DOI: 10.15415/jotitt.2015.31005

Enhancements In Sorting Algorithms: A Review

SHAMA RAHEJA AND VINAY KUKREJA

Chitkara University, Punjab, India

Email: shama.rani@chitkara.edu.in

Received: December 18, 2014| Revised: February 24, 2015|Accepted: May 20, 2015

Published Online: June 29, 2015

Abstract: One of the important issues in designing algorithms is to arrange a list of items in particular order. Although there is a large number of sorting algorithms, sorting problem has concerned a great compact of research, because efficient sorting is important to optimize the use of other algorithms. In many applications, sorting plays an important role as to easily handling of the data by arranging it in ascending or descending order.[2] In this paper, we are presenting enhancements in various sorting algorithms such as bubble sort, insertion sort, selection sort, and merge sort. A sorting algorithm consists of comparison, swap, and the use of assignment operations. Bubble sort, selection sort and insertion sort are algorithms, which are easy to understand but have the worst time complexity of O (n^2). The new algorithms are discussed, analyzed, tested, and executed for reference. Enhanced selection sort is based on sorting the items by making it slightly faster and stable sorting algorithm. Modified bubble sort is an modification on both bubble sort and selection sort algorithms with O (n log n) complexity instead of O (n^2) for bubble sort and selection sort algorithms. [3] [1].

Keywords: sorting techniques, enhanced sorting, bubble sort, selection sort, insertion sort, merge sort, complexities.

1. INTRODUCTION TO SORTING ALGORITHMS

An algorithm is a finite set of phases defining the solution topa rticular problem. [1].

An algorithm can be expressed in English likelanguage, called pseudo code, In a programming language, or in theform of a f lowchart.[2] One of the basic problems of computer science is sorting a list of items.

Journal on Today's Ideas – Tomorrow's Technologies, Vol. 3, No. 1, June 2015 pp. 73–82

The main purpose of sorting is to arrange the data into particular order. [3] There are several advanced algorithms, which works on fewer amounts of elements. Some algorithms are suitable for floating-point numbers and specific range. [3] [1]

The bubble sort, insertion sort, selection sort, and are Very simple algorithms, often included in the textbooks to intr oduce algorithms and sorting, having runtime complexity of O (n2) making them unrealistic to useThe selection sort has a sli ghtbetter running time than the simplest bubble sort algorithm and worsethan the insertion sorts do. It yields the improvements over bubble sorts are 60%.

However, the insertion sort is over twice asfast as the bub ble sort. A sorting algorithm is said to be stable if two objects with equal keys appears in the same order in sorted output as they performed in the same order in sorted output as they appear in the input unsorted arrays.

Some sorting techniques are stable by nature is insertion sort, merge sort and bubble sort. [2] [5]

2. EVERY ALGORITHM MUST FULFILL THE FOLLOWING PRINCIPLES

A. Input: There are zero or more values which are supplied from outside.
B. Output: At least one value is produced.
C. Definiteness: Each step must be clear and unambiguous.

In general, sorting means re-arrangement of data items according to a well-defined arrangement. The task of sorting algorithm is to transform the novel unsorted sequence to sorted sequence. Different sorts are classified in different categories: run time, memory usage, stability, comparison/non-comparison etc. [2] [6]

When comparing various sorting algorithms, there are several things to deliberate. The first is usually runtime. When dealing with increasingly large collections of data, ineffective sorting algorithms can be too slow for practical use.

A second consideration is taken is memory space. An earlier algorithm that needs the recursive calls naturally involve creating copies of the data to be sorted. In some surroundings where memory space is best, certain algorithms may be impractical. In other cases, it is possible to modify the algorithms without creating copies of data. However, this modification may also come at the cost of some of the performance advantage. [2][1][3][5]

A third consideration is stability. Stability is defined as the results of occurring the elements which comparatively the same. In stable sorting

techniques, those elements whose comparison key is the same will remain in the same order after sorting as they were before sorting. In an unstable sorting, no assurance is made as to the relative output order of those elements whose sort key is the same.[2]

3. THE RECOMMENDED MODIFIED SORTING ALGORITHMS

In the successive discussion, sorting is assumed to be in the ascending order for standardization. The earlier algorithms are concisely described before the discussion of the proposed algorithms for reference.

1) BUBBLE SORT: In the earlier bubble sort, in a single reiteration a data item is shifted to the end until a larger item is found. In this case, no swap operation takes place and this indicates that the element located just before the larger item is the largest until that location and it is assured that, no data item beyond this can reach the end in the next pass. Therefore, the next pass can start from the location where no swap operation occurs. Where enhanced bubble sort completely affects the complexity of the classical bubble sort algorithm. [4]

The enhanced bubble sort guaranteed range between the former maximum and the just found current maximum no value is greater than the former maximum. [6] Therefore, it is also wastage of time to look for a value greater than the former maximum in this range. Then, in the next iteration, it is safe for to look for the maximum from the location of the current maximum and can put the former maximum element at the location just before the current maximum value by interchanging appropriately to minimize the spaces in the search. [1] [7]

Algorithm: EBSA (a[], len)

Here **a** is the unsorted input list and len is defined as length of an array. After accomplishment of an algorithm the array must be sorted.

1. Set top = -1
2. Repeat steps 3 to 5 for i = **0** to n-2
3. If (top<0)
Push zero on stack
End if
4. Pop stack and put in val
5. Repeat step 6 for j = val to n-i-2
6. If (a[j]>a[j+1])
Interchange a[j] and a[j+1]

Else
Push j on the stack
End if [14][7]

One important thing to note here that in this process multiple local maximums can be discovered in a single pass and all of them must be remembered to use in later passes of the algorithm. A stack can be used to accomplish this. When a new maximum value is found, there should be an interchange between the earlier maximum value and the value that located just before the current maximum value and this new location of the old maximum value should be pushed on the stack. When the list is to be completed, the top item in the stack is measured the location of the former maximum value. Then the latest maximum value is swapped with the last item and the location of the current maximum value is considered as the starting point in the next iteration to look for the maximum. The next pass starts by assigning the value at the location stored at top in the stack to the current maximum. [, 4] [6]

ANALYSIS: That is why the best-case complexity is O(n) instead $0(n^2)$ of the classical bubble sort algorithm. The worst-case and average case complexity remains O (n^2), the same as before modification. Again, here, the stack can grow as large as the array to be sorted. Therefore, the space complexity is O (n).

A) *TIME COMPLEXITY:* Let the input list consists of n items.As in sorting algorithms, number of comparisons is used as a measure of computations required for sorting. It is experimented in the above algorithm that the outer loop is always executed but the inner loop (statement 5) is problem instance dependent. Therefore, the complexity of this algorithm is not deterministic and three special situations need to be allocated for complexity analysis.

BEST CASE: In every pass (statement 5) will be executed just once and later the comparison in (statement 6) will be proficient also once and hence, T(n) = n-1 = O(n). The best-case complexity of the classical selection sort is $0(n^2)$.

WORST CASE: In the iteration no k, when k-1 items are already sorted and n-k+1 items are still unsorted, n-k comparisons are required to find the largest element. Thus in this case T(n) = (n-1) + (n-2) +(n-3) + ……. + 3 + 2+1= n(n-l)/2 = (n2-n)/2 = $0(n^2)$.

AVERAGE CASE: An item is greater than half of the items in the list and smaller than the other half in the list. Therefore, in iteration k, when k-1 items are already sorted and n-k+1 items are unsorted, (n-k)/2 comparisons are required to find the current maximum. This leads to T(n) = ((n-1) + (n-2) + (n-3) +…….+ 3+2 + l)/2 = n(n-l)/4 = (n2-n)/4 = 0(n2).

B) *SPACE COMPLEXITY:* It is understandable that the enhanced selection sort algorithm uses a stack.in worst case; the stack can be as large as the size of the data array to sort. The space complexity of the proposed algorithm is O (n). However, the earlier selection sort algorithm does not use such stack. The enhanced algorithm's memory requirement is incompletely double of that of the earlier algorithm. However, classical computers always bias for increased performance with the cost of some extra memory in case of time-space trade off scenarios. [6]

(2). *SELECTION SORT:* The steps performed in earlier selection sort were huge and unnecessary comparisons and large run time. [2] We propose a new selection sorting technique for double-ended selection sort. All the code of elementary sorting techniques are in the books, simply found on the web, but the use of "double sorting" is not available and offer speed improvement over the normal selection sort. With double-ended selection sort, the average number of comparison is indistinctly reduced. Here, idea of using two chosen elements concurrently, applies to "selection sort", through possibility of enhance speed up 25% to 35%. Therefore, easily within the reach of beginners who understand the basic concept of this new sorting. [9] [8]

4. DOUBLE SELECTION SORT

The double selection sort starts from two elements and searches the entire list until it finds the maximum value and minimum value. The sort places the lowest value in the first place and maximum value in the last place, selects the second and second last element and searches for the second smallest and largest element. This process continues until the complete list is sorted.

In other words, a takes an elementary sorting algorithm designed to minimize the number of conversations that are performed. It is working by creation of N-l passes over the reduction-unsorted portion of the array, each time selecting the lowest and highest value. Those values are moved into their final sorted position with one exchanges pieces. [14]

Algorithm :ESSA(a[], Len)

Here a is the unsorted input list and length is the length of array. After the ending of the algorithm array will become sorted. Variable max keeps the location of the current maximum.

1. Repeat steps 2 to 9 until Len = l
2. If stack is empty
   Push 0 in the stack
   End if

3. Pop stack and put in max
4. Set count=max+1
5. Repeat steps 6 and 7 until count<Len
6. If (a[count]>a[max]
a. Push count-1 on stack
b. Interchange data at location count-1 and max
c. Set max=count
End if
7. Set count=count+1
8. Interchange data at location len-1 and max
9. Set len=len-1 [14][8]

ANALYSIS: The same analysis for the enhanced bubble sort algorithm is applicable for the enhanced bubble sort algorithm. That is the best-case complexity is O(n) instead O(n2). The average case and the worst-case complexity remains $O(n^2)$, the same as the classical version. Again, here, the stack can grow as large as the array to be sorted. Therefore, the space complexity is (n).

(3). INSERTION SORT: To insert a single item in the earlier insertion sort, many write operations were necessary to perform. A linear array has two sides and both of them can be considered to insert an element in it. It is natural that there will be unequal number of required shifts to insert an item along the left or the right side.[2] Therefore, it would be cost effective in terms of memory writes to insert an item along the side which demands less number of shifts. As data can be shifted from left or right, there must be enough space in the left to hold the moved data. This needs equal number of cells of the original array in both the left and the right sides. A momentary array with a double length of the original array can come to aid in this situation. The initial item of the unique array should be copied in the middle of the temporary array. Succeeding items would be added either to the left or to the right or to be inserted along the left side or the right side of the temporary array according to the need.[l]

Algorithm: EISA (a [], Len)

Here *a* is the unsorted input list and length is the length of array and b is a temporary array of size 2*length. After completion of the algorithm array will become sorted.

1. Set left = Len
2. Set right = Len 3.
3. Setb [left] = a[0]

4. Repeat steps 5 to 9 for i = 1 to len-1
5. If (a[i]>=b[right]) 5a. Set right = right+1
   Set b[right]=a[i]
   Go to step 4
   End if
6. 1f(a[i]<=b[left])
6a. Set left = left-1
   Set b[left] = a[i];
   Go to step 4
   End if
7. Set loc = right
8. Repeat while (a[i]<b[loc])
   Set loc = loc-1; 9.
9. If(right-loc<loc-left)
   Set j=right+1
   Repeat steps 9 while (j>loc+l) 9bx. Set b[j]=b[j-l]
   Set j=j-1
   Set right=right+l
   Set b[loc+l]=a[i]

Else

   Set j=left-1
   Repeat step 9bx and 9by while (j<*loc)* 9bx. Set b[j]=b[j+l]
   Set j=j+1
   Set left = left-1
   Set b[loc] = a[i]
   End if
10. Repeat steps 10a and 10b for i = 0 to n-l
   Set a[i]=b[left]
   Set left = left+1

ANALYSIS: The enhancement of the proposed algorithm is by reducing shift operation, which does not necessarily reduce any comparisons. Hence, the time complexity of the enhanced insertion sort is exactly same as the classical version, which are O(n), $O(n^2)$, and $O(n^2)$ for the worst, average and best respectively. Here, a temporary array of double length of the list to be sorted is used. So the space complexity is 2n or O (n). [7,8]

(4) MERGE SORT: In a merge sort algorithm, the best case occurs when the two sub arrays are already in ascending order. In this case, each element of the left sub array is compared to the first element of the right sub array.[8] As all

the elements of the left sub array are smaller than the smallest element (first element) of the right sub-array, they are derivative to the main array section by section at each iterative step. Now, the right sub array is appended to the main array without any additional comparisons. In this case, the number of comparisons is half the size of the main array. [2] [4] [5]

In the proposed modification to the existing merge sort algorithm, the fact that the two sub arrays to be merged are previously sorted is being too used. Thus, the best case can be recognized if the last element of left sub-array is less than first element of right sub array. If this case rises at any recursive calls, the two arrays will be appended to the main array without any additional comparisons.[12][13]

Algorithm EMSA(P, a, b, c)

```
1   nl←b-a+1
2   n2←c-b
3   create arrays L[1...N1+1] and R[l...N2+l]
4   for i←l to Nl
5   do L[i]← P[a+i-l]
6   for j ← 1 to n2
7   do R[j] ← P[b+j]
8   L[Nl+l]← ∞
9   R[N2+l] ← ∞
l0  i ← 1
11  j← l
12  if (L[Nl]<R[l])
13  for b ← a to b
14  P[b] ← L[i]
15  P[b+i] ← R[i]
16  i←i+l
17  else if(L[l]>R[N2])
18  for b ← a to b
19  P[b] ← R[i]
20  P[b+i] ← L[i]
21  i ← i+l
22  else
23  for k ← b to c
24  ifL[i]<R[j]
25  P[k] ← L[i]
26  i ← i+1
27  elseP[k]←[13]
```

ANALYSIS: Although the overall time complexity remained the same (O (N log N)), the number of comparisons between the array elements in some particular Cases have been reduced. The experimental examines showed that when we sort a list of numbers in random order, the improved algorithm workings almost as efficiently as the present algorithm. However, in the best case, the number of comparisons has been reduced drastically (by nearly 86%). Merge sort, the one we are modifying the copies the contents of the input array into the temporary array, and then copies the temporary array back into the input array. So it recursively sorts the input array, placing the two sorted halves into the temporary array. Then it merges, placing the sorted sequence into the input array as it goes. The improvement is that this double copying is wasteful can be done without. His suggestion is that: We can make it so that each call to merge only copies in one way, but the calls to merge alternate the direction. [12]

5. CONCLUSIONS

A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input unsorted array. Some sorting algorithms are stable by nature like Insertion sort, Merge Sort, Bubble Sort, etc. In this paper a review on modifications of existing algorithms, efforts are made to point out some deficiencies in earlier work related to sorting algorithms. By going through all the experimental results and their analysis, it is concluded that the proposed algorithm is efficient. In all existing algorithms, first complete list is entered, then the list is processed for sorting, but in case of proposed approach, the list is sorted simultaneously. The proposed sorting technique saves the time for traversing the list after entering all the elements, as it sorts all elements before entering any new.

REFERENCES

[1] Cormen T., Leiserson C, Rivest R., and Stein C, Introduction to Algorithms, McGraw Hill, 2001.

[2] D. Knuth, “The Art of Computer programming Sorting and Searching”, 2nd edition, Addison-Wesley, vol. 3, (1998).

[3] Deitel H. and Deitel P., C++ How to Program, Prentice Hall, 2001.

[4] E. Horowitz, S. Sahni and S. Rajasekaran, Computer Algorithms, Galgotia Publications

[5] Harish Rohil , Manisha, International Journal of Computer Trends and Technology (IJCTT) volume 14 number 1 – Aug 2014 ISSN:2231–2803 “Run Time Bubble Sort – An Enhancement of Bubble Sort”. http://www.cs.fit.edu/~pkc/classes/writing/hwl3/song.pdf

[6] Jehad Alnihoud and Rami Mansi, “An enhancement in Major Sorting Algorithms,” The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010.

[7] M. A. Bender, M. Farach-Colton and M. A. Mosteiro, “Insertion Sort is O(n log n)”, Proceedings of the Third International Conference on Fun With Algorithms (FUN), (2004), pp. 16–23

[8] Mansotra and Kr. Sourabh, “Implementing Bubble Sort Using a New Approach,” in proceedings of 5th National Conference; INDIACom-2011.

[9] Md. Khairullah “ Enhancing Worst Sorting Algorithms” International Journal of Advanced Science and Technology Vol. 56, July, 2014

[10] S. Chand, T. Chaudhary and R. Parveen, “Upgraded Selection Sort”, International Journal on Computer Science and Engineering (IJCSE), ISSN: 0975-3397, vol. 3, no. 4, (2011), pp.1633–1637.

[11] S. Jadoon, S. F. Solehria, S. Rehman and H. Jan, “Design and Analysis of Optimized Selection Sort Algorithm”, International Journal of Electric & Computer Sciences (IJECS-IJENS), vol. 11, no. 01, pp. 16–22.

[12] Shubham Saini, Bhavesh Kasliwal. “MODIFIED MERGE SORT ALGORITHM”. International Journal For Technological Research In Engineering Vol. 1, Issue. 1, Sep–2013

[13] Song Qin. Merge Sort Algorithm. Florida Institute of Technology.

[14] T. S. Sodhi, S. Kaur and S. Kaur, “Enhanced Insertion Sort Algorithm”, International Journal of Computer Applications, vol. 64, no. 21, (2013), pp. 35–39.