Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-117

The Rapid Sort

Mriganka Sarmah , Heisnam Rohen Singh

***Abstract*— Sorting is arranging a collection of elements either in ascending or descending order. There are various applications of sorting algorithm in every field of science. Already there exist different sorting algorithms with different complexities. In worst case, the best known complexity is O(n log n). In this, a sorting algorithm is developed and compared with the existing sorting algorithm. It is found the new algorithm is much better than the existing sorting algorithm like the Quick Sort, Merge sort etc. This algorithm is much better for closely related datasets. To sort the element in reverse order it can accomplished the sorting in O(n).**

***Keywords*— Sorting, complexity, worst case, reverse order.**

I. Introduction

A. *Sorting:*

It is nothing but arranging a collection of elements in a sequence i.e. either ascending or descending order. There are lot of application of sorting where data need to be arranged. For example the details of the students in a class can be easily analysed if the students can be arranged in alphabetical order. So, to arrange them the sorting algorithm can be used. Already there exists lots of sorting algorithm like Quick sort, Merge sort, Insertion sort, Radix sort etc.

B. *Analysis of algorithm:*

In order to find which algorithm is better than other we need to compare these algorithms. To compare, the complexity of these algorithms need to be calculated.

There are two types of complexity. They are:

i) Space Complexity: It gives the total amount of memory requires to perform the algorithm.

ii) Time Complexity: It gives the total amount of time requires to perform the algorithm.

Now-a-days, when we talk about the complexity, we mean time complexity as, the memory becomes very cheap. One way of comparing is based on the exact running time of all algorithms but it depends upon processor and language used.

Mriganka Sarmah(*Author*)
Assam down town University
India

Heisnam Rohen Singh(*Author*)
Assam down town University
India

Even if the processor and language are same, calculating the exact time is difficult as it would require CPU utilization may be different. The time complexity is dependent on the number of input. So, it is expressed in term of number of input or input size. Two algorithms can be compared by using the rate of growth function, f(n) of the algorithms expressed in term of number of input n. the algorithm with lesser rate of growth function is better than the other. If the rate of growth function is high when the numbers of input increase the number of operation also increase.

II. Existing Sorting Algorithm

There are lots of sorting algorithms. Some of the common sorting algorithms are given here.

1. Bubble sort[1][2]: In this algorithm the larger elements are pushed back to one end. It compared the two consecutives elements if the second element is smaller than first one the two elements are swapped and the larger element is pushed at back. It continues doing this for each pair of adjacent elements to the end of the data set. It is continue till no swapping is done.
2. Insertion sort[3][4]: This algorithm is just like the technique of arranging cards in card playing. It works by taking elements from one by one from the list and inserting them in their correct position into a new sorted sequence.
3. Selection sort[5][6] : In this sorting algorithm the smallest is element from list is found and swapped with the first element. The second smallest is swapped from second element. These are repeated until all elements are sorted.
4. Quick sort[7][8]: In this algorithm, an element is chosen as pivot element and in each step the exact position of the pivot element is found. A pivot partition the elements in two part, one parts consist of all elements less than the pivot and other consisting of all elements greater than pivot. The same step is repeated for each partition.
5. Merge sort[9][10]: This algorithm is based on Divide and Conquer technique. The lists of elements are divided in smaller sorted list. And these small sorted lists are merged in single sorted list.

III. RAPID SORT

In this sorting first, the minimum and maximum element from the sequence is found and placed in extreme ends of the sequence. These ends are mark as lower bound and upper bound i.e. minimum and maximum element are put at lower-bound and upper-bound respectively.

Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-117

Consider the following sequence of element:

4	5	12	1	14	7	8	20	6	3

In the above example the minimum element is 1 and maximum element is 20. So 1 is put in the first mark by lower bound and the element 20 is put at last, mark as upper bound.

lower_bound *upper_bound*

1	5	12	4	14	7	8	3	6	20

The other remaining elements in the sequence are placed alongside nearer extreme end which is calculated by comparing the absolute values of |**minimum - element**| and |**maximum - element**|. If the element is nearer to the lower bound the lower bound is set to the new element position, otherwise the upper bound is set to the new element position.

In the example , the next element is 5 and the element is nearer to the element in lower bound. Hence, the element 5 is placed along side lower-bound. Lower-bound is reset to second position.

lower_bound *upper_bound*

1	5	12	4	14	7	8	3	6	20

Next element 12 has distance of 11 from minimun and 6 from maximum, hence it was placed along side upper bound. The upper_bound is reset to ninth position.

lower_bound *upper_bound*

1	5	6	4	14	7	8	3	12	20

Element 6 has distance of 5 from minimum and 12 from maximum, hence it was placed along side lower_bound. This number 6 is compared with the element in lower_bound i.e. 5.

As 6 is larger than 5 the position of 6 is found and lower_bound is reset.

lower_bound *upper_bound*

1	5	6	4	14	7	8	3	12	20

Number 4 has distance of 3 from minimum and 16 from maximum, hence it was placed along side lower_bound. After that the exact position of the new element 4 is found by comparing in lower half. lower_bound is reset to next position.

lower_bound *upper_bound*

1	4	5	6	14	7	8	3	12	20

For element 14, Place near upper_bound After finding the exact position.

lower_bound *upper_bound*

1	4	5	6	3	7	8	14	12	20

Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-117

			lower_bound				*upper_bound*		
1	4	5	6	3	7	8	12	14	20

For element 3, the scenario is as shown below.

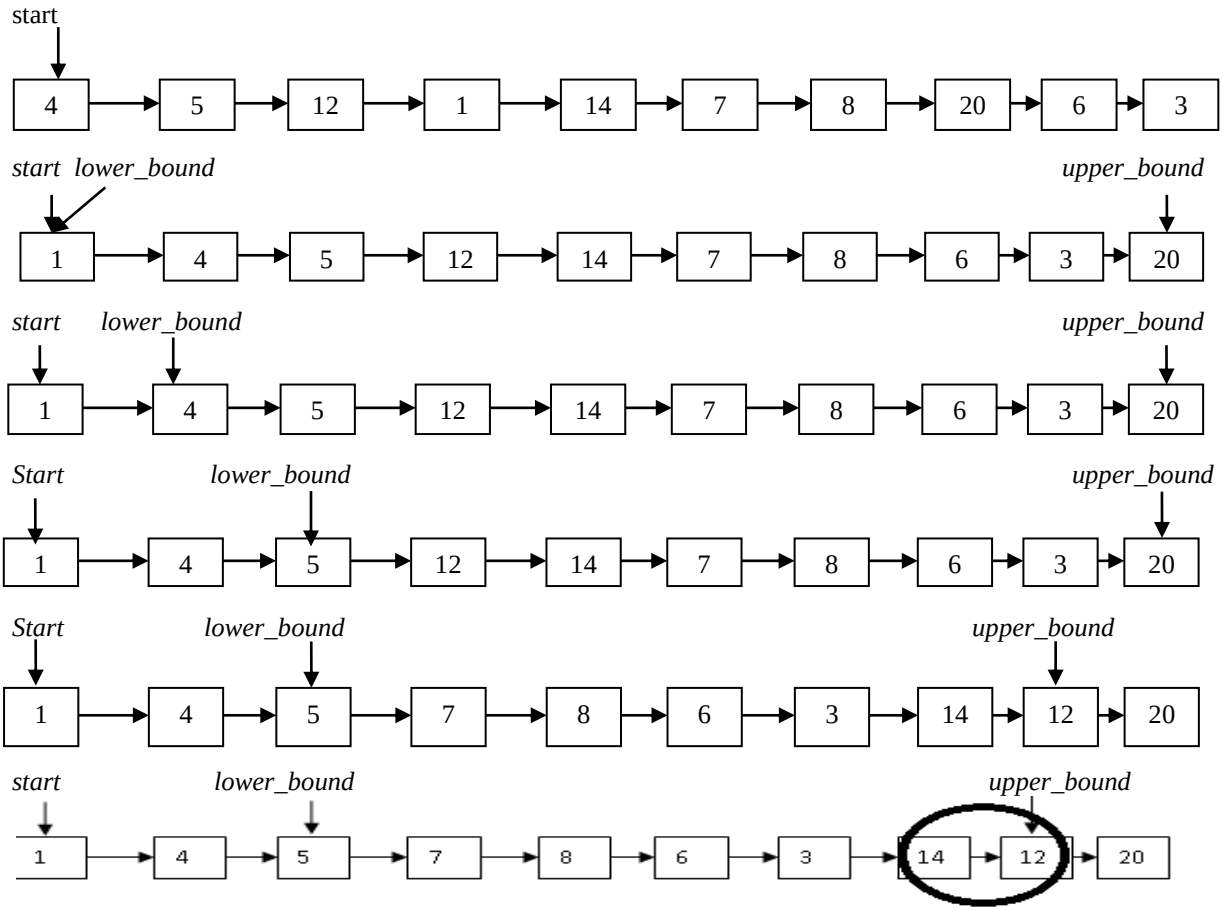
			lower_bound				*upper_bound*		
1	4	5	6	3	7	8	12	14	20

				lower_bound			*upper_bound*		
1	3	4	5	6	7	8	12	14	20

Similarily, for element 7 and 8 the exact position can be found. And at last the *lower_bound and upper_bound* will be differ by 1 at that moment the operation is stopped.

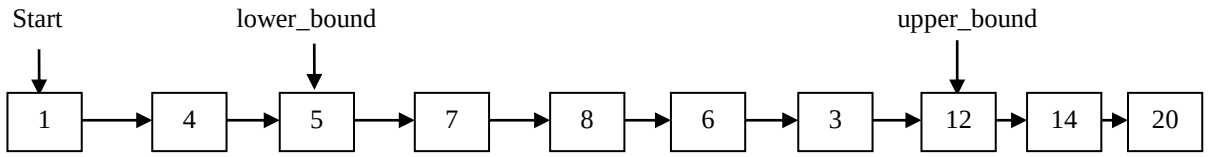
					lower_bound	*upper_bound*			
1	3	4	5	6	7	8	12	14	20

In this algorithm, for each element after finding the nearer end the exact position need to find. To find the position linear search [11][12] or binary search [13][14] can be used. And in positioning the element shifting of the elements may involved. To overcome this shifting, different data structure like linked list [15] can be used. In that, shifting can be just changing some pointers.

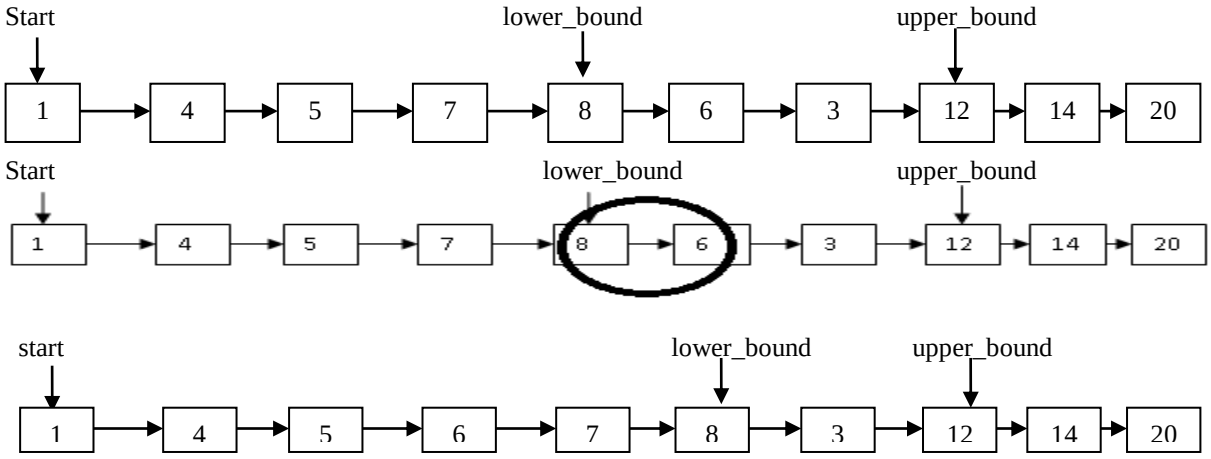


Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

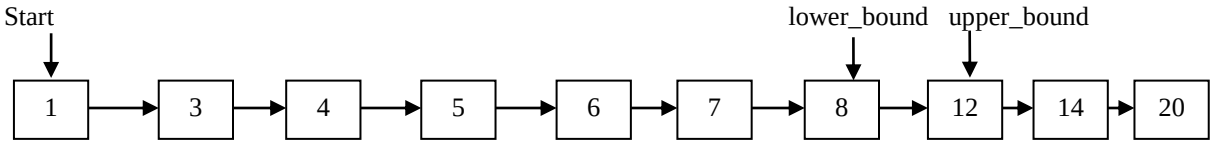
ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-117



Number 7 and 8 is as shown below.



Finally Number 3 is linked in its proper position. The number of exchanges might be trivial but links are formed in O(1) time.



The idea discussed might be involving a double link list.

IV. PROPOSED ALGORITHM

We assume we are sorting an array in ascending order.

Begin

From an Array of n numbers find the *minimum(A)* and *maximum(A).*

insert *minimum(A)* at the beginning and *maximum(A)* at the end.

Set *lower_bound=0, upper_bound=Last*

/*lower_bound points to the recent most element on left side. upper_ bound points to the recent most element on right side*/

Get *next_number*.

Dist(lb)=abs(*minimum(A) - next_number)*

Dist(ub)=abs(*maximum(A) - next_number)*

/*steps : To find out the proximity towards either of the far ends*/

If Dist(lb) <Dist(ub)

Insert *next_number* to the right of *lower_bound*

Else

Insert *next_number* to the left of *upper_bound*

Set tempLB=*lower_bound*, tempUB=*upper_bound*

While (*next_number* < A[tempLB]) /*finding the position of the number in lower half*/

Insertion _Sort(*next_number*, A[tempLB])

tempLB- - /*Resetting the lower_bound*/

While(***not true*** : *next_number* < A[tempUB] /*finding the position of the number in upper half*/

Insertion_Sort(*next_number* , A[tempUB]))

tempUB++ /*Resetting the upper_bound*/

Repeat for all remaining elements.

End

V. RESULTS AND DISCUSSION

The experiment was performed using the linear data structure array. The datasets is generated using the C in-built rand() function. This algorithm is compared with the common existing algorithm like Bubble sort, Insertion sort Quick sort, Merge sort. The complexity of the algorithm depends mostly upon the number of comparisons to do the

Proc. of the Intl. Conf. on Advances In Engineering And Technology - ICAET-2014

ISBN: 978-1-63248-028-6 doi: 10.15224/ 978-1-63248-028-6-01-117

sorting. So, in this the numbers of comparison of various algorithms are compared and are shown in the following figure. This experiment is fully programmed using turbo C in a windows 7 platform.

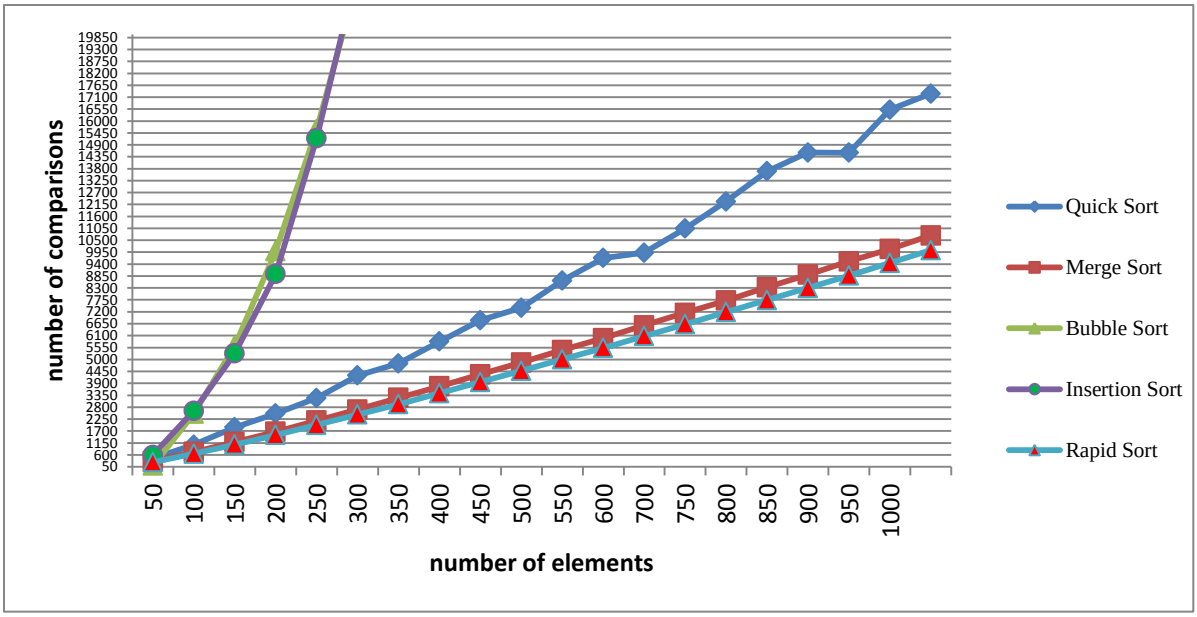


Figure 1: number of comparisons vs number of elements

VI. CONCLUSION

Based on the experience a chart is shown above the number of comparisons with the number of inputs of different sorting. It has been found that our algorithm is better than the existing sorting algorithm as the rate of growth is much slower than the other.

Acknowledgment

We would like to thanks Assam down town University, Guwahati, Assam for helping and supporting us to carry on this research.

References

[1] Yedidyah Langsam, Moshe J. Augenstein, Aron M. Tenenbaum, Data Structures using C and C++, Pearson Prentice Hall, 2007,pp 355-358.

[2] Owen Astrachan. Bubble Sort: An Archaeological Algorithmic Analysis. SIGCSE 2003 Hannan Akhtar. Available at: http://www.cs.duke.edu/~ola/papers/bubble.pdf.

[3] Robert Sedgewick, Algorithms, Addison-Wesley 1983 (chapter 8 p. 95)

[4] Yedidyah Langsam, Moshe J. Augenstein, Aron M. Tenenbaum, Data Structures using C and C++, Pearson Prentice Hall, 2007,pp 381-382.

[5] Yedidyah Langsam, Moshe J. Augenstein, Aron M. Tenenbaum, Data Structures using C and C++, Pearson Prentice Hall, 2007,pp 367-368.

[6] S.K. Srivastava, Deepali Srivastava, Data Structures through C in depth, BPB publication, 2011, pp 421- 424.

[7] T.H.Cormen , C.E.Leiserson, R.L.Rivest, C Stein, Introduction to Algorithms, 2nd ed.,PHI,pp 145-148.

[8] Yedidyah Langsam, Moshe J. Augenstein, Aron M. Tenenbaum, Data Structures using C and C++, Pearson Prentice Hall, 2007,pp 358-364.

[9] Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, Fundamentals of Computer Algorithms, Uiversities Press, 2009, pp 159-167.

[10] S.K. Srivastava, Deepali Srivastava, Data Structures through C in depth, BPB publication, 2011, pp 434-444.

[11] Yedidyah Langsam, Moshe J. Augenstein, Aron M. Tenenbaum, Data Structures using C and C++, Pearson Prentice Hall, 2007,pp403-405.

[12] S.K. Srivastava, Deepali Srivastava, Data Structures through C in depth, BPB publication, 2011, pp 472-473.

[13] Yedidyah Langsam, Moshe J. Augenstein, Aron M. Tenenbaum, Data Structures using C and C++, Pearson Prentice Hall, 2007,pp 410-413.

[14] S.K. Srivastava, Deepali Srivastava, Data Structures through C in depth, BPB publication, 2011, pp 473-476.

[15] Yedidyah Langsam, Moshe J. Augenstein, Aron M. Tenenbaum, Data Structures using C and C++, Pearson Prentice Hall, 2007,pp 202-207.

About Authors(s):

 Mriganka	B.Tech IT from Sikkim Manipal University in 2008. M.Tech IT from Sathyabama University in 2011. Assistant Professor at CSE dept. of Assam down town University since 2012. Research area of interest is Grid Computing and Artificial Intelligence.
 Rohen	B.Tech CSE from Maharastra Institute of Technology in 2009. M.Tech IT from Tezpur University in 2012. Assistant Professor at CSE dept. of Assam down town University since 2012. Research area of interest is Algorithms and Social Networking.