



Impact of Ascending and Descending Input Patterns on Sorting Techniques

¹Md. Aman Khan, ²Rohini Nag, ³Subhashis Majumder

Department of Computer Science and Engineering,

Heritage Institute of Technology, Kolkata, West Bengal 700 107, India.

Email: ¹aman.khan1991@gmail.com, ²rohini.nag22.5@gmail.com, ³subhashis.majumder@heritageit.edu

Abstract— Sorting is a process of arranging elements in an orderly manner, based on the value of a certain key which exists for each element. This key may be the value of the element itself or some other property associated with it. Sorting plays an important role in computer science as it is often performed, either in a solitary manner or as a part of some other algorithm. Thus its efficiency in terms of time and space is a crucial factor. Given a sequence of n elements, the number of comparisons required to sort them in the ascending order will be different than that of the number required for descending order. The objective of this paper is to analyze the performance of different sorting techniques with respect to the ascending and descending patterns in the elements to be sorted and to determine the importance of being able to predict whether it will be easier to sort the elements in ascending order or descending order.

Index Terms—Ascending Order, Descending Order, Sorting, Sorting Analysis.

I. INTRODUCTION

In computer science, sorting algorithms are used to rearrange input data in an orderly manner, either in ascending order or in descending order. Sorted data may be the primary requirement for the user or it may be required by some other algorithm that is to be performed. Because of its usage in so many situations, a sorting algorithm needs to be efficient in terms of both time and space and hence it has been a subject of extensive research in computer science from the very early days. There are many sorting algorithms available, ranging from simple to fairly complex [5].

II. TYPES OF SORTING ALGORITHMS

Types of sorting algorithms available can be broadly classified into two the following two types [3, 4] –

A. Comparison Sort

The data is analyzed by comparing the key value of elements with other elements and decisions are taken

based on these comparisons. Some examples are Bubble Sort, Insertion Sort, Quick Sort, Merge Sort and Heap Sort.

B. Non-Comparison Sort

The data is sorted based on the pattern of the key value without it being compared with other key values. It is also called content based sorting. Some examples are Radix Sort, Counting Sort and Bucket Sort.

In this paper we will be focusing on several comparison sorts and try to determine that for a given input, if it can be determined whether it is easier to sort it in ascending or descending order and by how much can the time taken for sorting be reduced.

Given in Table I are the asymptotic time complexities for best, worst and average cases of a few comparison sorting techniques.

Table I: Asymptotic Time complexities of the most popular comparison sorts [1, 3, 4, 5]

Type of Sorting	BEST CASE	AVERAGE CASE	WORST CASE
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

III. ANALYSIS OF SORTING TECHNIQUES

All the sorting Algorithms can be performed in two ways, Ascending or Descending. However, for a given input pattern, the number of comparisons required for sorting it in ascending order may differ from the number of comparisons required to sort the same array in descending order. Thus if we can determine from the input pattern,

whether an Ascending Sort will be easier or a Descending Sort without actually sorting them, we can reduce the time required for sorting large number of elements by a considerable amount. We will be analysing the different sorting techniques by taking all possible permutations of n elements and finding the average number of comparisons required for sorting them. For each value of n we will find out three values, namely Avg_Asc, Avg_Desc and Avg_Min.

Definition 2.1. For a particular value of n , Avg_Asc refers to the average number of comparisons required to sort n elements in ascending order. It is averaged over $n!$ different permutations.

Definition 2.2. For a particular value of n , Avg_Desc

refers to the average number of comparisons required to sort n elements in a descending order. It is averaged over $n!$ different permutations.

Definition 2.3. For a particular value of n , Avg_Min refers to the average number of comparisons required to sort n elements if it can be correctly predicted for all individual patterns whether Ascending Sort or Descending Sort will require lesser comparisons. It is averaged over $n!$ different permutations.

All possible permutations of n elements are sorted for $n=2$ to 11. For any sequence of n distinct element, since there exists a reverse sequence in the set of all possible permutations of n elements, values of Avg_Asc and Avg_Desc will be exactly equal for each n .

Table II: Analysis of Bubble Sort

Count Type	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10	N=11
Avg_Asc	1.5	4.33	8.62	14.43	21.81	30.80	41.43	53.72	67.7	83.36
Avg_Desc	1.5	4.33	8.62	14.43	21.81	30.80	41.43	53.72	67.7	83.36
Avg_Min	1	3.33	6.75	11.67	17.96	25.77	35.10	46	58.5	72.59
$\Delta\text{Avg}\%$	33.33	23.08	21.74	19.17	17.67	16.32	15.26	14.35	13.58	12.91

Definition 2.4. Let us define a parameter $\Delta\text{Avg}\%$ to analyse the difference between Avg_Asc and Avg_Min.

$$\Delta\text{Avg}\% = \left(\frac{\text{Avg_Asc} - \text{Avg_Min}}{\text{Avg_Asc}} \right) \times 100$$

A. Bubble sort

Improved Bubble Sort has a best case time complexity of $O(n)$, average case time complexity of $O(n^2)$ and worst case time complexity of $O(n^2)$. However, being able to correctly choose between ascending and descending sort does have a considerable impact. Let us analyze this impact for permutations of n elements where n ranges from 1 to 10. Values have been rounded off to two decimal places. As can be viewed from Table II, values of Avg_Asc and Avg_Desc are always the same. From now on we will only show values of Avg_Asc in the Tables instead of both. Also to be noted that the values shown in the Table II are rounded off and this will apply for all future data presented in Tables.

B. Insertion sort

Insertion sort is a sorting technique with a best case time complexity of $O(n)$, whereas average and worst case time complexity of $O(n^2)$ [1, 3, 4]. It sorts almost sorted patterns in a very short amount of time. However, if a pattern that is almost sorted in descending order has to be sorted in ascending order, it takes more time. Thus it will be very helpful if it can be predicted beforehand, in which order should elements be sorted, so that sorting time can be reduced. Let us analyze insertion sort from Table III

and understand how much impact can be made by being able to predict the more efficient order of sorting.

C. Quick sort

Quicksort, or partition-exchange sort, has a best and average time complexity of $O(n \log n)$ [1, 3, 4] and worst case of $O(n^2)$ to sort n elements. It works based on shifting the pivot element to its correct position in each pass. The impact of being able to predict whether ascending sort will be better or descending sort has been analyzed in Table IV. However, here the benefit appears to be less than bubble sort and insertion sort, at small values of n .

D. Merge sort

Merge sort is a comparison based sorting algorithm with same best case, worst case and average case time complexity $O(n \log n)$ [1, 3, 4]. As it follows a divide and conquer approach, the initial pattern of elements provided do not lead to any impact on the number of comparisons required as can be seen from the analysis in Table V. It actually makes the same number of comparisons whether an already sorted sequence is provided or highly disorganized sequence is provided.

E. Heap sort

Heap Sort is a comparison sort which improves upon the linear time of a single pass of a normal comparison sort by using a logarithmic priority queue. There will be a total of n passes required to sort n elements. Its best case, worst case and average case has the same time complexity of $O(n \log n)$ [1, 3, 4]. The analysis can be seen in Table VI.

IV. COMPARISON OF ALL SORTING TECHNIQUES

The impact of being able to predict the better order of sorting is not same for all sorting techniques. Thus, we need to perform a comparative study on the different sorting techniques based on the values of $\Delta\text{Avg}\%$. As it can be seen from table vii, merge sort has the least impact of being able to predict the better order of sorting. However, there will be a considerable improvement in bubble sort and insertion sort if we can predict the better order of sorting correctly for a given input pattern.

V. PREDICTING THE BETTER ORDER OF SORTING

Now that we have analyzed the importance of being able to predict the easier order of sorting based on number of comparisons required, let us try to devise an algorithm to predict which order of sorting will be easier for a given set of elements. For that we need to understand the following two terms.

Definition 4.1. Asc_pairs for a particular set of n elements $S=\{S_1, S_2, S_3 \dots S_n\}$ refers to the number ascending pairs, that is the number pairs of elements S_i and S_{i+1} such that $S_i \leq S_{i+1}$

Definition 4.2. Desc_pairs for a particular set of n elements $S=\{S_1, S_2, S_3 \dots S_n\}$ refers to the number of descending pairs, that is the number pairs of elements S_i and S_{i+1} such that $S_i \geq S_{i+1}$.

Table III: Analysis of Insertion Sort

Count Type	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10	N=11
Avg Asc	2.5	4.5	7	10	13.5	17.5	22	27	32.5	38.5
Avg Min	2	3.67	5.83	8.37	11.33	14.8	18.75	23.15	27.99	33.33
$\Delta\text{Avg}\%$	20	18.5	16.7	16.33	16.09	15.45	14.8	14.27	13.87	13.44

Table IV: Analysis of Quick Sort

Count Type	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10	N=11
Avg Asc	2	4.33	7.17	10.4	13.97	17.82	21.92	26.25	30.77	35.48
Avg Min	2	3.67	6.33	9.32	12.65	16.23	20.07	24.12	28.36	32.77
$\Delta\text{Avg}\%$	0	15.38	11.63	10.42	9.45	8.92	8.43	8.12	7.83	7.63

Table V: Analysis of Merge Sort

Count Type	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10	N=11
Avg Asc	2	5	8	12	16	20	24	29	34	39
Avg Min	2	5	8	12	16	20	24	29	34	39
$\Delta\text{Avg}\%$	0	0	0	0	0	0	0	0	0	0

Table VI: Analysis of Heap Sort

Count Type	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10	N=11
Avg Asc	1	3	6.5	10.95	15.13	19.79	25.8	32.5	38.6	45.24
Avg Min	1	3	6.17	10.2	14.23	19	25	31.3	37.3	43.78
$\Delta\text{Avg}\%$	0	0	5.13	6.85	5.95	4.01	3.2	3.67	3.46	3.22

Table VII: Comparative Analysis of $\Delta\text{Avg}\%$ for Sorting Techniques

Type of sorting	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10	N=11
Bubble sort	33.33	23.08	21.74	19.17	17.67	16.32	15.26	14.35	13.58	12.91
Insertion sort	20	18.5	16.7	16.33	16.09	15.45	14.8	14.27	13.87	13.44
Quick sort	0	15.38	11.63	10.42	9.45	8.92	8.43	8.12	7.83	7.63
Merge Sort	0	0	0	0	0	0	0	0	0	0
Heap sort	0	0	5.13	6.85	5.95	4.01	3.2	3.67	3.46	3.22

Table VIII: Acc% for Comparison Sorts

Type of sorting	N=2	N=3	N=4	N=5	N=6	N=7	N=8	N=9	N=10	N=11
Bubble sort	100	83.33	83.33	74.17	73.89	69.39	69.75	66.57	66.84	64.59
Insertion sort	100	66.7	91.67	76.67	71.94	65.12	71.56	66.86	63.96	61.2
Quick sort	100	66.7	66.7	58.33	58.61	56.57	55.46	54.68	54.05	53.39
Heap sort	100	100	50	50.83	56.39	55.06	49.19	48.49	52.46	51.08

Based on the values of Asc_pairs and Desc_pairs we make the following predictions.

- An ascending sort will take lesser comparisons if $\text{Asc_pairs} > \text{Desc_pairs}$.
- A descending sort will take lesser comparisons if $\text{Asc_pairs} < \text{Desc_pairs}$.
- If $\text{Asc_pairs} = \text{Desc_pairs}$ we perform an Ascending Sort.

We analyse out sorting techniques to find out the accuracy of out predictions. The accuracy for merge sort is always 100 percent as the order of sorting does not have any impact in the algorithm. Thus the analysis of merge sort has been left out from Table VIII.

Definition 4.3. Acc% refers to the percentage of accuracy of prediction of the better order of sorting.

As can be realized from the Table VIII, our prediction tends to give good results for bubbles sort and insertion sort. Prediction in Quicksort and Heap sort tends to be very weak. There is no need of any prediction for Merge sort as the number of comparisons required in merge sort are fixed and only dependent on the number of elements to be sorted. Hence, bears no importance to the pattern of the elements.

VI. CONCLUSION

We conclude that being able to predict the optimal order of sorting for a given set of elements will help to reduce the number of comparisons by a considerable amount. We can also predict the optimal order of sorting to some

extent on the basis of the Asc_pairs and Desc_pairs. Further research needs to be done on effective reduction in comparisons if the prediction is applied. Also we need to analyze the results for larger number of elements by using other means of average case analysis [2].

REFERENCES

- [1] Nidhi Chhajed, Imran Uddin and Simarjeet Singh Bhatia, "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.
- [2] Pankaj Sareen, "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.
- [3] Donald Knuth, "Art of Computer Programming Volume 3: Sorting and Searching", Second Edition (Pearson Education Singapore Ltd, 1998).
- [4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms", Second edition (MIT, 2010).
- [5] Sonal Beniwal and Deepti Grover, "Comparison Of Various Sorting Algorithms: A review", International Journal of Emerging Research in Management & Technology, Volume-2, Issue-5, May 2013.

