International Journal of Engineering Research & Management Technology

ISSN: 2348-4039

September-2017 Volume 4, Issue-5

Email: editor@ijermt.org www.ijermt.org

# ENHANCEMENT AND ANALYSIS OF SORTING ALGORITHMS

**Lal Singh Dhaila,**
SRM University, Ghaziabad

**ABSTRACT:**
Sorting algorithm is one of the most fundamental concepts to arrange the items in a proper way. Since a long time several research is made on how to minimize sorting time using different algorithms. Sorting language is important part of the programming language to make the program execute at mush more faster rate and so in daily life it has its own impact. Searching for a particular data in a sorted list makes process of sorting more affable as compared to search data in an unsorted database. Sorting algorithm and techniques are used to make the data search complexity easier and to make the large corporates store the data safely and generate the report in minimum amount of time. Several researches have been made in this area to make the sorting algorithm easier and fastest. Algorithms have been develop to minimize the time complexity of the code that have been used for the sorting purpose and to increase the quality of code that have been designed for a specific purpose.

**KEYWORDS:**
Selection sort, bubble sort, insertion sort, quick sort, merge sort, number of swaps, time About four key words or phrases in alphabetical order, separated by commas.

## INTRODUCTION

Sorting algorithm can be used to arrange the different item in an arranged way in some list. Most of the orders that are used is numerical order and lexicographical order. Sorting algorithm is used for efficient sorting of the input data in an ordered list and which can be used as for efficient report generation. The output of the sorting must satisfy the two criteria:

1. Non-Decreasing output of the order.
2. Permutation is the output of the order.

Further, the data is often taken to be in an array, which allows random access, rather than a list, which only allows sequential access, though often algorithms can be applied with suitable modification to either type of data.

Since the starting of computing, there have been various research happened in which is in turn because of the complexity of the sorting algorithm, several thesis have been proposed for better sorting algorithm. Taking as an example Bubble Sort was analyzed at 1956. Bubble sort as we've identified it is called a ``push-down'' sort. Bubble sort is considered as one of the simplest program to code. A fundamental limit of comparison sorting algorithms is that they require linear arrhythmic time – O(n log n) – in the worst case, though better performance is possible on real-world data (such as almost-sorted data), and algorithms not based on comparison, such as counting sort, can have better performance. Bubble sort has the worst case and average complexity both $O(n^2)$, where n defines the number of items being sorted.

Sorting algorithms are widespread algorithms in computer science ,this provides a widespread medium to calculate and develop a feasible sorting algorithm that can be used in various other fields which in turn can help in developing more user friendly software's. Several other things have been introduced that have been used to maintain the genuinely and accountability of the software being developed concepts, such as big O notation, divide and conquer algorithms, data structures such as heaps and binary trees, randomized algorithms, best, worst and average case analysis, time-space tradeoffs, and upper and lower bounds.

J-1 passes are carried out in classical selection sort algorithm against sorting of the J elements in a data list. To excute this whole list is checked first from the starting element to the last elements present in the list, comparison have been done to check which element is small and which one is large, so as to change the location of the specific element and changing the location of the element will give us the sorted array in which we can easily do the sorting. But the other logic is instead a to get the whole iteration done the minimum element location is find out or the value of the element at that specific element is noted and compared with the rest of the element in the list to retrieve the minimum or maximum element in the list.

The enhanced selection sort algorithm is based on this idea. The bubble sort also places the largest element in the proper location in each pass. As the bubble sort and selection sort are closely analogous, the enhancement of the bubble sort is done with the same method. In the classical insertion sort a sorted portion is maintained and in each pass of the algorithm a data item from the unsorted portion is inserted into the sorted portion from a certain side such that with the additional item it remains sorted. Considering just one side to insert leads many shift operations, which can be reduced if both sides of the sorted list is considered to insert a data item. The enhanced insertion sort incorporates this strategy

## SORTING ALGORITHMS

### A. SELECTION SORTING ALGORITHM

Selection sort algorithm works on the base of finding the minimum element in an array and repeatedly loop around the list of unsorted array to find the less element than the current. Once the element with the less numeric value found the position of that element is shifted to the first and that the larger number is shifted to the one position next to the smallest element found in first iteration.

This is algorithm is used by maintaining the two subarrays in a given array:

a. One subarray which is already sorted
b. For unsorted element there is another subarray
c. Below is one simple code for the sorting of the elements in an array using selection sort algorithm.

// Java program for slection sort

```
S:1 class SortingSelection
S:2 {
S:3 void sort(int arrsort[])
S:4 {
S:5 int n = arrsort.length;
S:6
S:7 // One by one move boundary of unsorted      subarray
S:8 for (int l = 0; l < n-1; l++)
S:9 {
S:10 // Find the minimum element in unsorted array
S:11 int minel = l;
S:12 for (int j = l+1; j < n; j++)
S:13 if (arrsort [j] < arrsort [minel])
S:14 minel = j;
S:15
S:16 // Swap the found minimum element with the first
S:17 // element
S:18 int temp = arrsort [minel];
S:19 arrsort [minel] = arrsort [l];
S:20 arrsort [l] = temp;
S:21 }
S:22 }
S:23
S:24
S:25 void printArray(int arrsort [])
S:26 {
S:27 int n = arrsort.length;
S:28 for (int l=0; l<n; ++l)
S:29 System.out.print(arrsort [l]+" ");
S:30 System.out.println();
S:31 }
S:32
S:33
S:34 public static void main(String args[])
S:35 {
S:36 SelectionSort obsort = new SelectionSort();
S:37 int arrsort[] = {64,25,12,22,11};
S:38 obsort.sort(arrsort);
S:39 System.out.println("Array Sorted Order ");
S:40 obsort.printArray(arrsort);
S:41 }
S:42 }
```

Selection sort spends most of the time in finding the minimum element in the unsorted part of the list.

**TIME COMPLEXITY:**

Sorting based on input given , Suppose the size of list be as N here. In selection sort number of comparison is used as measure of number of time loop need to run to make the list sorted. In above program it is clearly seen that the outer loop is is always executed but the internal loop execution is always uncertain as it always depends whether the value in that element is equal or not .Well can be concluded that the complexity is always situational and doesn't have a fixed value.

**BEST CASE:**

The code of line S:8 will execute once time as compared to that of the S:12 coding line and hence, T(n) = n-1 = O(n). The best-case complexity of the classical selection sort is O(n2).

***Worst case*:** In the iteration number j when the number of j-1 elements of the list is already sorted at v that time n-j+1 number of the elements of the list are still in an unsorted order , at that time only j-k comparisons will be required to find the largest number of element in the given list of an array. Hence in this case we can deduce that the worst case scenario can be the following

T(n) = (n-1) + (n-2) +(n-3) + ……. + 3 + 2 + 1 = n(n-1)/2 = (n2-n)/2 = O(n2).

**AVERAGE CASE**:

There may be the probability that half of the number present in the list may be smaller than the number of remaining half elements. Now here the when it will be the iteration J ,n-j+1 item s of the array will already be sorted out , in such case only (n-J)/2 comparison will be required to sort the array. This leads to T(n) = ((n-1) + (n-2) + (n-3) + ……. + 3 + 2 + 1)/2 = n(n-1)/4 = (n2-n)/4 = O(n2).

**SPACE COMPLEXITY**:

Enhanced sorting algorithm mostly used stack for sortin purposes. In this case on scenario which can be sai as the worst scenario will be the one in which the size of the stack can be as large as the size of the data array to be sorted. Hence O(n) is the propose complexity of this algorithm. In such case the memory requirement in this case will be the double of that classical algorithm.

## B. BUBBLE SORT ALGORITHM

Bubble sort work on the belief of changing the order of the element until whole elements in the array is sorted out and when there is no more swapping of the elements is happening that case can be considered as the case in which all the elements of the array is considered to be sorted and array will be in a sorted form. Bubble sort is considered as one of the simplest type of sorting algorithm that can be used for sorting purposes.

**Algorithm: Sequential-Bubble-Sort (A)**

```
fori← 1 to length [A] do
for j ← length [A] down-to i +1 do
  if A[A] < A[j - 1] then
    Exchange A[j] ↔ A[j-1]
```

Below is the advanced sorting algorithm better than then normal algorithm. Steps and algorithm to reproduce the sorting code is mentioned below :

1. First we have to insert all the elements in array that have to be sorted.
2. Define the variable and initialize the variables .We will initialize two variable at the starting where the firstele will be initialized with 0 and lastele will be initialized with the size of the index -1.
3. Once the variable will be initialized we will call the function mentioned in below code “SortingBubble”, followed by passing the size of the list of the array that have to be sorted and followed by passing first and last element index numbers . All these three values will be passed as parameters to the function.
4. The role of the function “SortingBubble” is to find the minimum value element in the list and to find the maximum element in the list and swap them in order to form a sorted ordered list.
5. In the "Enhanced Bubble Sort" function, the operation now is to find the maximum and the minimum elements and saving the index value of the max of the array in the variable maxcounter, and the index value of the min in the variable mincounter.
6. Place the maximum value in the lastele and at the same time place the minimum element in the firstele of the array, saving the value at first index and the last index in some variable which will be used in later stages .
7. Start decreasing the lastele value by one and on the other hand increase the values of firstele by one . During this process the size of the array while calling the first time will be size-2 and will repeatedly decreased with 2 .
8. Calling the " SortingBubble " array again and again till the size of the array will be greater than one. Once this will b e achieved we will return the sorted array.

```
S:1 Function SortingBubble(array, sizearr, firstele ,lastele )
S:2 if sizearr > 1 then
S:3 var temp := 0, maxctrc := lastele
S:4 var minctrc := firstele
```

```
S:5 var max := array(lastele ),min := array(firstele )
S:6 for a:= firstele  to lastele  do
S:7 if array(a) ≥  max then
S:8 max := array(a)
S:9 maxctrc := a
S:10 end if
S:11 if array(a) < min then
S:12 min := array(a)
S:13 minctrc := a
S:14 end if
S:15 end for
S:16 if firstele ==maxctrc AND
S:17 astindex==minctrc then
S:18 array(firstele ):= min
S:19 array(lastele ) := max
S:20 else
S:21 if firstele ==maxctrc AND
S:22 lastele  ≠  minctrc then
S:23 temp := array(lastele )
S:24 array(lastele ) := max
S:25 array(firstele ) := min
S:26 array(minctrc) := temp
S:27 else
S:28 if firstele  ≠  maxctrc AND
S:29 lastele ==minctrc then
S:30 temp := array(firstele )
S:31 array(firstele ):= min
S:32 array(lastele ) := max
S:33 array(maxctrc):= temp
S:34 else
S:35 temp := array(firstele )
S:36 array(firstele ):= min
S:37 array(minctrc):= temp
S:38 temp := array(lastele )
S:39 array(lastele ):= max
S:40 array(maxctrc):= temp
S:41 end if
S:42 end if
S:43 end if
S:44 firstele  := firstele  + 1
S:45 lastele  := lastele  - 1
S:46 sizearr := sizearr – 2
S:47 return SortingBubble
S:48 (array,sizearr,firstele ,lastele )
S:49 else return array
S:50 end if
```

## RUN-TIME COMPLEXITY ANALYSIS

Bubble sort unlike selection sort can terminate early. The bubble sort will not run more than n time it is so because the outer loop in the program will not run more than n times mean it will not run more than n iterations.

```
   S:1  def bubble_sort(L):
   S:2  for i in range(len(L)):
   S:3  swapped = False
   S:4  for j in range(len(L)-1-i):
   S:5  if L[j] > L[j+1]:
   S:6  L[j], L[j+1] = L[j+1], L[j]
   S:7  print("Successfully swapped ", L[j], "and", L[j+1])
   S:8  print(L)
   S:9  swapped = True
   S:10 else:
   S:11 print("Swapped doesn’t required at this point of time                    ", L[j], "and",
L[j+1])
   S:12 print(L)
   S:13if the swipping is not take place:
   S:14 return
   S:15
   S:16 print("====================================")
   S:if main
   S:18 L = [2, 3, 4, 5, 1]
   S:19 bubble_sort(L)
```

```
Swapped doesn’t required at this point 2 and 3
[2, 3, 4, 5, 1]
Swapped doesn’t required at this point 3 and 4
[2, 3, 4, 5, 1]
Swapped doesn’t required at this point 4 and 5
[2, 3, 4, 5, 1]
Swapped 1 and 5
[2, 3, 4, 1, 5]
====================================
Swapped doesn’t required at this point 2 and 3
[2, 3, 4, 1, 5]
Swapped doesn’t required at this point of time                    3 and 4
[2, 3, 4, 1, 5]
Successfully swapped 1 and 4
[2, 3, 1, 4, 5]
====================================
Swapped doesn’t required at this point 2 and 3
[2, 3, 1, 4, 5]
Successfully swapped 1 and 3
[2, 1, 3, 4, 5]
====================================
Successfully swapped 1 and 2
```

```
[1, 2, 3, 4, 5]
=====================================
```

We can therefore conclude that the in the **worst case**, Bubble Sort does not return before performing all n iterations of the outer loop.

Let's now figure out the worst-case runtime complexity of Bubble Sort for a list of length n by counting how many times the inner block repeats.

At iteration 0, the block runs for n-1-0 times At iteration 1, the block runs for n-1-1 times At iteration n-1, the block runs for n-1-(n-1)=0 times

So in total, the block runs

∑n− 1i=0(n− i− 1)∑i=0n− 1(n− i− 1) times.
∑n− 1i=0(n− i− 1)=n2− ∑n− 1i=0i− n=n2− n(n− 1)/2− n=n2− n2/2+n/2− n=n2/2− n/2∑i=0n− 1(n− i− 1)=n2 − ∑i=0n− 1i− n=n2− n(n− 1)/2− n=n2− n2/2+n/2− n=n2/2− n/2

(To compute ∑n− 1i=0i∑i=0n− 1i, we use the fact that ∑mj=1j=m(m+1)/2∑j=1mj=m(m+1)/2, and substitute m=n− 1m=n− 1.)

We can therefore conclude that the worst-case runtime complexity of Bubble Sort if O(n2)O(n2), just like Selection Sort.

## CONCLUSION

The above described technique and enhancement propose above have proposed a great enhancement over the classical algorithm. We have achieved this notch progress by elimination the comparison that were unnecessary and not required during the whole process, this type of unwanted requirement are deadlock while doing the sorting for a particular program or while developing the specific software product. This technique can save lots of time and cost of the client during the process of software development. This research shows that most of the comparison that have been dine during the process of sorting an unordered list can be avoided and hence this can help in developing software that will be much more efficient and countable.

## ACKNOWLEDGMENT

The preferred spelling of the word “acknowledgment” in American English is without an “e” after the “g.” Use the singular heading even if you have many acknowledgments. Avoid expressions such as “One of us (S.B.A.) would like to thank ... .” Instead, write “F. A. Author thanks ... .” **Sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page**.

## REFERENCES

1. G. O. Young, “Synthetic structure of industrial plastics (Book style with paper title and editor),” in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.
2. W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.
3. H. Poor, An Introduction to Signal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.
4. Smith, “An approach to graphs of linear forms (Unpublished work style),” unpublished.
5. H. Miller, “A note on reflector arrays (Periodical style—Accepted for publication),” IEEE Trans. Antennas Propagat., to be published.
6. J. Wang, “Fundamentals of erbium-doped fiber amplifiers arrays (Periodical style—Submitted for publication),” IEEE J. Quantum Electron., submitted for publication.
7. C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.
8. Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interfaces(Translation Journals style),” IEEE Transl. J. Magn.Jpn., vol. 2, Aug. 1987, pp. 740–741 [Dig. 9th Annu. Conf. Magnetics Japan, 1982, p. 301].
9. M. Young, The Techincal Writers Handbook. Mill Valley, CA: University Science, 1989.

10. J. U. Duncombe, “Infrared navigation—Part I: An assessment of feasibility (Periodical style),” IEEE Trans. Electron Devices, vol. ED-11, pp. 34–39, Jan. 1959.
11. S. Chen, B. Mulgrew, and P. M. Grant, “A clustering technique for digital communications channel equalization using radial basis function networks,” IEEE Trans. Neural Networks, vol. 4, pp. 570–578, July 1993.
12. R. W. Lucky, “Automatic equalization for digital communication,” Bell Syst. Tech. J., vol. 44, no. 4, pp. 547–588, Apr. 1965.
13. S. P. Bingulac, “On the compatibility of adaptive controllers (Published Conference Proceedings style),” in Proc. 4th Annu. Allerton Conf. Circuits and Systems Theory, New York, 1994, pp. 8–16.
14. G. R. Faulhaber, “Design of service systems with priority reservation,” in Conf. Rec. 1995 IEEE Int. Conf. Communications, pp. 3–8.
15. W. D. Doyle, “Magnetization reversal in films with biaxial anisotropy,” in 1987 Proc. INTERMAG Conf., pp. 2.2-1–2.2-6.
16. G. W. Juette and L. E. Zeffanella, “Radio noise currents n short sections on bundle conductors (Presented Conference Paper style),” presented at the IEEE Summer power Meeting, Dallas, TX, June 22–27, 1990, Paper 90 SM 690-0 PWRS.
17. J. G. Kreifeldt, “An analysis of surface-detected EMG as an amplitude-modulated noise,” presented at the 1989 Int. Conf. Medicine and Biological Engineering, Chicago, IL.
18. J. Williams, “Narrow-band analyzer (Thesis or Dissertation style),” Ph.D. dissertation, Dept. Elect. Eng., Harvard Univ., Cambridge, MA, 1993.
19. N. Kawasaki, “Parametric study of thermal and chemical nonequilibrium nozzle flow,” M.S. thesis, Dept. Electron. Eng., Osaka Univ., Osaka, Japan, 1993.
*20.* Pandey A.,Bansal K.K.(2014): “Performance Evaluation of TORA Protocol Using Random Waypoint Mobility Model” *International Journal of Education and Science Research Review* Vol.1(2)
*21.* Tiwari S.P.,Kumar S.,Bansal K.K.(2014): “A Survey of Metaheuristic Algorithms for Travelling Salesman Problem ” *International Journal Of Engineering Research & Management Technology* Vol.1(5)
22. G. W. Juette and L. E. Zeffanella, “Radio noise currents n short sections on bundle conductors (Presented Conference Paper style),” presented at the IEEE Summer power Meeting, Dallas, TX, June 22–27, 1990, Paper 90 SM 690-0 PWRS.
23. J. G. Kreifeldt, “An analysis of surface-detected EMG as an amplitude-modulated noise,” presented at the 1989 Int. Conf. Medicine and Biological Engineering, Chicago, IL.
24. J. Williams, “Narrow-band analyzer (Thesis or Dissertation style),” Ph.D. dissertation, Dept. Elect. Eng., Harvard Univ., Cambridge, MA, 1993.
25. Flores, “Analysis of Internal Computer Sorting”, ACM, vol. 7, no. 4, (1960), pp. 389- 409.
26. G. Franceschini and V. Geffert, “An In-Place Sorting with O(n log n) Comparisons and O(n) Moves”, Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science, (2003), pp. 242-250.
27. O. O. Moses, “Improving the performance of bubble sort using a modified diminishing increment sorting”, Scientific Research and Essay, vol. 4, no. 8, (2009), pp. 740-744.
28. Knuth, “The Art of Computer programming Sorting and Searching”, 2nd edition, Addison-Wesley, vol. 3, (1998).
29. Kapur, P. Kumar and S. Gupta, “Proposal of a two way sorting algorithm and performance comparison with existing algorithms”, International Journal of Computer Science, Engineering and Applications (IJCSEA), vol. 2, no. 3, (2012), pp. 61-78.
30. W. Thimbleby, “Using Sentinels in Insert Sort”, Software-Practice and Experience, vol. 19, no. 3, (1989), pp. 303-307.
31. Bingo Sort, http://xlinux.nist.gov/dads/HTML/bingosort.html.
32. Exact-Sort, http://www.geocities.ws/p356spt/.
33. https://www.tutorials point.com
34. Shell D., “A High Speed Sorting Procedure,”Computer Journal of Communications of the ACM, vol. 2, no. 7, pp. 30-32, 1959.
35. Thorup M., “Randomized Sorting in O(n log logn) Time and Linear Space Using Addition,Shift, and Bit Wise Boolean Operations,”Computer Journal of Algorithms, vol. 42, no. 2, pp. 205-230, 2002.
36. Aho A., Hopcroft J., and Ullman J., The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.
37. Astrachanm O., Bubble Sort: An Archaeological Algorithmic Analysis, Duk University, 2003.
38. Deitel H. and Deitel P., C++ How to Program, Prentice Hall, 2001.