P. Dhivakar *et al*, International Journal of Computer Science and Mobile Applications,
Vol.1 Issue. 6, December- 2013, pg. 1-10 **ISSN: 2321-8363**

Dual Sorting Algorithm Based on Quick Sort

[1]P. Dhivakar, [2]G. Jayaprakash

[1]PG Student, [2]PG Student, Department of CSE
M. Kumarasamy College of Engineering (Autonomous), Karur, TamilNadu, India
dhivakarit92@gmail.com
bloomingwithjp@gmail.com

Abstract--- In growing computer science world, a sorting algorithm is an efficient algorithm which performs an important task that puts elements of a list in a certain order or arranges a collection of items into a particular order (i.e. either ascending or descending order. Sorting the data element has been developed to arrange the array values in various ways for a database. For example, sorting will order an array of numerical data's from lowest to highest or from highest to lowest, or arrange an array of strings or characters into alphabetical order. We are proposing a dual sorting algorithm based on the Quick Sort. In that First element taken as pivot element one and last element taken as pivot element two. The Sorting required at least two elements. It will start from the second element. Our proposed approach have compared with the existing sorting algorithm for Best case, Worst case, Average case, it provided the better results.

Keywords- Dual Sorting Algorithm; Swapping; Average case; Best case; Worst Case

I. INTRODUCTION

Sorting is any process of regulating items according to a certain sequence or in different sets, and therefore, it has been categorized into two things.

1. Ordering: arranging or regulating items in some ordered sequence,
2. Categorizing: grouping and labeling items with similar properties together (by sorts).

In computer science, sorting is one of the most extended researched subjects because of the need to speed up the operation on thousands, billions or millions of records during a search operation. Most simple sorting algorithms involve two steps which are compare two items and swap two items or copy one item. It will continue to executing over and over until the data is sorted. Because sorting is important to optimizing the use of other algorithms in computer science such as binary search or linear search. It has been the subject of wide research in computer science, and some very difficult methods have been developed. Each method of sorting algorithm has different cases of performance, they are worst case, when the integers are not in order and they have to be swapped at least once. The term best case is used to describe the way an algorithm behaves under optimal conditions. For example, the best case for a simple linear search on an array occurs when the desired element is the first in the list. Average case is similar to worst case, but in average case, the integers are not in order. Computational complexity defined in terms of number of swaps. Sorting methods perform various numbers of swaps in order to sort a data. Memory usage is also a factor in classify the sorting algorithms.

Sorting algorithms are sometimes characterized by big O notation in terms of the performances that the algorithms yield and the amount of time that the algorithms take, where n is integer. Big O notation describes the limiting behaviour of a function when the argument tends towards a particular value or infinity, usually in terms of simpler functions. Big O notation allows its users to simplify functions in order to concentrate on their growth rates. The different cases that are popular in sorting algorithms are O(n) is fair, the graph is increasing in the smooth path. O(n^2): this is inefficient because if we input the larger data the graph shows the significant increase. It means that the larger the data the longer it will take. O(n log n): this is considered as efficient, because it shows the slower pace

increase in the graph as we increase the size of array or data.

Name	Average Case	Worst Cast	Stable
Bubble Sort	O(n^2)	O(n^2)	Yes
Insertion Sort	O(n^2)	O(n^2)	Yes
Merge Sort	O(n log n)	O(n log n)	Yes
Quick Sort	O(n log n)	O(n^2)	No
Cockatail Sort	O(n^2)	O(n^2)	Yes
Selection Sort	O(n^2)	O(n^2)	No
Dual Sort	O(n log n)	O(n^2)	No

II. RELATED WORK

Bubble Sort It is a straightforward and simple method sorting data that is used in computer science. The algorithm starts at the beginning of the data set. It compares the first two elements, and if the first is greater than the second, it swaps them. It continues doing this for each pair of adjacent elements to the end of the data set. It then starts again with the first two elements, repeating until no swaps have occurred on the last pass.

Insertion sort

It is a simple sorting algorithm that is relatively efficient for small lists and mostly-sorted lists, and often is used as part of more sophisticated algorithms. It works by taking elements from the list one by one and inserting them in their correct position into a new sorted list. In arrays, the new list and the remaining elements can share the array's space, but insertion is expensive, requiring shifting all following elements over by one

Selection sort

It is a sorting algorithm, specifically an in-place comparison sort. It has $O(n^2)$ complexity, making it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity, and also has performance advantages over more complicated algorithms in certain situations.

Shell sort

Shell sort was invented by Donald Shell in 1959. It improves upon bubble sort and insertion sort by moving out of order elements more than one position at a time. One implementation can be described as arranging the data sequence in a two-dimensional array and then sorting the columns of the array using insertion sort. Although this method is inefficient for large data sets, it is one of the fastest algorithms for sorting small numbers of elements.

Quick sort

Quick sort is a divide and conquer algorithm which relies on a partition operation: to partition an array, we choose an element, called a pivot, move all smaller elements before the pivot, and move all greater elements after it. This can be done efficiently in linear time and in-place. We then recursively sort the lesser and greater sub lists. Efficient

implementations of quick sort (with in-place partitioning) are typically unstable sorts and somewhat complex, but are among the fastest sorting algorithms in practice. Together with its modest O(log n) space usage, this makes quick sort one of the most popular sorting algorithms, available in many standard libraries. The most complex issue in quick sort is choosing a good pivot element; consistently poor choices of pivots can result in drastically slower $O(n^2)$ performance, but if at each step we choose the median as the pivot then it works in O(n log n). Merge sort It is a comparison-based sorting algorithm. In most implementations it is stable, meaning that it preserves the input order of equal elements in the sorted output.

Cocktail sort

Also known as bidirectional bubble sort, cocktail shaker sort, shaker sort (which can also refer to a variant of selection sort), ripple sort, shuttle sort or happy hour sort, is a variation of bubble sort that is both a stable sorting algorithm and a comparison sort. The algorithm differs from bubble sort in that sorts in both directions each pass through the list. This sorting algorithm is only marginally more difficult than bubble sort to implement, and solves the problem with so-called turtles in bubble sort

III. PROPOSED SYSTEMS

A . System model

Our proposed approaches using Dual sorting algorithm based on the Quick sort. In that we are taking two pivot elements. First element as a pivot element one and last element as a pivot element two. First we have compare the two pivot element. If the pivot element one is grater then the pivot element two means swapping operation has been performed.

We can start sorting from the second element. If the pivot element one is compared with the second element. If the pivot element one is greater than the second element means the swapping operation has been performed otherwise It will compare to the pivot element two. It will larger than the second element. Then it will not perform any operation otherwise It will perform swapping operation. So each iteration at least two element was started. It will reduce the number of iteration. Consider as the following algorithm

Algorithm 1: Dual Sorting Algorithm

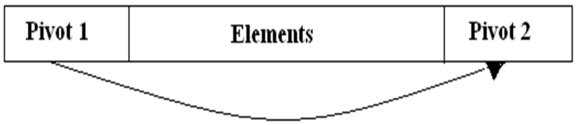
Requirement: Getting the input from the user that should be stored into the array

1. Selecting the first element as an pivot1
2. And selecting the second element as an pivot2
3. To comparing the pivot1 and pivot2
4. If *pivot1> pivot2*
5. Then perform *swap(pivot1, pivot2)*
6. else
7. Continue from the second element. Assigning the second position to variable *i*
8. To compare the i^{th} position element to the pivot1
9. If*(pivot1> i^{th} element)*
10. Then perform swapping operation
11. else
12. compare to the second pivot element
13. if*(pivot2< i^{th} element)*
14. Then perform swapping operation
15. end if
16. end if
17. incrementing the i^{th} positon value
18. goto step8

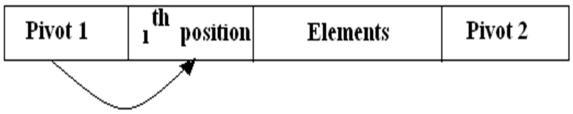
19. perform until the last element
20. end if
21. stop the process

Algorithm Description

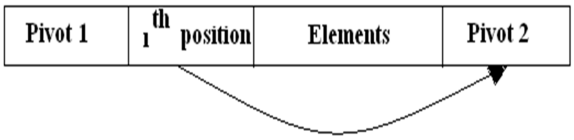
In above algorithm taking the first element as a pivot element one (p1) and last element as a pivot element two(p2). We compare the *P1* and *P2*. If *P1* is greater than the *P2* means swapping operation performed.



And second element position is the i^{th} position. And the second element compared with the *P1* if p1 is greater than the i^{th} position element means swapping operation performed



otherwise it will compare with the p2 element. if p2 element greater than i^{th} position element means swapping operation performed. It will continue until the p1 position is higher than the p2 position.



Example:

Consider as a following elements going to sort such as 94, 24, 96, 47, 84, 36, 39, 72 first element 95 compared with 72. (95> 72) it will perform swapping operation. And continue with second element 24. It will compare with 72. 72 is greater than 24. It will perform swapping operation. And continue with 96. It is not greater than 24, so it will compare with 95. 96 is greater than 95. It will perform swapping operation.

In step4, the element 47 compared with element 24. 24 is smaller than the 47, so it will compare to the element 96. 96 greater than 47. Does not perform any operation and move to the next position. The element 84 compare with the element 24. 24 smaller than the 84. It will compare with the 96. 96 is greater than the 84. It will does not perform any operation and move to the next position. 36 compared with 24. 24 smaller than 36 so it will compare with 96. 96 greater than 36. It will not perform any operation. And move to the next position. 39 compare with 24 and does not perform any operation.

In Iteration2 starting with second element as pivot1 and previous to the last element as a pivot2 the same procedure continued. Until reach the pivot2 position.

Iteration 1:

Steps	**95**	24	96	47	84	36	39	**72**
Step1	**72**	**24**	96	47	84	36	39	95
Step2	**24**	72	**96**	47	84	36	39	95
Step3	24	72	**96**	47	84	36	39	**95**
Step4	**24**	72	95	**47**	84	36	39	96
Step5	24	72	95	**47**	84	36	39	**96**
Step6	**24**	72	95	47	**84**	36	39	96
Step7	24	72	95	47	**84**	36	39	**96**
Step8	**24**	72	95	47	84	**36**	39	96
Step9	24	72	95	47	84	**36**	39	**96**
Step10	**24**	72	95	47	84	36	**39**	96
Step11	24	72	95	47	84	36	**39**	**96**

Iteration 2:

Steps	24	**72**	95	47	84	36	**39**	96
Step1	24	**39**	95	47	84	36	72	96
Step2	24	**39**	**95**	47	84	36	72	96
Step3	24	39	**95**	47	84	36	**72**	96
Step4	24	**39**	72	**47**	84	36	95	96
Step5	24	39	72	**47**	84	36	**95**	96
Step6	24	**39**	72	47	**84**	36	95	96
Step7	24	39	72	47	**84**	36	**95**	96
Step8	24	**39**	72	47	84	**36**	95	96
Step9	24	**36**	72	47	84	**39**	95	96

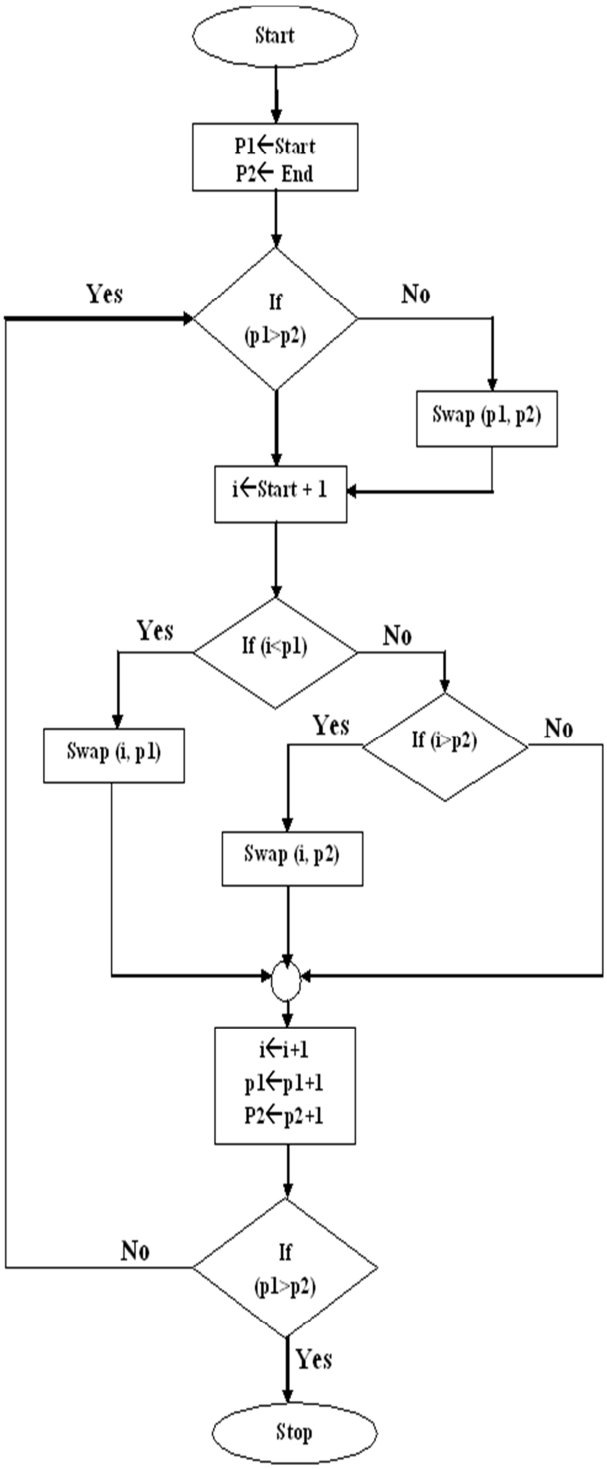
Iteration 3:

Steps	24	36	**72**	47	84	**39**	95	96
Step1	24	36	**39**	47	84	**72**	95	96
Step2	24	36	**39**	**47**	84	72	95	96
Step3	24	36	39	**47**	84	**72**	95	96
Step4	24	36	**39**	47	**84**	72	95	96
Step5	24	36	39	47	**84**	**72**	95	96
Step6	24	36	39	47	**72**	**84**	95	96

Iteration 5:

Steps	24	36	39	**47**	**72**	84	95	96

Flow chart Representation

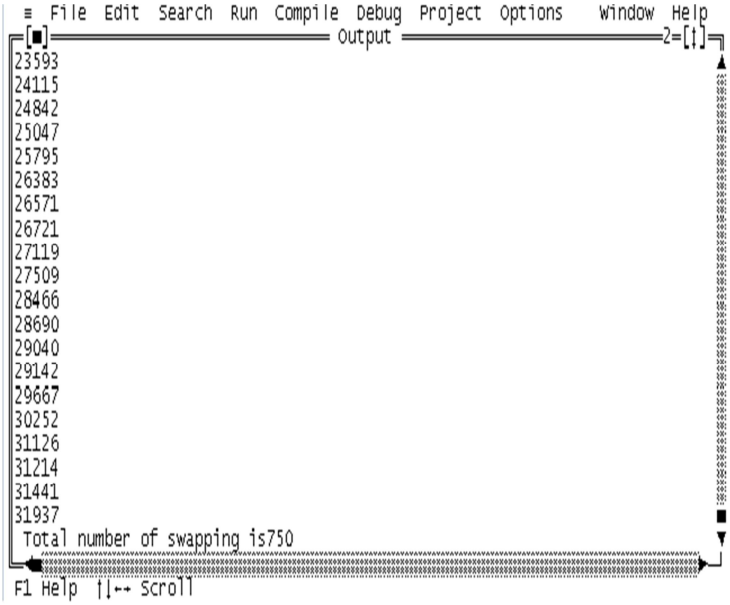


IV. PERFORMANCE EVALUATION

This Section presents the program results and performance analysis of the proposed algorithms. Further this factor affects the worst case, Base case, average case analysis and Number of swapping.

Execution results

The program execution has been carried out using a c++ programming Language. First we are taking 100 Random numbers for reading. We are calculating number of swapping occurred during the sorting process. The following output 100 numbers of elements



And we are taking reading for 1000 number of elements using the random function. We are calculating number of swapping.

And we are taking reading for 10000 number of elements using the random function. We are calculating number of swapping.

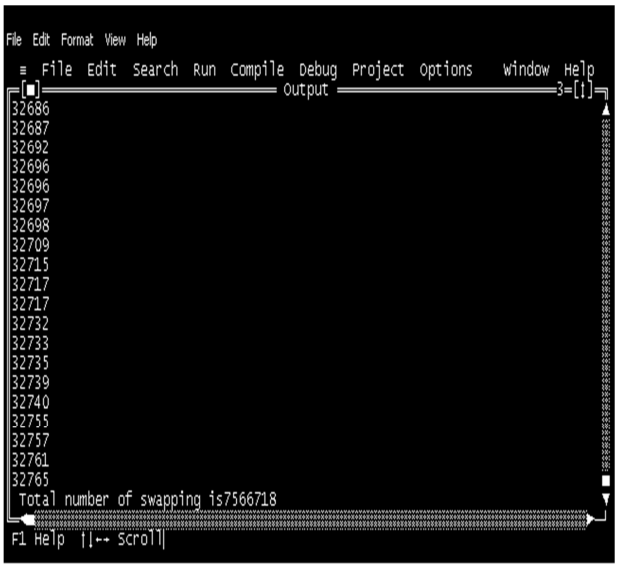
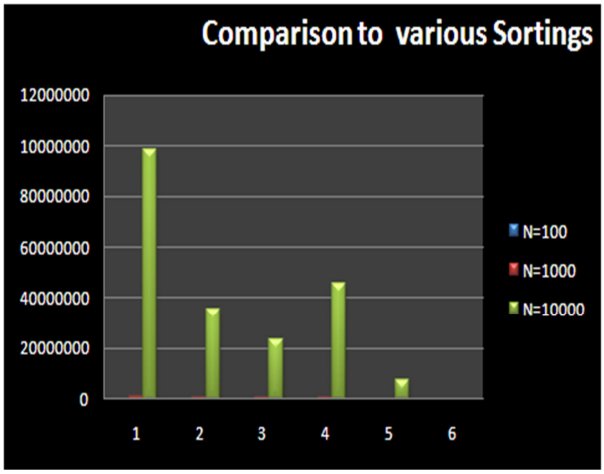


Table: Comparison of the number of swaps relates to several sorting algorithms

Sort	N=100	N=1000	N=10000
Bubble Sort	7890	879604	98530146
Selection Sort	4913	267548	35648754
Insertion Sort	3146	243578	23453685
Cocktail Sort	4897	574970	45367598
Dual Sort	750	83995	7566718

In following graph comparison to the number of element taken and number of swapping.



V. CONCLUSION

Base on our experiment, the charts displaying the relationship between various sorting and number of swapping show that the Dual sort is the fastest algorithm compare to Bubble sort, Insertion sort, Selection sort, Quick Sort and Cocktail sort. It also prove the literature about big O notation, that O(n log n) is considered efficient, because it shows the slower increase in the graph as we increase the size of the array. The diagram shows that Dual sort is efficient for both small and large integers. Although the worst case, it makes O(n^2) comparison, typically, Dual sort is significantly faster in practice than other O(*n* log *n*) algorithms. On the other end of the efficiency scale, the Bubble sort is notoriously slow but is conceptually the simplest of the sorting algorithms and this that reason is a good introduction to sorting. In terms of swapping, the Bubble sort performs the greatest number of swaps because each element will only be compared to adjacent elements and exchanged if they are out of order. Cocktail sort is a slight variation of bubble sort. The table shows that the Cocktail sort performs fewer swaps than the Bubble sort. The reason for this is that bubble sort only passes through the list in one direction and therefore can only move items backward one step each iteration. Insertion Sort sorts small array fast, but big array very slowly. Quick sort is fastest on average. Very slow sorting occurs sometimes due to unbalanced partition.. Merge sort is stable in that two elements that are equally ranked in the array will not have their relative positions flipped.

IV. REFERNCES

1. The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison–Wesley, 1997. ISBN 0-201-89685-0. Pages 138–141 of Section 5.2.3: Sorting by Selection.

2. Sorting by Insertion", The Art of Computer Programming, 3. Sorting and Searching (second ed.), Addison-Wesley, 1998, pp. 80–105, ISBN 0-201-89685-0.

3. Sedgewick, Robert (1983), "8", Algorithms, Addison-Wesley, pp. 95ff, ISBN 978-0-201-06672-2.

4. Ciura, Marcin (2001). "Best Increments for the Average Case of Shellsort". In Freiwalds, Rusins. Proceedings of the 13th International Symposium on Fundamentals of Computation Theory. London: Springer-Verlag. pp. 106–117. ISBN 3-540-42487-3.

5. David M. W. Powers, Parallelized Quicksort and Radixsort with Optimal Speedup, Proceedings of International Conference on Parallel Computing Technologies. Novosibirsk. 1991.

6. Paul E. Black and Bob Bockholt, "bidirectional bubble sort", in Dictionary of Algorithms and DataStructures (online), Paul E. Black, ed., U.S. National Institute of Standards and Technology. 24 August 2009. (accessed: 5 Feb 2010)

Authors Profile

Mr. P. Dhivakar doing Final year M.E CSE in M.Kumarasamy college of Engineering (Autonomous), Karur. He was completed his B.TECH IT in M.Kumarasamy College Of Engineering, Karur in 2012. His interested areas are Wireless Sensor Networks, Data Structure and Algorithms, Network Security.

Mr.G.Jayaprakash doing first year M.E CSE in M.Kumarasamy College of Engineering (Autonomous), Karur. He was completed his B.TECH IT in M.Kumarasamy College Of Engineering, Karur in 2012. His interested areas are Data Structures and Algorithms, Wireless Sensor Networks.

