SQUARE ROOT SORTING ALGORITHM

[1]Mir Omranudin Abhar, [2]Nisha Gatuam

[1]M.Tech Student, [2]Assistant Professor

[1,2]Department of Computer Science & Engineering

[1,2]AP Goyal Shimla University, Shimla, India

Abstract: In this paper, we are going to introduce the Square root sorting algorithm. We study the best case and worst case of Square root sorting algorithm, and we compare this algorithm with some of the algorithms that are already existed.
A Sorting Algorithm is used to rearrange a given array elements according to a comparison operator on the elements. So far there are many algorithms that have been used for sorting like: Bubble sort, insertion sort, selection sort, quick sort, merge sort, heap sort etc. Each of these algorithms are used according to the list of elements of specific usage and they have specific space complexity and time complexity as well. The Square root sorting algorithm has the least time complexity comparing to some of the existing algorithms, especially in case of the best case, and in the worst case it also has less time complexity than some of the existing algorithms, which we will discuss it in coming pages of this paper.

***Keyword*- Algorithm, Time Complexity, Space Complexity, Sorting.**

I. INTRODUCTION

Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently [5]. Sorting is one of the most important and valuable parts in the data structure, Sorting is nothing but arrangement of a set of data in some order or a process that rearranges the records in a file into a sequence that is sorted on some key.
Different methods are used to sort the data in ascending or descending orders. The different sorting methods can be divided into two categories:

- Internal sorting(sorting of data items in the main memory)
- External sorting (when the data to be sorted is so large that some of the data is present in the memory and some is kept in auxiliary memory)

Arrays (sets of items of the same type) are stored in the fast, high-speed random access internal storage on computers, whereas files are stored on the slower but more spacious External store [2].
The Square root sorting algorithm is the type of sorting of internal algorithms that deals with the sorting of data items in the main memory, which, in particular, has the best result compared to previous algorithms. We combine this algorithm with one of the best-performing algorithms in the best case and worst case scenarios, we use the merge sort or quick sort algorithm, because these two sorting algorithms are among the best of available algorithms. We will discuss about it later.

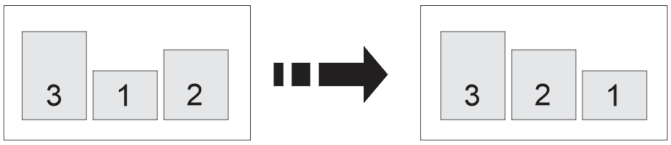

Figure.1. Sorting.

II. PROBLEM FORMULATION

The goal of designing the SRS (Square Root Sorting) algorithm is to reduce the time complexity of the memory.
We use one of the best sorting algorithms in combining this scheme of sorting from data items, which if we find better algorithms in the future we can give it a place. But the current algorithm doesn't have any problem by using merge sort.

This is how our algorithm works:

First, we divide the list of data using square root operation. We will have equal numbers of elements in each part except last part, it might contain less elements than other parts.
The second step is to sort each part separately using one of the available algorithms, which has the least time complexity.
The third step is to start comparing data items and swapping.
The comparison method is such that we compare the first element of the first part with the first element of the other sections,
If the first element of the other part is larger, we will not perform the comparison with other elements of that section. It is because if the first element of one of the segments is larger, it states that other elements of this section are also larger than the first element of the first part.
We can find the smallest element in the first swapping operation and put it at first index of the array.
Similarly, we start the comparison with the second element of the first part. We still compare it with only the first elements of the other segments. If one of the first elements of the other part is smaller than the first element of the first part, we will perform the swap operation. This means that the second small element is in the second index of array.
Now, if the swap element of that section is larger than the next element, we will perform the sorting operation with the same algorithm we have selected twice over.
If others are in the sorted order, there is no need to be sorted again. Similarly, it is time to compare the third element of the first part. We do the same thing twice before the end.

III. ALGORITHMS

Step 1. $n \leftarrow length$ of List $A, r \leftarrow \sqrt{n}$,$s = 0$.
Step 2. Divide the list A in r part and every part have r item in that list.
Step 3. $for(p = 1\ to\ r, p++)$ part of list A individual sort, with the Merge sort or other Algorithms.
$merge(part.p)$
Step 4. The first item will compare with the first item of seconds part.

```
for(p = 1 to r, p++){
    for(i = 1 to r, i++){
        if(part.(p)[i] ≤ part.(p + 1)[s]){
            s++
        }else{
            A ← part.(i)[1]
            B ← part.(i + 1)[s]
            part.(i + 1)[s] ← A
            part.(p)[i] ← B
            merge(part.(i + 1))
        }
    }
    s = 0
}
```

Step 5. Return A

Complexity Analysis: The complexity of a sorting algorithm measures the running time as a function of the number of n items to be sorted. Each sorting algorithm S will be made up of the following operations, where $A_1, A_2 \ldots A_n$ contain the items to be sorted and B is an auxiliary location.

1. Comparisons which test whether $A_i < A_i$ or test whether$A_i < B$.
2. Interchanges which switch the contents of A_i and A_j *or* A_i *and* B.
3. Assignments which set $B = A_i$ and then set $A_j = B$ or $A_j = A_i$.

Generally the complexity function measures only the number of comparisons, since the number of other operations is at most a constant factor of the number of other operations is at most a constant factor of the number of comparisons. Suppose space is fixed for one algorithm then only run time will be considered for obtaining the complexity of algorithm [5,4].
Complexity analysis of the proposed SR Sorting Algorithm for best case and worst case is discussed here with the help of examples.

A. Best Case:
In this case the array is already sorted and the terminating conditions is obtained after traversing the array twice. In this case, the complexity of the SR algorithm is$O(n\ (\log\sqrt{n}))$.

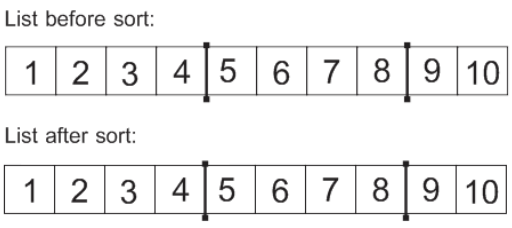

Figure. 2. Best Case SRS Algorithm.

B. Average Case:

In this case the array data items used the random number, In this case, the complexity of the SR algorithm is $O((n \log \sqrt{n}) + cn)$.

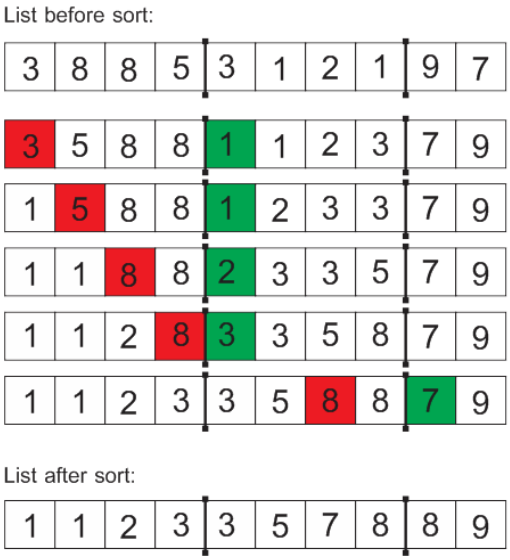

Figure. 3. Average Case SRS Algorithm.

C. Worst Case:

Reverse sorted array which involves every element to be moved to $(n + 1)^{th}$ position from its initial i^{th} position. In this case, the complexity of the SR algorithm is $O(n(\log \sqrt{n}) + cn)$.

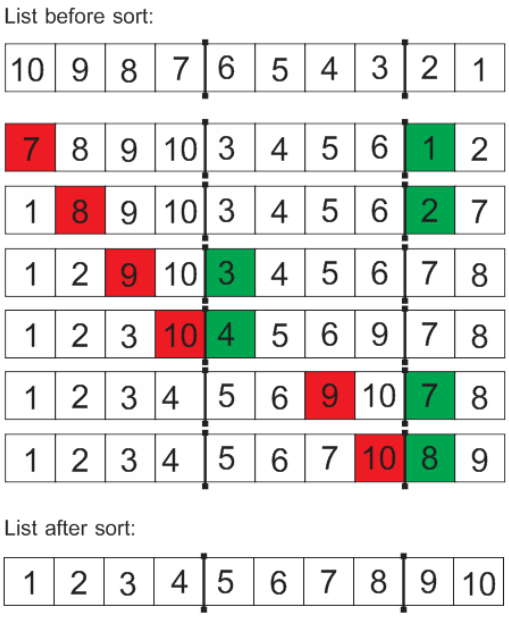

Figure. 4. Worst Case SRS Algorithm.

IV. COMPARE SRS ALGORITHM WITH OTHER SORTING ALGORITHM

Which sorting algorithm is the fastest? This question doesn't have an easy or unambiguous answer, however. The speed of sorting can depend quite heavily on the environment where the sorting is done, the type of items that are sorted and the distribution of these items.

For example, sorting a database which is so big that cannot fit into memory all at once is quite different from sorting an array of 100 integers. Not only will the implementation of the algorithm be quite different, naturally, but it may even be that the same algorithm which is fast in one case is slow in the other. Also sorting an array may be different from sorting a linked list, for example [1].

In order to better understand the advantages and disadvantages of an algorithm, there is a need for time that the algorithm should be adapted in different criteria's so that the results obtained can be examined in order to understand the advantages and disadvantages of the algorithm.

When we discuss the complexity of the SR algorithm, we have achieved the results after it was implemented in Java programming language which indicates that the algorithm performs a comparative and substitute comparison with previous algorithms.

This algorithm can still be called a technique that replaces any sorting algorithm that exists up to now or in the future in order to reduce the time complexity of the algorithm and when we compare merge sort with this algorithm, the results obtained from 100 to 102400 data items are presented in the following way:

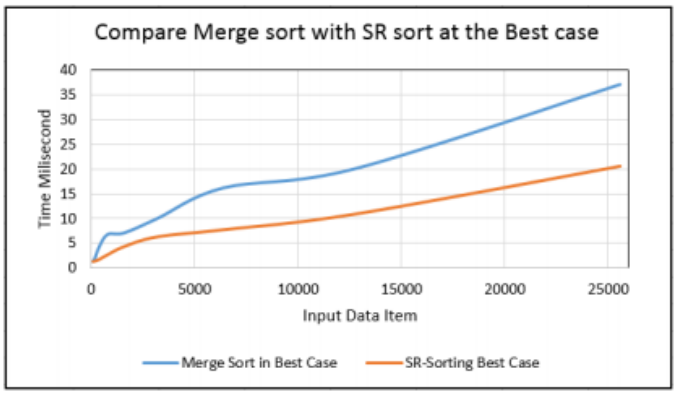

Figure. 5. Compare Merge Sort with SR-Sort at the best case.

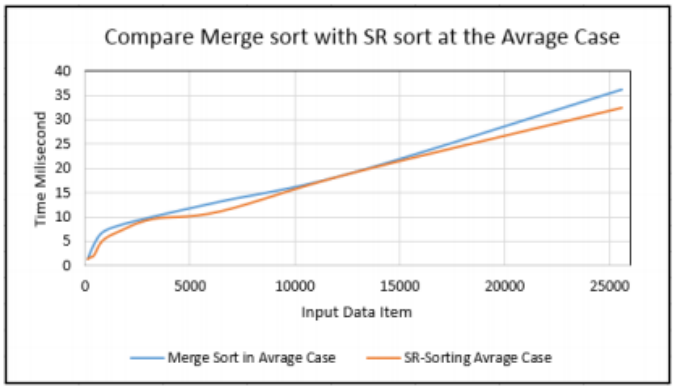

Figure. 6. Compare Merge Sort with SR-Sort at the average case

.

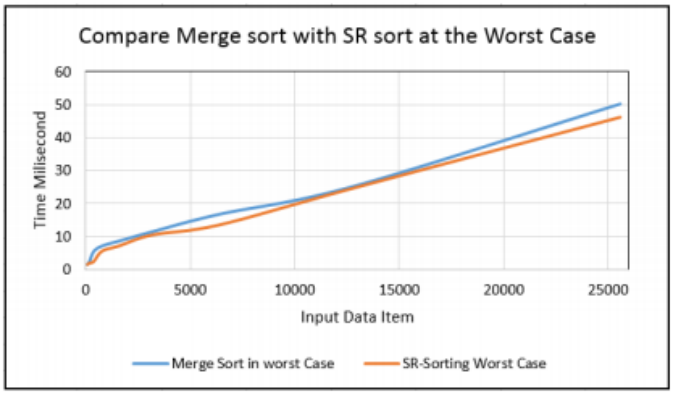

Figure. 7. Compare Merge Sort with SR-Sort at the best case

The following table illustrates the complexity of some sorting algorithms with SR sorting algorithms.

ALGORITHMS	BEST CASE	AVERAGE CASE	WORST CASE
SR Sort	$n\log(\sqrt{n})$	$n\log(\sqrt{n}) + cn$	$n\log(\sqrt{n}) + cn$
Merge Sort	$n\log(n)$	$n\log(n)$	$n\log(n)$
Bubble Sort	n	n^2	n^2
Insertion Sort	n	n^2	n^2

Selection Sort	n^2	n^2	n^2

V. CONCLUSION

In this research paper, we discussed the SR algorithm. The performance of the algorithm and comparison with other existing algorithms in terms of time and space complexity. This algorithm is one of the best algorithms with less time and space complexity, especially in best case and better performance in worst case than some other existing algorithms.
In case of time complexity, the SR algorithm has better performance than Insertion Sort, Bubble Sort and Selection Sort. And in case of best case, it performs better than merge sort although merge sort performs better in worst case.

VI. ACKNOWLEDGEMENTS

First of all, I would like to thank the most merciful and the most gracious Allah who teaches and shows us the way of happiness of this and next word by the agency of his messenger the owner of honor Mohammad (peace be open him) and give command for learning the knowledge. I am grateful from the all lectures especially my supervisor Asst.Prof.Ms NISHA GAUTAM for his valuable contributions, patience and guidance through this study. I am grateful to my parents for their hidden help in terms of prayers and faith in me, which actually empowered me to fulfill this task. And last, I am grateful to all my entourage, classmates and companions who helped me on the journey of university career and made it easier. I would also like to thank my entourage, classmates and companions especially from my partner Eng. Ehsan Bayat who helped me on the journey of university career and made it easier.

And last but not least, I am grateful to my parents for their hidden help in terms of prayers and faith in me, which actually empowered me to fulfill this task.

REFERENCES

[1] Comparison of several sorting algorithms. http: //warp.povusers.org/SortComparison/ .Accessed: 2018-11-13.
[2] Tata McGraw-Hill Education. Data Structures and Algorithms Using C. 2010.
[3] Harish Chugh Gaurav Kumar. \Empirical Study Of Complexity Graphs for Sorting Algorithms". In :(2013).
[4] seymour Lipschutz. “Data Structures With C". In: 2013 .
[5] R.Mariselvi G.Santhana Devi C.Rajalakshmi C.Durai V.P.Kulalviamozhi M.Muthulakshmi. Performance Analysis of Sorting Algorithm , IEEE Journals . 2015.
[6] Indradeep hayran and Pritee Khanna. Couple Sort, PDGC , 2016.
[7] Hoda Osama ,Yasser Omar and Amr Badr. Mapping Sorting Alogirhtm , IEEE, July 2016.
[8] Chen Hongyan, Wan Junwei, Lu Xianli , Research and implementation of database high performance sorting algorithm with big data, IEEE,2017
[9] Wei Wang, Big Data, Big Challenges , IEEE , 2014