International Journal of Engineering Sciences & Emerging Technologies, Dec. 2024
ISSN: 22316604 Volume 11, Issue 4, pp: 894-901 ©IJESET

A COMPREHENSIVE STUDY ON THE DESIGN AND ANALYSIS OF COMPARISON – BASED SORTING ALGORITHMS

Aryan Sharma[1], Mahendra Singh Sagar[2]
[1]Department of Computer Science & Engineering (AI & ML),
Moradabad Institute of Technology, Moradabad, India
[2]Department of Computer Science & Engineering (AI & ML),
Moradabad Institute of Technology, Moradabad, India
[1]aryan070sharma@gmail.com
[2]mahendra.singh12jan@gmail.com

ABSTRACT

This technical paper gives you a comprehensive study on the design and analysis of four distinct algorithms. The algorithms include 'Insertion Sort', 'Selection Sort', 'Merge Sort', 'Quick Sort'. Each one of the algorithms has been chosen for their unique and relevance on the target problem. The primary goal of this paper is to compare the performance, efficiency, and scalability of these algorithms which includes time and space complexity. The results obtained from the comparative analysis shows the valuable guidance to practitioners and researchers seeking to make informed algorithmic choices. The perception obtained from this research aim to inform future algorithmic development and deployment strategies in order to achieve boost performance and efficiency.

KEYWORDS
Sorting, Algorithms, Complexity.

1. INTRODUCTION

In the design and study of algorithms, Sorting is the process of putting information in a structured way. By sorting the data, it makes the search undisturbed which helps in searching the data faster and easier. Dictionary can be the most simple example of sorting. Comparison – Based sorting can be defined as comparing the data values with each other and arranging it in ascending or descending order. Sorting also includes one of the most valuable characteristics know as complexity which helps to identify the efficiency of the comparison – based sorting algorithms. Complexity is divided into two parts – Time Complexity and Space Complexity.
The time required in the computation of the algorithm is known as time complexity and the space required in the execution of the algorithm is known as space complexity. Time complexity have three cases – Best case, Average case and Worst case. And other properties of sorting algorithms are In-place, not in-place. In-place property can be defined as the data which is sorted by the algorithm does not require any extra space for it, it only uses the given data space. While not in-place can be defined as the data which is sorted by the algorithm required the additional space for the sorting operation, it cannot sort the given data int data space.

2.RELATED WORK

Very few literatures are available on Comprehensive Study based on sorting algorithms.
The objective of Sandeep Kaur Gill et al.'s paper is to examine the sorting algorithms' time complexity: Comparative and non-comparative sorting algorithms are two distinct types of algorithms based on the methodology used to sort the set of values. [1]. Neelam Yadav et al demonstrates the various sorting methods found in structures of data, such as fast, insertion, heap, and merge sorting. Every algorithm

uses a different format to attempt to solve the sorting problem[2].In order to show how this algorithm works to minimize run time, Khalid Suleiman Al-Kharabsheh et al. compared the Grouping Comparison Sort (GCS) against conventional algorithms such as Selection sort, Quick sort, Insertion sort, Merge sort, and Bubble sort [3].

3. SORTING ALGORITHMS: A DETAILED ANALYSIS

3.1 Insertion Sort: In insertion sort, it picks the first element from the given array and iterate to find and put the element on the desired place. It sorts the array by comparison and using the shifting operations to shift the place of the element. The simplest example of insertion sort is arranging the playing cards.

The following is a list of the easy methods to do the insertion sort:

Step 1: If it is the first element, assume that it has already been sorted. Return to 1.

Step 2: Select the next piece and lock it separately with a key.

Step 3:At this point, compare each and every element in the array that the key was used to sort.

Step 4: Check to see if the next item in the sorted collection is smaller than the one you are looking at.

Step 5: Enter the number.

Step 6:Until the array is sorted, step six is to repeat.

Example:

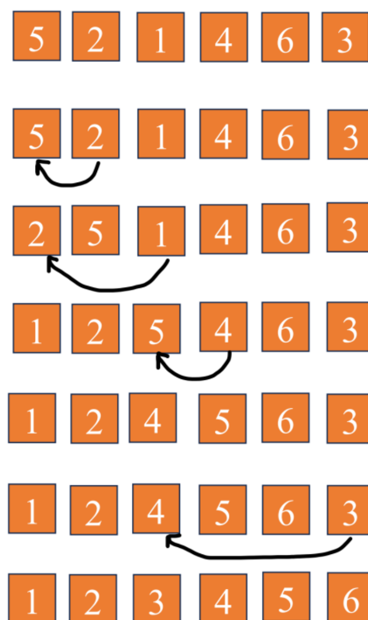

Fig 3.1 Insertion sort example

***Time complexity analysis of Insertion Sort*:**

3.1.1 ***Best Case***: If the array is in ascending order already

No. of comparison	No. of swap
0	0
1	
1	
1	
1	

Total no. of comparison = (n-1)

Total no. of swap = 0

Thus, complexity is O (n^2)

3.1.2 ***Worst Case*:** When the given array is given in descending order.

No. of comparison	No. of swap
0	0
1	1
1	1
1	1
1	1
1	1
.	.
.	.
.	.
.	.
n-1	n-1

Total no. of comparison = n(n-1)/2

Total no. of swap = n(n-1)/2

Thus, complexity is O (n^2)

3.1.3 ***Average Case*:** Similar to worst case as we have to iterate n-1 times

Thus, complexity is O (n^2)

3.2 Selection Sort: In selection sort, an element is chosen from the provided array, compared to determine which element is the smallest in each iteration, and then shifted from the unsorted array to the sorted array.

The steps in the selection sort algorithm are as follows:

Step 1: Choosing The input list is split into the sorted and unsorted sublists by the algorithm. The list is initially unsorted in its entirety.

Step 2:Identifying the Minimum in It begins by presuming that the minimal value is the first entry in the unsorted sublist. The lowest element is then found by repeatedly iterating through the unsorted sublist.

International Journal of Engineering Sciences & Emerging Technologies, Dec. 2024
ISSN: 22316604 Volume 11, Issue 4, pp: 894-901 ©IJESET

Step 3: Swapping: The least element in the unsorted sublist is found, and it is then swapped with the unsorted sublist's initial element. By doing this, the sorted sublist is essentially increased by one element, and the unsorted sublist is decreased by one member.

Step 4: Repeat: For the remaining unsorted sublist, steps 2 and 3 are repeated. The minimal element is located and positioned correctly within the expanding sorted sublist.

Step 5: Finalization: The procedure keeps going until the whole list is sorted. The smallest unsorted sublist element that is still present at the end of each iteration is added to the end of the sorted sublist until all unsorted elements have been eliminated, producing a fully sorted list.

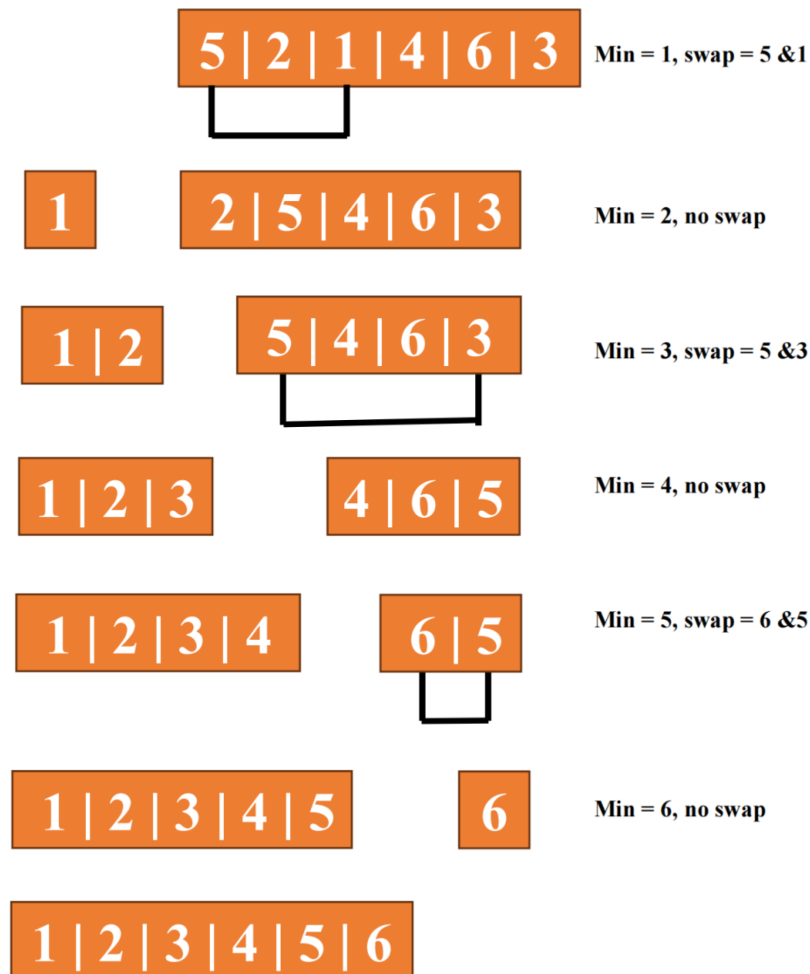

Fig 3.2.selection sort example

3.2.1 Time complexity analysis of Selection Sort:

In selection sort best, worst and average all case have the same complexity as there are two loops and the loop used are nested loop. Both loops iterate for n times therefore, n*n

Thus, complexity is O (n^2)

***3.3 Merge Sort*:** In merge sort, an unsorted array is split into smaller sub-arrays. These sub-arrays are then sorted and put back together to make a single sorted array by matching their parts. It repeatedly divides the array into half until all the sub – arrays are only left with one element and then combines them together while sorting in the merging step.

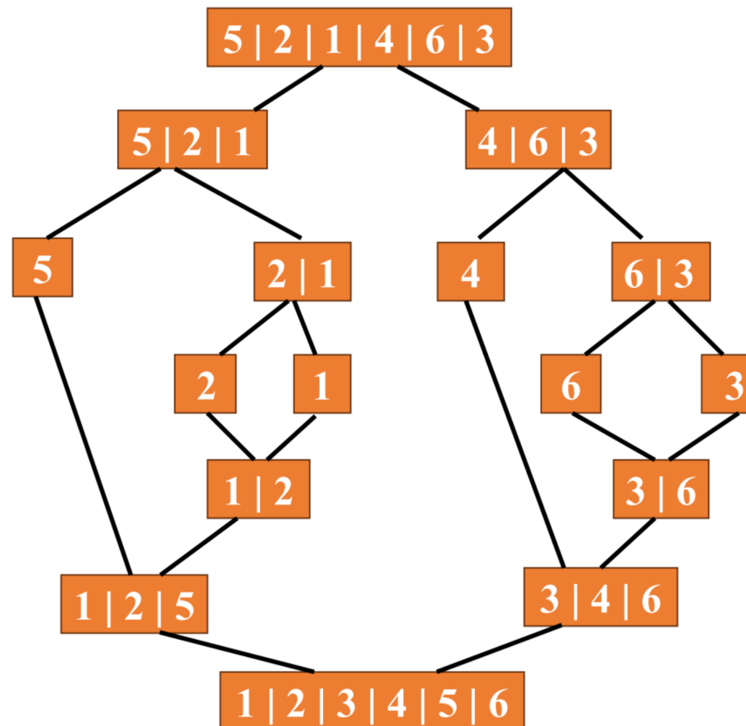

Fig 3.3 Merge sort example

3.3.1 Time complexity analysis of Merge Sort:

Assume that T(k) = Time spent sorting each of the k elements.
Moreover, M(k) = Time spent combining the k elements.
Now, $T(n) = 2 * T(n/2) + M(n)$
$= 2 * T(n/2) + constant * n$
Here, n/2 are split into another pair of halves. Thus
$= 4 * T(n/4) + 2 * n * constant$

.
.
.

$= 2^k * T(n/2^k) + k * n * constant$

Here, $n/2^k = 1 * k = \log_2 n$
From, $T(n) = n * T(1) + n * \log_2 n * constant$
$= n + n * \log_2 n$
Thus, complexity is $O(n * \log_2 n)$

3.4 Quick Sort: In quick sort, it is a divide and conquer algorithm which works by choosing a pivot element from the given data in array and portioning it into two sub – array depending if they are lesser or greater than the pivot element, it works recursively. Some of the ways of choosing a pivot are as follows -

- One option for the pivot is to choose a random pivot from the provided array.
- The pivot point in the specified array might be either the leftmost or the rightmost element.
- Decide on the median to be the pivot point.

International Journal of Engineering Sciences & Emerging Technologies, Dec. 2024
ISSN: 22316604 Volume 11, Issue 4, pp: 894-901 ©IJESET

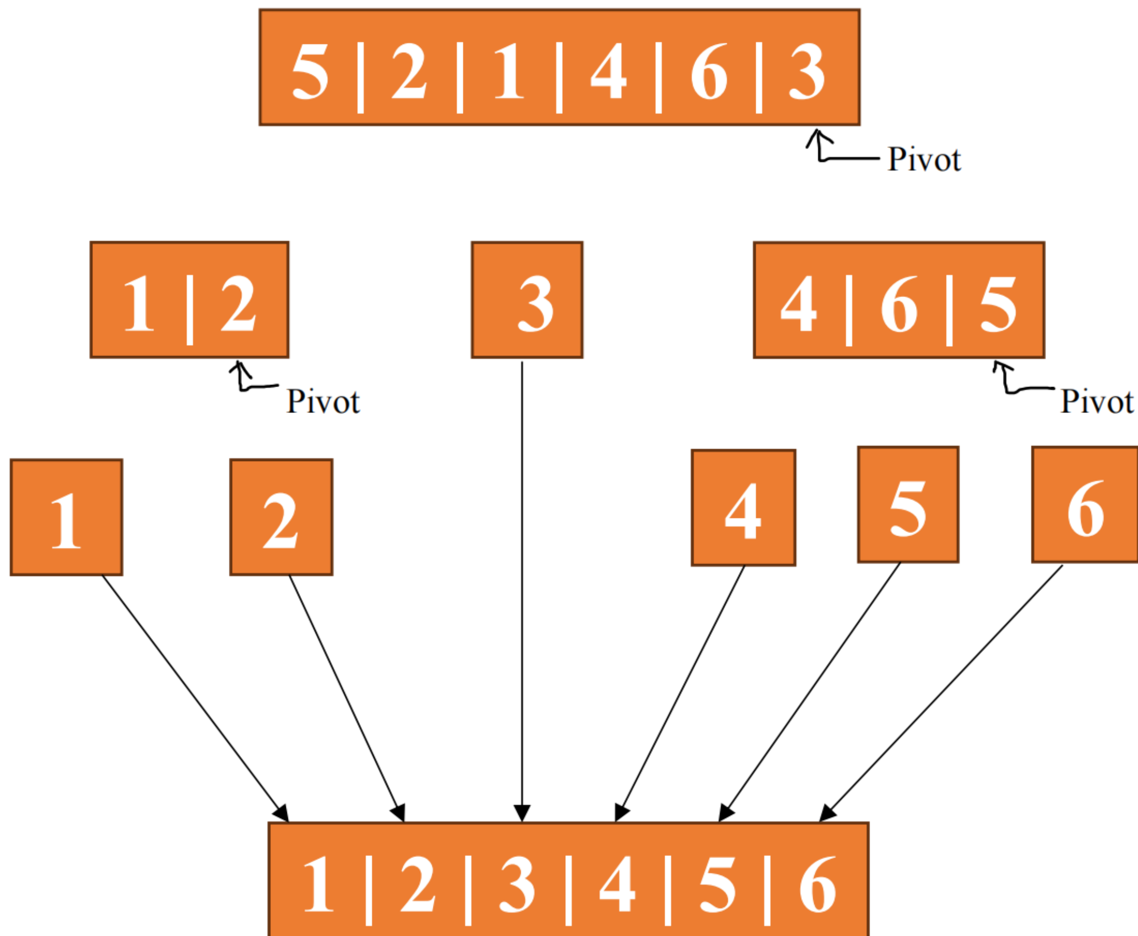

Fig 3.4 Quick sort example

3.4.1 Time complexity analysis of Quick Sort:

3.4.1.1 Best Case: When the pivot element divides the given array into two equal sub – array.

Thus, $T(n) = 2 * T(n/2) + n * \text{constant}$
$T(n) = 4 * T(n/4) + 2 * n * \text{constant}$
.
.
$T(n) = 2^k * T(n/2^k) + k * \text{constant} * n$
Then, $2^k = n$
$k = \log_2 n$
Therefore, $T(n) = n * T(1) + n * \log_2 n$
So, complexity is $O(n * \log_2 n)$

3.4.1.2 Average Case: Same as best case but occurs when pivot element divides the array into k into n – k size of two sub – arrays.

Thus, complexity is $O(n * \log_2 n)$

3.4.1.3 Worst Case: The worst-case scenario is when the array has already been sorted and a quick sort algorithm gets used.

Thus, complexity is $O(n^2)$

3.5 Discussion

Aspects	*Insertion Sort*	*Selection Sort*	*Merge Sort*	*Quick Sort*
Time Complexity	$O(n^2)$ average and worst-case Best case: O(n) when the array is already sorted	$O(n^2)$ average and worst-case Best case: $O(n^2)$ when the array is already sorted	$O(n \log_2 n)$	$O(n \log_2 n)$ average , $O(n^2)$ worst-case Best case:
Space Complexity	O(1)	O(1)	O(n)	$O(\log_2 n)$
Stability	Stable	Not stable	Stable	Not stable
Adaptive	Yes	No	No	No
Best for	Small arrays, nearly sorted arrays	Small arrays, nearly sorted arrays	Large arrays, linked lists	General purpose, large arrays
Worst for	Large arrays, completely reverse sorted	Large arrays, completely reverse sorted	Not applicable	Already sorted arrays

4. CONCLUSION

In this research, we have taken one common array to check the different computational capabilities, adaptivity, stability, time complexity and space complexity of all four distinct algorithms – Insertion sort, Selection sort, Merge sort and Quick sort. All four algorithms have some strength such as working on small arrays or nearly sorted array, small arrays or nearly sorted array, working on large array or linked list and for general purpose, large array respectively. The algorithms also have some weakness such as not good for large arrays or completely reverse sorted, good for large arrays or completely reverse sorted, not applicable for merge sort and worst for already sorted arrays respectively.

REFERENCES

[1]Sandeep Kaur Gill et al “A Comparative Study of Various Sorting Algorithms”Special Issue based on proceedings of 4th International Conference on Cyber Security (ICCS) 2018.
[2] Neelam Yadav, SangeetaKumari “SORTING ALGORITHMS” International Research Journal of Engineering and Technology (IRJET) Volume: 03 Issue: 02 | Feb-2016
[3] Khalid Suleiman Al-Kharabsheh et al “Review on Sorting Algorithms A Comparative Study” International Journal of Computer Science and Security (IJCSS), Volume (7) : Issue (3) : 2013
[4] A.D. Mishra, D.Garg, “Selection of Best Sorting Algorithm”, International Journal of Intelligent Processing, 2(2): pp.363- 368, 2008
[5] Thomas H. Cormen, Chales E. Leiserson, Ronald L. Rivest and Clifford Stein, “Introduction to Algorithms”, MIT press, 2009.
[6] Yang, Ping Yu, Yan Gan, “Experimental Study on the Five Sort Algorithms”, IEEE, 2011, 978-1-4244-9439-2/11
[7] Dongchen Jiang, Miao Zhou, “A Comparative Study of Insertion Sorting Algorithm Verification”, IEEE, 2017, 978-1- 5090-6414-4/17
[8] Avinash Shukla, Anil Kishore Saxena, “Review of Radix Sort & Proposed Modified Radix Sort for Heterogeneous Data Set in Distributed Computing Environment”, International Journal of Engineering Research and Applications (IJERA), Vol. 2, Issue 5, pp.555-560,2012
[9] Tanvi Puri, Anuj Kumar Jain, Anjana Sangwan, “Design And Analysis Of Optimized Counting Sort Algorithm (OCSA)”, Spvryan’s International Journal of Engineering Science and technology (SEST), Issue 1 , Volume 1 - Oct. 2014
[10] Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, “Fundamentals of Computer Algorithms”, Galgotia Publications Pvt. Ltd., 1998
[11] Michael T. Goodrich, Roberto Tamassia, “Algorithm Design”, Wiley India (P) Ltd, 2002
[12] Udit Agarwal, “Algorithms Design and Analysis”, Dhanpat Rai & Co. Pvt. Ltd., 2012
[13] E.V.Krishnamurthy, S.K.Sen, “Numerical Algorithms”, Affiliated East-West Press Private Limited, 1986
[14] Reina Setiawan, “Comparing Sorting Algorithm Complexity Based on Control Flow Structure”, IEEE, 2016, 978-1-5090- 3352-2.

[15] Neetu Faujdar, Shipra Saraswat, " The detailed experimental analysis of bucket sort", IEEE, 2017, 978-1-5090-3519-9
[16] Sorting in linear time. http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap0 9.htm, March 2014
[17] A.D. Mishra, D.Garg, "Selection of Best Sorting Algorithm", International Journal of Intelligent Processing, 2(2): pp.363- 368, 2008
[18] Thomas H. Cormen, Chales E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms",MIT press, 2009.
[19] Udit Agarwal, "Algorithms Design and Analysis", Dhanpat Rai & Co. Pvt. Ltd., 2012
[20] E.V.Krishnamurthy, S.K.Sen, "Numerical Algorithms", Affiliated East-West Press Private Limited, 1986