SORTING ALGORITHMS

Pooja Nayak, Rajat Wason, Sahil Mudgal
Student(b.tech VIth sem) Department of Computer science
Dronacharya College Of Engineering,Gurgaon-123506

Abstract- Sorting is an important data structure operation, which makes easy searching, arranging and locating the information. This paper includes all the types of sorting algorithms and also their comparison. Algorithms are the finite set of instructions. An algorithm must be unambiguous. There are some properties of the algorithms. Such as finiteness, definiteness, input, output, effectiveness. In this we discuss about merge sort, bubble sort, selection sort and quick sort.

I. INTRODUCTION

One of the basic problems of computer science is ordering a list of items. There are a number of solutions to this problem, known as sorting algorithms . Some sorting algorithms are simple and spontaneous, such as the bubble sort. Others, such as the quick sort are enormously complex, but produce super-fast results.

There are several elementary and advance sorting algorithms. All sorting algorithm are problem specific meaning they work well on some specific problem and do not work well for all the problems. All sorting algorithm are, therefore, appropriate for specific kinds of problems. Some sorting algorithm work on less number of elements, some are suitable for floating point numbers, some are good for specific range, some sorting algorithms are used for huge number of data, and some are used if the list has repeated values. We sort data either in statistical order or lexicographical, sorting numerical value either in increasing order or decreasing order and alphabetical value like addressee key.
The formal definition of the sorting problem is as follows:

Input: A sequence having n numbers in some random order
(a1, a2, a3, ….. an)
Output: A permutation (a'1, a'2, a'3, ….. a'n)
$a'1 \leq a'2 \leq a'3 \leq \ldots.. a'n$

For instance, if the given input of numbers is (59, 41, 31, 41, 26, 58), then the output sequence returned by a sorting algorithm will be (26, 31, 41, 41, 58, 59).

Various other factors which include:

- The size of the array or sequence to be sorted,
- The extent up to which the given input sequence is already sorted,
- The probable constraints on the given input values,
- The system architecture on which the sorting operation will be performed,
- The type of storage devices to be used: main memory or disks.

Almost all the available sorting algorithms can be categorized into two categories based on their difficulty. The complexity of an algorithm and its relative effectiveness are directly correlated. A standardized notation i.e. Big O(n), is used to describe the complexity of an algorithm. In this notation, the O represents the complexity of the algorithm and n represents the size of the input data values. The two groups of sorting algorithms are

$O(n^2)$, which includes the bubble, insertion, selection sort and O(nlogn) which includes the merge, heap & quick sort.

II. COMPARISON OF SORTING ALGORITHMS

	Time					
Sort	Avg.	Best	Worst	Space	Stability	Remarks
Bubble sort	O(n^2)	O(n^2)	O(n^2)	Constant	Stable	Always use a modified bubble sort
Selection Sort	O(n^2)	O(n^2)	O(n^2)	Constant	Stable	Even a perfectly sorted input requires scanning the entire array
Insertion Sort	O(n^2)	O(n)	O(n^2)	Constant	Stable	In the best case (already sorted), every insert requires constant time
Merge Sort	O(n*logn)	O(n*logn)	O(n*logn)	Depends	Stable	On arrays, merge sort requires O(n) space; on linked lists, merge sort requires constant space
Quick Sort	O(n*logn)	O(n*logn)	O(n^2)	Constant	Stable	Randomly picking a pivot value (or shuffling the array prior to sorting) can help avoid worst case scenarios such as a perfectly sorted array.

A. BUBBLE SORT :

Bubble sort is a basic sorting algorithm that performs the sorting operation by iteratively comparing the adjacent pair of the given data items and swaps the items if their order is reversed. The worst case as well as average case complexity of bubble sort is $O(n^2)$, where n represents the total number of items in the given array to be sorted.

ALGORITHM:

- Repeat Steps ii and iii for K=1 to N-1
- Set PTR = 1
- Repeat while PTR<= N-K
  1) If DATA[PTR] > DATA[PTR+1], then
  2) Swap DATA[PTR] and DATA[PTR+1]
  3) Set PTR = PTR+1
- Exit.

1) Advantage: Simplicity and ease-of-implementation

2) Disadvantage: Code inefficient

B. Selection Sort [Best/Worst: O(N^2)]

Scan all items and find the smallest. Swap it into position as the first item. Repeat the selection sort on the remaining N-1 items.

ALGORITHM:

- Repeat Steps ii & iii for K=1 to N-1
    - Set MIN = DATA[K] and LOC = K
    - Repeat for J= K+1 to N
    - If MIN > DATA[J] then
        - MIN= DATA [J]
        - LOC = DATA [J]
        - LOC = J
    - Set TEMP = DATA[K]
    - DATA [K] = DATA[LOC]
    - DATA[LOC] = TEMP
- Exit

1) Advantage: Simple and easy to implement

2) Disadvantage: Inefficient for large lists, so similar to the more efficient insertion sort, the insertion sort should be used in its place.

C. **Insertion Sort [Best: O(N), Worst: O(N^2)]**

Start with a sorted list of 1 element on the left, and N-1 unsorted items on the right. Take the first unsorted item and insert it into the sorted list, moving elements as necessary:

ALGORITHM:

- Set A[0] = -∞
- Repeat Steps iii to v for K = 2 to N
- Set TEMP = DATA[K] and PTR = K
- Repeat
- while TEMP < DATA[PTR]
    - Set DATA[PTR+1] = A[PTR]
    - Set PTR =PTR – 1
    - Set DATA[PTR+1] = TEMP
- Exit.

1)Advantage: Relative simple and easy to implement. Twice faster than bubble sort.

2) Disadvantage: Inefficient for large lists.

D.Quicksort [Best: O(N lg N), Avg: O(N lg N), Worst:O(N^2)]

There are many versions of Quicksort[7], which is one of the most popular sorting methods due to its speed (O (N lgN) average, but O (N^2) worst case).

1) **Using external memory**
2) **Using in-place memory**
3) **Using in-place memory with two pointers:**

Advantage: Fast and efficient

Disadvantage: Show horrible result if list is already sorted.

***E*. Merge Sort [Best: O(N lg N), Avg: O(N lg N), Worst:O(N^2)]**

Merge sort is based on the divide-and-conquer paradigm. Its worst-case running time has a lower order of growth than insertion sort

1) Divide Step If a given array *A* has zero or one element, simply return; it is already sorted. Otherwise, split *A*[*p* .. *r*] into two subarrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*], each containing about half of the elements of *A*[*p* .. *r*]. That is, *q* is the halfway point of *A*[*p* .. *r*].

2) Conquer Step
Conquer by recursively sorting the two subarrays *A*[*p* .. *q*] and *A*[*q* + 1 .. *r*].

Advantage: Well suited for large data set.

Disadvantage: At least twice the memory requirements than other sorts**.**

III. CONCLUSION

As the algorithm has already proved itself well in the case of integer values, this could be applied to other complex data types and its performance could be evaluated. One such data type could be characters or even strings. This could be helpful in sorting the character strings and be applied in, for example, contacts in mobile phones, words in dictionary etc.

The two dimensional matrices are being used to store and represent massive data in many fields, such as meteorology, engineering design and management. Efficient two-dimensional sorting algorithms are needed when these data are sorted by computers. The proposed algorithm could act as a base for the development of effective two dimensional sorting algorithms that could serve the purpose.

For further evaluation purposes, the algorithm can be compared with merge, cocktail or heap sort etc. Although these algorithm have their own specific implementation constraints, performance analysis with these or some other existing sorting algorithms will give be useful.

REFERENCES

[1] Sultanullah Jadoon, Salman Faiz Solehria, Mubashir 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.

[2] You Yang, Ping Yu, Yan Gan, (2011) "Experimental Study on the Five Sort Algorithms", International Conference on Mechanic Automation and Control Engineering (MACE).

[3] Charles E. Leiserson, Thomas H. Cormen, Ronald L. Rivest, Clifford Stein (2009) "Introduction To Algorithms", MIT Press, 3rd Ed. p.5-7, 147-150.

[4] Seymour Lpischutz, G A Vijayalakshmi Pai (2006) "Data Structures", Tata McGraw-Hill Publishing Company Limited, p.4.11, 9.6, 9.8.