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 $O(n\log n)$, implying that, on average, comparison sorts cannot be faster than $O(n\log n)$. 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 $O(n)$, $O(n\log n)$, $O(n^2)$. That means $O(n)$ is the best for complexity because it gives the lowest execution time then $O(n\log n)$ 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 $O(n\log n)$. 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(n^2). 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].


(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 :

<table>
<thead>
<tr>
<th>39</th>
<th>9</th>
<th>18</th>
<th>2</th>
<th>20</th>
<th>14</th>
<th>17</th>
<th>19</th>
</tr>
</thead>
</table>
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)

\[ T(n) = (n-1) + (n-2) + \ldots + (n-(n-1)) \]

\[ T(n) = O(n^2) \]

(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

<table>
<thead>
<tr>
<th>NAME OF SORTING</th>
<th>TIME TAKEN BY SORTING TECHNIQUES (in ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>BUBBLE</td>
<td>300.932</td>
</tr>
<tr>
<td>SELECTION</td>
<td>249.193</td>
</tr>
<tr>
<td>INSERTION</td>
<td>164.330</td>
</tr>
<tr>
<td>QUICK</td>
<td>5.056</td>
</tr>
<tr>
<td>LEAST-FIRST</td>
<td>184.237</td>
</tr>
</tbody>
</table>
### FOR 20000 ELEMENTS

<table>
<thead>
<tr>
<th>NAME OF SORTING</th>
<th>TIME TAKEN BY SORTING TECHNIQUES (in ms)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUBBLE</td>
<td></td>
<td>1051.443</td>
<td>964.823</td>
<td>971.687</td>
<td>1064.721</td>
<td>1038.167</td>
<td>962.219</td>
<td>1008.84</td>
</tr>
<tr>
<td>SELECTION</td>
<td></td>
<td>661.053</td>
<td>616.351</td>
<td>621.842</td>
<td>561.889</td>
<td>633.003</td>
<td>618.731</td>
<td>356.1097</td>
</tr>
<tr>
<td>INSERTION</td>
<td></td>
<td>335.365</td>
<td>367.161</td>
<td>347.608</td>
<td>320.101</td>
<td>334.882</td>
<td>401.541</td>
<td>618.8116</td>
</tr>
<tr>
<td>LEAST-FIRST</td>
<td></td>
<td>606.946</td>
<td>510.439</td>
<td>608.805</td>
<td>558.589</td>
<td>622.907</td>
<td>616.927</td>
<td>588.56</td>
</tr>
</tbody>
</table>

**Bar Diagram**

- **Bubble Sort**: 1008.84 ms
- **Insertion Sort**: 356.1097 ms
- **Selection Sort**: 618.8115 ms
- **Quick Sort**: 10.93 ms
- **Merge Sort**: 12.58 ms
- **Least-First Sort**: 588.6 ms
## FOR 30000 ELEMENTS

<table>
<thead>
<tr>
<th>NAME OF SORTING</th>
<th>TIME TAKEN BY SORTING TECHNIQUES (in ms)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUBBLE</td>
<td></td>
<td>2244.853</td>
<td>2207.052</td>
<td>2376.420</td>
<td>2207.364</td>
<td>2214.017</td>
<td>2313.600</td>
<td>2260.551</td>
</tr>
<tr>
<td>SELECTION</td>
<td></td>
<td>1276.382</td>
<td>1262.456</td>
<td>1317.8480</td>
<td>1270.115</td>
<td>1287.416</td>
<td>1286.485</td>
<td>1283.45</td>
</tr>
<tr>
<td>INSERTION</td>
<td></td>
<td>630.764</td>
<td>719.781</td>
<td>751.354</td>
<td>710.108</td>
<td>704.126</td>
<td>706.747</td>
<td>703.81</td>
</tr>
<tr>
<td>LEAST-FIRST</td>
<td></td>
<td>1114.122</td>
<td>1137.429</td>
<td>1157.431</td>
<td>1125.803</td>
<td>1114.427</td>
<td>1217.150</td>
<td>1144.394</td>
</tr>
</tbody>
</table>

## FOR 40000 ELEMENTS

<table>
<thead>
<tr>
<th>NAME OF SORTING</th>
<th>TIME TAKEN BY SORTING TECHNIQUES (in ms)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUBBLE</td>
<td></td>
<td>4068.297</td>
<td>4110.165</td>
<td>4010.122</td>
<td>4084.865</td>
<td>4004.996</td>
<td>4008.773</td>
<td>4047.8697</td>
</tr>
<tr>
<td>SELECTION</td>
<td></td>
<td>2124.927</td>
<td>2123.295</td>
<td>2067.118</td>
<td>2126.148</td>
<td>2125.300</td>
<td>2180.036</td>
<td>2124.470</td>
</tr>
<tr>
<td>INSERTION</td>
<td></td>
<td>1197.157</td>
<td>1206.384</td>
<td>1118.178</td>
<td>1183.461</td>
<td>1204.298</td>
<td>1200.976</td>
<td>1185.0757</td>
</tr>
</tbody>
</table>
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


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