International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 07 | July -2017 www.irjet.net p-ISSN: 2395-0072

# Study on sorting algorithm and position determining sort

**S. Harihara Sudhan[1], C. Kalaiarasan[2]**

*[1]Student of B.E. Computer Science and Engineering, SNS College of Engineering*
*[2]Dean of Computer Science, SNS College of Engineering*

---

**Abstract-** *Sorting is an important task in many computer applications. Efficiency is a crucial factor when the amount of data is large. Memory allocation in operating systems, networks and databases use sorting concept. There are many ways to implementing different sorting algorithms. Though the real challenge lies in the implementation and the theoretical concept is of mere importance. The new sorting algorithm proposed uses the divide and conquer technique to increase the time efficiency. A new sorting algorithm has been put forth and its advantages and disadvantages have been discussed. The proposed algorithm is compared with other existing sorting algorithms. Finally, the possible implementations of this algorithm have been implemented.*

**Key Words-** sorting, time complexity, space complexity, quick sort, selection sort, algorithm, quasilinear

## 1. Introduction

### 1.1 Concept of sorting

Sorting is considered as an initial task before many processes like searching, data management and database system [1]. Sorting is a process of rearrangement a list of elements to the correct order since handling the elements in a certain order is more efficient than handling randomizes elements [2]. A sorting algorithm is a method that can be used to place a list of unordered items into an ordered sequence which minimizes search time. The sequence of ordering is determined by a key. Various sorting algorithms exist, and they differ in terms of their efficiency and performance. An important key to algorithm design is to use sorting as a basic building block, because once a set of items is sorted, many other problems become easy. So, almost all computer based solutions use any one of the sorting methods.

Many interesting and good sorting algorithms have been proposed. Every algorithm has its own advantages and disadvantages. For example, the selection sort has a poor efficiency when dealing with huge lists. In contrast, the quick sort performs well for large amount of data. The performance of a sorting algorithm depends on the data and the machine used for sorting, which is called as order of sorting algorithm.

The common operation performed in a sorting algorithm is comparison and assignment. Selection of suitable sorting algorithm depends on the input data, available main memory, extent to which the data has been sorted and disk space. In order to calculate the performance of an algorithm the execution time and space required for the successful completion of the algorithm are considered. Since sorting algorithms are common in computer science, some of its context contributes to a variety of core algorithm concepts.

### 1.2 Classification based on efficiency

Sorting algorithms can be classified into three basic groups based on their sorting efficiencies. Some of these groups and representative algorithms are:

a. **Linear time-** An algorithm takes linear time if its complexity is O(n). The runtime increases linearly with size of the input data. Linear time is the best possible time complexity in situations where the algorithm has to sequentially read its entire input. Sorting algorithms such as Bucket sort, Flash sort, and Radix sort run in linear time. Selection problem can be solved in O(n) if the array is sorted.

b. **Polynomial time-** An algorithm runs in polynomial time if a polynomial expression in the size of the input for the algorithms can upper bounded its running time. Some examples are Bubble, Selection and Insertion sort.

**Strong and weak polynomial time**

The algorithm runs in strongly polynomial time if,

1. The number of operations in the arithmetic model of computation is bounded by a polynomial in the number of integers in the input instance; and
2. The space used by the algorithm is bounded by a polynomial in the size of the input.

An algorithm which runs in polynomial time but which is not strongly polynomial is said to run in weakly polynomial time.

c. **Linearithmetic time-** A linearithmetic time is a special case of quasilinear where the exponent k on logarithmic term is one. It is a function of n log n. A line arithmetic term grows faster than a linear term but slower than any polynomial term of n with exponent greater than 1. Heap sort and smooth sort are example. In many cases, the $n \log n$ running time is simply the result of performing a $\Theta(\log n)$ operation $n$ times. Comparison sorts require at least linearithmetic number of comparisons in the worst case because $\log(n!) = \Theta(n \log n)$, by Stirling's [3] approximation. They also frequently arise from the recurrence relation $T(n) = 2T(n/2) + O(n)$.

## 2. Literature Review

There are many sorting algorithms and it is not possible to consider all of them. Hence only basic and the most popular algorithms are reviewed.

### 2.1 Selection Sort

Selection sort is the simplest sorting technique. It has O (n^2) time complexity, making it inefficient on large lists. Although it has many comparisons, it does the least amount of data moving. That means if your data has small keys but large data area, then selection sorting may be the quickest [4]. But this algorithm is not stable because the relative order with the same value is not maintained.

**Disadvantages**

1. The primary disadvantage of the selection sort is its poor efficiency when dealing with a huge list of items;

2. The selection sort requires n-squared number of steps for sorting n elements.

### 2.2 Quick sort

In this sort an element called pivot is identified and that element is fixed in its place by moving all the elements less than that to its left and all the elements greater than that to its right. Since it partitions the element sequence into left, pivot and right it is referred as a sorting by partitioning [5].

**Disadvantages**

1. The slightest disadvantage of quick sort is that its worst case is similar to average performance of bubble, insertion or selection sorts.

### 2.3 Merge sort

Merge sort is a divide and conquer algorithm. It divides the list into two approximately equal sub lists then it sorts the sub lists recursively [6]. Merge sort is a stable sort and is more efficient at handling slow-to-access sequential media.

**Disadvantage**

1. Requires extra space;
2. Requires more space than other sorts.

Some comparative study [7] [8] [9] [10] have been carried out in this field and situations of better suitemate for these algorithms (Table 1) are clearly notified.

**Table-1**

| Comparison | | | |
|---|---|---|---|
| **Name of the algorithm** | **Average Time Complexity** | | **Stable (or) not** |
| | **Average case** | **Worst case** | |
| Selection sort | O(n^2) | O(n^2) | Not stable |
| Quick sort | O(nlog2n) | O(n^2) | Not stable |
| Merge sort | O(nlog2n) | O(nlog2n) | Stable |

## 3. Proposed technique

### 3.1 Description

The new algorithm could be viewed as an extension of the selection sort. It uses the divide-and-conquer strategy. An element is selected and positioned in its exact place after getting compared with all the other elements in the list by a swap operation. An additional array stores the details about the locations that have been already sorted. These details are examined while considering a divide. A divide happens if the largest or smallest element in the list has been fixed. The least or highest element becomes a separate block. These separate blocks will not be considered in further iterations. Hence reducing the number of times the basic operation is executed. No recursion is used in this process.

**3.2 Algorithm of the technique**

**Initialize** *low* as 0

**Initialize** *high* as *n*

**for** $i = 0$ to $n$

*a[i]* = 0

**end for**

**while** *low* < *high*-1 **do // loop1**

**Initialize** *location* and *l* as *low*

**for** *j=low* to *high* **//loop2**

**if** *ar[j]* is less than *ar[l]* **then**

*location* = *location* + 1

**end if**

**end for**

**while** *ar[l]* is equal to *ar[location]* ***and*** *location* is not equal to *l* ***do***

***//*loop3**

**if** *a[location]* is equal to 1 **then**

*location = location* + 1

**else**

*a[location] = 1*

*location = location* + 1

**end if**

**end while**

*a[location]* = 1

*temp* = *ar[l]*

*ar[l]* = *ar[location]*

*ar[location]* = *temp*

**if** *a[low]* is equal to 1 **then**

**while** *a[low]* is equal to 1

*low* = *low* + 1

**end while**

**else if** a*[high]* is equal to 1 **then**

**while** *a[high* - 1*]* is equal to 1

*high* = *high* - 1

**end while**

**end if**

**end while**

***Example:***

| Array to be sorted | 9 | 4 | 8 | 7 | 7 | 5 | 2 | 6 | 7 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|
| Additional space | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 7 | 4 | 8 | 7 | 7 | 5 | 2 | 6 | 7 | 9 |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | 5 | 4 | 8 | 7 | 7 | 7 | 2 | 6 | 7 | 9 |
| | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| | 8 | 4 | 5 | 7 | 7 | 7 | 2 | 6 | 7 | 9 |
| | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| | 7 | 4 | 5 | 7 | 7 | 7 | 2 | 6 | 8 | 9 |
| | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | 2 | 4 | 5 | 7 | 7 | 7 | 7 | 6 | 8 | 9 |
| | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| | 2 | 4 | 5 | 7 | 7 | 7 | 7 | 6 | 8 | 9 |
| | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| | 2 | 4 | 5 | 7 | 7 | 7 | 7 | 6 | 8 | 9 |
| | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| | 2 | 4 | 5 | 6 | 7 | 7 | 7 | 7 | 8 | 9 |
| | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**Figure 1:** Position determining sort example

### 3.3 Factors analyzed

The analysis of algorithm defines that the estimation of resources required for an algorithm to solve a given problem. Sometimes the resources include memory, time and communication bandwidth [11].

**a. Additional space requirements**

The technique requires another array (index array) of the same size as that of the original array. The list differentiates the sorted and unsorted elements of the data. This algorithm also requires two pointers (or locations) that will point to the elements in the lowest and highest index under consideration in the list.

**b. Algorithm complexity**

**I) Run time complexity**

There are two cases of execution –

The main (loop1) loop comparing *low* and *high* iterates n-1 times and the number of iterations of this loop is reduced when the number of elements placed during a cycle increase which is one, normally.

**i) More than one element is placed**

When one or more elements are present in their exacted positions with its duplicates present in the array, multiple elements are placed at the same time. A loop (loop3) checks for redundant elements and arranges them.

**ii) Only one element is placed**

When only one element is placed loop3 is not executed. But in both the cases the loop (loop2) that does the comparison task is executed. This loop finds the appropriate position of the element.

Worst case : $O(n^2)$

Average case : $O(n^2)$

Best case : $O(n^2)$

**II) Space complexity**

The algorithm requires an additional array whose size is one less than the size of the array to be sorted. But unlike in merge sort the whole n-1 locations are not required throughout the operation. Once the lowest and highest elements that are considered are in place the extra space allocated can be freed immediately.

**c. Behaviour on already or nearly sorted array**

An already sorted or nearly sorted array does not bring any change to the runtime or space complexity of the technique. Each element is considered to check for its appropriate position in both the cases.

## 4. Conclusion

The performance of selection, quick and merge sort has been evaluated. This was done using literature material of relevant work. A new technique was implemented that had the aim of exploiting the least amount of data moving like in selection sort and it was achieved. The problem of the proposed technique is its inability to handle redundant data without extra memory. The proposed technique was implemented in C++ language and tested with multiple inputs. Future efforts to minimize the memory usage of the sorting technique will make it more useful.

## 5. References

[1] D.E. Kunth, The Art of Computer Programming: Vol. 3, Sorting and Searching, 2nd printing, Addison- Wesley, Reading, MA, 1975

[2] P. Adhikari, Review on Sorting Algorithms, " A competitive study in two sorting algorithms", Mississippi state university, 2007

[3] Keith Conrad, Stirling's formula. Available in http://www.math.uconn.edu/~kconrad/blurbs/analysis/stirling.pdf.

[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] Khalid Suleiman Al-Kharabsheh, Review on Sorting Algorithms a Comparative Study, International Journal of Computer Science and Security (IJCSS), Volume (7) : Issue (3) : 2013

[6] Katajainen, Jyrki; Pasanen, Tomi; Teuhola, Jukka (1996, MAR). "Practical in-place mergesort". Nordic Journal of Computing. (3). pp. 27–40.

[7] A. Tridgell, Efficient Algorithms for Sorting and Synchronization, Ph.D. Thesis, Dept. of Computer Science, the Australian National University, 1999.

[8] S. Jadoon, S. F. Solehria and M. Qayum, (2011) "Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm: a Comparative Study" International Journal of Electrical & Computer Sciences, IJECS-IJENS, Vol: 11 No: 02.

[9] Y. Yang, P. Yu, Y. Gan, (2011) "Experimental Study on the Five Sort Algorithms", International Conference on Mechanic Automation and Control Engineering (MACE).

[10] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. ACM Computing Surveys, 24:441–476, 1992.

[11] Karunanathi .A, A Survey, Discussion and comparison of sorting algorithms, Umea University, June 2014