contained lots of duplicate values.
- In the sorting program fourteen different sized (100, 200, 400, 800, etc.) arrays were created and populated those arrays with the data from the files with the random numbers.
- All those arrays of different sizes were sorted using Quick Sort, Merge Sort and Insertion Sort programs. The programs were run for several times. Time taken by to sort each array by each sorting algorithm was tabulated, calculated average time taken by each algorithm.
- Discussion was written based on the experimental output and the literature reviews.

Theories

The sorting process includes comparisons, movements and swapping of elements of the list (array). The elements are moved or swapped as necessary after comparing elements of the list with each other. It is not necessary that the number of movements and the number of comparisons to be equal, these two operations depend on the size of input (size of list), nature of data in the input. The number of movements and comparisons play vital role in the calculation of the time complexity of the algorithms.

Time complexity of an algorithm is defined as the time taken by an algorithm to complete execution and is expressed as a function of the size of a problem. "The limiting behavior of the complexity of an algorithm with the increase in size of input is called the asymptotic time complexity." The asymptotic complexity of an algorithm determines the size of problems that can be solved by the algorithm. If an algorithm processes 'n' number of input data in cn^2 time, where c is some constant, then we say that the time complexity of the algorithm is $O(n^2)$, "order of n^2" [2].

Elements of the list are sorted by comparing elements of the list with each other. Knowing number of comparisons is not possible, an approximate value is computed. The number of comparisons and movements is approximated with the big-Oh notation by mentioning the order of magnitude of these numbers, the order of magnitude can vary as the initial order of the data plays important role on that magnitude[3]. Big-Oh notation provides the worst-case time-complexity for the algorithms, that is, whatever the worst situation be the algorithm will not take more than specified amount of time to complete execution. Following questions are important when calculating efficiency of sorting:

- How much time computer take to order ordered data?
- Does algorithm recognize that data is ordered or not?

Some sorting algorithms perform same number of operations regardless of the initial ordering of data, some don't. It is not necessary that number of movements and number of comparisons to be equal for all the algorithms.

Quicksort

Quicksort is considered as one of the fastest sorting methods. Quicksort is based on the divide-and-conquer paradigm. It consists of three-steps: Divide, Conquer, and Combine to sort an array A[p..r].

- Divide: Find any element (first, last, middle etc.) of the array A[1…r] and let the position of that element be q. Divide array A into two sub arrays (partitions) A[p…q-1] and A[q+1…r] such that elements less than or equal to element A[q] are stored in A[p..q-1] and elements greater than or equal to element A[q] are stored in A[p + 1...r].
- Conquer: Sort the two subarrays (partitions) A[p..q-1] and A[q + 1...r] by recursive or non-recursive calls to quicksort.
- Combine: The subarrays are sorted in place, no work needed to combine them. The entire array is sorted.

The running time of quicksort is affected by the fact where the partitions created are of equal size or not, the partitions depend on the element used for partitioning. If the partitions are even then the quicksort runs asymptotically as fast as merge sort otherwise it can run asymptotically as slow as insertion sort [7]. The initial ordering of the data also affects the performance of quicksort, quicksort does not perform well with the sorted data. If already sorted sequence (list) with no duplicates and first element is chosen as pivot, then the Quicksort would take a quadratic number of steps[2]. The Big-Oh notation for quicksort is O(nlogn).

Insertion Sort

Insertion sort is an efficient algorithm for sorting a small sized lists or arrays. Insertion sort works the way many people use to sort a hand of playing cards. Initially hand is empty, one card at a time is picked and placed in its proper place by shifting other cards if necessary. Correct position of the card is found by comparing the picked card with cards already in the hand.

The time taken by the Insertion sort depends on the input size, if the number of elements is very high the insertion sort will be slow but if the number of elements is small then it will be fast. The worst-case running time of insertion sort is $O(n^2)$ where n is the number of elements in the list to sort. If the list is already sorted then insertion sort will be fast as movements of the elements are not done, only comparison of data is done.

Merge Sort

A basic method to write good algorithm based on the divide-and-conquer paradigm is to divide a list into approximately equal halves instead of unbalanced halves. Merge Sort divides list of equal sizes (Alfred V. Aho, 2000). Mergesort algorithm nearly follows the divide-and-conquer paradigm. It operates as follows:

Divide: Divide the n-element list to two subsequences of size n/2.
Conquer: Sort the two subsequences recursively using merge sort.
Combine: Merge the two sorted subsequences and get sorted list.

The running time of the merge sort is O(nlogn) where n is the size of list (array). The order of elements in the list does not affect the running time of mergesort.

Related Works

Oladipupo has compared quicksort with merge sort and found Quick sort to be faster for smaller sized arrays than merge sort and merge sort was faster for larger sized arrays [6]. An experiment was performed to see which is the fastest sorting algorithm among considered six different algorithms: Selection Sort, Insertion Sort, Merge Sort, Quick Sort, Bubble Sort, and Comparison Sort using a program written in C++ with data set of random sequence of sizes 10000, 20000, and 30000. The result of the experiment showed that the time taken by all the algorithms are similar for the smaller sized data sets, but Quick Sort was very fast in sorting larger sized data sets than other sorting algorithm[4]. A comparative study between median, heap and quick sort on the varying sized data set was performed using program written in C language and found heap sort to be better than median sort and quick sort in terms of space usage and taken required to sort data sets[1]. V.P. Kulalvaimozhi [8] has studied performance analysis of several sorting algorithms (Bubble, Insertion, Selection, Quick, Merge, Heap, Binary Tree, Shell, Address Calculation, Radix sort) using program developed in C++ language and found Quick sort to the preferred one for the program which need very fast data sorting, and have also found quick sort taking longer running time is some occasions.

Results and Discussion

The comparison between Quicksort, MergeSort and Insertion Sort was performed using JAVA programs. Those JAVA programs were executed to sort different natured arrays with varying number of elements and time taken to sort those different arrays with varying size were tabulated in table 1, 2, and 3 respectively. The tables show size of arrays, time (in nanoSeconds) taken by the sorting algorithms to sort the arrays, name of sort algorithm (among three) which took longest time to sort array, comparison between quicksort and mergesort and comparison between mergesort and insertion sort.

Table 1 Time taken by QS, MS, and IS to sort arrays of different sizes with unique ordered data

					Maximum Time taken		
Runs	Size	QS	MS	IS	Among three	Between MS & IS	Between QS & MS
1	100	217167	90300	6633	QS	MS	QS
2	200	919833	354900	20000	QS	MS	QS
3	400	5520800	551767	66500	QS	MS	QS
4	800	5394733	1174600	363100	QS	MS	QS
5	1600	8462300	1709867	209333	QS	MS	QS
6	3200	2187600	1807900	452867	QS	MS	QS
7	10000	19157933	3285100	575700	QS	MS	QS
8	20000	70421200	4932333	1363867	QS	MS	QS
9	40000	274845367	7470967	3352100	QS	MS	QS
10	80000	1350954067	12996133	2110367	QS	MS	QS
11	160000	5487292333	20252067	563300	QS	MS	QS
12	320000	22291074667	21436367	1042333	QS	MS	QS
13	400000	40991830800	23809067	814900	QS	MS	QS
14	471000	58209834933	46904967	310300	QS	MS	QS

Table 1 shows the time taken to sort list of number of different sizes by considered sorting algorithms. For the list of sorted unique data of different sizes, both quicksort and merge sort are seen to underperform compared to insertion sort. Merge sort took less time to sort all the data set compared to the quicksort.

Table 2 Time taken by QS, MS, and IS to sort arrays of different sizes with unique unordered data

					Maximum Time taken		
Runs	Size	QS	MS	IS	Among three	Between MS & IS	Between QS & MS
1	100	39600	94633	82233	MS	MS	MS
2	200	242200	439100	788100	IS	IS	MS
3	400	347833	561167	2545100	IS	IS	MS
4	800	748633	997533	6693033	IS	IS	MS
5	1600	1234333	1816267	3651333	IS	IS	MS
6	3200	1517633	3358033	2381567	MS	MS	MS
7	10000	6317633	2578467	12817400	IS	IS	QS
8	20000	3485333	6704400	50206100	IS	IS	MS
9	40000	6842733	12410800	197061067	IS	IS	MS
10	80000	10424067	23007633	789233833	IS	IS	MS
11	160000	15409833	42087733	3177193133	IS	IS	MS
12	320000	30602000	39279100	3458826900	IS	IS	MS
13	400000	39095300	71526500	23521605300	IS	IS	MS
14	471000	45520467	86155567	34343372733	IS	IS	MS

Table 2 shows that Quicksort is very fast when the list is not sorted and values are unique. In only one case (dataset with size 10000), quicksort was inferior than MergeSort, but for the remaining datasets quicksort was found very efficient . For the unique data sets, insertion sort was found to be very slow than quicksort and mergesort.

Table 3 Time taken by QS, MS, and IS to sort arrays of different sizes with unordered data with duplicates (more than 30% repetitions)

					Maximum Time taken		
Runs	Size	QS	MS	IS	Among three	Between MS & IS	Between QS & MS
1	100	131800	81833.33	77066.66667	QS	MS	QS
2	200	794966.6667	604166.7	474466.6667	QS	MS	QS
3	400	3043500	573200	491300	QS	MS	QS
4	800	5171100	1029033	1090666.667	QS	IS	QS
5	1600	5489100	1413933	2627066.667	QS	IS	QS
6	3200	6485966.667	3315300	3005766.667	QS	MS	QS
7	10000	46644100	3417300	1487966.667	QS	MS	QS
8	20000	177853933.3	5331967	581000	QS	MS	QS
9	40000	805718200	10584400	977833.3333	QS	MS	QS
10	80000	3575455400	21256000	1774500	QS	MS	QS
11	160000	12150656833	26695300	715772100	QS	IS	QS
12	320000	12789312233	40157767	2601263700	QS	IS	QS
13	400000	13110925767	57453000	11959905233	QS	IS	QS
14	471000	13060171133	68823633	17281442133	IS	IS	QS

Table 3 shows that if the dataset contains lots of duplicate values, then the quicksort is slow compared mergesort and insertion sort; Megresort is better than quicksort, and between merge sort and insertion in some data sets merge sort is better, in other data sets insertion sort. Merge sort is found better than Insertion sort for larger data sets.

Quicksort is considered as the best sorting algorithms, and most of the related articles considered for this study also supported this fact. But the experimental result of this research work showed that quick sort is better for unsorted list with unique values only; quicksort proved to be better than mergesort and insertion sort for almost all the considered unsorted lists of different sizes with unique values. But for ordered data set and data set with lots of duplications quicksort was found to be inefficient. For ordered set of data, insertion sort was found better than quick sort and merge sort. For data set with duplicate values with big size lists mergesort was found better than quicksort and insertion sort.

The performance of quicksort is affected by: choice of the pivot, and the initial ordering of the data. It is difficult to get pivot which can distribute data evenly between two sub arrays; if the data is ordered then the sublists are just to be created as pivots will be already in their respective places; and if there are duplicates then there will be possibility of lots of data to hang near the pivot.

The measured time for sorting data by the considered algorithms will vary greatly from machine to machine, from trial to trail even on the same machine as many other processes will be running in the background sharing Central Processing Time (CPU) and the memory of the machine[5]. This fact is necessary to consider while comparing algorithm.

Conclusion

The experimental results of this research work showed that quicksort is good for unsorted list with unique values only for both small and large size lists. In case of sorted list, insertion sort was found better than quicksort and mergesort and for list with duplicate values mergesort was found better for larger arrays than quicksort and insertion sort. The nature and size of list have good affect over the sorting algorithms and also in how is experiment conducted. There will be background processes sharing CPU and main memory which will affect the time complexity of the algorithms.

References

[1] Adesina, O. (2013). A Comparative Study of Sorting Algorithms. *African Journal of Computing & ICT*, 199 - 205.

[2] Alfred V. Aho, J. E. (2000). *The Design and Analysis of Computer Algorithms.* Singapore: Addison Wesley Longman.

[3] Drozdek, A. (2001). *Data Structures and Algorithms in Java.* Singapore: Brooks/Cole Thomson Learning.

[4] Khalid Alkharabsheh, I. A. (2013). Review on Sorting Algorithms A Comparative Study. *Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani & Nabeel Imhammed Zanoon International Journal of Computer Science and Security (IJCSS), Volume (7) : Issue (3)*, 120 - 126.

[5] Michael T. GoodRich, R. T. (2014). Data Structures and Algorithms in Java. In R. T. Michael T. GoodRich, *Data Structures and Algorithms in Java* (pp. 164 - 165). Wiley.

[6] Oladipupo, E. T. (2020). Comparative Study of Two Divide and Conquer Sorting Algorithms: QuickSort and MergeSort. *Procedia Computer Science*, 2532-2540.

[7] Thomas H. Cormen, C. E. (2002). *Introduction to Algorithms.* Connaught Circus: Prentice-Hall of India.

[8] V.P.Kulalvaimozhi, M. R. (2015). Performance Analysis of Sorting Algorithm. *International Journal of Computer Science and Mobile Computing*, 291-306.