© 2014 IJIRT | Volume 1 Issue 6 | ISSN : 2349-6002

# AN ENHANCEMENT OF MAJOR SORTING ALGORITHMS

Proteep Benerjee, Ritu Verma, Sagar Dudega

*Abstract*- **One of the fundamental issues in computer science is ordering a list of items. Although there is a huge number of sorting algorithms, sorting problem has attracted a great deal of research; because efficient sorting is important to optimize the use of other algorithms. This paper presents two new sorting algorithms, enhanced selection sort and enhanced bubble Sort algorithms. Enhanced selection sort is an enhancement on selection sort by making it slightly faster and stable sorting algorithm. Enhanced bubble sort is an enhancement on both bubble sort and selection sort algorithms with O(nlgn) complexity instead of O(n2) for bubble sort and selection sort algorithms. The two new algorithms are analyzed, implemented, tested, and compared and the results were promising.**

*Index Terms*- **Enhanced selection sort, enhanced bubble sort, selection sort, bubble sort, number of swaps, time complexity.**

## I. INTRODUCTION

Information growth rapidly in our world and to search for this information, it should be ordered in some sensible order. Many years ago, it was estimated that more than half the time on many commercial computers was spent in sorting. Fortunately this is no longer true, since sophisticated methods have been devised for organizing data, methods which do not require that the data be kept in any special order [9]. Many algorithms are very well known for sorting the unordered lists. Most important of them are Heap sort, Bubble sort, Insertion sort and shell sort [17]. As stated in [5], sorting has been considered as a fundamental problem in the study of algorithms, that due to many reasons:

• The need to sort information is inherent in many applications.

• Algorithms often use sorting as a key subroutine.

• In algorithm design there are many essential techniques represented in the body of sorting algorithms.

• Many engineering issues come to the fore when implementing sorting algorithms. Efficient sorting is other algorithms that require sorted lists to work important to optimize the use of correctly; it is also often in producing human-readable output. Formally, the output should satisfy two major conditions:

• The output is in non-decreasing order.

• The output is a permutation, or reordering, of the input. Since the early beginning of computing, the problem has attracted many researchers, perhaps due to sorting the complexity of solving it efficiently. Bubble sort was analyzed as early as 1956 [2]. Many researchers considered sorting as a solved problem. Even so, useful new sorting algorithms are still being invented, for example, library sort was first published in 2004. Sorting algorithms are prevalent in introductory computer science classes, where the abundance of algorithms for the problem provides a gentle introduction to a variety of core algorithm concepts [1, 19]. In [1], they classified sorting algorithms by:

• Computational complexity (worst, average and best behavior) of element comparisons in terms of list size (n). For typical sorting algorithms good behavior is O(n log n) and bad behavior is $\Omega(n^2)$. Ideal behavior for a sort is O(n). Sort algorithms which only use an abstract key comparison operation always need $\Omega$(n log n) comparisons in the worst case.

• Number of swaps (for in-place algorithms).

• Stability: stable sorting algorithms maintain the relative order of records with equal keys (values). That is, a sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.

• Usage of memory and other computer resources. Some sorting algorithms are “in place”, such that only O(1) or O(log n) memory is needed beyond the items being sorted, while others need to create auxiliary locations for data to be temporarily stored.

• Recursion: some algorithms are either recursive or non recursive, while others may be both (e.g., merge sort).

• Whether or not they are a comparison sort. A comparison sort examines the data only by comparing two elements with a comparison operator. In this paper, two new sorting algorithms are presented. These new algorithms may consider as selection sort as well as bubble sort algorithms. The study shows that the proposed algorithms are more efficient, theoretically, analytically, and practically as compared to the original sorting algorithms. Section 2 presents the concept of enhanced Selection Sort (SS) algorithm and its pseudocode. Furthermore, the implementation, analysis, and comparison with selection sort are highlighted. Section 3 introduces enhanced bubble sort algorithm and its pseudocode, implementation, analysis, and comparison with bubble sort. Also, a real-world case study for the proposed algorithms is presented in section 4. Finally, conclusions were presented in section 5.

## II. ENHANCED SELECTION SORT

### 2.1. Concept

Inserting an array of elements and sorting these elements in the same array (in-place) by finding the maximum element and exchanging it with the last element, and then decreasing the size of the array by one for next call. In fact, the Enhanced Selection Sort (ESS) algorithm is an enhancement to the SS algorithm in decreasing number of swap operations, making the algorithm to be data dependent, and in making it stable. The differences between ESS and SS algorithms are discussed in section 2.5.

### 2.2. Procedures

The procedures of the algorithms can be described as follows:

• Inserting all elements of the array.

• Calling the "Enhanced Selection Sort" function with passing the array and its size as parameters.

• Finding the maximum element in the array and swapping it with the last index of the same array.

• Decreasing the size of the array by one.

• Calling the "Enhanced Selection Sort" function recursively. The size of the array is decremented by one after each call of the "Enhanced Selection Sort" function. Operationally, the (size) after the first call became (size-1), and after the second call became (size-2), and so on.

### 2.3. Pseudocode

In simple pseudocode, enhanced selection sort algorithm might be expressed as:

```
function enhanced selection sort (array , size)
1 if size > 1 then
2 var index, temp, max
3 index := size-1
4 max := array(index)
5 for a:= 0 to size-2 do
6 if array(a) ≥ max then
7 max := array(a)
8 index := a
9 end if
10 end for
11 if index ≠ size-1 then
12 temp := array(size-1)
13 array(size-1) := max
14 array(index) := temp
15 end if
16 size := size-1
17 return Enhanced Selection Sort (array , size)
18 else
19 return array
20 end if
```

2.5. Comparison with SS Algorithm SS algorithm, works by selecting the smallest unsorted item remaining in the array, and then swapping it with the item in the next position to be filled. The selection sort has a complexity of O(n2) [8, 11]. In simple pseudocode, selection sort algorithm might be expressed as:

```
Function SelectionSort(array, size)
1 var I, j
2 var min, temp
3 for I := 0 to size-2 do
4 min := I
5 for j := i+1 to size-1 do
6 if array(j) < array(min)
7 min := j
8 end if
9 end for j
10 temp := array(i)
11 array(i) := array(min)
12 array(min) := temp
13 end for I
```

The main advantage enhanced selection sort over selection sort algorithms is: selection sort always performs O(n) swaps while enhanced selection sort depends on the state of the input array. In other words, if the input array is already sorted, the ESS does not perform any swap operation, but selection sort performs n swap operations. Writing in memory is more expensive in time than reading, since EBS performs less number of swaps (read/write) then it is more efficient than selection sort when dealing with an array stored in a secondary memory or in EEPROM (electrically erasable programmable read only memory). However, there are many similarities between ESS and SS algorithms.

To prove that ESS algorithm is relatively faster than SS algorithm, we implement each of them using C++, and measure the execution time of both programs with the same input data, and using the same computer. The built-in function (clock ()) in C++ is used to get the elapsed time of the two algorithms.

```
#include<iostream.h>
#include<ctime>
#include <cstdlib>
int sort(int[], int);
void main()
{ . . . .
 clock_t Start, Time;
 Start = clock();
 // the function call goes here
 Time = (clock() – Start);
 cout<<"Execution Time : "<<Time<<" ms."<<endl;
}
```

3. Enhanced Bubble Sort

The history of Bubble sort algorithm may elaborate as follows:

In 1963 FORTRAN textbook [13] states the following code to what so called "Jump-down" sort.

```
1 void JumpDownSort(Vector a, int n){
2 for(int j=n-1; j> o; j--)
3 for(int k=0; k< j;k++)
4 if (a[j] < a[k])
5 Swap(a,k,j);}
```

3.1. Concept and Procedures of EBS

The proposed algorithm is considered as an enhancement to the original Bubble sort algorithm and it works as follows:

Inserting an array of elements and sorting these elements in the same array (in place) by finding the minimum and the maximum elements and exchanging the minimum with the first element and the maximum with the last element, and then decreasing the size of the array by two for next call.

The detailed procedures of the algorithm can be summarized as follows:

1. Inserting all elements of the array.
2. Defining and initializing two variables, (firstindex = 0) and (lastindex = size-1).
3. Calling the "Enhanced Bubble Sort" function with passing the array, its size, firstindex, and lastindex as parameters of the function.
4. In the "Enhanced Bubble Sort" function, the operation now is to find the maximum and the minimum and saving the index value of the max of the array in the variable maxcounter, and the index value of the min in elements the variable mincounter.
5. Put the max in the lastindex and min in the firstindex of the array without losing the last values of the first index and the last index of the original array.
6. Decreasing the value of lastindex by one and increasing the value of firstindex by one. Operationally, the size of the array after the first call became (size-2), and after the second call it actually became (size-4), and so on.
7. Calling the "Enhanced Bubble Sort " array recursively while the size of the array is greater than one (size>1). Then returning the sorted array

## III. CONCLUSIONS

In this paper, two new sorting algorithms are presented. ESS has O(n2) complexity, but it is faster than SS, especially if the input array is stored in secondary memory, since it performs less number of swap operations. SS can be specially implemented to be stable. One way of doing this is to artificially extend the key comparison, so that comparisons between two objects with other equal keys are decided using the order of the entries in the original data order as a tie-breaker. ESS is stable without the need to this special implementation EBS is definitely faster than BS, since BS performs O(n2) operations but EBS performs O(nlgn) operations to sort n elements. Furthermore, the proposed algorithms are compared with some recent sorting algorithms; shell sort and enhanced shell sort. These algorithms are applied on a real-world case study of sorting a database of (12500) records and the results showed that the EBS also increases the efficiency of both shell sort and enhanced shell sort algorithms.

## REFERENCES

[1] Aho A., Hopcroft J., and Ullman J., The Design and Analysis of Computer Algorithms, Addison

Wesley, 1974.
[2] Astrachanm O., Bubble Sort: An Archaeological Algorithmic Analysis, Duk University, 2003.
[3] Bell D., “The Principles of Sorting,” Computer Journal of the Association for Computing Machinery, vol. 1, no. 2, pp. 71-77, 1958.
[4] Box R. and Lacey S., “A Fast Easy Sort,” Computer Journal of Byte Magazine, vol. 16, no. 4, pp. 315-315, 1991.
[5] Cormen T., Leiserson C., Rivest R., and Stein C., Introduction to Algorithms, McGraw Hill, 2001.
[6] Deitel H. and Deitel P., C++ How to Program, Prentice Hall, 2001.
[7] Friend E., “Sorting on Electronic Computer Systems,” Computer Journal of ACM, vol. 3, no. 2, pp. 134-168, 1956.