COMPARATIVE STUDY AND PERFORMANCE ANALYSIS OF VARIOUS SORTING TECHNIQUES

Ramesh M. Patelia[1], Shilpan D. Vyas[2], Parina S. Vyas[3]

[1]Department of M.Sc. (IT) & HOD of BCA, AMCOST, Anand, Gujarat (India)
[2]Department of M.Sc. (IT) & BCA, AMCOST, Anand, Gujarat (India)
[3]Department of E&C, Shree P.M. Patel College of E&C, Anand, Gujarat (India)

ABSTRACT

Sorting is the process to arrange the data in either ascending or descending order. Sorting technique is a technique that does sorting i.e. arranges the data in either ascending or descending order. Various sorting techniques are Bubble sort, Insertion sort, Selection sort, Merge sort, Heap sort and Quick sort. This paper is an attempt to study, analysis and compare these different sorting techniques on the basis of various performance metrics such as best case, average case and worst case time complexity.

Keywords: Bubble, Permutation, Pivot, Sorting, Time complexity.

I. INTRODUCTION

A sorting algorithm is an algorithm that puts elements of a list in a certain order. The output must satisfy two conditions:

1. The output is in a particular order either ascending or descending.
2. The output is a permutation (reordering) of the input.

Two main factors on which the performance of a program depends are the amount of computer memory consumed and time required to execute it successfully. Time complexity of a program is the amount of time required to execute successfully.

II. BUBBLE SORT

2.1 Concept

The main idea behind the bubble sort is to compare the top element of an array with its successor, swapping the two if they are not in the proper order. The algorithm, which is a comparison sort, is named for the way smaller elements "bubble" to the top of the list. Although the algorithm is simple, it is too slow and impractical for most problems even when compared to insertion sort. This sorting technique is not efficient in comparison to other sorting techniques. But for small lists this sorting techniques works fine.

2.2 Analysis

The best case involves performing a one iteration which requires n-1 comparisons. Therefore, the best case is O(n). The worst case performance of bubble sort is n(n-1)/2 comparisons and n(n-1)/2 interchanges. Therefore, it is $O(n^2)$. The average case performance of bubble sort is $O(n^2)$.

III. INSERTION SORT

3.1 Concept

The main idea behind the insertion sort is to inserts in the i^{th} pass the i^{th} element in its correct place. The performance of insertion sort depends on the initial ordering of the elements. There are two for loops in the algorithm of insertion sort. The first for loop is always executed for n times, but the execution of second for loop depends on the ordering of elements. We will explore three case best, worse and average case for insertion sort.

3.2 Analysis

The best case is when the elements are already in sorted order. In this case, the second for loop will be executed in the following pattern for n elements

$F(n) = 1 + 1 + \ldots\ldots\ldots + 1$ upto n times

$= n$

$= O(n)$

The worst case will be when the elements are in reverse order. The second for loop will be executed in the following pattern.

$F(n) = 1 + 2 + 3 + 4 + \ldots\ldots\ldots + n$

$= n(n-1)/2$

$= O(n^2)$

The average case will be as shown below:

$F(n) = 1/2 + 2/2 + 3/2 \ldots\ldots\ldots + n/2$

$= n(n-1)/4$

$= O(n^2)$

IV. SELECTION SORT

4.1 Concept

The main idea behind the selection sort is to successive elements are selected from a file or array and placed in their proper position. It is the simplest sorting technique.

4.2 Analysis

The time required to sort the data does not depend upon the order of the elements. There is no difference in the number of iteration for a list which is sorted already or which has its elements in reverse order. The only difference will be with the number of comparison and with the assignment statements. Their execution frequency will be varying. Total number of comparison is ½ n(n-1). The best case, average case and worst case are same and they are

$F(n) = (n-1) + (n-2) + (n-3) + (n-4) \ldots\ldots\ldots + 1$

$= n(n-1)/2$

$= O(n^2)$

It is more efficient than the bubble sort or insertion sort, because there are no more than N -1 actual interchanges. Although there is no significance gain compare to bubble or insertion sort in run time: its efficiency is also $O(n^2)$ for n data items.

V. MERGE SORT

5.1 Concept

The main idea behind the Merge sort is to divide a file into two sub-files. These files are compared, one pair of record at a time, and merged by writing them to other files for the further comparisons. Conceptually, a merge sort works as follows:

1. Divide the unsorted list into *n* sublists, each containing 1 element (a list of 1 element is considered sorted).
2. Repeatedly merge sublists to produce new sorted sublists until there is only 1 sublist remaining. This will be the sorted list.

5.2 Analysis

In sorting *n* objects, merge sort has an average and worst-case performance of O(*n* log *n*). Merge sort is more efficient than quick sort for some types of lists if the data to be sorted can only be efficiently accessed sequentially, and is thus popular in languages such as Lisp, where sequentially accessed data structures are very common.

VI. HEAP SORT

6.1 Concept

The main idea behind the Heap sort is that a two-phase sort procedure that uses a full binary tree structure. In computer programming, heap sort is a comparison-based sorting algorithm. Heap sort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the smallest element and moving that to the sorted region. The improvement consists of the use of a heap data structure rather than a linear-time search to find the minimum.

6.2 Analysis

Consider the timing analysis of the heap sort. Since we are using a complete binary tree, the worse case analysis is easier than average case. The worst case at each step involves performing a number of comparisons which is given by the depth of the tree. This observation implies that the number of comparison is O(*n* log *n*).The average case is more complex but it can be also O(*n* log *n*).

VII. QUICK SORT

7.1 Concept

A pivot (middle) item near the middle of the array is chosen, then moves are made such that item on one side of the pivot are smaller than the pivot and those on the other side are larger. This procedure is applied recursively until the whole array is sorted.

7.2 Analysis

To analyze the quick sort algorithm, note that for a list of n, if the partition always occurs in the middle of the list, there will again be logn divisions. In order to find the split point, each of the n items needs to be checked

against the pivot value. The result is nlogn. In the worst case, the split points may not be in the middle and can be much skewed to the left or the right, leaving a very uneven division. In this case, sorting a list of n items divides into sorting a list of 0 items and a list of np-1 items. Then sorting a list of n-1, divides into a list of size 0 and list of size n-2 and so on. The result is an $O(n^2)$ sort with all of the overhead that recursion requires. The Best case = O(n log n), Average Case = O(n log n) and Worst Case = $O(n^2)$.

VIII. COMPARISONS OF SORTING TECHNIQUES

The following table gives the comparative study of different sorting techniques such as Bubble sort, Insertion sort, Selection sort, Merge sort, Heap sort and Quick sort.

Sorting Techniques	Time Complexity			Remarks
	Best case	Average case	Worst case	
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	Good for small n (n<=100)
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$	Good for almost sorted data
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	Good for partially sorted data and small n.
Merge Sort	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	Good for External file sorting
Heap Sort	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	Excellent
Quick Sort	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	Excellent

IX. CONCLUSION

In this paper, we provide descriptions of several sorting techniques for small and large data. We also provide classification of these techniques according to their time analysis. From the above analysis, it can be said that, Bubble sort, Selection sort and Insertion sort are fairly straight forward and time complexity of Average case & Worst case both are same. Merge sort, Heap sort and Quick sort are good sorting techniques in terms of time complexity.

REFERENCES

Books

[1] Tremblay J. & Sorenson P. G. : An Introduction to Data Structures with Applications, 2nd Edition, McGraw-Hill International Edition, 1987.

[2] Singh Bhagat & Naps Thomas : Introduction to Data Structures, Tata McGraw-Hill Publishing Co. Ltd.,1985.

[3] R. B. Patel : Expert Data Structures with C, Third Edition, Khanna Book Publishing Co. (P) Ltd., Delhi.

Journal Papers

[4] Comparative Analysis &Performance of Different Sorting Algorithm in Data Structure. IJARCSSE, Volume 3, Issue 11, November 2013, pg 500-507

Websites

[5] http://en.wikipedia.org/wiki/Sorting_algorithm