IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Least First Sort - New sorting algorithm

Swarna Saha

Narula Institute of Technology

Kolkata, India

Sauna Roy

Narula Institute Of Technology

Kolkata, India

Subhasree Bhattacharjee

Narula Institute of Technology

Kolkata, India

Abstract: Sorting is the process of organising data or specific elements into meaningful order so that analysis can be done more efficiently. In this paper, we are intending to introduce a new sorting algorithm called Least-First Sort. In this technique, after each stage of comparison, the smallest element will come first. We have compared the algorithm with other sorting algorithms. We have done the comparison with 10000 to 60000 elements. To the best of our knowledge, the newly proposed algorithm results the least run time than Bubble Sort and Selection Sort.

Keywords: *sorting; complexity; smallest; algorithm.*

I. Introduction

The process of sorting the data is a fundamental algorithmic problem in the data structure. It helps to solve many problems in our daily life as well as in the technological world. A programmer face with many tasks, many problems and sorting algorithms help them to solve the tasks. Nowadays there are so many different sorting algorithms have been developed. They are developed by using some methods like divide and conquer, insertion, randomization, exchange, merging etc. [1]. There are too many sorting algorithms, but all are not predominating. Dominate algorithms are implemented in industrial case [2].

Primarily sorting algorithms are used depending on what they are needed for. This means we need a different sorting technique for different circumstances. In our daily life, sorting algorithms are used to sort the data. For example, if there is a database table that has some attributes like Name, Roll No., Age, etc. and Roll No. is the primary key. Then it will be easy to arrange the data according to their Roll No. using a sorting method, in case the data are randomly arranged.

There are many sorting algorithms out there that we can choose for our specific purposes. While considering which algorithm to choose, we need to consider many factors like different kinds of complexities. There are two types of complexities in the sorting method. Time complexity and space complexity. It is very important to reduce the complexity. Generally sorting algorithms is divided into two categories [3].

1. Comparison Sorting

This type of sorting technique compares elements at every step of the algorithm to decide where the individual elements should be of another element.
Comparison sorts are normally more straightforward to achieve than integer sorts but compare sorting technique is limited by a lower bound of *O(nlogn)*, implying that, on average, comparison sorts cannot be faster than *O(nlogn)*. A lower bound for an algorithm is the *worst-case* running time of the *best* possible algorithm for a given problem.
Time complexity depends on the number of elements used in the sorting algorithm [4]. The complexity in ascending order is $O(n)$, $(nlogn)$, $O(n^2)$. That means $O(n)$ is the best for complexity because it gives the lowest execution time then $(nlogn)$ then $O(n^2)$.
Some of the most famous comparison sort example includes, Quick sort [5], Heap sort [6], Shell sort, Merge sort, Intro sort, Insertion sort, Selection sort, Bubble sort, etc.

2. Integer Sorting (also known as Non- Comparison Sorting)
Integer sorts are called counting sorts. Integer sorts do not perform comparisons, so they are not limited by $\Omega(nlogn)$. Integer sorts determine for each element *x*, how many elements are less than *x*. If there are 14 elements, that are less than *x*, then *x* will be placed in the 15th slot. This knowledge is used to place each element into the correct opening immediately—no need to rearrange the lists.

In non-comparison based sorting, elements of an array are not compared with each other to find the sorted array.

- **Radix sort –**
  Best, average and worst-case time complexity: nk
  where k is the maximum number of digits in elements of an array.
- **Count sort –**
  Best, average and worst-case time complexity: n+k
  where k is the size of the count array.
- **Bucket sort –**
  Best and average time complexity: n+k
  where k is the number of buckets.
  Worst-case time complexity: n^2
  if all elements belong to the same bucket. [7]

II. Overviews and analysis of some well-known sorting techniques

Bubble Sort

This sort is a comparison based sort. Here sorting is progressed by comparing a data of the array with the next data of the array. The algorithm works until (n-1) pass where n is the number of elements Complexity of this sort $O(n^2)$ in the worst and average case. It is a simple and less complex sorting technique.

Selection sort

This sorting algorithm follows the method selection of the smallest. That means to select an element of the array and compare all the elements of the array with it. The process is working until n passes where n is the number of elements [2].

The complexity of this sorting is O(n2). But this sort is not efficient for a large array.

Insertion sort

This sort is a compared based sort. The elements of the array are compared with each element. After comparing the element takes its position in some particular order. This procedure follows the game of cards. It works by inserting an element at a particular position.

III. Working procedure and algorithm of least first sort

PROCEDURE:

The basic process of the working of the least first sort is given as follows:

(a) In Pass 1, A[0] and A[N-1] are compared, then A[0] is compared with A[N-2], A[0] is compared with A[N-3], and so on. Finally, A[0] is compared with A[1]. Pass 1 involves n–1 comparisons and after comparing and swapping the smallest element takes place at the A[0].

(b) In Pass 2, A[1] and A[N-1] are compared, then A[1] is compared with A[N-2], A[1] is compared with A[N-3], and so on. Finally, A[1] is compared with A[2]. Pass 2 involves n–2 comparisons and after comparing and swapping the next smallest element takes place at the A[1].

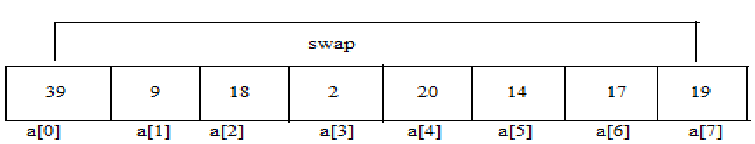
(c) In Pass 3, A[2] and A[N-1] are compared, then A[2] is compared with A[N-2], A[0] is compared with A[N-3], and so on. Finally, A[2] is compared with A[3]. Pass 3 involves n–3 comparisons and after comparing and swapping the 3rd smallest element takes place at the A[2].

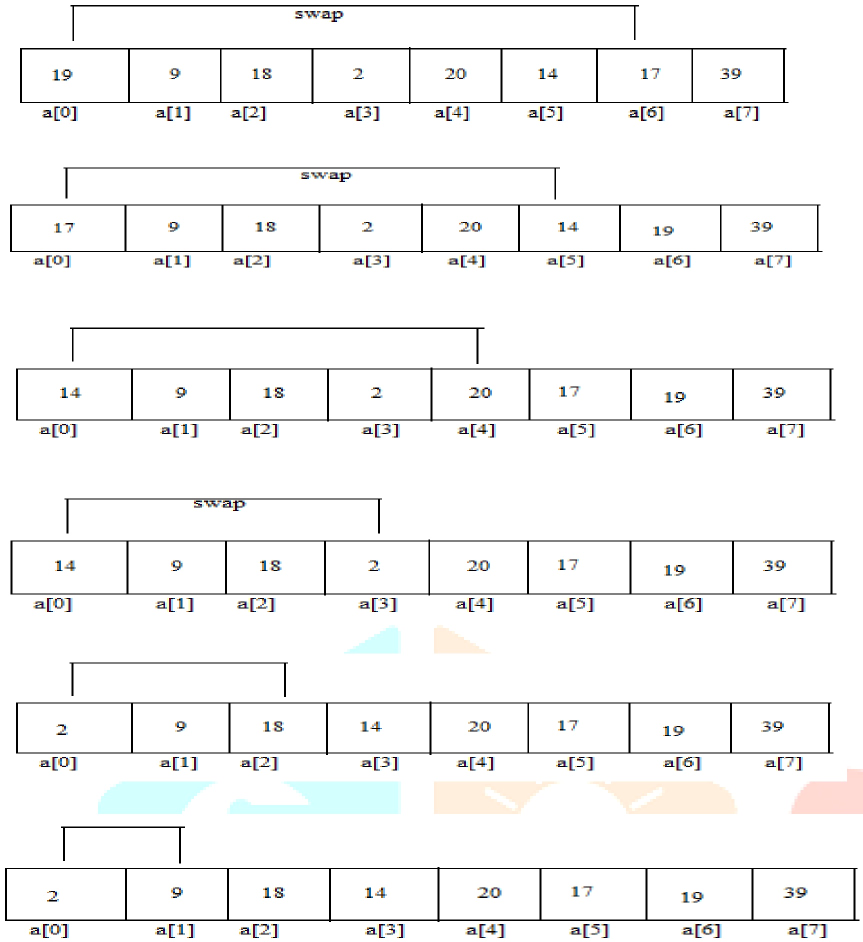
(d) In Pass n–1, A[N-1] and A[N] are compared so that A[N-1]<A[N]. After this step, all the elements of the array is arranged in ascending order.

EXAMPLE :

39	9	18	2	20	14	17	19
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]

PASS





Similar passes for other elements.

Algorithm

STEP 1: Initialize

STEP 2: Repeat for l=0 to n

STEP 3: Repeat for g=(n-1) to l

STEP 4: if(a[l] >a[g])

swap a[l] and a [g]

End if

End loop

STEP 5: End loop

STEP 9: END

IV. Case study of least first sort

In this section we are analysing the performance of LEAST FIRST SORTING and comparing the performance with other well-known sorting techniques.

(i) Worst Case:

When data is sorted either in ascending or descending order then best case will occur. In this case only comparison is done. No swapping is required.

For n elements,

- in the first pass (n-1) comparison required
- in the second pass (n-2) comparison needed

So, total comparison needed=T(n)

=(n-1)+(n-2)+..........+(n-(n-1))

$=O(n^2)$

(ii) Average Case:

Performance of sorting when evaluated in average case then random data need to be considered.

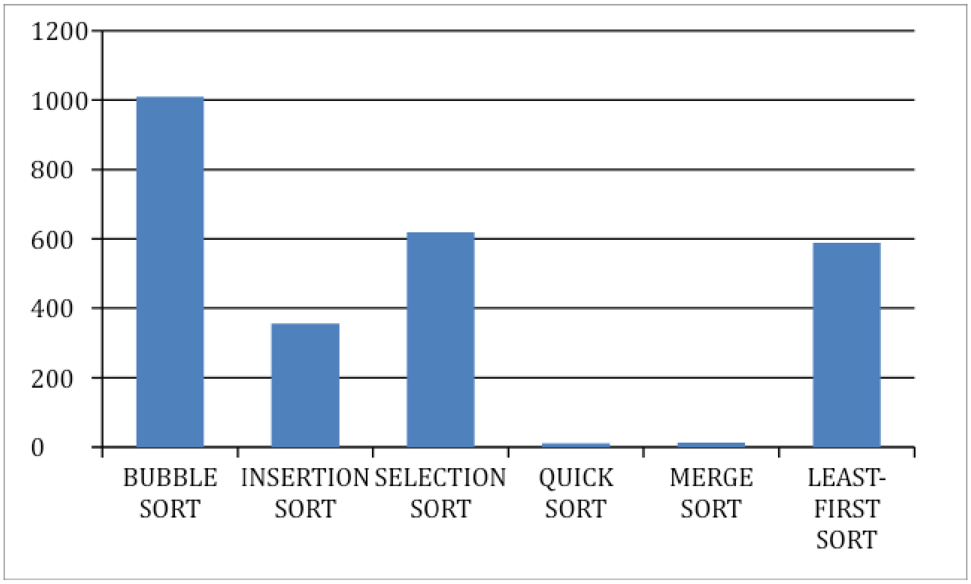
In random case, for 10000 elements time required is 187. 721ms.

Total comparison required is $O(n^2)$ in this case.

V. Comparison with well-known Sorting Algorithms

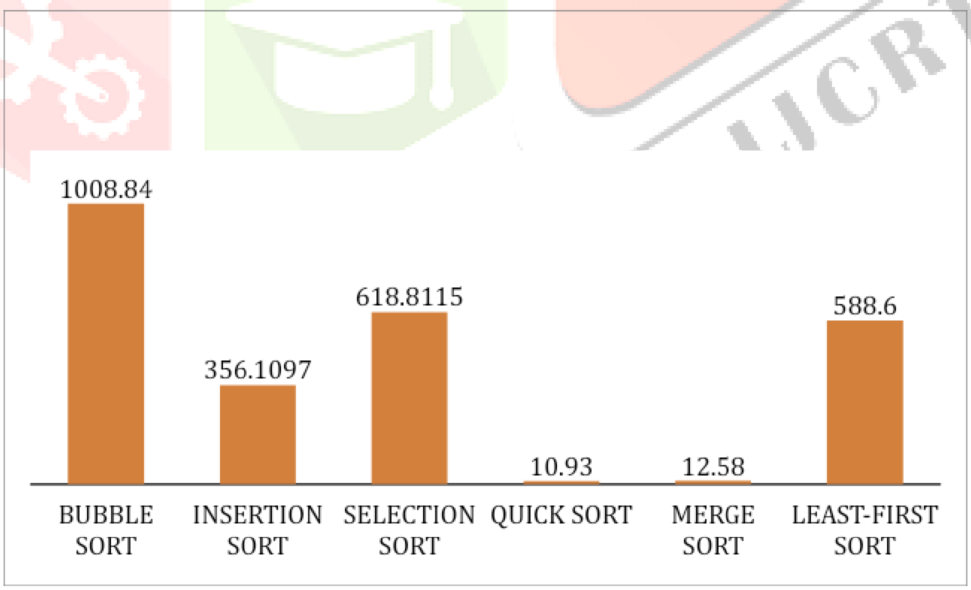
FOR 10000 elements

NAME OF SORTING	TIME TAKEN BY SORTING TECHNIQUES(in ms)						
	1	2	3	4	5	6	AVERAGE
BUBBLE	300.932	343.711	237.215	237.784	323.579	239.213	244.406
SELECTION	249.193	208.314	242.128	203.832	164.751	263.339	221.926
INSERTION	164.330	154.089	108.882	73.659	167.035	145.056	135.5085
QUICK	5.056	4.659	3.717	5.066	5.058	5.061	4.7695
MERGE	4.714	4.715	6.508	5.129	6.439	6.516	5.670
LEAST-FIRST	184.237	221.219	207.751	178.859	201.951	132.309	187.721



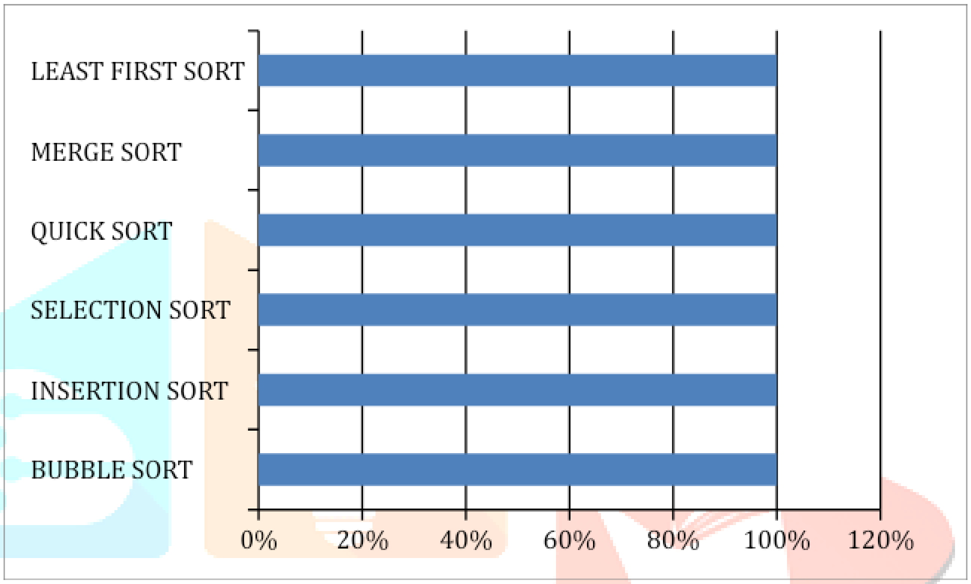
FOR 20000 ELEMENTS

NAME OF SORTING	TIME TAKEN BY SORTING TECHNIQUES(in ms)						
	1	2	3	4	5	6	AVERAGE
BUBBLE	1051.443	964.823	971.687	1064.721	1038.167	962.219	1008.84
SELECTION	661.053	616.351	621.842	561.889	633.003	618.731	356.1097
INSERTION	335.365	367.161	347.608	320.101	334.882	401.541	618.8115
QUICK	11.171	10.614	11.030	11.013	10.700	11.025	10.93
MERGE	13.516	12.866	13.605	13.441	8.278	13.781	12.58
LEAST-FIRST	606.946	510.439	608.805	558.589	622.907	616.927	588.6



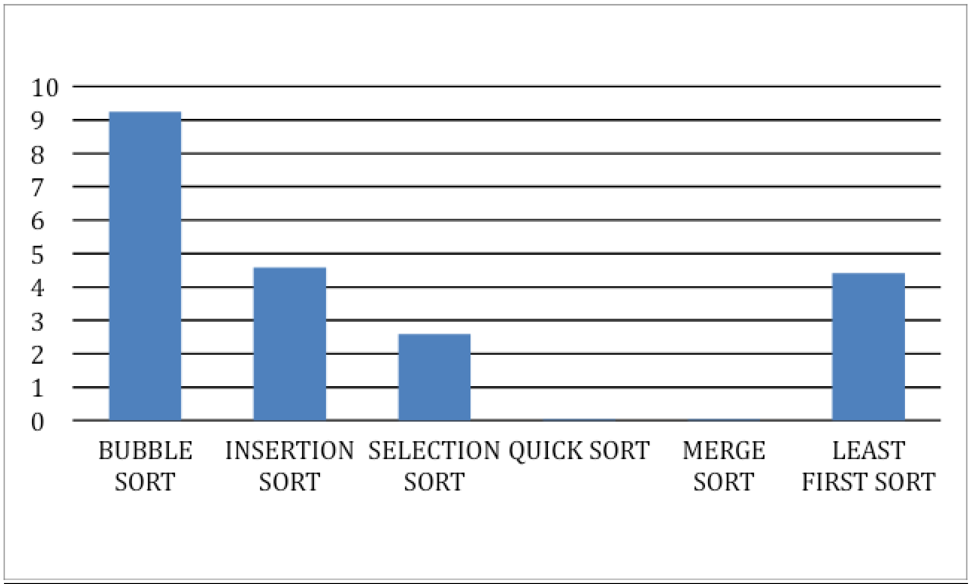
FOR 30000 ELEMENTS

NAME OF SORTING	TIME TAKEN BY SORTING TECHNIQUES(in ms)						
	1	2	3	4	5	6	AVERAGE
BUBBLE	2244.853	2207.052	2376.420	2207.364	2214.017	2313.600	2260.551
SELECTION	1276.382	1262.456	1317.8480	1270.115	1287.416	1286.485	1283.45
INSERTION	630.764	719.781	751.354	710.108	704.126	706.747	703.81
QUICK	12.181	13.525	13.568	15.196	16.482	13.551	14.084
MERGE	18.179	8.980	11.212	20.153	20.018	18.832	16.229
LEAST-FIRST	1114.122	1137.429	1157.431	1125.803	1114.427	1217.150	1144.394



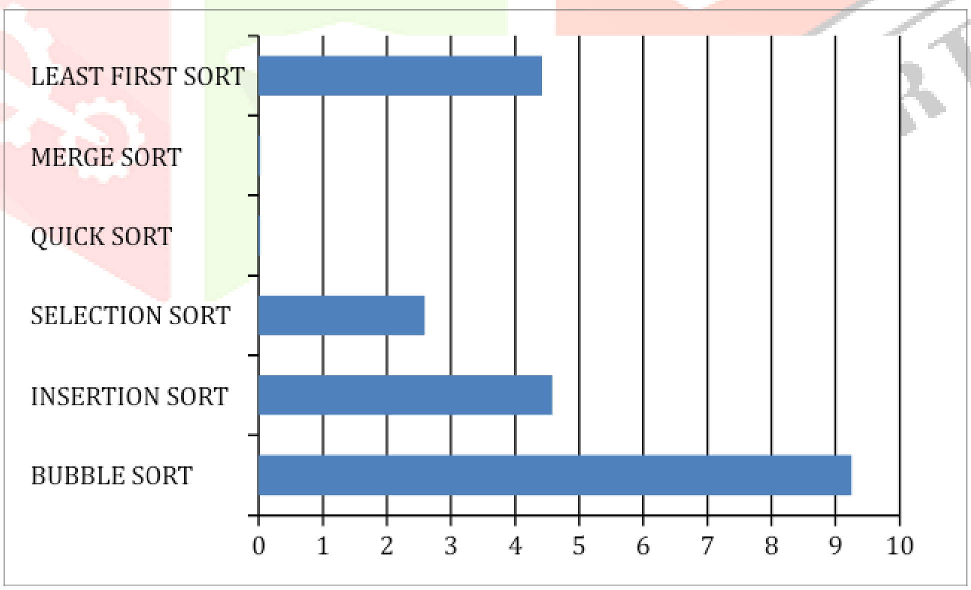
FOR 40000 ELEMENTS

NAME OF SORTING	TIME TAKEN BY SORTING TECHNIQUES(in ms)						
	1	2	3	4	5	6	AVERAGE
BUBBLE	4068.297	4110.165	4010.122	4084.865	4004.996	4008.773	4047.8697
SELECTION	2124.927	2123.295	2067.118	2126.148	2125.300	2180.036	2124.470
INSERTION	1197.157	1206.384	1118.178	1183.461	1204.298	1200.976	1185.0757
QUICK	11.717	19.717	19.175	21.824	15.771	18.933	17.856
MERGE	25.633	16.279	13.816	23.541	25.901	24.675	21.640
LEAST-FIRST	2007.400	2066.947	2075.914	1998.457	1958.633	1962.924	2011.7125



FOR 50000 ELEMENTS

NAME OF SORTING	TIME TAKEN BY SORTING TECHNIQUES(in ms)						
	1	2	3	4	5	6	AVERAGE
BUBBLE	6331.342	6426.341	6363.233	6335.358	6384.409	6384.030	6370.7855
SELECTION	3275.189	3271.414	3188.651	3187.897	3275.119	3271.018	3244.88
INSERTION	1850.325	1779.815	1845.694	1827.848	1798.386	1855.709	1826.296
QUICK	23.530	22.617	23.545	22.715	23.319	19.381	22.5178
MERGE	32.461	29.421	22.488	31.270	22.509	29.412	27.9268
LEAST-FIRST	3054.896	3151.587	3067.161	3137.647	3108.880	3090.979	3101.8583



VI. Conclusion

In this paper comparison between our newly proposed algorithms least first sort and other sorting techniques have been done. In this sorting, the time taken for a different number of an element is compared with another sorting method. It is found that the running time of least first sort is lower than bubble sort and selection sort. But higher than quick sort and merge sort. In future, we will try to reduce complexity by introducing a divide and conquer strategy.

VII. Reference

[1]International Journal of Engineering and Advanced Technology (IJEAT), ISSN: 2249 – 8958, Volume-3, Issue-1, October 2013

[2]International Journal of Innovative Technology and Exploring Engineering (IJITEE), ISSN: 2278-3075, Volume-8 Issue-12, October 2

[3]Communications of ACM SIGPALN, Vol.31, No.3, March, 1996, ACM, pp.28-36

[4]S. Jadoon , S.Solehria, S.Rehman and H.Jan.(2011,FEB). " Design and Analysis of Optimized Selection Sort Algorithm".11. (1),pp. 16-21.

Available: http://www.ijens.org/IJECS%20Vol%2011%20Issue%2001.html.

[5] C.A.R. Hoare, "Algorithm 64: Quicksort," Communications of the ACM, Vol. 4 , pp. 321, 1961.

[6] J.W.J. Williams. "Algorithm 232: Heapsort," Communication of the ACM, No.7, pp.347-348, 1964.

[7] Available at: https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/

[8] Cormen, T.H., Leiserson, C.E., & Rivest, R.L. Introduction to Algorithms (2nd ed.). Prentice Hall of India private limited, New Delhi-110001 (2001).