A Comparative Study of Different Types of comparison Based Sorting Algorithms in Data Structure

Article · March 2014

CITATIONS
5

READS
1,749

3 authors:

Kamlesh Kumar Pandey
Dr. HariSingh Gour Vishwavidyalaya (A Central University), Sagar, India
25 PUBLICATIONS **42** CITATIONS
SEE PROFILE

Rajesh Bunkar
APS University
8 PUBLICATIONS **15** CITATIONS
SEE PROFILE

Kamlesh kumar Raghuvanshi
Ramanujan college, University of Delhi,Delhi
4 PUBLICATIONS **7** CITATIONS
SEE PROFILE

Volume 4, Issue 2, February 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Comparative Study of Different Types of comparison Based Sorting Algorithms in Data Structure

Kamlesh Kumar Pandey[1], Rajesh Kumar Bunkar[2]
Computer Science, IGNTU Amarkantak.
India

Kamlesh Kumar Raghuvanshi[3]
Asst.Professor,PGDAV College Delhi
India

Abstract— there are many popular problems in different practical fields of computer science, record applications, Networks and Artificial intelligence. These basic operation and problems is sorting algorithm; the sorting problem has attracted a great deal of research. This research paper presents the different types of comparison Based sorting algorithms of data structure like insertion, selection, bubble, quick and merges. Each algorithm is solving to specific sorting problem in a different formats. This research provides a detailed study of how all the five algorithms work and their algorithm, advantage, disadvantage and then compares them on the basis of various parameters like time complexity and space complexity.

Keywords— Comparisons, Insertion Sort, Selection Sort, Bubble Sort, Quick Sort, Merge Sort, Time Complexity.

I. INTRODUCTION

A sorting algorithm is an algorithm that puts elements of a list in a certain order, such as increasing and decreasing. The most-used orders are numerical order and lexicographical order. A number of sorting algorithms have been developed like include insertion sort , selection sort , bubble sort, quick sort and merge sort are comparison based sort .There is another class of sorting algorithms which are non comparison based sort. This paper gives the brief introduction about comparison based sorting algorithms and compare to each sorting in different most important parameter like time complexity, space complexity, Stability etc. When we sort to any type of List, array etc. then time we compare one element to another element on the list after that we swap this element. This type of sorting is called comparison based sorting. Algorithm and property are every sorting algorithm is different because every algorithm is sort the data in different time and different memory allocation technique.

The sorting methods can be divided into two categories:
1. **Internal Sorting: -** if data are sorted at a time in main memory this type of sorting is called internal sorting.
2. **External Sorting**: - if data are sorted in auxiliary memory (hard disk, floppy, tape etc) this type of sorting is called External Sorting

II. WORKING PROCEDURE OF COMPARISON BASED SORTING ALGORITHMS

1. Insertion Sort

Insertion sort is a naive algorithm that belongs to the family of comparison sorting. Insertion sort is an example of an incremental algorithm; it builds the sorted sequence one number at a time. In this sorting we can read the given elements from 1 to n, inserting each element into its proper position through comparison. Here n-1 pass (step) are require for sorting time. For example, the card player arranging the cards dealt to him. The player picks up the card and inserts them into the appropriate position. At each step, we place in the item into its proper place.

Algorithm:- Here K,J is a variable which value is a element position and A is Array[1-N].

INSERTION_SORT (A)

```
1. For K=2 to length[A]  (for pass)
      2. item= A [K], J=K-1  (for minimum number K-1  comparison)
      3. WHILE J>0 and item<A[J]
          4. A[J+1]=A[J]
          5. J=J-1
          END WHILE LOOP
6. A [J+1]=item
.END FOR LOOP
```

Example:

	A[1]	A[2]	A[3]	A[4]	A[5]
DATA	55	33	44	11	22
PASS 1	55	33	44	11	22
PASS 2	33	55	44	11	22
PASS 3	33	44	55	11	22
PASS 4	11	33	44	55	22
SORTED DATA	11	22	33	44	55

2 .Selection Sort

Selection sort belongs to the family of in-place comparison sorting. This algorithm is called selection sort because it works by selecting a minimum element in each pass (step) of the sort. In this method, to sort the data in increasing order, the first element is compared with all the elements. If first element is greater than smallest element than interchanged the position of elements. So after the first pass, the smallest element is placed at the first position. The same procedure is repeated for 2nd element and so on until the element of list is sorted.

Algorithm: - Here I, K,LOC is a variable which value is a element position, A is Array [1-N] and min is minimum value of array A.

SELECTION_SORT (A)

1. for I=1 to length[*A*]-1 (finding minimum value for pass)
   2. min=A [I]
   3. for K=I+1 to length[*A*] (for comparison)
      4. if (min>A [I])
      5. min=A [K], Loc=K
      [End if]

      [End of inner loop]

   6. Swap (A [Loc],A[I])
   [End of OUTER loop]
7. Exit

Example:

	A[1]	A[2]	A[3]	A[4]	A[5]
DATA	55	33	44	11	22
	55	33	44	11	22
PASS 1	33	55	44	11	22
	33	55	44	11	22
	11	55	44	33	22
	11	55	44	33	22
	11	55	44	33	22
PASS 2	11	44	55	33	22
	11	33	55	44	22
	11	22	55	44	33
	11	22	55	44	33
PASS 3	11	22	44	55	33
	11	22	33	55	44
	11	22	33	55	44
PASS 4	11	22	33	44	55
SORTED DATA	11	22	33	44	55

3. Bubble Sort

Bubble sort belongs to the family of comparison sorting. The Bubble Sort is the simplest sorting technique and multiple swapping process to apply to every pass, in which smallest data element are moved ('bubbled up') to the top of the list. In this sorting method, we compare the adjacent members of the list to be sorted, if the top of item is greater than the item immediately below it, they are swapped. Unfortunately, it is a slowest sorting method as a compare selection and insertion sort.

Algorithm:- Here I,K is a variable which value is a element position and A is Array[1-N].

BUBBLE_SORT (A)

1.For I=1 to length[*A*]-1 (for pass)
   2. For k=1 to length[*A*]-I(for comparison)
   3. If A[K]>A[K+1]
      4. Swap [A(K) , A(K+1)]

[End if]
[End of inner loop]
[End of outer loop]
5.Exit

Example:

	A[1]	A[2]	A[3]	A[4]	A[5]
DATA	55	33	44	11	22
PASS 1	55	33	44	11	22
	33	55	44	11	22
	33	44	55	11	22
	33	44	11	55	22
	33	44	11	22	55
PASS 2	33	44	11	22	55
	33	44	11	22	55
	33	11	44	22	55
	33	11	22	44	55
PASS 3	33	11	22	44	55
	11	33	22	44	55
	11	22	33	44	55
PASS 4	11	22	33	44	55
	11	22	33	44	55
SORTED DATA	11	22	33	44	55

4. Quick Sort:

Quick sort was developed by Sir Charles Antony Richard Hoare (Hoare 1962). It belongs to the family of exchange sorting. Quick sort is an in-place, divide-and-conquer, massively recursive sort and it is also known as a partition-exchange sort.

Divide: firstly divided the list by choosing a partitioning element (pivot element) . one list contains all element lass than or equal to the partitioning element and the other list contains all element greater than the partitioning element .

Conquer: after these two list are recursively partitioned in the same way till the resulting lists become trivially small to sort by comparison.

Combine: at last we then go on combining the sorted smaller list to produce the sorted list of the entire input element.

Algorithm: Here A is Array of [1-N] element

QUICK_SORT (A,N)
1.QUICK(A,1, N)
2.Exit

Given a sub array A[p …. r] such that p <= r -1,this subroutine rearranges the input sub array into two sub arrays, A[p .. q-1] and A[q +1.. r], so that each element in A[p .. q-1] is less than or equal to A[q] and each element in A[q +1..r] is greater than or equal to A[q] .Then the subroutine outputs the value of q.

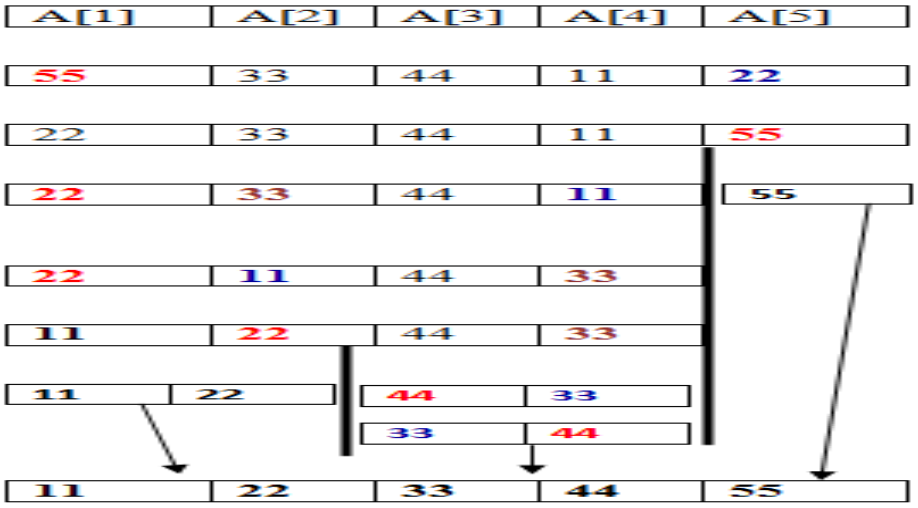
QUICK(A, p, r)
1. if p >=r then return
2. q = PARTITION(A, p, r)
3. QUICK(A, p, q - 1) Recursive call to Quick
4. QUICK(A, q +1, r)
5.Exit

Use the initial value of A[r] as the pivot in the sense that the keys are compared against it. Scan the keys A[p..r -1] from left to right and Flush to the left all the keys that are greater than or equal to the pivot.

PARTITION(A, p, r)
1. x = A[r]
2. i = p -1
3. for j = p to r - 1 do
4. if A[j] <=x then
5. i =i+1
6. Exchange A[i] and A[j]
END IF
7. Exchange A[i+1] and A[r]

8. return i+1
END FOR LOOP
9. Exit

Example:



5. Merge Sort

Merge sort was invented by John von Neumann and belongs to the family of comparison-based sorting. This algorithm is also based on Divide-and-Conquer approach.

Divide: Given a sequence of N elements A[1],….,A[N], the general idea is to breaks into two data sets A[1],….A[N/2] and A[n/2+1],….,A[N].

Conquer: Each set is individually sorted by recursively

Combine: finally resulting sorted sequence are merged to produce a single sorted sequence of N elements

Conceptually, a merge sort works as follows

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

Algorithm :- To sort the entire sequence A[1 .. N], make the first call to the procedure MERGE-SORT (A, 1, N).

MERGE_SORT (A, p, r)

1. IF p < r then // Check for base case
   2. q = FLOOR[(p + r)/2] // Divide step
   3. MERGE (A, p, q) // Conquer step.
   4. MERGE (A, q + 1, r) // Conquer step.
   5. MERGE (A, p, q, r) // Conquer step.
   End if
6. Exit

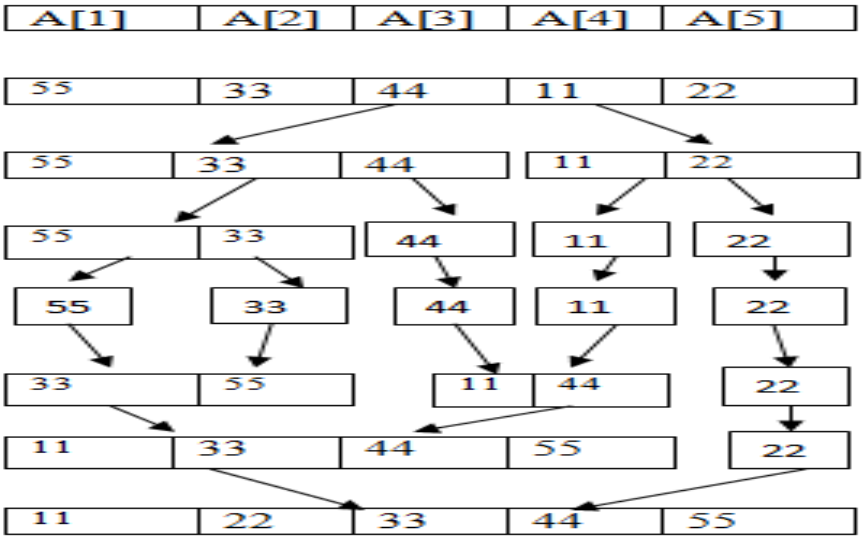
MERGE (A, p, q, r)

1. $n_1 \leftarrow q - p + 1$
2. $n_2 \leftarrow r - q$
3. Create arrays $L[1 \ldots n_1 + 1]$ and $R[1 \ldots n_2 + 1]$
4. FOR $i \leftarrow 1$ TO n_1
   5. DO $L[i] \leftarrow A[p + i - 1]$
   Exit for
6. FOR $j \leftarrow 1$ TO n_2
   7. DO $R[j] \leftarrow A[q + j]$
   Exit for
8. $L[n_1 + 1] \leftarrow \infty$
9. $R[n_2 + 1] \leftarrow \infty$
10. $i \leftarrow 1$
11. $j \leftarrow 1$
12. FOR $k \leftarrow p$ TO r
    13. IF $L[i] \leq R[j]$ then
        14. $A[k] \leftarrow L[i]$

15. i ← i + 1
16. ELSE A[k] ← R[j]
17. j ← j + 1

Exit if
Exit for
18. Exit

Example:



III. COMPARITIVE STUDY OF ALL ALGORITHMS

TABLE I COMPARISON OF COMPARISON BASED SORTING TECHNIQUES ON VARIOUS PARAMETERS

Parameter	Insertion Sort	Selection Sort	Bubble Sort	Quick Sort	Merge Sort
Total no of comparison	$(N-1)N/2=0(n^2)$	$N(N-1)/2=0(n^2)$	$N^2-2N+1=0(n^2)$	$N(N+1)/2=0(n^2)$	Each pass N comparison so N * log N
Time Complexity					
1.Best Case	O(N)	$O(N^2)$	O(N)	O(N log N)	O(N log N)
2.Average Case	$O(N^2)$	$O(N^2)$	$O(N^2)$	O(N log N)	O(N log N)
3.Worst Case	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	O(N log N)
Space Complexity	O(1)	O(1)	O(1)	O(1)	O(N)
Sorting concept (Method)	Insertion	Selection	Exchange	Partitioning	Merging
Algorithm	Incremental algorithm	Straightforward algorithm	Straightforward And Interchanging Algorithm	Divide and Conquer Algorithm	Divide and Conquer Algorithm
STABLE	Yes	No	Yes	typical in place sort is not stable; stable versions exist	Yes
Type	Internal	Internal	Internal	Internal	Internal and External
Implementation on programming	Easy	Easy	Easy	Hard	Hard
In-Place	YES	YES	YES	YES	NO
Strategy	Scan all the elements & insert the smallest element before largest	Scan all the elements & sort the list	Scan all the elements & bubble up largest element	CONCEPT OF PIVOT ELEMENT	Divides An Array Into Two Separate Lists

TABLE 2 ADVANTAGE/DISADVANTAGE OF COMPARSION BASED SORTING TECHNIQUES

Sorting Techniques	Advantage	disadvantage
Insertion Sort	1. It is an in-place sorting algorithm so the space requirement is minimal. 2. given to good performance in a small list	1. Useful only few elements a list
Selection Sort	1. No additional temporary storage is required.	1. Inefficient for large lists. 2.Unstable but can be implemented as a stable sort
Bubble Sort	1.Straightforword,simple and easy to implement 2. No additional temporary storage is required	1.Very inefficient for large list of element
Quick Sort	1.Fast and efficient	1.Unstable sort and complex for choosing a good element
Merge Sort	1. Well suited for a large data set. 2. A fast recursive sorting.	1. At least twice the memory requirements then other sorts.

IV. CONCLUSIONS

This paper discusses five comparison based sorting algorithms and their example. Merge Sort and Quick Sort are faster for large lists as a compare Bubble Sort, Selection Sort and Insertion Sort. In this region quick sort is best algorithm. We have compared the various sorting algorithm on the basis of various factors like complexity, memory required, working concept, advantage, disadvantage etc.

REFERENCES

[1] Eshan Kapur, Parveen Kumar and Sahil Gupta,"Proposal Of A Two Way Sorting Algorithm And Performance Comparison With Existing Algorithms*" International Journal of Computer Science, Engineering and Applications* (IJCSEA) Vol.2, No.3, June 2012.

[2] C.A.R. Hoare, *Quicksort*, Computer Journal, Vol. 5, 1, 10-15 (1962)

[3] Knuth, D.E., 1988. *The Art of programming-Sorting and Searching*. 2nd Edn.,Addison Wesley, ISBN: 020103803X.

[4] Cormen, T.H. et al. *Introduction to Algorithms*. 2nd Edn., 2001. ISBN: 0262032937

[5] H. M. Mahmoud, R. Modarres, and R. T. Smythe. *Analysis of quickselect: An algo-rithm for order statistics*. ITA – Theoretical Informatics and Applications, 29(4):255–276, 1995.

[6] Ahmed M. Aliyu, Dr. P. B. Zirra "A Comparative Analysis of Sorting Algorithms on Integer and Character Arrays" The International Journal Of Engineering And Science (IJES) Volume 2,Issue 7,Pages 25-30,2013 ISSN(e): 2319 – 1813 ISSN(p): 2319 – 1805

[7] Madhavi Desai, Viral Kapadiya "Performance Study of Efficient Quick Sort and Other Sorting Algorithms for Repeated Data" National Conference on Recent Trends in Engineering & Technology 13-14 May 2011

[8] Ashutosh Bharadwaj, Shailendra Mishra, "Comparison of Sorting Algorithms based on Input Sequences" *International Journal of Computer Applications (0975 – 8887) Volume 78 – No.14, September 2013*

[9] Comparison of Sorting Algorithms (On the Basis of Average Case) Pankaj Sareen.

[10] A Comparison Based Analysis of Four Different Types of Sorting Algorithms in Data Structures with Their Performances.

[11] Data Structures by Seymour Lipschutz and G A Vijayalakshmi Pai (Tata McGraw Hill companies), Indian adapted edition-2006,7 west patel nagar,New Delhi-110063

[12] Introduction to Algorithms by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, fifth Indian printing (Prentice Hall of India private limited), New Delhi-110001

[13] Data structures using c and c++ by Yedidyah langsam,Aaron M. Tenenbaum second Indian printing (Prentice Hall of India private limited), New Delhi-110001

[14] An Introduction to Data Structures with Application by Jean-paul Tremblay Tata McGraaw Hill

