www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 6 Issue 4 April 2017, Page No. 21006-21010
Index Copernicus value (2015): 58.10 DOI: 10.18535/ijecs/v6i4.37

Minimising the Execution Time of Selection Sort Algorithm

Manish Kumar, Ms. Monika Malhotra, Ms. Deepali Ahuja
Research Scholar (CSE)
MD University Rohtak, Haryana, India
manishdahiya2008@gmail.com
Assistant Professor (CSE)
WCTM Gurugram
Assistant Professor (CSE)
WCTM Gurugram

ABSTRACT:
Sorting has made their importance in computing field as well as the applications we face in real life. So far many algorithms come into existence in current era. In this research paper we present a new sorting algorithms i.e. Enhanced division Selection sort. We try to enhance the performance of Selection sort by doing some modifications in the pseudo code. A comparison also has done to do analysis between existing Selection Sort and Enhanced division Selection Sort. After comparison we found that the new algorithm has produced better results and good performance during the running time.

Keywords: Algorithm, Sort, Selection, Research, Optimize, Developments, Experiment, Enhanced.

I. INTRODUCTION

The importance of the information is very essential in today time of the world. The information should be ordered in a sensible manner. Many years back, it was estimated that much of the time was spent in sorting. But now, that is not true because many complex methods come into existence for processing data before its use. Sorting a list of items is a very basic problem in computer science. Organizing the randomly scattered elements in a particular order is called sorting. There are very well known algorithms for sorting purpose are present for sorting unsorted lists. Some algorithms are simple and some are complex in nature. The Process of sorting having a lot of importance because of various reasons

- Sorting information is essential when an application needs that information.
- Some algorithms use sorting algorithms as a support.
- In computing world, issues rises when sorting algorithms are implement.

Efficiency is always important during sorting process. This field always attracts researchers since the beginning of computing. Many researchers considered that sorting is a solved issue. But still algorithms got discovered. Each sorting algorithm has their own working environments. Some are suitable for solving large lists; some are suitable for small lists. There are various factors that must be included for sorting algorithms.

- Availability of memory that is used for sorting operations. This is called space complexity.
- Number of exchanges of elements of list under sorting
- Recursion: some sorting algorithms are recursive in nature and some are non recursive.
- Stability.

In this paper a new sorting algorithm is presented. That is enhanced division Selection sort. The algorithm is enhancement over the classical selection sort. The enhanced sorting algorithm is

more efficient, tested and reliable compared to the classical selection sort.

II. LITERATURE REVIEW

Developments in year 2010

In Research Paper[1] enhancements in selection sort are discovered by Rami Mansi and Jehad Alnihoud. ESS(Enhanced Selection Sort) is faster than selection sort and has O(n2) complexity, when the given elements array is stored in memory of secondary type like Hard disk, Tape drive.

Developments in year 2012

Arora Nitin, Kumar vivek and Kumar Suresh in Research paper[2] published Counting position sorting and novel sort algorithm. The Counting position sort works as each element position is count first in the given array. The smaller elements are counted and their position is then fixed in the array.

In Research paper[3], Wasi Haider Butt and Abdul Wahab discovered Concatenate sort and relative Split algorithm. They compared the algorithm with other algorithm and it is implemented.

Developments in year 2013

Ms Khairullah in Research paper[4] proposed new enhanced selection sorting algorithm. In this paper the selection sort is enhanced by remembering intermediate points using variables, by which the position is stored up to which no swapping is taking place and finding a largest up to this position then this maximum element is placed just before the position at which swap in taking place. So in next iteration the comparison starts from here. For insertion sort, a slight different approach is applied. A new blank array is taken and an element placed at first position in array is removed and shifted to the centre of the new blank array then one by one a element is removed and inserted in new array using both the ends. If an element is greater than element inserted at the centre , in that case the element will be shifted and placed at the back side in the array otherwise the element is placed before the centre element from the front end.

Sonal Beniwal and Deepti Grover in research paper[5] presented their views on comparison between various sorting algorithms.

Pankaj Saren in his research paper[6] also shows the comparison between sorting algorithms on average cases

Hence in this way various enhancements had been taking place and continue to enhance the performance of these algorithms.

Classical working of selection sort algorithm is as follows

Selection Sort

Selection sort algorithm works by selecting smaller element in each pass and placed one after the other in increasing order starting from the front side. The smallest element is placed at the first position and then the next higher element and likewise the rest of the elements secured their positions in the array. At the end the array attain its sorted form contains sorted elements. It is also simple sorting algorithm.

III. EXISTING SELECTION SORT

In Existing Selection Sort the array got sorted by selecting smallest element and shifted at the first position by exchanging of elements if required. It also uses two loops in pseudo code to sort the list of items. The pseudo code is as follows

SELECTION SORT(A)

n ← length[A]

for j ←1 to n-1

smallest ← j

for I ← j+1 to n

if A[i] < A[smallest]

then smallest ← i

exchange ([A], A[smallest])

For example, if the Selection Sort were used on the array, 6, 4, 3, 2, 7, 8, 5, 9. Each pass would be like as shown in Table 1:

Pass-1	**2**	**4**	**3**	**6**	**7**	**8**	**5**	**9**
Pass-2	**2**	**3**	**4**	**6**	**7**	**8**	**5**	**9**
Pass-3	**2**	**3**	**4**	**6**	**7**	**8**	**5**	**9**
Pass-4	**2**	**3**	**4**	**5**	**7**	**8**	**6**	**9**
Pass-5	**2**	**3**	**4**	**5**	**6**	**8**	**7**	**9**
Pass-6	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Pass-7	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**

Table-1. Selection sorting for input values 6,4,3,2,7,8,5,9

In Selection sort the number of comparisons is always the same and the two loops will run for the specified number of times whether the given array is sorted or partially sorted.

IV. PROBLEM STATEMENT

The main drawback of Selection sort is the number of useless comparisons that are performed even after the array got sorted and a lot of comparisons for fixing only one element at a time. If the array is big the comparisons are even more making it impractical for handling large data set.

V. OBJECTIVE

Our objective is to avoid these useless comparisons and reduce the time complexity of the algorithm.

VI. PROPOSED ENHANCED DIVISION SELECTION SORT ALGORITHM

In this algorithm, sorting is performed in three phases.

Phase-1: In this phase the given array is searched for the smallest and largest element and their position is stored using variables. When the whole array is scanned then a mid value is calculated by using smallest and largest element. i.e.

(Smallest+largest)/2.

Phase-2: In this phase, the array is divided into two parts using this mid value calculated during first phase. The elements which are lower or equal to mid value are placed in the part start from front end. The elements which are larger than this mid value are placed in the other portion. The first part of array is up to that point where the last element which is less or equal to mid value.

Phase-3: In this phase the two arrays are sorted separately using classical selection sort algorithm. One sub array in increasing order and the other in decreasing order.

ALGORITHM

PROPOSED_SELECTION_SORT(A[ ] , N)

```
1:  Initialize  s, g
2:  Repeat For loop from i=1 to i<=n
3   if((i==1) && (A[i]>A[i+1])) then
      s=i+1  and  g=i
    else if((i==1) && (A[i]<A[i+1])) then
s=i and g=i+1
else if(A[i]<=A[s]) then
s=i
else if(A[i]>=A[g]) then
g=i
swap A[1] and A[s]
End For loop
4.  initialize m= (A[s]+A[g])/2 and b=n
5.  Repeat For loop from i=1 to n-1
              if(A[i]>m)  then
                    if(i==b) then
           b=i-1
           break outer For loop
  Repeat For loop from j=b to  j>=i
                  if(A[j]<=m)  then
swap A[i] and A[j]
               b=j-1
                  break inner For loop
                  else if(i==j-1)
                  b=b-2
                  break inner For loop
                  End inner For loop
                  if(i==b) then
                  break outer For loop
End outer For loop
6.     Repeat For loop from  i=2 to i<=b-1
          h=i
       Repeat For  loop from  j=i+1  to  j<=b
      if(A[j]<=A[h]) then
      h=j
    End inner For loop
     swap A[i] and A[h]
     End outer For loop
7.    Repeat For loop from   i=n to i>=b+2
```

```
    h=i
Repeat For loop from   j=i-1  to   j>=b+1
 if(A[j]>=A[h]) then
  h=j
End inner For loop
 swap A[i] and A[h]
 End outer For loop
9.  Stop
```

By dividing the array into two parts using the mean value which is calculated by using of smallest and largest elements. We reduce the number of comparisons between elements since the two sub arrays got sorted separately and there is no need to do the merging as the division is performed like that merging is not required at all. In the average case, the number of comparisons and exchanging of elements are always less compared to normal selection sort.

Let us take the previous example

The given array is as 6, 4, 3, 2, 7, 8, 5, 9.

Phase-1

The value of s=2, g=9 and m=(9+2)/2=5

Then the array formed after replacing the smallest element to 1st position and largest element to last position is as shown below

2	4	3	6	7	8	5	9

Phase-2

The array divided into two parts. Each element is compared with the value of m and placed to front side if it is less or equal to m and placed towards end side otherwise. The array becomes

1st part

2	4	3	5	7	8	6	9

2nd part

Phase-3

The array then sorted in two parts separately from 1st to 4th elements and from 5th to 8th elements. The first sub array sorted in three passes as shown in Table-2.

Pass-1	2	4	3	5
Pass-2	2	3	4	5
Pass-3	2	3	4	5

Table-2: Enhanced division selection sort for 1st part of array

The 2nd part got sorted in three passes as shown I Table-3

Pass-1	6	7	8	9
Pass-2	6	7	8	9
Pass-3	6	7	8	9

Table-3: Enhanced division selection sort for 2nd part of array

VII. IMPLEMENTATION

In order to test this proposed algorithm for its efficiency the algorithm was implemented in C language on Turbo C++ with operating system window 7. But it was sufficient to sort less no of data item up to 15000 because of memory problem.

VIII. ANALYSIS AND RESULTS.

Both algorithms are compared on the same elements of unordered list. In order to make a comparison of the proposed algorithms with the existing Selection sort, a number of tests were conducted for small as well as large number of elements. A Comparative study of execution time for the number of inputs is shown in tabular form as in Table-4 below

No of Elements Given as input for sorting	Time taken by Selection sort (in sec)	Time taken by Optimized Division Selection sort (in sec)	Difference (in sec)
1000	0.05	0.00	0.05
2000	0.16	0.05	0.11
3000	0.43	0.16	0.27
4000	0.54	0.21	0.33
5000	0.60	0.32	0.28
6000	0.82	0.38	0.44
7000	1.26	0.49	0.77
8000	1.42	0.54	0.88
9000	1.86	0.60	1.26
10000	2.08	0.60	1.48

Table-4: Comparative study of execution time of Selection Sort and Enhanced Division Selection Sort

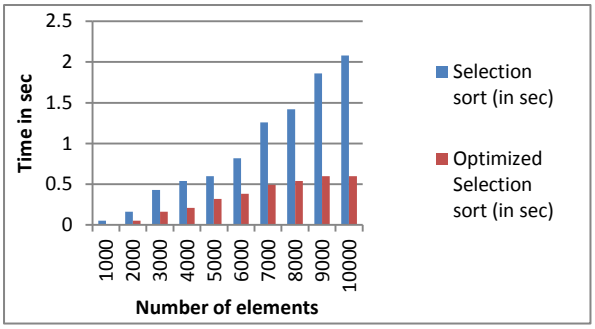

Fig-1: Number of inputs vs. CPU time in (sec)

In Fig-1, the X Axis shows the number of elements and the Y Axis shows the time elapsed in seconds.

IX. CONCLUSION

In this research paper we try to find the issues related to selection sort and earlier work performed in this field. After doing detailed experiment in this field and analyze the various results we reach to conclusion that enhanced selection sort algorithm is efficient and giving better results. The complexity is much better than the existing Selection sort algorithm even in worst cases. In average case also, the complexity is always less than $O(n^2)$. The field of sorting is so vast that still more research will take place.

REFERENCES

[1] Jehad Alnihoud and Rami Mansi, *"An Enhancement of Major Sorting Algorithms,"* The International Arab Journal of Information Technology, Vol.7, No. 1, January 2010.

[2] Arora Nitin, Kumar vivek and Kumar Suresh. *"A Novel Sorting Algorithm and Comparison with Bubble Sort and Insertion Sort,"* International Journal of Computer Applications (0975-8887) vol. 45, No. 1, May 2012.

[3] Abdul Wahab Muzaffar, Naveed Riaz, Juwaria Shafiq and Wasi Haider Butt, " *Relative Sp lit and Concatenate Sort (RSCS-VI)",* International Journal of Computer Theory and Engineering vol. 4, No. 2, April 2012.

[4] Md. Khairullah " *Enhancing Worst Sorting Algorithms*" International Journal of Advanced Science and Technology Vol. 56, July, 2013.

[5] Sonal Beniwal, Deepti Grover, *"Comparison of various sorting algorithms"* , International Journal of Emerging Research in Management &Technology ISSN: 2278-9359 (Volume-2, Issue-5) , May 2013

[6] Sareen Pankaj, *" Comparison of Sorting Algorithms (On the Basis of Average case)"*, International Journal of Advanced Research in Computer Science and software Engineering ISSn: 2277128x, volume 3, Issue 3, March 2013, pp. 522-532