kvrabhishek

by Rani Clement king

Submission date: 22-Jul-2022 03:50AM (UTC-0500)
Submission ID: 1867201200
File name: KVR_Abhishek_Paper.docx (850.58K)
Word count: 1721
Character count: 9239

Implementation and Analysis of Double Linked List Sorting

Abhishek S
UG Student,
Department of AI and ML,
RNSIT, Bengaluru, India
abhishekiyer.2002@gmail.com

Dr. Rama Satish K V
Associate Professor,
Department of AI and ML,
RNSIT, Bengaluru, India
ramasatishkv@gmail.com

***Abstract*— This paper works on sorting the given set of elements effectively in both Sort-ascending and Sort-descending order. There are numerous methods/algorithms for sorting a given group of random integers wherein each of these algorithms take unique time to execute the sorting operation. Basic idea of time complexity was highly considered during the design of this algorithm and we make use of doubly linked list in achieving this.

Keywords — Doubly linked list, Sorting, Array.

I. INTRODUCTION

Sorting can be defined as the process of systematic arrangement of elements.

Analysis of any algorithm is done by two parameters: Space complexity and time complexity. Space complexity is the memory space expected for the general execution of the program and time complexity can be defined as the total time taken depending on various factors such as processor speed, compiler used, etc. If the current advancements are kept in mind, space complexity can be ignored and hence time complexity becomes the only deciding parameter.

$O(n \log n)$, $O(n)$, $O(n^2)$ are the general time complexities which are majorly seen among the different sorting algorithms after considering their best, average and worst cases.

We have made use of doubly linked list which contains two links associated with each of the node, that is the left link (llink) and the right link (rlink). If the sorting is done in ascending order, we can easily convert the same into descending order by swapping the front and last pointers.

II. LITERATURE SURVEY

Some of the algorithms for sorting are:

❖ *Insertion Sort:*

This method works well with tiny and almost sorted values and is frequently used as part of more complicated algorithms. This algorithm pulls factors from the array one by one and inserts them into a new sorted list using their prescribed function. Insertion sort has O(n) time complexity.

❖ *Selection Sort:*

This sort has $O(n^2)$ complexity, it is inefficient on large arrays and frequently works worse than insertion sort algorithm. In favourable cases, the selection type has an advantage over more sophisticated algorithms. This method distinguishes the smallest factor, trades it with the worth in the exhibit's underlying number, and rehashes until the end of the rundown. It requires n swaps and is generally used in situations where swapping is prohibitively expensive.

❖ *Quick Sort:*

It is basically a divide-and-conquer strategy. An element known as a pivot is chosen when an array is partitioned into nodes. All factors lower than the pivot are shifted before it, while all factors higher than it are shifted after it. The time it takes to accomplish this procedure is linear. The lesser and increased sub arrays are then sorted recursively. This results in $O(n \log n)$ time complexity.

❖ *Bubble Sort:*

It is the most basic method of sorting an array, but it is also the slowest. The principal thought behind this kind is to thoroughly analyze two nearby values, as well as to switch them in the event that they are in some unacceptable succession. The algorithm has O(n) behavior in the good case condition, where the listing is already sorted. Bubble sort begins at the very commencement of the array. If the first element is greater than the second, it compares the two and swaps them. This is done recursively for every pair of adjoining factors up to the array's end. Bubble sort has time complexity of $O(n^2)$ for its average and worst cases, so it is rarely used to sort gigantic arrays.

❖ *Merge Sort:*

The array is divided into n subarrays, each with one component, using this sorting algorithm (an array of 1 component is regarded to be sorted). It then combines subarrays one by one to build new sorted subarrays until only one is left. This collection has now been sorted. Merge sort has a complexity of O (n log n).

❖ *Radix sort:*

The given set of elements are sorted one after the other based on its least and most significant bit. This sorting technique is also called as bucket sort as 10 buckets with values 0 to 9 are used. This algorithm is not effective for different range of elements.

III. IMPLEMENTATION CODE/ALGORITHM

Program is implemented in the given link:

https://onlinegdb.com/-fVxnqAtF

Code snippet:

```
if(first==NULL)
    {
      first=new;
       last=first;
    }
    else if(first->rlink==NULL)
    {
       if(new->data <= first->data)
        {   new->rlink=first;
            first->llink=new;
            first=new;
        }
        else
        {   first->rlink=new;
            new->llink=first;
            last=new;
        }
    }
    else
    {
        temp=first;
        while(temp!=NULL)
        {
            if(new->data <= temp->data)
            {  if(temp==first)
                {
                  new->rlink=temp;
                  temp->llink=new;
                 first=new;
                 return;
                }
                else
                {
                 new->rlink=temp;
                 temp->llink->rlink=new;
                 new->llink=temp;
                 temp->llink=new;
                 return;
                }
            }
            else if(temp->rlink==NULL)
            {
                temp->rlink=new;
                new->llink=temp;
                return;
            }
            else if(new->data >= temp->data)
            {
                if(new->data <= temp->rlink->data)
                {
                    new->rlink=temp->rlink;
                    temp->rlink=new;
                    temp->rlink->llink=new;
                    new->llink=temp;
                    return;
                }
                else if(new->data > temp->rlink->data)
                {
                    temp=temp->rlink;
                }
            }
        }
    }
```

Working of proposed method:

The elements to be sorted are initially stored in an array. Later, each element one after the other are inserted in a doubly linked list. The initial element acts as the first node when insertion operation takes place for the first array element. Later, successive elements of array are considered one after the other and insertion of each of the element takes place in the doubly linked list.

Comparison of element takes place and then the insertion of the same happens when its correct place is identified. This algorithm even works when identical inputs are provided. It is also noted when the null value is encountered and the first and last pointers are updated respectively.

Analysis of proposed method:

This algorithm has O(n) time complexity and is very flexible with huge input set of elements.
The elements are entered in an array as shown below:

0	1	2	3	4
40	**20**	**10**	**50**	**30**

Figure 1- Initial Array Elements

Step 1- The first element of array becomes the first node:

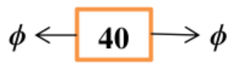

Figure 2- Formation of initial node

Step 2- The second element of array is then inserted:

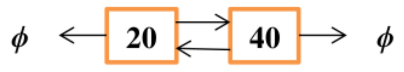

Figure 3- Doubly linked list after insertion

Step 3- Array element in 2nd index number will be inserted:

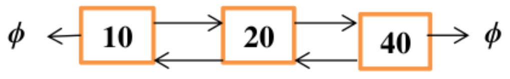

Figure 4- Doubly linked list after insertion

Step 4- Array element in the 3rd index number will be inserted:

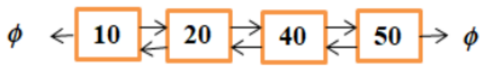

Figure 5- Doubly linked list after insertion

Step 5- The final array element is considered:

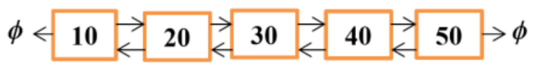

Figure 6- Final sorted output

IV. Results Discussion

The proposed method has been examined for the arrays of following variety of factors 5000, 10000, 20000 and 30000. The evaluation with different sorting algorithms is as follows.

1. The examination between the proposed technique and Bubble sort comprising of various number of components

TABLE I: BUBBLE SORT VS. PROPOSED METHOD

# of values	Bubble Sort	Proposed method
5000	0.54945	0.060513
10000	1.64835	0.193082
20000	3.29670	0.860823
30000	4.94505	2.922519

Graph for the above comparison is as demonstrated-

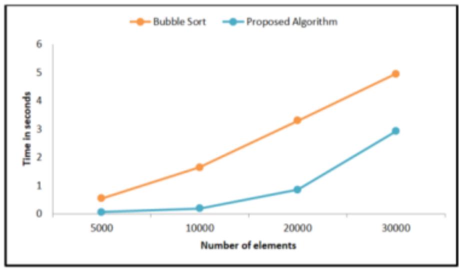

Figure 7- Graph between proposed method and bubble sort.

In the Figure 7, blue line i.e. the proposed method consumes less time compared to existing bubble sort i.e. orange line.

2. The examination between the proposed technique and selection sort comprising of various number of components.

TABLE II: SELECTION SORT VS. PROPOSED METHOD

# of values	Selection Sort	Proposed method
5000	0.54945	0.060513
10000	1.74635	0.193082
20000	3.09570	0.860823
30000	4.94505	2.922519

Graph for the above comparison is as demonstrated-

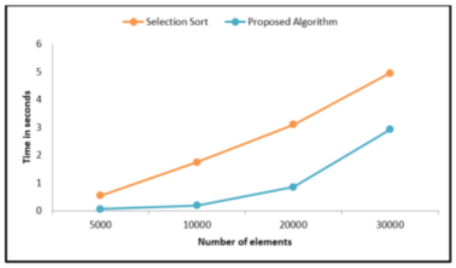

Figure 8- Graph between proposed method and selection sort.

In the above graph, blue line depicts the proposed method and theorange line depicts the selection sort.

3. The examination between the proposed technique and Merge sort comprising of various number of components.

TABLE III- MERGE SORT VS. PROPOSED ALGORITHM

# of values	Merge Sort	Proposed method
5000	0.54945	0.060513
10000	1.64835	0.193082
20000	2.74725	0.860823
30000	3.84615	2.922519

Graph for the above comparison is as demonstrated-

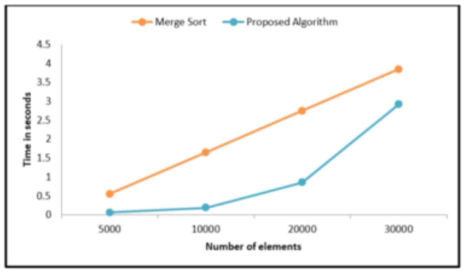

Figure 9- Graph between proposed method and merge sort.

In the above graph, blue line depicts the proposed method and theorange line depicts the merge-sort.

4. The examination between the proposed technique and Insertion sort comprising of various number of components.

TABLE IV-INSERTION SORT VS. PROPOSED ALGORITHM

# of values	Insertion sort	Proposed method
5000	0.54945	0.060513
10000	1.09890	0.193082
20000	2.74725	0.860823
30000	4.39560	2.922519

Graph for the above comparison is as demonstrated-

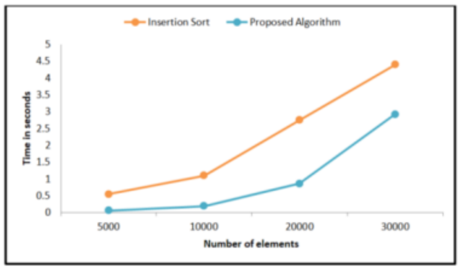

Figure 10- Graph between proposed method and insertion sort

In the Figure 10, proposed algorithm (blue line) consumes less time compared to insert-sort method (orange line).

5. The examination between the proposed technique and Array-Indexed sort comprising of various number of components.

TABLE V – ARRAY-INDEXED SORT VS. PROPOSED METHOD

# of values	Array-Indexed sort	Proposed method
5000	2.74725	0.060513
10000	4.94505	0.193082
20000	9.34066	0.860823
30000	15.38462	2.922519

Graph for the above comparison is as demonstrated-

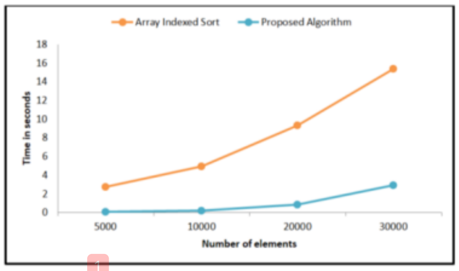

Figure 11- Graph between proposed method and array indexed sort.

In the above graph, blue line depicts the proposed method and theorange line depicts the array-indexed sort.

REFERENCES

[1] H. C Thomas, E. L Charles, L. R Ronald, and S. Clitlord, "Introduction to Algorithms", Second Edition MIT Press and 609 McGraw- Hill, ISBN 0-262-03293-7. Section 1.1 Algorithms, pp.5, 2001

[2] K. Donald, "The Art of Computer Programming Sorting and Searching", Third Edition. Addison-Wesley, ISBN 0-201- 89685- O.Section 5.2.2 Sorting by Exchanging, pp 106-110, 1997.

[3] L. Seymour, "Theory and Problems of Data Structures", Schaum's Outline Series: International Edition, McGraw-Hill. ISBN 0-07-099130-8. Section 9.4: Quick Sort. pp. 324-325, 1986.

[4] Insertion sort, http://en.wikipedia.org/wikiIInsertion_sort, 2011.

[5] Indexed Sorting Algorithm for natural numbers", IEEE, pp. 606- 609,2011.

[6] Insertion_sort,https://en.wikipedia.org/wiki/Insertion_S ort

[7] Merge_Sort, https://en.wikipedia."org/wiki/Merge_sort

[8] Bubble_Sort https://en.wikipedia.org/wiki/Bubble_sort

[9] Rama Satish K V, Manoj M Kaushik, "A Holistic Method of Linked List sorting", Proceedings of NCCT, RNSIT, Bengaluru, 2021.

ACKNOWELDGEMENT

This research was supported by RNS Institute of Technology, Bangalore, India. We are so grateful to Dr. M. K. Venkatesha, Principal at RNS Institute of Technology for his comments on an earlier version of the manuscript, although any errors are our ownand should not reflect negatively on these illustrious individuals' reputations.

We like to thank Dr. Rama Satish K. V from the bottom of our hearts for assistance with Application Development, and for comments that greatly improved the manuscript.

We also want to express our gratitude to Dr. Harsha for sharing his knowledge with us during the course of this research, and we thank our family members for their immense support.

We thank our friends from various departments who provided insight and expertise that greatly assisted the research. We are also appreciative for the feedback we have received on this research topic from our readers.

AUTHORS PROFILE:

Abhishek S is currently pursuing Bachelor of Engineering (B.E.) in Artificial Intelligence and Machine Learning (AI-ML) from RNSIT. His area of interest includes Data Structures and algorithms, Artificial Intelligence and Machine Learning.

Dr. Rama Satish is an associate professor at Department of AIML, RNSIT. He holds M.Tech and Ph.D. from VTU. His Area of interest includes Web Services, Cloud Computing, Artificial Intelligence, Machine Learning, Data Science and Big Data Processing.

kvrabhishek

ORIGINALITY REPORT

11%	4%	8%	0%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Shruti Rishabh Pandey. "A HEURISTIC APPROACH OF SORTING USING LINKED LIST", 2018 Second International Conference on Computing Methodologies and Communication (ICCMC), 2018 Publication	6%
2	www.irjmets.com Internet Source	2%
3	Rama Satish K V. "Deep Clustering Assisted Authentication and Data Sharing Approach in Mapreduce Enabled Environment", 2021 International Conference on Circuits, Controls and Communications (CCUBE), 2021 Publication	1%
4	www.skymem.com Internet Source	1%
5	www.cscjournals.org Internet Source	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		