Volume 5, Issue 1, January 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Avi Selection Sort Algorithm

Avinash Bansal
Assistant Professor (CSE) GNIT,
Mullana, Ambala (Haryana), India

Abstract— In the case of rearranging an array with N elements either in ascending or in descending order; we find that, sorting algorithms such as the Bubble, Insertion and Selection Sort have a quadratic time complexity. In this paper, we introduce Avi Selection sort – a new algorithm to sort N elements array by selecting two data elements, smallest and largest simultaneously. We evaluate time complexity $O(N^2)$ of Avi Selection sort theoretically and empirically. Amongst the three, Insertion sort is the fastest [2]. Our results show a different way to sort the data elements in quadratic time complexity and experimentally prove that its actual time is much faster than insertion sort in average case.

Keywords — Sorting, Avi, Time Complexity $O(N^2)$, Selection.

I. INTRODUCTION

The process of arranging the data elements of an array either in ascending or in descending order is called sorting [1]. Explore engine / Search box uses sorting algorithm. When we explore some key word online / offline, the feedback information is brought to us sorted by the importance of the web page online and as per the data in our local machine offline [2]. The sorting techniques - Bubble, Selection and Insertion Sort, have $O(N^2)$ time complexity. In this paper we introduce Avi Selection Sort which is also able to rearrange data elements of an array in ascending / descending order. Avi Selection sort works as selection sort algorithm. In this process, simultaneously we find the smallest and the largest data elements of the array and place them at first and largest position respectively. Similarly, in next iteration we find the second smallest and second largest element of the array and placed them at 2^{nd} and 2^{nd} last position respectively. In the same way whole data elements must be in a sorted order. Our main motive is to introduce Avi Selection Sort as one more efficient algorithm that can sort a list of array elements in $O(N^2)$ time. We evaluate the $O(N^2)$ time complexity theoretically and practically. Rather than the quadratic nature of complexity some sorting algorithms like Merge, Quick and Heap sort have $O(N \log_2 N)$ complexity which is the minimum complexity on RAM's model to sort the data. There are some more algorithms which have O(N) time complexity but these are not based on RAM's model / Comparison method e.g. Radix, Bucket and Counting sort which are dependent on the nature of data elements or occurrence of data elements [3].

Time complexities of different sorting algorithms are represented in Table I. In Table I under 'order' column, down arrow show that if we move down complexity will increases. In next section we describe some existing sorting algorithms: - Bubble Sort, Insertion Sort and Selection Sort in brief.

TABLE I SORTING ALGORITHM COMPLEXITY

S. No.	Sorting Algorithm	Time Complexity	Order
1.	Radix Sort Algorithm Bucket Sort Algorithm Counting Sort Algorithm	O (N) [3]	↓
2.	Merge Sorting Algorithm Quick Sort Algorithms Heap Sort Algorithm	O ($N \log_2 N$) [3]	
3.	Bubble Sort Algorithm Insertion Sort Algorithm Selection Sort Algorithm	O (N^2) [3]	

II. RELATED WORK

A. *Bubble Sort*

Bubble sort [4] works in the following process: We keep passing through the list, exchanging adjacent element, if the list is in unordered form; when the list is in sorted order, no exchanges of data elements are required.

B. Insertion Sort

Insertion Sort [5] works as follows: We insert an element into its proper place in the previous sub-list. For the iteration, we identify two regions, sorted region and unsorted region. We take one element from the unsorted region and "insert" it in the sorted region. The elements in the sorted region will increase by one after the iteration. We repeat this on the rest of the unsorted region without the first element.

C. Selection Sort

Selection sort [6] works as following: It is based on finding the smallest element in the list and placing it at the first position. Then the next smallest element is found and placed at the second position and so on until we are left with one element.

Amongst the three, Insertion sort is the fastest [2].

III. PROPOSED WORK

There are many sorting algorithm which sort the data either in increasing or in decreasing order. Avi Selection Sort based on selection sort. It selects the smallest as well as largest data element in a single iteration and places them at their right position. Smallest element is placed at the first position and largest element is placed at last position of the array. In the same way we find and place the second smallest and second largest element at their right position. Same procedure is followed by the rest of the data elements and in this we way we get the final sorted output of data elements.

Let Input: Array a[0…n-1], Algorithm: in 1st iteration for i=0, after executing it we got the first smallest and largest elements and placed them at a[0] and a[n-1] place. In second iteration 2nd smallest and 2nd largest data elements are placed at a[1] and a[n-2] places respectively. Total number of iteration for outer loop is (n/2). At last we get the final sorted output. Fig. 1 shows the algorithm for Avi Selection sort.

```
Avi_Selection_Sort (a, n)
//Let A is a linear array with n elements i.e. a[0: n-1]
                                                          Times
1.  for i = 0 to n/2-1                                    n/2
2.      min = a[i]                                        n/2-1
3.      max = a[i]                                        n/2-1
4.      p1 = i                                            n/2-1
5.      p2 = i                                            n/2-1
6.      for j = i+1 to j <= n-i-1                         Σ(i=0 to n/2) tj
7.          If min > a[j]                                 Σ(i=0 to n/2) tj-1
8.              min = a[j]                                Σ(i=0 to n/2) tj-1
9.              p1=j                                      Σ(i=0 to n/2) tj-1
10.         else if max < a[j]                            Σ(i=0 to n/2) tj-1
11.             max = a[j]                                Σ(i=0 to n/2) tj-1
12.             p2=j                                      Σ(i=0 to n/2) tj-1
13.     If p2 == i && p1 == (n-i-1)                       n/2-1
14.         Exchange a[p1] with a[p2]                     n/2-1
15.     Else if p2 == i && p1!= (n-i-1)                   n/2-1
16.         Exchange a[i] with a[p1]                      n/2-1
17.         Set p2 = p1                                   n/2-1
18.         Exchange a[n-i-1] with a[p2]                  n/2-1
19.       Else                                            n/2-1
20.         Exchange a[i] with a[p1]                      n/2-1
21.         Exchange a[n-i-1] with a[p2]                  n/2-1
```

Fig. 1 Avi Selection Sort Algorithm

Output: Array a[0...n-1] in ascending order.

We can understand this concept with the help of an example. Assume there are total eight data elements which we want to sort. We will explain it with the help of various figures and subfigures which shows step by step working of the algorithm.

All figures (2-5) shows that index number of the data elements like a[0], a[1]…a[7] whereas 0,1,…7 are index number and data element row shows the actual data elements. Variables like p1, p2, min and max represents position (p1) for minimum data element, position (p2) for maximum data element, min contained the minimum value of data element and max contained the maximum value of data element. By default at the start of the algorithm, these entire variables have zero value. In our explanation, 'step' word is related to the number assigned to that line in the algorithm shown in fig. 1. For example step 3 means the 3rd line of algorithm shown in fig.1 which is max = a[i].

All figures (2-5) have subfigures also. As we see in fig. 2a (subfigure of fig. 2) which shows the initial array and variables settings. For initial value of i = 0 and j = 1 is shown in fig. 2a

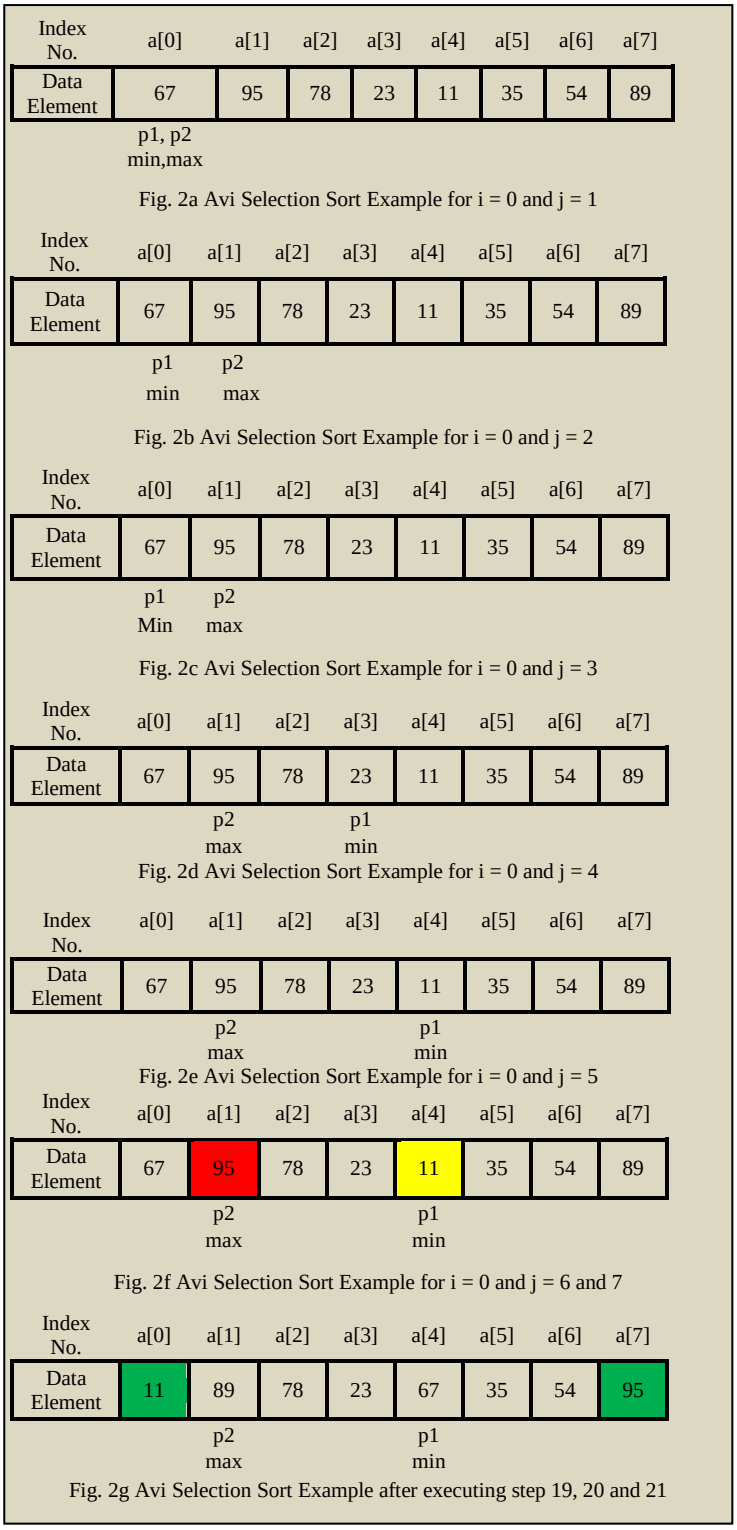

Fig. 2 Avi Selection Sort Example for i = 0 and various value of j from 1 to 7

In fig. 2a we see p1and p2 represent '0' value where as min and max represent '67' value (for either a[p1] or a[p2]). At step 1 of Avi Selection Sort algorithm, we see in fig. 1; variable i works for i=0, 1, 2, 3 for n = 8 because step 1 moves for i=0 to (n/2-1).After assigning the values to the variables we reach at step 6. At step 6 for i=0, j works for j=1, 2, 3, 4, 5, 6, 7 [i.e. (n-i-1) = (8-0-1) = 7]. For i=0 and j=1 at step 7, min > a[1] / 67>95 condition false move to step 10 as shown in fig.1, max < a[1] / 67<95 condition true. Now max = a[1] / max = 95 and p2=1. After j=1; we get values for variable are p1 = 0, p2 = 1, min = 67 and max = 95. Now move back to step 6 for further value of j.

Now in fig. 2b, for i=0 and j=2 at step 7, min > a[2] / 67>78 condition false move to step 10, max < a[2] / 95<78 condition false. No change in positions and values of variables. Here again move back to step 6.

Now in fig. 2c, for i = 0 and j = 3 at step 7, min > a[3] / 67>23 condition true. Then min = a[3] / min = 23 and p1 = 3. After executing j=3 we get variables value are p1 = 3, p2 = 1, min = 23 and max = 95. Here again move back to step 6, for further value of j.

Now in fig. 2d, after executing j = 3, for i = 0 and j = 4 at step 7, min > a[4] / 23>11 condition true. Then min = a[4] / min = 11 and p1 = 4. After executing j=4; we get variables value are p1 = 4, p2 = 1, min = 11 and max = 95. Move to step 6 again.

Similarly in fig. 2e, after executing j = 4, for i = 0 and j = 5 at step 7, min > a[5] / 11>35 condition false move to step 10, max < a[5] / 95<35 condition false. No change in positions and values of variables.

As we see in fig. 2f, after executing j = 6 and 7, for i = 0 step 7 and step 10 fail for rest value of j, so no change in positions and values of variables. For i = 0 step ends now. For understanding we have used red cell colour for largest data element, yellow cell colour represents the smallest data element and green cell colour represent two things / points. First point (achieved after swapping) at the right position means that smallest element at first position and largest element at last position respectively. Second point is, now both; not considered as the part of the array; rest iteration work for the rest of array. Same thing is shown here in fig. 7 by their cell coloured as red and yellow.

After that control moves to step 13, If p2 == i && p1 == (n-i-1) / if 1==0 && 4== (8-0-1) = 7 which a false. Now control moves to step 15, if p2 == i && p1! = (n-i-1) / if 1==0 && 4!= (8-0-1) = 7 which is again false. Now control moves to step 19 which is true by default after that step 20 and 21 executes. As shown in fig. 8 after executing step 19, 20 and 21. Here green cell colour shows their relative right position and is settled at that position.

Fig. 2g shows that, two elements min and max set at their respective right position. Minimum value is set at location a[0] and maximum value set at location a[n-1] (a[7] in our case). Both are at the end of the array whereas smallest at the front and largest at the rear.

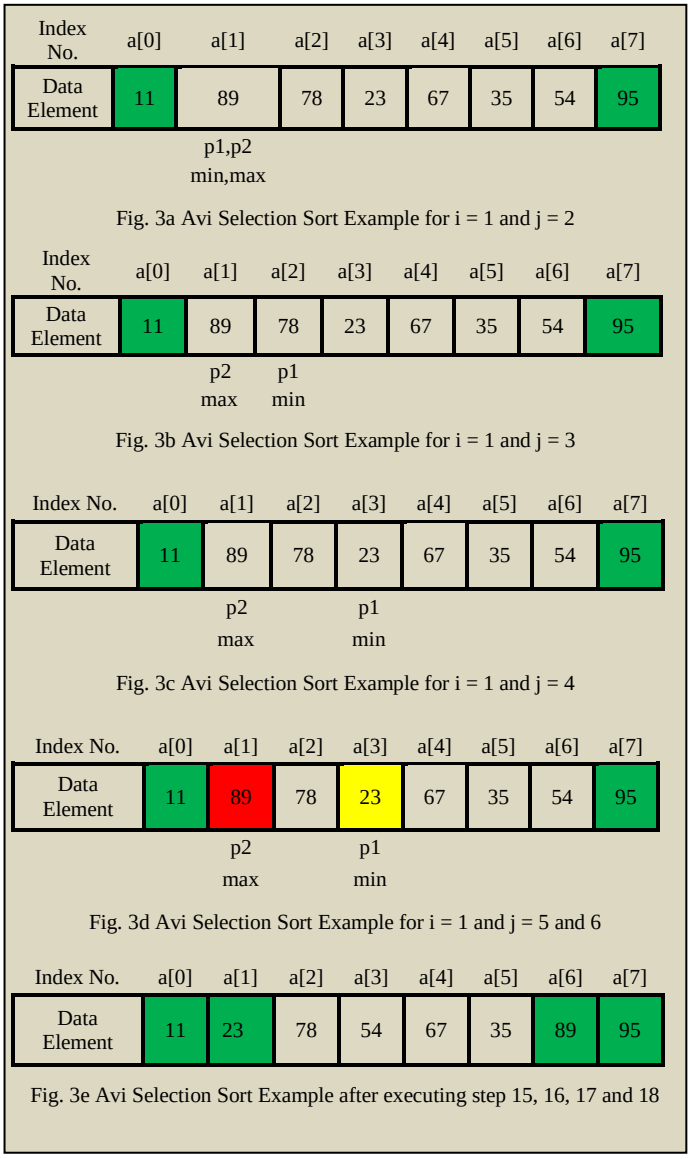

Fig. 3a Avi Selection Sort Example for i = 1 and j = 2

Fig. 3b Avi Selection Sort Example for i = 1 and j = 3

Fig. 3c Avi Selection Sort Example for i = 1 and j = 4

Fig. 3d Avi Selection Sort Example for i = 1 and j = 5 and 6

Fig. 3e Avi Selection Sort Example after executing step 15, 16, 17 and 18

Fig. 3 Avi Selection Sort Example for i = 1 and various value of j from 2 to 6

Now control back move to step 1 for i = 1 and j = 2 as shown in fig. 3a (subfigure of fig. 3), again similar to fig. 2a here p1and p2 represent '1' value where as min and max represent '89' value. For i =1 and j = 2, at step 7, min > a[2] / 89 > 78 condition true. Then min = a[2] / min = 78 and p1 = 2.where as p2 = 1 and max = 89. Now move to step 6.

Now in fig. 3b, after executing j = 2, for i = 1 and j = 3 at step 7, min > a[3] / 78 > 23 condition true. Which means min = a[3] / min = 23 and p1 = 3, whereas no change in p2 and max variables. Now move to step 6.

As we see in fig. 3c, after executing j = 3, for i=1 and j=4 at step 7, min > a[4] / 23 > 67 condition fail. Now control move back to step 10 and check max <a [i] / 89 < 67, again condition fail. So there is no change in positions and values of variables. Now p1 = 3, p2 = 1, min = 23 and max = 89. Now move back to step 6.

As we see in fig. 3d, after executing j = 5 and 6, for i = 1 step 7 and step 10 fail for rest value of j, so no change in positions and values of variables. For i = 1 step ends for j = ((n-i-1) / (8-1-1) = 6. Here we marked again red cell for largest element and yellow cell for the smallest elements from the rest of the array. Minimum = 23 and maximum = 89.

After that control moves to step 13, If p2 == i && p1 == (n-i-1) / if 1==1 && 3== (8-1-1) = 6 which is false. Now control moves to step 15, if p2 == i && p1! = (n-i-1) / if 1==1 && 4! = (8-1-1) = 6 which is true. So that step 16, 17 and 18 will execute which actually placed these elements at their respective correct position shown in fig. 3e. Mark them green.

Index No.	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Data Element	11	23	78	54	67	35	89	95
			p1,p2 min,max					

Fig. 4a Avi Selection Sort Example for i = 2 and j = 3

Index No.	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Data Element	11	23	78	54	67	35	89	95
			p2 max	p1 min				

Fig. 4b Avi Selection Sort Example for i = 2 and j = 4

Index No.	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Data Element	11	23	78	54	67	35	89	95
			p2 max	p1 min				

Fig. 4c Avi Selection Sort Example for i = 2 and j = 5

Index No.	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Data Element	11	23	78	54	67	35	89	95
			p2 max			p1 min		

Fig. 4d Avi Selection Sort Example for i = 2 and j = 5

Index No.	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
Data Element	11	23	35	54	67	78	89	95
			p2 max			p1 min		

Fig. 4e Avi Selection Sort Example after executing step 13 and 14

Fig.4 Avi Selection Sort Example for i = 2 and various value of j from 3 to 5

Now control move back to step 1, for i = 2 and j = 3 as shown in fig. 4a (Subfigure of fig. 4), which is similar to fig. 2a, here

p1 = p2 = 2 whereas min = max = 78.

Now in fig. 4a we see for i = 2 and j = 3, at step 7, min > a[j] / 78 > 54 which is true. Then min = a[i] / min = 54 and p1 = 3 whereas no change in p2 and max variables. Move to step 6.

After that, now for i = 2 and j = 4, as shown in fig. 4b, at step 7, min > a[j] / 54 > 67 which is false. Now we move to step 10, max < a[j] / 78 < 67 which is also false. So there is no change in value of p1, p2, min and max variables.

After executing i =2 and j = 4, for i = 2 and j = 5 we see at step 6 for i=2, j works for (n-i-1) / (8-2-1) = 5 where we reached for j = 5. As shown in fig. 4c, at step 7, min > a[j] / 54 > 35 which is true. So that min = 35 and p1=5 whereas no change in p2 and max variables value.

After that at step 6 loop works for i = 2 and j = 6 which is false. So that again we marked yellow and red cell for smallest and largest data element respectively are shown in fig. 4d.

After that control moves to step 13, If p2 == i && p1 == (n-i-1) / if 1==1 && 5== (8-2-1) = 5 which is true. So that step 14 will execute, swap the data elements and place their accurate position; shown in fig. 4e. Mark them with green cell colour.

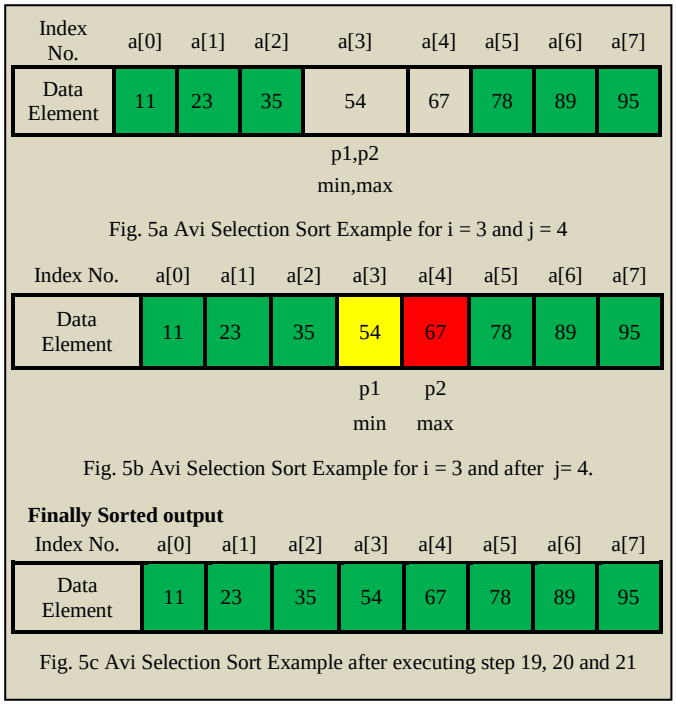

Fig. 5a Avi Selection Sort Example for i = 3 and j = 4

Fig. 5b Avi Selection Sort Example for i = 3 and after j= 4.

Fig. 5c Avi Selection Sort Example after executing step 19, 20 and 21

Fig. 5 Avi Selection Sort Example for i = 3 and value of j = 4

Now control again, move back to step 1 for i = 3 and j = 4 as shown in fig. 5a (subfigure of fig. 5), again similar to fig. 2a here p1 and p2 represent '3' value whereas min and max represent '54' value. For i =3 and j = 4 at step 7, min > a[j] / 54 > 67 which is false. Now we move to step 9, max < a[j] / 54 < 67 which is true. So that p2 = 4 and max = 67 whereas no change in p1 and min variables. Now control move back to step 6 again.

Value for i = 3 is the last iteration as we see in step 1 for i moves 0 to (n/2-1) / (8/2-1) = 3. For i=3, j move from j = i+1 = 4 to j <= (n-i-1) / j < (8-3-1) =4. Exactly two move one for j = 4 for right condition and one for j = 5 for false condition. Again we used yellow and red cell for the same purpose as said above which is shown in fig. 5b.

After that control moves to step 13, If p2 == i && p1 == (n-i-1) / if 4==3 && 3== (8-3-1) = 4 which a false. Now control moves to step 15, if p2 == i && p1! = (n-i-1) / if 4==3 && 3!= (8-3-1) = 4 which is again false. Now control moves to step 19 which is always true, after that step 20 and 21 execute placed them at right position, mark green as shown in fig. 5c.

Finally we get the sorted data element in ascending order.

Complexity of Avi Selection Sort / Theoretical Evaluation

The running time of the algorithm is the running time for each statement executed; a statement that takes c_i steps and executes n times will contribute $c_i n$ to the total running time. As shown in fig. 1, step1 to step 22; each step has some value, to which number of times the statement will execute shown in fig.1. Here assume c is constant for each statement. So total cost for Avi Selection sort is: ---

T(n) = c (n/2-4*(n/2-1)+ Σ(i=0 to n/2) tj+ 6*(Σ(i=0 to n/2)(tj-1)) + 9*(n/2-1)). Here we find that, in worst case the running time for this algorithm is an^2+bn+c. where a, b and c are any constant. It is a quadratic function of n, so its complexity is $O(n^2)$.

Experimental Evaluation

The effectiveness of the Avi Selection sort algorithm will be measured in CPU time; which is measured using the clock function on the machine with minimal background processes running, with respect to the size of the input array, and compared with the bubble, selection and insertion sort algorithms. The Avi Selection sort algorithm will run with the array size parameter set to: 1k, 3k, 5k, 7k, 9k, 11k, 13k and 15k over a range of varying-size arrays. Number of data elements is used as synthetic data elements of varying-length arrays with random numbers. The tests were run on PC running Windows 7 Ultimate (32 bit operating system) and the following specifications: Intel ® Atom ™ CPU N2600 at 1.60 GHz with 2 GB of RAM. Algorithms are run in Turbo C++ IDE.

Procedures

The procedure is as follows:

1. Store 15,000 records in an array
2. Choose 1,000 records

3. Sort records using Avi Selection sort, bubble, selection and insertion sort algorithm
4. Record CPU time
5. Increment array size by 2,000 each time until reach 15,000, repeat 3-5

Results and Analysis

Table II shows the actual execution time of different-different sorting algorithm (mentioned above) which also shows that Avi Selection sort is faster than insertion sort. When data taken randomly (by random function). In average case, over all Avi selection sort is the fastest among the all four sorting algorithms.

TABLE II execution of different sorting algorithm in average case (bubble, selection, insertion and avi selection sort)

S. No.	No. of Data elem e-nts	Testing time (seconds)			
		Bubble Sort	Selectio n Sort	Insertio n Sort	Avi Selection Sort
1	1000	0.16483 5	0.21978	0.164835	0.10989
2	3000	0.76923 1	0.714286	0.604396	0.494505
3	5000	1.75824 2	1.483516	1.263736	0.934066
4	7000	3.18681 3	2.472527	2.142857	1.483516
5	9000	4.83516 5	3.681319	3.241758	2.087912
6	1100 0	7.03296 7	5.164835	4.505495	2.912088
7	1300 0	9.50549 5	6.868132	5.879121	3.626374
8	1500 0	12.5274 7	8.791209	7.582418	4.615385

Fig. 6 shows the graphical representation of Table II. Fig. 6 graphically represents the comparison of execution time in average case.

In fig. 6, we see that; time taken by Avi Selection sort is very less as compared to the other sorting algorithms like bubble, selection and insertion sort. In our case we have applied this test on, up to 15000 data elements.

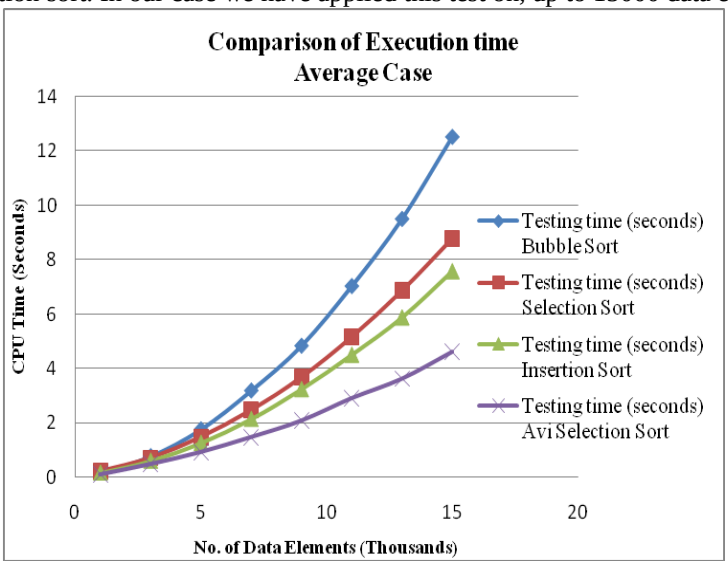

Fig.6 Comparison of CPU Execution time for different Sorting Algorithm (Average Case)

This graph shows that as the number of data elements increase; time taken by Avi Selection sort is much lower than other sorting (said above) algorithm.

Table III, also shows the actual execution time of different-different sorting algorithm (mentioned above) which also shows that Avi Selection sort lies in between selection and bubble sort in worst case. Here worst case is defined as when data elements are taken purely in reverse order.

TABLE III Execution of Different Sorting Algorithm in Worst Case (Bubble, Selection, Insertion and Avi Selection Sort)

S. No.	No. of Data elements	Testing time (seconds)			
		Bubble Sort	Selection Sort	Insertion Sort	Avi Selection Sort
1	1000	0.164835	0.164835	0.10989	0.10989
2	3000	0.549451	0.494505	0.32967	0.494505
3	5000	1.263736	0.989011	0.549451	1.043956
4	7000	2.142857	1.483516	0.769231	1.758242
5	9000	3.351648	2.197802	0.989011	2.637363
6	11000	4.67033	2.967033	1.208791	3.681319
7	13000	6.263736	3.901099	1.428571	4.835165
8	15000	8.131868	4.89011	1.703297	6.208791

Fig. 7 shows the graphical representation of Table III. Fig. 7 graphically represents the comparison of execution time in worst case.

In fig. 7, we see that; time taken by Avi Selection sort is lies in between selection and bubble sort. As said above, in this case again we applied this test on, up to 15000 data elements. By this we can predict as the number of data elements increase time taken by Avi Selection sort is always lies in between selection and bubble sort; in worst case only.

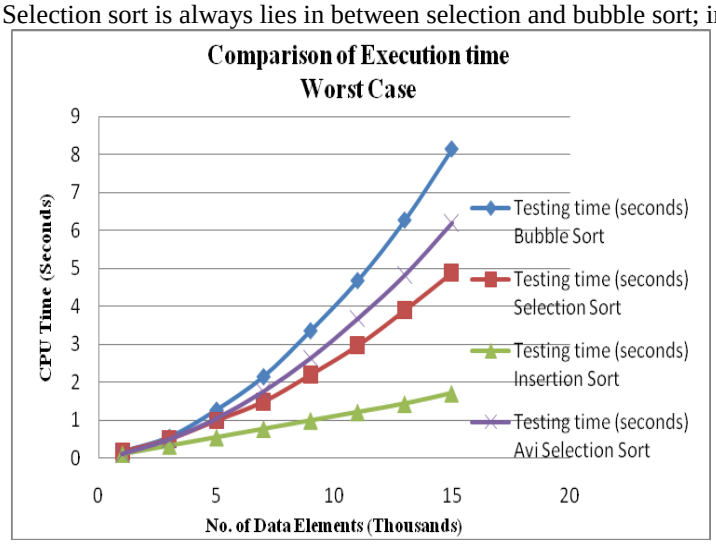

Fig. 7 Comparison of CPU Execution time for different Sorting Algorithm (Worst Case)

This algorithm takes more time (worst case) than selection sort because this algorithm has some more condition (if statement) due to which it have to pay some extra time.

IV. CONCLUSIONS

In this paper we have introduced Avi Selection Sort algorithm which is based on comparison method, having $O(N^2)$ time and correct sorting algorithm. Avi sort is based on selection sort in-which we select two elements (smallest and largest) simultaneously and place them in their relative position. Theoretical and empirical method shows that the complexity of Avi Selection sorting is $O(N^2)$. In average case Avi Selection sort's execution time is minimum and faster than insertion sort. In worst case; execution time lies in between bubble and selection sort.

The major limitation of this algorithm is its quadratic nature which consumes very much time to sort data elements. In future we shall try to reduce the complexity of Avi Selection sorting algorithm by modifying the algorithm or by designing a new algorithm.

REFERENCES

[1] Sorting Algorithm. Available: http://en.wikipedia.org/wiki/Sorting_algorithm.

[2] Song Qin, *Merge Sort Agorithm*,CS.fit.edu Available: *cs.fit.edu/~pkc/classes/writing/hw13/song.pdf.*

[3] T.H Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *pp. 23-40,159-164, 174-178, 194-200 Introduction to Algorithms*, PHI Learning Ptd., 2012. ISBN-978-81-203-4007-7

[4] Owen Astrachan, *Bubble Sort: An Archaeological Algorithmic Analysis*, SIGCSE 2003. Available: http://www.cs.duke.edu/~ola/papers/bubble.pdf

[5] Sedgewick, *Algorithms in C++, pp.98-100*, Addison-Wesley, 1992.

[6] P.J. Deitel and H.M.Deitel, *C++ How to Program, Sixth Edition, pp.931-932*, PHI Learning Ptd, 2008, ISBN-978-81-203-3496-0.

