Comparative Stack Space Performance Analysis Of O(nlogn) Sorting Algorithms

Abhinav Yadav, Dr. Sanjeev Bansal

Abstract— Sorting Algorithms are of great significance in various areas and is therefore a fundamental research topic in Computer Science. These algorithms pave the way for other operations (e.g. insert, delete, search) to execute faster on some set of records. O(nlogn) algorithms have been experimented with randomly generated records. Based on the experiments and analysis, we have summarized the result statistically and analytically in this paper. It has been analyzed that on the basis of the nature of input choosing a specific sort algorithm sometimes with some variation is of vital importance, change in the nature of input leads to huge difference in the execution time and memory consumption of these algorithms.

Index Terms— Sorting, Quick Sort, Merge Sort, Heap Sort, nlogn, Stack Space.

1 Introduction

In order to perform the operations on the data efficiently it is important that the data must be in an order, either ascending or descending. Sorting is a procedure that orders data and thus enhances the efficiency of other operations to be performed on the data. We have performed experiments on three sorting algorithms for analytical study: Quick Sort, Merge Sort, and Heap Sort. All the three algorithms belong to O(nlogn) category as there execution time is of order O(nlogn). For experiments randomly generated large data sets have been used. Three types of input sequence were used: Unsorted, Sorted, Reverse Sorted. We have focused the analysis on the three factors: Time Complexity, Space Complexity and Maximum Stack Utilization at a particular time. Output produced by all the algorithms (sorted order) is an ascending sequence, and dynamic memory allocation has been used.

2 The Three Sorting Algorithms

The three sort algorithms that were experimented with are Quick Sort, Merge Sort and Heap Sort. For large data set these algorithms are very popular and efficient. These algorithms are heavily used with some variations in order to handle some exceptional worst case situations. Algorithms used for study are defined first in this section. Performances and results of the algorithms have been summarized in the next section.

2.1 Quick Sort

The recursive quick sort procedure implemented in C language for experiments is from [1]. The partition procedure used is Hoare's Partition [1]. The time complexity of this Quick Sort is O(nlogn) except when the input sequence is sorted or reverse sorted i.e. the worst case. In worst case the algorithm requires $\sum_{i=0}^{n-1} i = 0(n^2)$ comparisons.

If algorithm partitions the input sequence in two equal parts, best case occurs with the running time complexity of O(nlogn) which follows equation(1) :

$$\mathbf{T(n)} = \begin{cases} \mathbf{\Theta(1)}, & \mathbf{if\ n \le 1} \\ \mathbf{2T\left(\frac{n}{2}\right) + \Theta(n)} & \mathbf{otherwise} \end{cases} \qquad \mathbf{(1)}$$

When the input sequence is sorted or reverse sorted the algorithm partitions the input sequence in two parts such that one part contains one element and the other contains n-1 elements and the algorithm follows equation (2) which gives the time complexity of order O(n2).

$$\mathbf{T(n) = T(n-1) + \Theta(n)} \qquad \mathbf{(2)}$$

When unbalaced partion is made by the algorithm e.g. equation (3) , (4) the running time complexity of algorithm is O(nlogn).

$$\mathbf{T(n)} = \begin{cases} \mathbf{O(1)}, & \mathbf{if\ n \le 1} \\ \mathbf{T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + \Theta(n)} & \mathbf{otherwise} \end{cases} \qquad \mathbf{(3)}$$

$$\mathbf{T(n)} = \begin{cases} \mathbf{O(1)}, & \mathbf{if\ n \le 1} \\ \mathbf{T\left(\frac{n}{4}\right) + T\left(\frac{3n}{4}\right) + \Theta(n)} & \mathbf{otherwise} \end{cases} \qquad \mathbf{(4)}$$

2.2 Merge Sort

The Merge Sort algorithm used for experiment is from [2]. The algorithm divides the array in two equal parts, sorts both the parts recursively and then using Merge procedure merges both the sub arrays.

This algorithm Merge Sort follows equation (1) and its

- *Abhinav Yadav is currently pursuing masters degree program in Computer Science & Engineering, Amity University, India.*
- *E-mail: abhinavanihba@gmail.com*
- *Prof. (Dr.) Sanjeev Bansal currently holds position of Director –Amity Business School, Amity University, India.*
- *E-mail: sbansal1@amity.edu*

running time complexity is O(nlogn).

2.3 Heap Sort

The Heap Sort procedure is from [2][3]. The Heap Sort algorithm has a running time complexity of O(nlogn). The recursive Max_Heapify procedure of the algorithm is used to maintain the Max Heap property of the array visualizing it as a tree i.e. each child of a node is lesser than the node itself. Here children of ith node are (2*i)th node and (2*i+1)th node. There are n-1 calls to the procedure Max_Heapify in Heap_Sort procedure. All n-1 calls to Max_Heapify in Heap_Sort procedure takes O(logn) time. The procedure Max_Heapify follows equation (5).

$$\mathbf{T(n) \leq T\left(\frac{2n}{3}\right) + \Theta(1)} \qquad \mathbf{(5)}$$

3 Experimental Study

Performance measurements have been done on Intel Core i5 CPU @ 2.40 GHz. and 3.42 GB RAM, with Windows XP Operating System installed. GCC compiler is used for executing programs. For running time calculations time functions are used and data generation is done using random function.

3.1 Experimental Results

When the input given to the algorithms is a random sequence and the input scale varied from 10,00,000 to 1,00,00,000 time taken by the algorithms to execute is demonstrated by table 1 and figure 1.

TABLE 1

TIME COSTS OF ALGORITHMS UNDER UNORDERED SEQUENCE

No. Of Records	Quick Sort	Merge Sort	Heap Sort
1000000	.218	0.593	0.562
2000000	.438	1.359	1.281
3000000	.687	1.906	2.093
4000000	.937	2.578	2.968
5000000	1.172	3.281	3.891
6000000	1.438	3.953	4.828
7000000	1.703	4.656	5.781
8000000	1.938	5.328	6.797
9000000	2.203	6.079	7.812
10000000	2.469	6.766	8.828

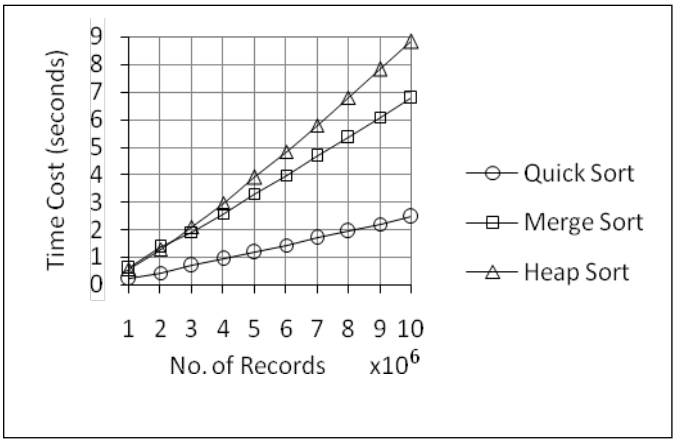

Figure 1. Time Cost comparison of algorithms under unordered sequence

For unordered sequence figure 1 shows that Quick Sort from the very beginning i.e. for 1000000 records outperformed the other two algorithms and with the increase in no. Of records the running time difference between Quick Sort and the other two increased heavily. Up to 3000000 no. Of records the difference between time costs for Merge Sort and Heap Sort was not much but as the no. Of records became larger Merge Sort showed more efficiency.

When the input sequence is sorted the time costs of the three algorithms are demonstrated by table 2 and figure 2.

TABLE 2

TIME COSTS OF ALGORITHMS UNDER SORTED SEQUENCE

No. Of Records	Quick Sort	Merge Sort	Heap Sort
100000	0.187	0.000	0.000
200000	0.485	0.015	0.015
300000	1.031	0.015	0.031
400000	1.734	0.032	0.031
500000	2.688	0.047	0.047
600000	3.766	0.063	0.047
700000	5.141	0.063	0.062
800000	6.719	0.078	0.078
900000	8.609	0.093	0.093
1000000	10.469	0.094	0.094

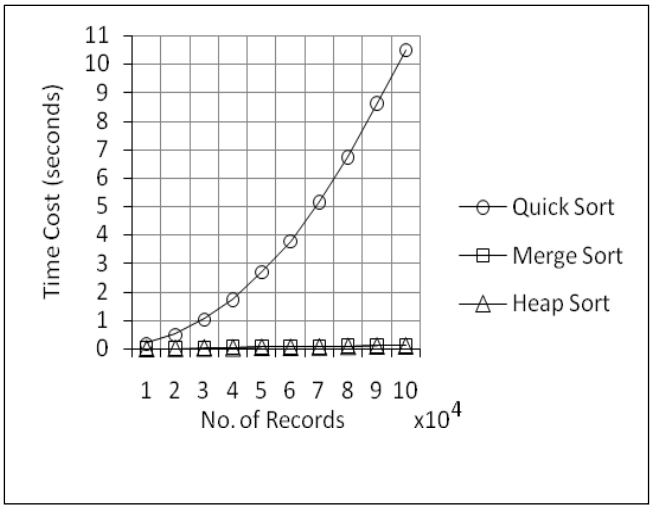

Figure 2. Time Costs comparison of algorithms under sorted sequence

As table 2 and figure 2 depicts when the input sequence is sorted from it is the Quick Sort that becomes slow sort. Merge Sort and Heap Sort took almost same time to execute on the same input size. Where Merge Sort and Heap Sort took less than 1 second to execute on the input size of 1000000 records, Quick Sort took more than 1 second to execute on the input size of 300000. So it is obvious that using Quick Sort for sorting a sorted input sequence could be troublesome.

When the input sequence is in reverse sorted order the time costs of the three algorithms are demonstrated by table 3 and figure 3.

TABLE 3

TIME COSTS OF ALGORITHMS UNDER REVERSE SORTED SEQUENCE

No. Of Records	Quick Sort	Merge Sort	Heap Sort
100000	0.203	0.000	0.000
200000	0.469	0.015	0.015
300000	1.031	0.031	0.016
400000	1.719	0.032	0.032
500000	2.672	0.047	0.032
600000	3.813	0.063	0.047
700000	5.125	0.063	0.062
800000	6.766	0.078	0.063
900000	8.469	0.078	0.078
1000000	10.360	0.109	0.094

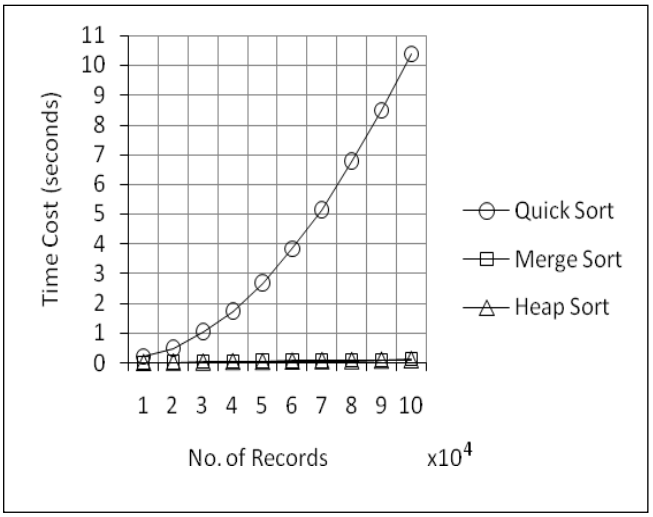

Figure 3. Time Costs comparison of algorithms under reverse sorted sequence

When the input sequence was reverse sorted the running time costs of the three algorithms were almost same as under the sorted sequence. Quick Sort was the worst and has quadratic growth as shown in figure 3. The difference between the running time costs of Merge Sort and Heap Sort was very less.

3.2 Performance Analysis

In order to analyse the performance of the three sorting algorithms, some factors have been taken into account. Firstly, time complexity i.e. time taken by an algorithm to execute. With the increase in the size of input running time also increases. So, if the size of input is n then we can have a function f(n) that determines the running time of algorithm. Secondly, space complexity which is all about the memory used by the algorithm to perform the task. Memory is an important aspect in order to analyze an algorithm because of its limitedness. Space Complexity can also be defined as a function of the size of input. Thirdly, maximum stack space consumed by an algorithm at a particular time. Because all the three algorithms are recursive they consume stack space for storing the information about the recursive procedure as they go deep in to recursion. Although the factor, stack space is a part of the memory factor yet it is important to analyze this separately because of memory constraints on compiler.

Quick Sort has O(nlogn) time complexity in best and average case. Space complexity is O(logn) in best and average case and that is due to stack space usage of the algorithm. In worst case Quick Sort has a time complexity of O(n2). Space Complexity in worst case could be O(n). Either for sorted sequence or reverse sorted sequence the stack depth becomes O(n). The reason behind is serving the request of recursion first for the longer sub array. Here it is important to implement the recursive procedure carefully using tail recursion and first serving the request of sorting the smaller sub array[4][5][6].

Merge Sort has O(nlogn) time complexity in best, average

and worst case. Space Complexity of Merge Sort is of order O(n) in all the cases, where as the stack space consumed is of order O(logn) because it always divides the array into two parts. In average case Quick Sort has an edge over Merge Sort but in the worst case scenario Merge Sort outperforms Quick Sort.

Heap Sort has O(nlogn) time complexity in best, average and worst case same as with the Merge Sort. Space Complexity of Heap Sort is O(n) in all the cases where as the maximum stack space consumption at a particular time is of order O(logn).

4 Conclusion

On the basis of the analysis above we could have a summarized result table below. In the table 4 below for each algorithm time complexity, space complexity and maximum stack space consumed are listed for best, average and worst case.

TABLE 4

COMPARISON OF SORTING ALGORITHMS ON THE BASIS OF VARIOUS FACTORS

Algorithms	Best Case		Average Case		Worst Case	
Quick	Time	O(nlogn)	Time	O(nlogn)	Time	$O(n^2)$.
	Space	O(logn)	Space	O(logn)	Space	O(n)
	Stack	O(logn)	Stack	O(logn)	Stack	O(n)
Merge	Time	O(nlogn)	Time	O(nlogn)	Time	O(nlogn)
	Space	O(n)	Space	O(n)	Space	O(n)
	Stack	O(logn)	Stack	O(logn)	Stack	O(logn)
Heap	Time	O(nlogn)	Time	O(nlogn)	Time	O(nlogn)
	Space	O(n)	Space	O(n)	Space	O(n)
	Stack	O(logn)	Stack	O(logn)	Stack	O(logn)

Quick Sort and Merge Sort operate on the records in a sequence and therefore they make efficient use of cache, whereas Heap Sort does not show cache efficiency. Whatever, for either sorted or reverse sorted sequence of input Merge or Heap is obviously the necessary choice, but when the real use of sorting operation comes in scenario i.e. when an unsorted sequence is to be sorted Quick Sort is the best option so far to choose over the other two algorithms.

References

[1] C. A. R. Hoare, (1961), "Algorithm 63, Partition" and "Algorithm 64, Quicksort", Communications of the ACM, Vol. 4, p. 321.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, (2009), Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001.

[3] J. W. J. Williams, (1964), "Algorithm 132 (Heapsort)". Communications of the ACM, 7:347-348.

[4] Donald Knuth, (1997), The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison-Wesley.

[5] Niklaus Wirth, (1976), (in English). Algorithms + Data Structures = Programs, Prentice-Hall.

[6] R. Sedgewick, (1978), "Implementing Quicksort Programs", Communications of the ACM 21, 10, 847–857.

[7] R. Sedgewick, (1997), Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching, 3rd Edition, Addison-Wesley.