ISSN No. 0976-5697

Volume 8, No. 1, Jan-Feb 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Comparative Study of Well Known Sorting Algorithms

Kazim Ali
MS Computer Science
Lahore Leads University, Pakistan

Abstract: .In computer science sorting is much basic and important problem. Sorting is also a fundamental problem in algorithm analysis and designing point of view. Therefore many computer scientists have much worked on sorting algorithms. Therefore as a computer science student, I also want to work on existing sorting algorithms and presenting my findings after studying some well-known sorting algorithms which are Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Heap Sort, Quick Sort, Counting Sort, Bucket Sort and Radix Sort. In this paper, we will study these algorithms and their time complexities. Time complexity is much important factor of success and popularity of any sorting algorithm. Time complexity means the running time of algorithms on a virtual machine or a real machine. The stability and in-placing of above algorithms will also discuss .Therefor time complexity comparisons are done in this paper and also determine the conditions where a specific algorithm is best. I will also present some related work of different people. My own comments and opinions on the previous related work will also be the part of this paper.

***Keywords*:** Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Heap Sort, Quick Sort, Counting Sort, Bucket Sort, Radix Sort, Time Complexity, Comparison .

1. INTRODUCTION

There are several algorithms which are used to solve the sorting problems. In sorting we have a number of elements like a_1, a_2, a_3,.....a_n as input and we have to determine the output as á'$_1 \leq$ á'$_2 \leq$ á'$_3$.....$\leq$ á'$_n$ and vice versa [1]. The data elements which are to be sorted, normally stored in an array data structure but it is not necessary we can use any other linear or non-linear data structure for storing purpose according to nature of data elements. Most software applications need to sort data in some order e.g. ascending or descending. When data will be sorted then there must be used some sorting algorithm. Therefore research in sorting algorithm is very important in computer science. In earlier days, People thought a commercial computer spent a significant amount to sort data [2]. There are many basic and advance level algorithms are available which are best for some specific sorting problem but not for all sorting problems. I think they depend upon amount of data, nature of data, format of input or perhaps machine specific. Anyway number of reasons can participate to select a sorting algorithm to solve a specific sorting problem.

In this research paper, I am tried to present a comparison based analysis of some simple and advanced level algorithms and explain these algorithms in a simple manner. Also present some previous comparative study of sorting algorithms of some other people which have done nicely. But in the field of knowledge we can criticize in a polite and civilized manner the work of other people which are great researcher, hard work and much respectable people in their fields. It is also a bare fact that the critical research is the base of knowledge. Unfortunately, this mistake will be done by me in this research paper in the study of related work and discussion section.

2. RELATED WORK AND DISCUSSION

[3] They have compared four already known algorithms, insertion sort, bubble sort, selection sort, merge sort and computed their running time on a real machine. They have input 10, 100, 1000 and 10000 elements to theses sorting algorithms and measure their time on these inputs. This is a nice work but problem is that it is a machine specific analysis. It must be machine independent. [4] Presents analysis of two sorting algorithms selection sort and Quick sort on integer and character arrays by running on CPU and measure time taken to sort both arras. Again good and nice effort but it is also machine specific analysis. [5] Presents a new approach to increase the efficiency of an Old Bubble Sort algorithm, good effort but not so useful in the presence of some efficient algorithms which have less time complexity. [6] Describe a comparative study of Selection Sort, Quick Sort, Insertion Sort, Merge Sort, Bubble sort and Grouping Comparison Sort. They also run on a real computer against different size of inputs and measure their running time on a specific computer. [7] Proposed an Enhancement of bubble sort algorithm whose performance is better than old Bubble sort algorithm on a real computer. But this performance is also on a specific computer. We do not know what performance on other machines or computing devices. [8] Compares three algorithms bubble sort, binary sort and merge sort nicely and compare their best case, average case and worst case time complexity and also discuss their stability. It is a machine independent analysis, which is a good approach. [9] Conduct a comparative study of five sorting algorithms bubble sort, selection sort, insertion sort, merge sort, quick sort and compare their average case, best case, worst case time and space complexity with help of C# language program. Also discussed stability and in placing of these algorithms. It is a good comparative study but machine and language dependent. [10] Has compared four algorithms quick sort, heap sort, and insertion sort, merge sort and presented their running time using turbo C++ compiler.

3. SORTING ALGORITHMS

In this section I will present a detail study of well-known sorting algorithms which is given as under.

3.1. Merge Sort:

Merge sort is based on divide and conquer strategy technique which is a popular problem solving technique. We will give a set of data items of length n as an input to the algorithm or procedure, stored in an array A or any other data structure which is best suited according to conditions. The merge sort algorithm is work as under

- Split array A from middle into two parts of length n/2.
- Sorts each part calling Merge Sort algorithm recursively.
- Merge the two sorting parts into a single sorted list.

The pseudo code of Merge Sort Algorithm is given under.

```
MERGE-SORT (array A, int p, int r)
{

 If (p < r)
 Then
 q   (p + r)/2
 MERGE-SORT (A, p, q)
 MERGE-SORT (A, q + 1, r)
 MERGE (A, p, q,  r)
}
```

The MERGE algorithm is given under

```
MERGE (array A, int p, int q,  int r)
{
 int B[p….r]
int i ←  k   ← p
int j ←  q + 1
 while (i ≤ q) and (j ≤ r)
 do if (A[i] ≤ A[j])
 then
 B[k++] ←  A[i++]
 else
B[k++] ←  A[j++]
 while (i ≤ q)
 do B[k++] ←  A[i++]
 while (j ≤ r)
 do B[k++] ←  A[j++]
 for i ←  p to r
 do A[i] ←  B[i]
}
```

The Merge Sort Algorithm takes Θ (nlogn) time. [1] Describes the whole method to calculate the running time using mathematically tools. Therefore I will not determine the time complexity of sorting algorithms. It is stable algorithm but not in-place.

3.2 Quick Sort:

Quick Sort is based on divide and conquers strategy. It is one of the fastest sorting algorithms which is the part of many sorting libraries. The algorithm is given by

```
QUICKSORT( array A, int p, int r )
{
 if (r > p)
 then
 i ←  a random index from [p..r]
 swap A[i] with A[p]
 q ←  PARTITION(A, p, r)
 QUICKSORT(A, p, q − 1)
 QUICKSORT(A, q + 1, r)
}
```

The partition algorithm is given by

```
PARTITION( array A, int p, int r)
{
 x ←  A[p]
 q ←  p
 for s ←  p + 1 to r
 do if (A[s] < x)
 then q ←  q + 1
 swap A[q] with A[s]
 swap A[p] with A[q]
 return q
}
```

The partition algorithm works as follows

- A[p] = x is the pivot value.
- A [p…q - 1] contains elements less than x.
- A [q + 1…s - 1] contains the element which are greater than or equal to x.
- A[s...r] contains elements which are currently unknown.

The running time of Quick Sort depends upon heavily on choosing the pivot element. Since selection of pivot element is randomly therefore running time is Θ (nlogn). However worst case running time is $\Theta(n^2)$ but it happens rarely. Quick sort is not stable but is an in-place [1].

3.3 Heap Sort:

A heap is a left complete binary tree which satisfies the heap order. There are two types of heap. The minimum heap contains the smallest value in the root node and the maximum heap contains the largest value in the root node. Heap sort algorithm works as follows

- Build a max heap of a given array A[1…n].
- Extract the largest value from the heap repeatedly.
- When largest element is removed then a whole is left at the root node.
- Replace with the last leaf to fix this problem.

The heap sort algorithm is given by

```
HEAPSORT( array A, int n)
{
 BUILD-HEAP(A, n)
 m ←  n
 while (m ≥ 2)
 do SWAP (A[1], A[m])
 m ←  m− 1
 HEAPIFY (A, 1, m)
}
```

The heapify procedure is given by

```
HEAPIFY( array A, int i, int m)
{
 l ←  LEFT(i)
 r ←  RIGHT(i)
 max ←  i
 if (l ≤ m) and (A[l] > A[max])
 then max ←  l
 if (r ≤ m) and (A[r] > A[max])
 then max ←  r
 if (max ≠ i)
 then SWAP (A[i], A[max])
 HEAPIFY (A, max, m)
}
```

The running time of heap sort is Θ (nlogn). The method to calculate the running time is given in [1].

3.4 Selection Sort:

The main idea of selection sort algorithm is given by

- Find the smallest element in the data list.
- Put this element at first position of list.
- Find the next smallest element in the list.
- Place at the second position of the list and continue until the whole data items are sorted.

The pseudo code of this algorithm is

```
SELECTION-SORT(A)
    for j ← 1 to n-1
        small ← j
        for i ← j + 1 to  n
              if A[ i ] < A[ small ]
                    small ← i
        Swap A[ j ] ↔ A[ small ]
```

The running time of Selection Sort Algorithm is $\Theta(n^2)$ and in-place sorting algorithm. It cannot be implemented as stable sort.

3.5 Insertion Sort:

The main idea of insertion sort is

- Start by considering the first two elements of the array data. If found out of order, swap them
- Consider the third element; insert it into the proper position among the first three elements.
- Consider the fourth element; insert it into the proper position among the first four elements and continue until array is sorted.

The pseudo code is

```
INSERTION-SORT (A)
  For j ← 2 to length [A]
     Do key ← A [j]
      Insert A[j] into the sorted sequence A [1 . . . j - 1].
     i ← j - 1
     While i > 0 and A[i] > key
       Do A [i + 1] ← A[i]
         i ← i - 1
     A [i + 1] ← key
```

The running time of insertion is $\Theta(n^2)$ and in-place sorting algorithm. It can be implemented as stable sort.

3.6 Bubble Sort:

The steps in the bubble sort can be described as below

• Exchange neighboring items until the largest item reaches the end of the array.

• Repeat the above step for the rest of the array.

The pseudo code is

```
Bubble-Sort (A)
   For I ← Length [A] down to 2
        Do for j← 2 to I
                Do if A [j-1] > A [j]
                      Then Swap A [j-1] ↔ A [j]
```

The running time of Bubble Sort is $\Theta(n^2)$ and in-place sorting algorithm. It can be implemented as stable sort.

3.7 Counting Sort:

- The Counting Sort is sorted the numbers whose range is 1 to k where k is small value.
- The main idea is to determine the rank of each element.
- The rank of element is the number of elements which are less than or equal to that number.
- After determine the rank, copy the numbers to their final position in the output array.
- Counting Sort algorithm takes $\Theta(n + k)$ time.
- If k is $\Theta(n)$ then Counting sort is $\Theta(n)$ time algorithm.
- It is stable but not in-place algorithm.

Here is the pseudo code

```
Counting-Sort ( array A, array B, int k)
 For i ← 1 to k
 Do C[i] ← 0
 For j ← 1 to length [A]
 Do C [A[j]]  ← C[A[j]] + 1
 For i ← 2 to k
 Do C[i] ← C[i] + C [i − 1]
 For j ← length [A] down to 1
 Do B[C [A[j]]] ← A[j]
      C [A[j]] ← C[A[j]] – 1
```

3.8 Bucket Sort:

- Let S be a list of n key-element items with keys in [0, N - 1].
- Bucket-sort uses the keys as indices into auxiliary array B:
- The elements of B' are lists, so-called buckets.
- **Phase 1:**
- Empty S by moving each item (k, e) into its bucket B[k].
- **Phase 2:**
- for i = 0 ,…., N - 1 move the items of B[k] to the end of S.

Here is the pseudo code

```
BUCKET-SORT (A)
 B [0….n - 1]
 n = A.length
 For i = 0 to  n - 1
  B [i] = NULL
 For i = 1 to n
 Insert A[i] into list B [nA[i]]
 For i = 0 to n - 1
 Sort list B[i] with insertion sort.
 Concatenate the lists B [0], B [1],……., B [n-1] in order.
```

The running time of bucket sort is $\Theta(n)$. It is stable and in-place algorithm.

3.9 Radix Sort:

- Sort on the most significant digit then the second most significant digit and continue.
- Sort the least significant digit first.

The pseudo code is

```
Radix-Sort (A, d)
For i=1 to d
Stable-Sort (A) on digit i
```

The running time of Radix-Sort is $\Theta(n)$. It is stable but not in-place algorithm.

4. COMPARISON TABLE

This table gives the comparison of time complexity or running time of above sorting algorithm in a short and precise manner which given as under.

Table No. 1:

Name of Algorithm	Worst Case Running Time	Average Case Running Time	Stable	In-Place
Mere Sort	Θ(nlogn)	Θ(nlogn)	Yes	No
Heap Sort	Θ(nlogn)	Θ(nlogn)	No	Yes
Quick Sort	$\Theta(n^2)$	Θ(nlogn)	No	Yes
Insertion Sort	$\Theta(n^2)$	$\Theta(n^2)$	Yes	Yes
Selection Sort	$\Theta(n^2)$	$\Theta(n^2)$	No	Yes
Bubble Sort	$\Theta(n^2)$	$\Theta(n^2)$	Yes	Yes
Counting Sort	Θ(n)	Θ(n)	Yes	No
Bucket Sort	Θ(n)	Θ(n)	Yes	Yes
Radix Sort	Θ(n)	Θ(n)	Yes	No

5. CONCLUSION

In this paper, I have discussed well known sorting algorithms, their pseudo code and running time. In the previous work section, the people have done a comparative study of sorting algorithms. Some have compared four or five different algorithms and their running time. Also some of them compared running time of algorithms on real computers on different number of inputs which is not much useful because in these days, the diversity of computing devices is very high. Manufacture companies change the specification s of their devices day by day. Its means that these type of comparative studies are only valid for some or one specific type of computers or computing devices, not all computers or computing devices. Therefore I have done this comparative study of sorting algorithms impendent of machines. I have compared the running time of their algorithms purely as a mathematical entity and tried to analyze as a generic point of view. I have discussed nine well known sorting algorithms and their running time which is given in the above table. From the table No.1, it is cleared that the running time of Merge Sort, Heap Sort and Quick Sort are Θ(nlogn) therefore these algorithms are also called Θ (nlogn) time algorithms. The Insertion Sort, Selection Sort and Bubble Sort take $\Theta(n^2)$ running time. These are called slow sorting algorithms and expensive in the sense of running time. Their use is not so much in these days. It is mathematically proved that any comparison based sorting algorithm take at least Θ(nlogn) running time [1]. Therefore we should select a sorting algorithm among Θ (nlogn) running time algorithms when sorting is comparison based. From pseudo of merge sort, it is cleared that it has used two arrays or any data structures which is draw back. In Heap Sort, if data items being sorting, initially store any other data structure than heap then they must be stored in a max heap or min heap data structure then will be sorted. I think it is also an overhead of Heap Sort algorithm. In quick sort, there is no need for extra storage it is a stable algorithm. Quick Sort is used in most of applications where sorting is needed. In this algorithm, we have only to care about the selection of pivot element or item. I think among the Θ(nlogn) running time sorting algorithms, the Quick Sort is the best choice. The Counting Sort, Bucket Sort and Radix Sort algorithms are linear time algorithms. The linear time algorithm are the fastest that an algorithm can run. But these algorithms are used under restrictive conditions. In my point of view, the linear time sorting algorithms are best when data items are numeric and in limited size like grades of students.

6. REFERENCES

[1] T. H. Cormen, C. E. Lieserson, R. L. Rivest and S. Clifford, "Introduction to Algorith", 3rd ed., The MIT Press Cambridge, Massachusetts London, England 2009.

[2] A. Jehad, M. Rami "An Enhancement of Major Sorting Algorithms," The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010

[3] B. Ashutosh, M. Shailendra, Comparison of Sorting Algorithms based on Input Sequences, International Journal of Computer Applications (0975 – 8887) Volume 78 – No.14, September 2013.

[4] M. A. Ahmed, 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.

[5] V. Mansotra, Kr. Sourabh, Implementing Bubble Sort Using a New Approach, Proceedings of the 5th National Conference; INDIACom-2011, Computing For Nation Development, March 10 –11, 2011.

[6] Khalid Suleiman Al-Kharabsheh, Ibrahim Mahmoud AlTurani, Abdallah Mahmoud Ibrahim AlTurani & Nabeel Imhammed Zanoon, Review on Sorting Algorithms, A comparative study.

[7] R. Harish, Manisha, Runtime Bubble Sort – An Enhancement of Bubble Sort, International Journal of Computer Trends and Technology (IJCTT) – volume 14 number 1 – Aug 2014.

[8] Savina, K. Surmeet, Study of Sorting Algorithm to Optimize Search Results, International journal of emerging trends and technology in computer science, Volume 2, Issue 1, January – February 2013.

[9] S. Pankaj, Comparison of Sorting Algorithms (On the Basis of Average Case), International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 3, March 2013.

[10] Chhajed. N, U. Imran , Simarjeet. S., B., A Comparison Based Analysis of Four Different Types of Sorting Algorithms in Data Structures with Their Performances, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 2, February 2013.