Analysis of Comparison Based Sorting Algorithms

Atitesh Gaurav
B.E Student
Department of Information Technology
Lakshmi Narain College of Technology Bhopal, India

Abstract— An algorithm is a set of steps of operations to solve a problem performing calculation, data processing, and automated reasoning tasks. Sorting is an important issue in Data Structure which creates the sequence of the list of items. Although numbers of sorting algorithms are available, it is all the more necessary to select the most efficient sorting algorithm. Sorting algorithms are usually judged by their efficiency. The efficiency of a sorting algorithm is related to the number of items being processed. Sorting is the procedure of ordering list of elements in ascending or descending with the help of key value in specific order. We will see how Comparison Based sorting algorithms perform operation and then distinguish them based on various constraints to come with outcome. Here efficiency refers to the algorithmic efficiency as the size of the input grows large and is generally based on the number of elements to sort. In this paper, we have implemented different types of Comparison Based sorting algorithms like Insertion sort, Selection sort and Merge sort with their analysis in term of worst, average, and best case.

Keywords: Algorithm, Complexity, Bubble Sort, Selection Sort, Quick Sort and Insertion Sort, Complexity, Merge Sort, Best Case, Average Case and Worst Case, Comparison Based Sorting Algorithms

I. INTRODUCTION

Here we do the Analysis of Comparison Based sorting algorithm .When we discuss Time complexity for algorithm it is how much time required for executing the sorting of array. In these we had find different cases. The running time of Best case an algorithm gives us a lower bound for any input. The running time of Worst case an algorithm gives us upper bound for any input.

There are different types of sorting Algorithms (ie Comparison Based sorting, without Comparison Based sorting) , we here focus only on Comparison Based sorting algorithm. Big-O, theta and omega notation measures in worst case average case and best case. The running time complexity of these sorting algorithm is in average case of Selection sort is (n^2), insertion sort is (n^2), merge sort is (nlogn) and quick sort is (nlogn)

II. INSERTION SORT

The Insertion sort is a most efficient comparison based algorithm for sorting a small number of elements. Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list. In this sorting algorithm at each iteration, insertion sort removes one element from the input data then first it will compare with first value of existing array if that value is smaller than existing one then it will came on first position. Otherwise one by one it will compare and take right position. It repeats until no input elements remain. Sorting is typically done in-place, by iterating up the array, growing the sorted list behind it, this algorithms consider as an example of an incremental approach. Its sorts the sequence of array one element at a time. If the given numbers are sorted, this algorithm runs in O(n) time. If the given numbers are in reverse order, the algorithm runs in O(n^2) time.

Here time complexity for best case in Θ(n) and for Worst case time complexity is Θ(n^2) and average case time complexity is Θ(n^2).

Algorithm for INSERTION SORT:-

1) For i <- 1 to Length(A)
a) x <- A[i]
b) j <- i
c) While j > 0 and A[j-1] > x
   1) A[j] <- A[j-1]
   2) j <- j - 1
d) A[j] <- x

III. SELECTION SORT

The method of selection sort is we continuously find the next largest (or smallest) element in the array and move it to its first position to last position in the sorted array. The selection sort algorithm is also used for small amount of data for sorting elements. This type of sorting is called Selection Sort as it works by repeatedly sorting elements.

In this sorting algorithm, first find the smallest in the array and exchange it with the element in the first position, then find the second smallest element and exchange it with the element in the second position, and continue in this way until the entire array is sorted. Selection sort spends most of its time trying to find the minimum element in the unsorted part of the array. In this one value was compared with every remaining value of array so more time required for execution Selection sort is quadratic in both the best case, worst and the average case, and requires no extra memory. Here, for worst case, best case and for average case time complexity remains same.

Algorithm for Selection Sort is:

SELECTION_SORT (Arr([0….N-1])
//An array Arr([0….N-1]) of ordered element
1) For I=0 To N-2
2) Do Min=I
3) For J=I+1 To N-1 Do
4) If Arr [J] < Arr [Min]
5) Min =J
6) Swap Arr [I] And Arr [Min]

A. Analysis:

For each i from 1 to n - 1, there is one exchange and n - i comparisons, so there is a total of n - 1 exchanges and n − 1 + n − 2 + ...+ 2 + 1 = n (n – 1)/2 comparisons

IV. BUBBLE SORT

Bubble Sort is an elementary sorting algorithm that works by repeatedly exchanging the adjacent elements, if necessary and when no exchanges are required, the file is sorted. The only significant advantage that bubble sort has over most other implementations, even quicksort, but not insertion sort, is that having ability to detect that the list is sorted efficiently is built into the algorithm. When the list is already sorted (best-case), the complexity of bubble sort remains only O (n).Bubble sort has worst-case and average complexity both O (n ^ 2) , where n is the number of items being sorted. It is not a practical sorting algorithm when n is large.

Algorithm: Sequential-Bubble-Sort (A)

1) For i← 1 to length [A] do
2) For j ← length [A] down-to i +1 do
3) if A[A] < A[j - 1]
4) then Exchange A[j] ↔ A[j-1]

A. Analysis

Here, the number of comparisons are 1 + 2 + 3 +...+ (n - 1) = n (n - 1)/2 = O (n^2) Clearly, the equation shows the n^2 nature of the bubble sort.

V. MERGE SORT

Merge sort is based on the divide-and-conquer approach. Merge sort algorithm is generally used for large size of data .It is one of the most popular sorting algorithms and a great way to develop credence in building recursive algorithms. it split the list of records in two smallest units after that it compare each of the element with adjacent list and sort the two pieces or units of data sets recursively, consequently it merges and sorted the all the elements in the list. Here we are dealing with sub problems, we state each sub problem as sorting a sub array

Here In this basically we do following steps:

1) Divide: we divided the n elements into n/2 sub parts.
2) Conquer: Sort the two parts recursively using merge sort algorithm.
3) Combine: Merge the two sorted parts using merge algorithm.

Algorithm for Merge Sort :

MERGE (A, p, q, r)

1) n1 ← q − p + 1
2) n2 ← r − q
3) Create arrays L[1 . . n1 + 1] and R[1 . n2 + 1]
4) FOR i ← 1 TO n1
5) DO L[i] ← A[p + i − 1]
6) FOR j ← 1 TO n2
7) DO R[j] ← A[q + j]
8) L[n1 + 1] ← ∞
9) R[n2 + 1] ← ∞
10) i ← 1
11) j ← 1
12) FOR k ← p TO r
13) DO IF L[i] ≤ R[j]
14) THEN A[k] ← L[i]
15) i ← i + 1
16) ELSE A[k] ← R[j]
17) j ← j + 1

MERGESORT (A, p, r)

1) IF p < r
2) THEN q ← [(p + r)/2]
3) MERGE (A, p, q)
4) MERGE (A, q + 1, r)
5) MERGE (A, p, q, r)

A. Analysis

Time Complexity Analysis of Merge Sort
Best Case Time Complexity – O (n log n)
Worst Case Time Complexity – O (n log n)
Average Case Time Complexity – O (n log n)

The Recurrence Relation for best, worst and average case is T (n) = 2 T (n/2) + n, So on solving we get T (n) = O (n log n)

VI. QUICK SORT

Quick Sort is a divide and conquer algorithm. Quick sort is not a stable search, but it is very fast and requires very less additional space, it's also called partition exchange sort. It relies on a partition operation, to perform partition on array, an element called a pivot is selected. All elements smaller than the pivots are moved before it and all greater elements are moved after it. (ie. Pivot element is an middle, the left side element is less value and right side element is greater value) This can be done efficiently in linear time and in-place. The lesser and greater sub-lists are then recursively sorted. The most complex issue in quick sort is in choosing a good pivot element, consistently poor choices of pivots can result in drastically slower O (n^2) performance, if at each step the median is chosen as the pivot then the algorithm works in O (n log n).Quick sort could be extremely economical algorithmic rule and is predicated on dividing of an array of knowledge with in smaller arrays. It is in-place since it uses only a small auxiliary stack. It requires only n logn time to sort n items. It has an extremely short inner loop. This algorithm has been subjected to a thorough mathematical analysis, a very precise statement can be made about performance issues. Quick sort works by partitioning a given array A[p ... r] into two non-empty sub array A[p ... q] and A[q+1 ... r] such that every key in A[p ... q] is less than or equal to every key in A[q+1 ... r].

Algorithm for Quick Sort

1) If(Low>High):
2) Return
3) Set Pivot=A[Low] 3.Set I=Low+1
4) Set J=High
5) Repeat Step 6 Until I>High Or A[I] > Pivot //Search For An Element Greater Pivot Then
6) Increment I By 1
7) Repeat Step 8 Until J<Low Or A[J] < Pivot //Search For An Element Smaller Than Pivot
8) Decrement J By 1
9) If I<J: //If Greater Element Is On The Left Of Smaller Element
10) Swap A[I] With A[J] 10.If I<=J:
11) Go To Step 5//Continue Search 11.If Low<J:
12) Swap A[Low] With A[J]//Swap Pivot With Last Element In First Part Of The List

13) Quicksort (Low,J-1)//Apply Quicksort On 1st Left To Pivot (or) 13.Quicksort(J+1,High)//Apply Quicksort On 1st Right To Pivot

A. Analysis of Quick Sort

1) Recurrence Relation for Worst Case Complexity

$T(n) = T(n-1) + n$

On solving this we get, $T(n) = O(n^2)$

2) Recurrence Relation for Average Case Complexity

$T(n) = a\,T(n/b) + f(n)$

On solving this we get $T(n) = O(n \log n)$

3) Recurrence Relation for Best Case Complexity

$T(n) = 2\,T(n/2) + n$

On solving this we get $T(n) = O(n \log n)$

Best Case Time Complexity – $O(n \log n)$

Worst Case Time Complexity – $O(n^2)$

Average Case Time Complexity – $O(n \log n)$.

VII. CONCLUSION

In this research paper we have studied about different Comparison Based sorting algorithms along with their calculating the performance analysis of time complexity. Every sorting algorithm has advantage and disadvantage. Various Sorting algorithms have been compared on the basis of different factors like complexity, number of passes, and number of comparison. After performing all the algorithms based on time complexity we have conclude that merge sort And Quick Sort is best in all kind of data because in both array is divided into two parts so this is faster than rest of sorting. Quick Sort is considered as most Practical Sorting Algorithm. This paper can be useful for analysis of Comparison Based sorting algorithms. Here, various algorithms is implemented in divide and conquer methodology. I tried to illustrate the different Comparison Based sorting algorithms .There is huge Important of the Comparison Based sorting algorithm in the Practical Applications in Software Industry.

REFERENCES

[1] T. Cormen, C. Leiserson, and R. Rivest, Introduction to Algorithms, McGraw-Hill, New York, NY 1992
[2] Aho A., Hopcroft J., and Ullman J., the Design and Analysis of Computer Algorithms, Addison Wesley, 1974.
[3] Fundamentals of Computer Algorithms by Ellis Horowitz
[4] D. Rajagopal, and K. Thilakavalli. "Comparison of different sorting algorithms in Data Structure based upon the Time Complexity" International Journal of Technology, Vol.6, no. 1, pp. 23- 30, 2016.
[5] D. Knuth, "The Art of Computer programming Sorting and Searching", 2nd edition, Addison Wesley, vol. 3, (1998).

