INTERNATIONAL JOURNAL FOR INNOVATIVE RESEARCH IN MULTIDISCIPLINARY FIELD ISSN: 2455-0620 Volume - 3, Issue - 10, Oct – 2017
Monthly, Peer-Reviewed, Refereed, Indexed Journal with IC Value: 86.87 Impact Factor: 5.60 Publication Date: 30/10/2017

# A Comparative Analysis of Sorting Algorithms on Quick Sort and Merge Sort

**Harsh Srivastava**[1], **Jyotiraditya Tripathi**[2] , **Bhawana Gautam**[3]
[1,2, 3] M.Tech. Student, Centre for Computer Science & Technology (Cyber Security)
Central University of Punjab, Bathinda (Punjab).

Email – [1] Harshmagic20@gmail.com [2] j.adityatripathi@rediffmail.com [3] bg96335@gmail.com

***Abstract:*** *Most sensible applications in computer programming would require the output to be organized in a sequential order. A plenty of sorting algorithms has been developed to reinforce the performance in the terms of computational complexity, memory and alternative factors. This paper is an attempt to check the performance of two sorting algorithm: Quick Sort and Merge Sort, with the aim of comparing their speed when sorting an integer and string arrays. Analysis of these two sorting algorithm was also carried out. The finding shows that Quick sort performs better than Merge sort. The study also indicates that the integer array have faster CPU time than string arrays although both have the upper bound running time $O(n^2)$.*

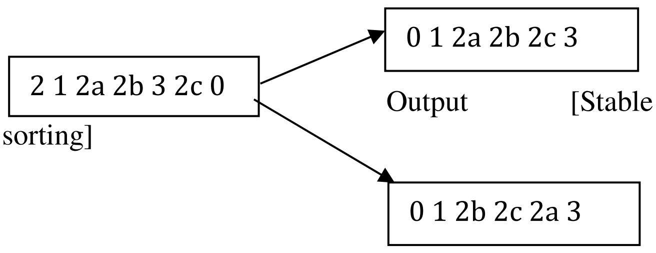
***Key Words:*** *Sorting, Stable Sorting Method, Inplace Sorting Method, Quick sort, Merge sort, Time Complexity and Space Complexity.*

## 1. INTRODUCTION:

Algorithms have a important role in computer science. As we all know that computer is based on instructions and an algorithm is simply a sequence of instruction that we people use to perform any task on computer. Therefore informally an algorithm is a well defined procedure that is used to solve a computational problem. There are different types of algorithms that depends upon the problems. Sorting algorithms are one of the such category. Sorting algorithms are used to arrange the data into a logical sequence. Sorting algorithms mostly work on an array or a list of elements that can be alphabets or numerals. There are many algorithms available for sorting some of that have quite simple working while some have complicated working.

## 2. STABLE SORTING METHOD:

Sorting algorithm is said to be stable if identical elements should occupy relative order as like in unsorted order.

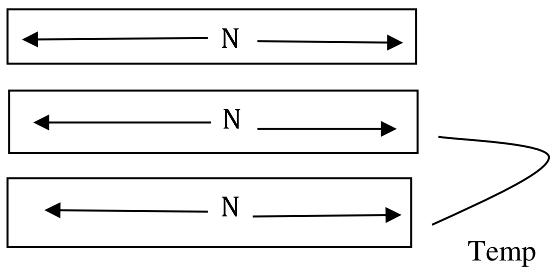


Output [Stable sorting]

Output [Unstable sorting]

- Stable sorting method is preferred if element size is too large(DB record sorting).
- Merge sort is stable sorting method.

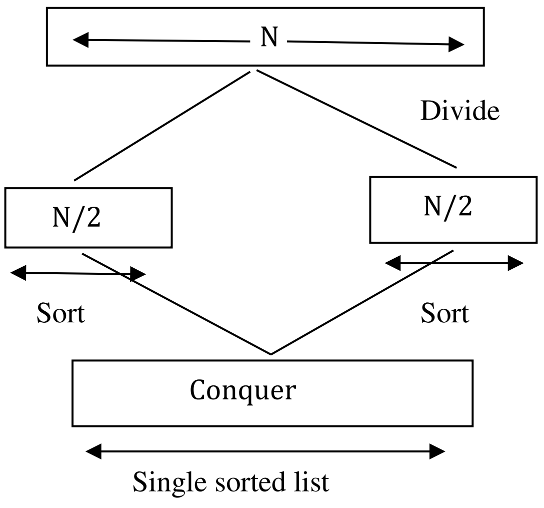
## 3. INPLACE SORTING METHOD:



Merge sort → Non-Inplace sorting

If elements exchanged in same array (input array) then sorting method is inplace.

### I. MERGE SORT



```
Algorithm: Merge sort (low, high)
        {
         //a [low…….high], array of n elements
        If(low<high)
        {
         mid=(low+high)/2
        n/2[Merge sort(low…mid)
        n/2[Merge sort(mid+1….high)]
```

0/n[Merge(low, mid, high)] }}

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 7 | 8 | 3 | 5 | 6 | 6 |

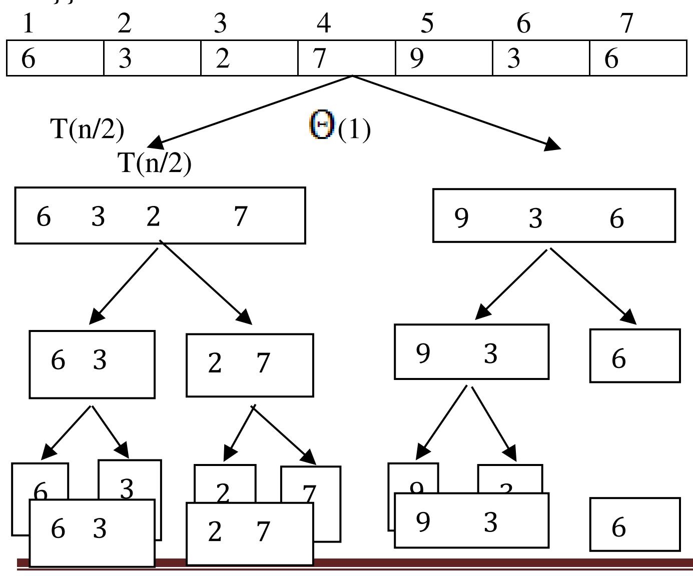
(a)

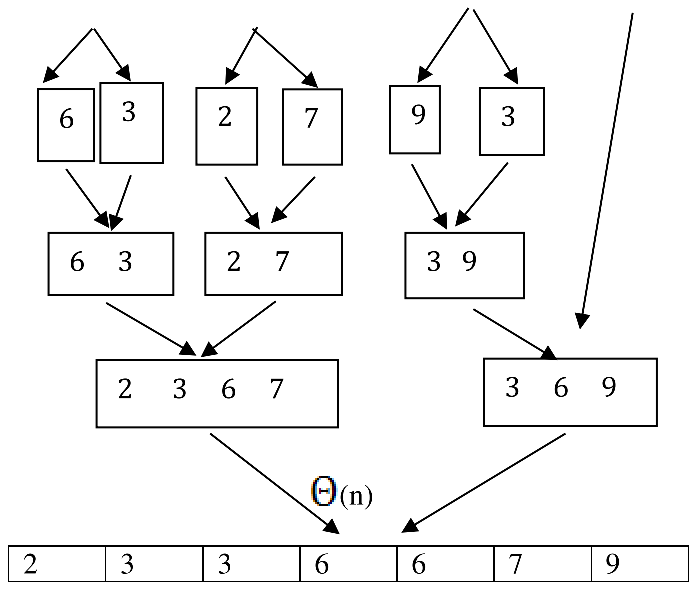
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 5 | 6 | 6 | 6 | 7 | 8 |

(b)

Auxiliary array

```
Algorithm merge(low, mid, high)
// To merge two sorted list a[low, mid] &a[mid+1, high] into one single sorted list.
h=low, j= mid+1, i=low;
while(h<=mid && j<=high)
if(a[h]<=a[j])
{
b[i]=a[h];
h=h+1;
}
else
{
b[i]= a[j];
j=j+1;
}
i=i+1;
}
If(h>mid)
For(k=j;k<=high;k++)
{
b[i]=a[k];
i=i++;
}
else
for(k=h;k<=mid;k++)
{
b[i]=a[k];
i=i+1;
}
for(i=low;i<=high;i++)
{
a[i]=b[i];
}}
```





In this algo while loop can run max(n-1) times & min n/2 times.

**Time Complexity:**

$$T(n)=\begin{cases} 2T(n/2) + CN, & n>1 \\ a=\Theta(1), & n=1 \end{cases}$$

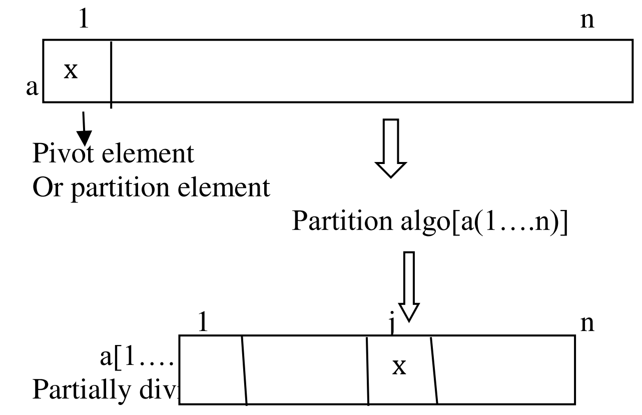
$T(n)= \Theta(n\log2n)$ in all cases.

**Depth of Recursion:**

$\Theta(\log 2n)$ in all cases.
Space complexity[exact I/p space]
Auxiliary array b[]=$\Theta$[n]
Stack space=$\Theta(\log n)$
$\Theta$[n] space complexity.

II. QUICK SORT

Unstable sorting algorithm & Inplace sorting algorithm.
Mainly used in real time. Tony Goore discovered Quick sort.



Algorithm: Quick sort(p,q)
{

```
//a[p……q] array of n elements
If(p<q)
{
j=partition(a,p,q)
Quicksort(a,p,j-1)
Quicksort(a,j+1,q)
}}
```

1. Stop increment of i until a[i] > x
2. Stop decrement of j until a[j] < x
3. If i<j swap(a[i], a[j])
4. Repeat 1, 2, 3 until i>=j
5. Swap(a[j], a[p])

Return j;

Partition element

```
Algorithm: Partition(a, p, q)
{
// a[p……q] array and a[q+1]=+∞
I = p, j = q+1
V=a[p]  //first element is pivot
Do
{
Do
{
I=i+1;
}
While(a[i]<v)
Do
{
J=j-1;
}
While(a[j]>v)
If(i<j) swap (a[i], a[j])
}
While(i<j)
Swap(a[j], a[p])
Return(j)
}
```

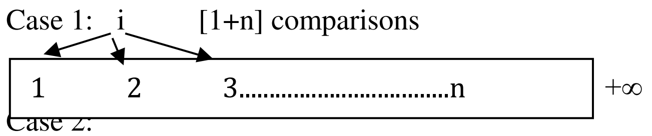
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 15 | 13 | 12 | 15 | 19 | 20 | 18 | 25 | 20 |

Quick(1,3) Quick(5,9)

+∞ is used to avoid array out and access if partition element in max element in array.

**Time Complexity for Partition Algorithm:**

Case 1: i [1+n] comparisons



Case 2:

| | | | | | | |
|---|---|---|---|---|---|---|
| n | n-1 | n-2 | …. | …… | …… | N |

[n+1] comparisons

Case 3:

Unsorted list +∞

n/2+(n/2+1) = n+1 comparisons

Time complexity of partition algo= $\Theta$[n]

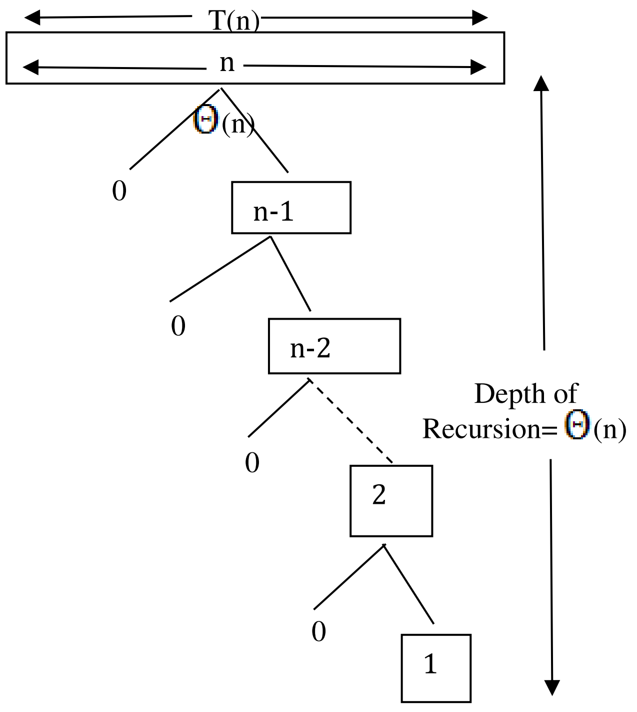
**Time Complexity of Quick sort:**

Worst case time complexity:

Quick sort behaves worst case of input array is already in sorted order.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 2 | 4 | 6 | 7 | 10 | 11 | 12 |

```
Quicksort(1,7)
{
Y=partition
Q.S.(1,0)→0 elements
Q.S.(2,7)→(n-1) elements | T(n-1)
}
```



$$T(n)=\begin{cases} T(n-1)+cn, & n>1 \\ a, & n<=1 \end{cases}$$

= n+n-1+………..+2+1

= $\Theta(n^2)$ [Time complexity of Quick sort in worst case]

Quick sort behaves we if input list divided as ‘c’ and ‘n-c’ elements in every partition.

b) Best case of Time complexity:

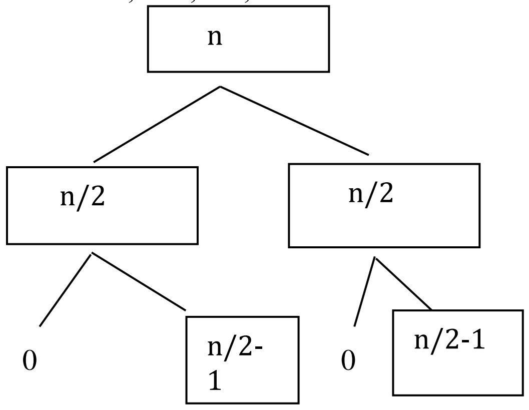
```
QS(p,q)
{
j=partition(a, p, q) → cn
QS(p, j-1) → n/2 elements T(n/2)
QS(j+1, Q) → n/2 elements T(n/2)
}
```

$$T(n)=\begin{cases} 2T(n/2)+cn, & n>=2 \\ a=\Theta(1), & n<=1 \end{cases}$$

Depth of recursion= $\Theta(\log 2n)$
T.C.= $\Theta(n\log n)$
Best case

c) Average case of Time complexity:

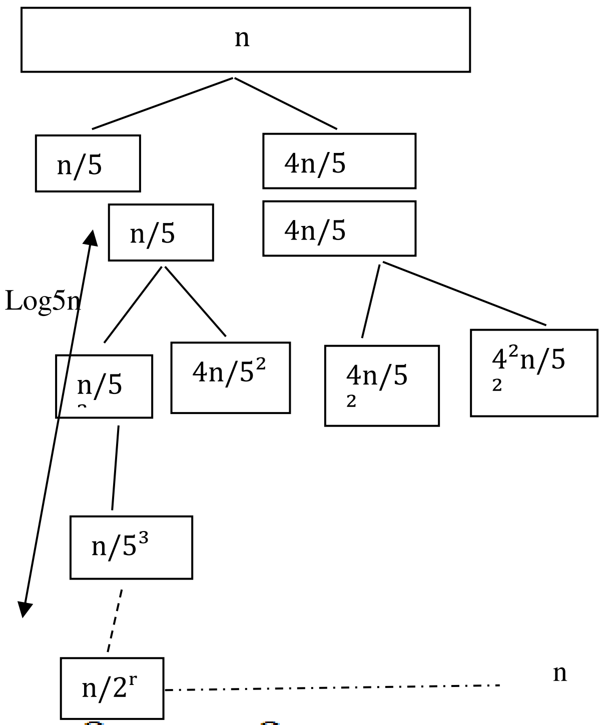
Case 1: BC, WC, BC, WC ……….



Time complexity= $\Theta(n\log n)$

Case 2: n/5 elements and 4n/5 elements in two lists after partition of n element.

$$T(n)=\begin{cases} T(n/2)+T(4n/5)+cn, & n>1 \\ A & n<=1 \end{cases}$$



T(n)= $\Theta(n\log 5/4n)$= $\Theta(n\log n)$

Average time complexity of Quick sort recurrence relation

$$T(n)=\begin{cases} T(\dot{\alpha}.n)+T((1-\dot{\alpha})n)+cn, & n>1 \\ A, & n<=1 \end{cases}$$

$0<\dot{\alpha}<1$
T(n)= $\Theta(n\log n)$
T(n)=T(n/100)+T(99n/100)+cn
$=\Theta(n \log 99/100\ n)= \Theta(n\log n)$

Quick sort is unstable sorting algorithm but inplace algo.

Space complexity of Quick sort(exclusive i/p space)

BC } AC } Stack space= $\Theta(\log n)$
WC } Stack space= $\Theta(n)$

If all elements are identical in input array then Time complexity of Quick sort is $\Theta(n\log n)$.

## 4. CONCLUSION:

| Name | **Quick Sort** | **Merge sort** |
|---|---|---|
| Best case | O(nlog n) | O(nlog n) |
| Average case | O(nlog n) | O(nlog n) |
| Worst case | O(n2) | O(nlog n) |
| Stable | No | Yes |
| Remark | In Place Sorting | Requires extra memory |

This term paper deliberate the comparison of two sorting algorithms. And it concludes that Merge sort performs better than the Quick sort but takes more memory at the time of sorting.

## REFERENCES:

1. Donald E. Knuth et al. "The Art of Computer Programming," Sorting and Searching Edition 2, Vol.3.
2. Cormen et al. "Introduction to Algorithms," Edition 3, 31 Jul, 2009.
3. D. Knuth, "The Art of Computer programming Sorting and Searching", 2nd edition, Addison-Wesley, vol. 3, (1998).
4. D. Mishra and D. Garg, "Selection of the best sorting algorithm", International Journal of Intelligent Information Processing, vol. 2, no. 2, (2008) JulyDecember, pp. 363-368.
5. C. A. R. Hoare, Algorithm 64: Quick sort. Comm. ACM, vol. 4, no. 7 (1961), pp. 321.

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)., ISSN(e): 2319 – 1813 ISSN(p): 2319 – 1805.
7. E. Horowitz, S. Sahni and S. Rajasekaran, Computer Algorithms, Galgotia Publications.
8. Horowitz, E., Sahni. S, Fundamentals of Computer Algorithms, Computer Science Press, Rockville. Md
9. Laila Khreisat, "Quick Sort: A Historical Perspective and Empirical Study", IJCSNS
10. T. H. Coreman, C. E. Leierson, R. L. Rivest and C. Stein, Introduction to Algorithms, 2nd edition, MIT Press.
11. John Darlington, Remarks on "A Synthesis of Several Sorting Algorithms", Springer Berlin / Heidelberg, pp 225-227,Volume 13, Number 3 / March, 1980.

**WEB REFERENCES:**

http://www.geeksforgeeks.org/iterative-quick-sort
https://en.wikipedia.org/?title=Merge_sort
https://en.wikipedia.org/?title=Quicksort
http://www.geeksforgeeks.org/forums/topic/merge-sort/