International Journal of Advanced Science and Technology
Vol.100 (2017), pp.25-36
http://dx.doi.org/10.14257/ijast.2017.100.03

Implementation of an Enhanced Sorting Technique to Sort Elements with DMA Model by Using Linux Based Environment

Gandi Ramakrishna[1], Uppe Nanaji[1], NakkaThirupathiRao[1],
Debnath Bhattacharyya[1] and Hye-jin Kim[2]

[1]*Department of CSE*
Vignan's Institute of Information technology
Visakhapatnam, AP, India
[2]*Sungshin Women's University,*
2, Bomun-ro 34da-gil,
Seongbuk-gu, Seoul, Korea
{gandiramakrishna,nakkathiru,nanajistiet,debnathb}@gmail.com
hyejinaa@daum.net
(Corresponding Author)

Abstract

Sorting is procedure of arranging elements from a collection in ascending or descending order. For example, a list of names could be arranged in sorted order alphabetically based on first character in telephone book. A list of states could be sorted by population by area. So, all members are necessary have knowledge of effective sorting method to deal with this problem with in an indefinite period. Some number of developed algorithms that were able to benefit from having a sorted list .In this paper, proposed algorithm is an enhanced sorting technique over power some disadvantage of traditional sorting algorithms by utilizing dynamic memory allocations properly. Proposed algorithm is contrast with current algorithms by using some aspects.

Keywords*:*

1. Introduction

Sorting is the technique of reorganizing a given group of objects in increasing or decreasing order. Sorting usage is to simplify to easily search for elements of the set, which has elements in sorted order [2]. Let see in telephone books, in income tax files, in tables of contents, in libraries, in dictionaries, in warehouses, and so on that stored objects are in sorted order, have to be efficiently searched and retrieved the objects. Even in child hood, taught to put their playing objects "in order", and they are learned with some sort of sorting before they learn anything about arithmetic in class. Sorting is a relative and necessitous activity, specifically in data processing in databases. What else would be easier to sort than data! Forever, our primary interest in sorting is fond to the even more elemental techniques used in the construction of algorithms .In the near future, all of them recognize their job is a unique case of sorting, usage of library routines properly make short work of the problem. Second, some of different sorting algorithms have been developed, each of which focused on a particular good idea or observation. Most of the pseudo code design models lead to the direction of significant sorting techniques, including randomization, incremental insertion, probabilistic analysis, master theorem for divide-and-conquer approach for recursive algorithm and analysis of algorithms. Most of the mathematical problems solved from characteristics and properties of these algorithms. Sorting is an efficient method to perform the work of organizing the elements in

ISSN: 2005-4238 IJAST

increasing or decreasing order. Sorting method, in most cases used in our daily life in real world as well as in many computer applications [1] [4].

1.1. Why Sorting?

Many computer analysts, researchers and scientists concluded that sorting to be most elemental problem in the study of algorithms [5,11]. Reasons are:

- At certain times an application internally needs to sort data. For example, to prepare customer statements in order, bank clerks sort the checks based on check number.
- Algorithms usually worth sorting as a key subroutine. For example, a procedure that impart graphical objects which are stratified on top of each other capability to order the objects admit to an "above" relation in order that it can draw these objects from bottom to top. We intend to see numerous algorithms that habit sorting as a subroutine.
- From large variety of sorting algorithms and we can able to draw, to employ a rich set of techniques. In evidence, many important techniques used throughout algorithm design appear in the sorting algorithms are developing since from few years ago.
- We can prove a nontrivial lower bound for sorting

Asymptotically, upper bounds match the lower bound, and so we learn that sorting algorithms are asymptotically optimal. In other case, sorting also to prove lower bounds for particular other problems [6,7,10].

- After implementing sorting algorithms, many engineering issues came to arise. The optimal sorting algorithm for a special situation depend on many factors, such as preceding knowledge about the keys and satellite information, the caches and virtual memory of the host computer, and the software circumstances. Many issues are best negotiating with at the algorithmic level, fairly than by "pinch" the code.

Sorting techniques are categorized into two kinds:

i) **Internal methods** -these sorting techniques are used wherever small number of elements in the list to be ordered. So that the complete sorting could be taken place in primary memory. All data that is to be processed at a time to sort in primary memory.

A proposed algorithm is an internal sorting technique.

ii) **External methods**–these sorting methods are used on larger lists. To sort a large amount of data needs secondary memory such as HDD to store data, which is not fit in primary memory. In primary memory holds the currently present sorted data only. The process of dividing, parts of data are sorted separately and merged together.

2. Literature Survey

2.1 Selection Sort Method

Selection sort hoist away an arrangement of passage way on unsorted elements. From first passage an element is chosen on few principles and stored in the appropriate location among list of elements. Principle for choosing an entry to be the lowest or biggest element. If lowest one is picking up, later to sort the elements in increasing order, exact position is towards the starting from the list. At present that exact entry is placed in first from the array list, this technique imitated on the other elements. This process is re-iterated for n-1 times .such the n-1 lower entries are ordered in the first n-1 passages, finally the biggest entry is placed at the end. For this, require n-1 passages [12]. Given below pseudo code is as follows:

for j←1 to n-1

do

search the lowest entry from j^{th} position to nth

position.

swap this entry with j^{th} position element.

done

From below example, let see how this technique works on the small number of elements:

10 3 6 8 5 9

- on passage 1 search from 1st to sixth to get lowest

swap second entry by first entry

3 10 6 8 5 9

- on passage 2 search from 2^{nd} to sixth to get lower entry ,and then swap fifth entry by 2^{nd} one

3 5 6 8 10 9

- on passage 3 search from 3^{rd} to sixth to get lower entry, and then exchange 3rd entry by third

3 5 6 8 10 9

- on passage 4 search from 4th to sixth to get lower entry, and then exchange 4th entry by fourth

3 5 6 8 10 9

- on passage 5 search from 5th to sixth to get lower entry, and then swap 5th entry by 6th

3 5 6 8 9 10 sorted.

2.2 Bubble Sort Method

Bubble sorting technique workout to improve in order of elements by contrasting entries in pairs and interchange among them whenever elements are in orders. This technique is reiterated up to the array list is in increasing or decreasing order. The following method compares adjacent elements [14].

The 1st and 2nd entries are in comparison and swapped whenever required, and then they are at appropriate locations relevant among them. Anyhow is the 2^{nd} entry is at exact position relevant to the 3rd entry. It clarifies by using a loop in which 1st entry is in contrast with 2^{nd} entry, 2^{nd} entry with 3^{rd} entry, 3^{rd} entry with 4^{th} entry until the last element in list, pseudo code is given below.

for k←0 to n-1

do

ifA[k] > A[K+1] then

exchangea[K] and a[K+1]

end-if

done

What does this accomplish? It `bubbles' the largest element to the end of the list in pass1.

Let us consider 10 3 6 5 8 9

compare first & second elements -> 3 10 6 5 8 9

compare 2^{nd}& 3^{rd} elements -> 3 6 10 5 8 9

compare 3^{rd}& 4^{th} elements -> 3 6 5 10 8 9

compare 4^{th}& 5^{th} elements -> 3 6 5 8 10 9

compare 5^{th}& 6^{th} elements -> 3 6 5 8 9 10

The biggest value has now bubbled to the end position in first pass. The given process is at present, reiterated as far as the n-l passes done. That is, in each pass larger element placed in correct position when compared with unsorted list. Finally, all elements in sorted order. The sorting technique pseudo code is given below

```
for  j:=0 to n-1
for  k:=0 to  n-j-2
if  A[K] > A[K+1]  then
      temp := A[K];
      A[K] := A[K+1];
      A[K+1] := temp;
end-if
done
done
```

After applying above pseudo code to given below example.
following intermediate steps are obtained:

initially 10 3 6 5 8 9

j = 0 3 6 5 8 9 10

j = 1 3 5 6 8 9 10

j = 2 3 5 6 8 9 10 no swap takes place on this pass

j = 3 3 5 6 8 9 10

j = 4 3 5 6 8 9 10

2.3 Insertion-Sort Method

Let us suppose that at some case in ordered procedure first i positions of elements in a list are in ascending or descending order. The fundamental concept of this method is to place the i+1th position of element into its exact position from beginning to ith position. The length of the ordered section increases to i+1. At first i is set to 1, surely the first number on its own fix a sorted list. After n-1 iterations of this procedure, the list is sorted [13]. Hence the pseudo code description:

```
for  i:=1 to n-1
do
place  i+1th entry in exact position from the  first to
```


$i+1^{th}$ entries.

done

For above pseudo code, interpretation is given below, where the ordered basic section at each stage is underlined:

4 7 5 8 10	i	=	1
7 5 8 10	i	=	2
5 8 10	i	=	3
8 10	i	=	4
10	i = 5		

The inclusion taken place when a clone of the i+1th position element re-iteratively contrasting with the ith, i-1th,... position elements up to an element is found which is lower, at each stage the bigger entry element is carried one place to the right. Wherever the exact position is found later the cloned element is included in the independent area accomplished by the preceding displacements of elements to the right side. Insertion in pseudo code is given below as follows:

```
Y:=A[i+1];    // assign
k := i;        // begin check at ith position
while Y < A[k]
do
    A[k+1] :=A[k];//displace an element to right side
    k--;          // check backward
done
A[k+1] := Y;    // cloned element is assigned to
```

and discussed in given below:

3 6 8 4 4 about to add

3 6 - 8 4 < 8 displace 8 to the right side

3 - 6 8 4 < 5 displace 5 to the right side

3 4 6 8 4 > 3 so add the element 4

Now the problem is from above pseudo code that is, whenever A[i+1] is lower among all A[0..i] ,later the loop aim infinitely running. Alone solution is to solve this problem is to assign k:=0 as an appropriate case. Thus given pseudo code is:

```
for i := 0 to n-2
do
    Y := A[i+1]    // assign to x
    k := i;        // begin check at  entry i
    while k > 0 && y < A[k]
    do
    A[k+1] ← A[k]; //displace an element
k--;          // check backward
```

```
        done
        if  k == 0 then
                if  Y< A[0] then
                k := -1;
                A[1] := A[0];//displace an element
                end-if
        end-if
        A[k+1] := Y; // copied element is assigned to
    done
```

The pseudo code is interpreted is given as an midway step from the procedure:

5 6 10 3 8 9 3 to be inserted

5 6 - 10 8 9 3<10 and j=3, move 10 right

5 - 6 10 8 9 3<6 and j=1, move 5 right

3 5 6 10 8 9 j=0, 3 < 5, move 5 right and

put 3 into a[0].

In determining, efficacy of this sorting method is observed once more the main loop is running n-1 times while running the inside loop 1, 2, 3...n-1 times. Swapping step works inside the main loop, that is inner loop.

2.4. Quick Sort Method

Divide-and-Conquer approach [8] is Quick Sort technique, and therefore this method works under both non-recursive and recursive algorithm. This approach uses master theorem is somewhat contrast from merge sort method. In merge sort, the partition step work on each entry itself as distinct list, and in the combine step is real work to combine sub lists as sorted list and so on until main list of elements are sorted. But in Quick sort, actual work starts in partition step(i.e., divide) to sort the list of elements. Truth is, combine step in quick sort does not work[9].

In practice, quick sort outrun merge sort, it convincing defeats selection sort method and insertion sort method.

At present quick sort uses this approach, ordering the sublist array [p..r], where at the beginning array list is array [0..n-1].

1. Partition the sub list array [p..r] By selecting the entry called pivot. Reorganize the entry elements in array [p..r], in case all entries in array [p..r] Which are lower or same as pivot moved left side to the pivot and all entries in the list array [p..r] are greater than pivot are placed right side to it. This process is called partitioning. At this point, elements may be in order or not, moved to left side of pivot element which are relevant among them, and the same situation happens for the entries to the right side of the pivot element. Just now, we see that each entry is around not known on the exact position of the pivot.

Practically, every time select the right end element from the sub list, array[r] element taken as pivot. Let us see the given example shown in Figure 1, if the sub array consists of [10,7,5,8,6], later select 6 as the pivot element. By using pivot partitioning the array list, array could be [5,6,10,8,7]. Let pivot element position be q, which is in sorted position.

2. Conquer repetitively ordering the sub list array [p..q-1],array elements displaced to the left side of pivot, and that are lower or equal in contrast to the pivot element and sub list array[q+1..r], here array entries displaced to the right side of the pivot element , and that are bigger in contrast to the pivot element.

3. Third step is Combine step, which is nothing to do in this method. Because conquer step recursively sorts the array elements by using pivot. The entries in list [p..r] do not help ,any way they are ordered. Here pivot in main array list and each sub-array list plays an important role in order to sort the elements

Let come to our previous example. After repetitively ordering the sub array lists using pivot, left sub array contains elements following the pivot element position is [5], and the right sub array lists followed by the pivot element position is [7,8,10]. So the left list has [5], 6 is successor for left list, and the right sub array which has [7, 8,10] is the successor for pivot element.

In the lowest part, each subarray contains almost one element, same as in external sorting technique. In external sorting, sub array not at all have empty that is no elements. In quick sort, if entries in the sub list's are all lower or greater than the element which is pivot.

Here in conquer step works over the repetitive sorting of the sub array lists. After partitioning in starting time as pivot element 6, we have left sub array of [5] and right sub array of [10,8,7].

Left sub list [5] to sort, only single element is there, automatically it is ordered. Partitioning stops here in base case.

Now order the sub list [10,8,7] of main array list, we select 7 as the pivot, after partitioning ,left sub array is empty, because no element is lower than or equal to pivot and right sub array is [8,10] to the right side of pivot element. After sub arrays are sorted, we have the pivot 7, and 7 followed the right sub array [8,10]. Again partitioning right sub list [8,10] where 10 as pivot element. In this way, recursively sort the elements using pivot elements until the base case reached.

Let given below figure depicts real working nature of quick sort technique. Here, array entries in shaded color are pivot elements in preceding repetitive calls, such that these entry values in these indexes never modified and never exchanged, because these values are in sorted order:

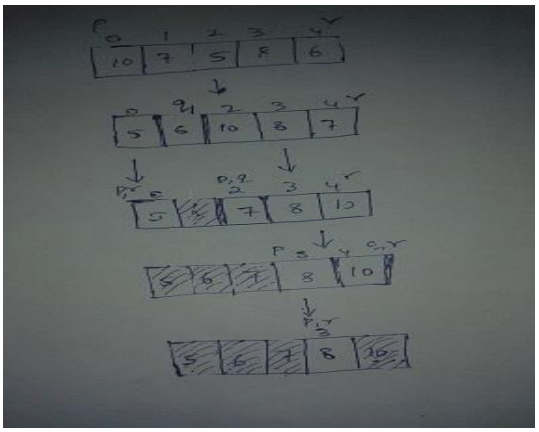

Figure 1.Quick Sort Example

2.5. Merge Sort Method

As long as of using divide-and-conquer approach to arrange in increasing order, commitment to determine what our sub problems are operating to noticing alike. The complete problem is to arrange entire array elements in increasing order. Suppose that a sub problem is to arrange sub array in ascending order. In

appropriate, we will assume of a sub problem as arranging the sub array in correct order beginning at position p and moving over position r. It is acceptable to own a sub array notation, so that array[p..r] represents sub array of array. Note that our problem is to arrange in increasing order for an array[0..n-1] of size n elements

Given below depicts the merge sort method uses divide-and-conquer approach:

1. In Divide, position q is finding by midway between positions p and r. Repeat this step as same as we found the midpoint in binary search until the single element in sublist.
2. In Conquer, recursively arrange the elements in sublists in increasing order from each of the two sub problems which are created in divide step. Let see we repeatedly arrange the sublists array[p..q], array[q+1..r] in correct order.
3. In Combine, merge the two order sublists into the single ordered sublist array[p..r].
   Condition required is sublists which have atmost one element when $p \geq r$, that means sublists with no elements or one element is already in correct order. So divide-conquer-combine approach applicable only when $p<r$.
   Let's see an example. An array [13, 6, 2, 11, 8, 10, 5, 1], so firstly sublist is the full array, array[0..7] (p=0 and r=7). This sublist has more than one element. So that from conditions, it is not a base case .
   Given below figure shows how to sort the given sub list using merge sort method in step by step manner.

- we measure q=3 in **divide** step.
- Sequence the two sublists array[0..3], which contains [13,6,2,11],and array[4..7], which contains [8,10,5,1] in correct order in **conquer** step. After backtrack from conquer step, two sublists ordered in increasing: array[0..3] contains [2,6,11,13] and array[4..7] contains [1,5,8,10], so from two sublists final array is [2,6,11,13,1,5,8,10].
- At last, combines the two correct ordered sublists from the first and second halfs, displaying the final increasing order array list [1,2,5,6,8,10,11,13] in **combine** step.
  According to what condition sublist array[0..3] develop into sorted order? Equivalently, sublists contain more than two elements, and conclude it's not a unit case. With the positions p=0, and r=3, calculate q=1, recursively order the array[0..1] ([13,6]) and array[2..3] ([2,11]), arranging the sublist array[0..3] containing [6,13,2,11], and combine the first and second half's to yield the list [2,6,11,13].

How did the sub array array[0..1] become sorted? With p=0 and r=1, compute q=0, recursively sort array[0..0] ([13]) and array[1..1] ([6]), resulting in array [0..1] still containing [13,6], and merge the first half with the second half, producing [6,13].

The sub arrays array[0..0] and array[1..1] are base cases

Most of the steps in merge sort are simple. You can check for the base case easily. Finding the midpoint qqq in the divide step is also really easy. You have to make two recursive calls in the conquer step. It's the combine step, where you have to merge two sorted sub arrays, where the real work happens. For the moment, let's assume that we know how to merge two sorted sub arrays efficiently and continue to focus on merge sort as a whole.

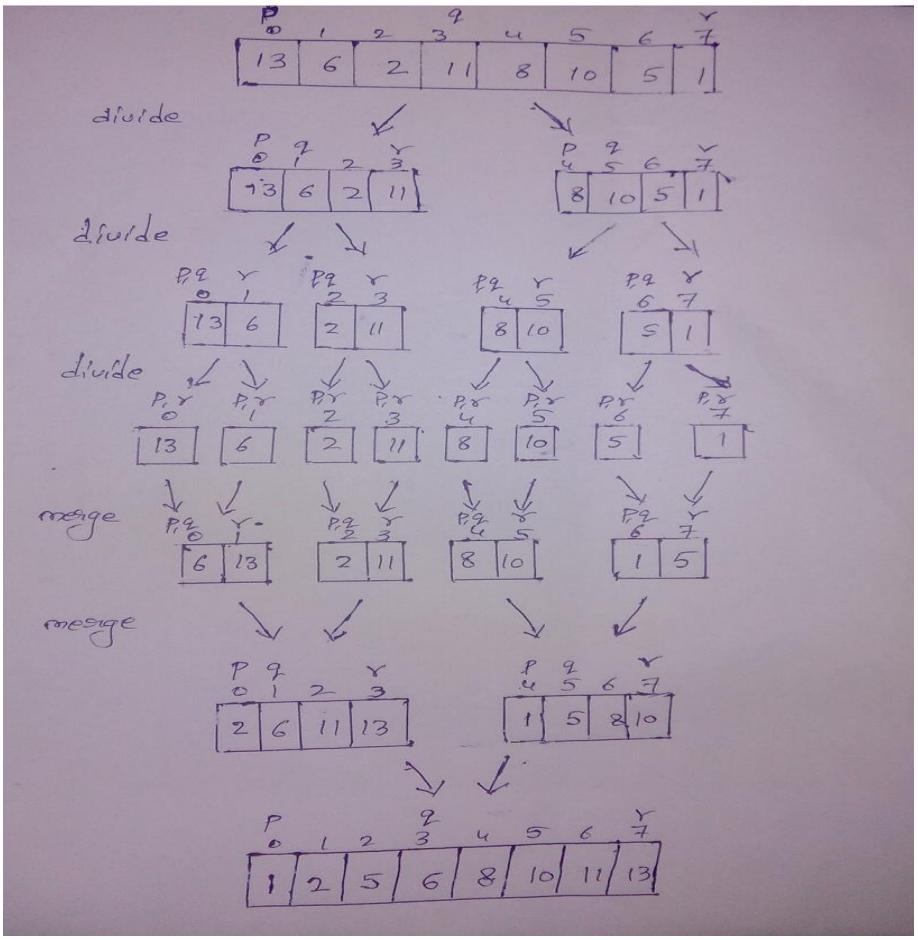

Figure 1. Merge Sort Example

3. Proposed Algorithm

Firstly, number of elements to be read, and then implementing the algorithm. Dynamically create the array .During reading each element, proposed algorithm finds the maximum element. Create the array dynamically using calloc function, the size of dynamic array is level to maximum element. Only deal with distinct values by using this algorithm. All values which are read from first dynamic array will arrange in correct order starting from 0^{th} position to *maxelement*-1^{th} position in another array. We observe that all the entries are displaced in ascending order in another array. Later preserve the memory i.e., which is essential to capture the entries which have values in the list and later discharge the additional memory in the array list using dynamic memory allocation.

Proposed Algorithm

Enhance Sort(pointr,x,n)

pointr—It is of integer type pointer points to starting location of the dynamically memory allocation with size n.

r—pointer is of integer type points to the beginning position of the dynamic memory allocation using calloc with size x

x—It's value is bigger value from all dynamic array elements

n—total count of values entered ,which are non-zero.

1) Read maximum value among all elements of dynamic array using pointer pointr. i.e., x

2) create the array list for x-entries with pointer r using dynamic memory allocation.
3) Select each non-zero value of Array using p to *non-zero value*-1 position of array using q;otherwise position values are zero upto x-1 position.
4) count:= 0
5) for i:=0 to x-1
6) if *(r+i)!=0 then
   a) *(r+count):=*(r+i)
   b) r++
7) end if
8) done
9) Now x-n memory locations should be release

Here Let see the example to sort values 23 16 32 17 8 in sorted order using this planned sorting algorithm
Given values are 23 16 32 17 8 .Sort them in ascending order

steps

1) find Maximum value from elements of array using pointer p. i.e., x
2) Create another dynamic array using pointer q with size x. Assign 0's to each location.

0	0	0	0	0

3) Assign non-zero values into the dynamic array.
   Which are read from using p.
   *(q+(v-1))=v

```
Elements stored in memory locations
 0 0 0 0 0 0 0 8 0 0 0 0 0 0 0 16 17 0 0 0 0 0 23 0 0 0 0 0 0 0 0 32
```

4) Copying all values to sequential positions**.**

```
Elements in sequencial indices after copying
 8 16 17 23 32 0 0 8 0 0 0 0 0 0 0 16 17 0 0 0 0 0 23 0 0 0 0 0 0 0 0 32
```

5) Now release memory of x-n elements
   After that,elements are in sorted order and also finds running time in sec's**.**

```
Elements after sorting
 8 16 17 23 32
The required running time to execute in seconds:0
```

In given below second example, from dynamic array of 20 elements finds the maximum element x i.e., 3456789 using proposed algorithm.

```
krishna@ubuntu:~/SortingUpdate$ ./a.out
Enter the no of elements
20
335 12 27 3456789 345 32 31 34 224 435 445 5567 555 498 789 234 223 322 111 7
```

Create another dynamic array with size x. Assign 0's to each location,and then sort the elements by copying all elements to sequential positions and release the remaining memory.

4. Conclusion

A Proposed algorithm shown to arrange the array elements in sorted order is effort for the non-duplicate entries. Presently, it is displayed memory, accordingly utilized in our policy using dynamic memory allocation efficiently with pointers an also shown how to release the memory after displaced the elements in sorted order

5. Future Work

Proposed algorithms alone take under consideration for non-duplication entries for this technique. So, in latter forthcoming analyzing to enhance the algorithm such that it could do task for repeated entries also. In future, necessity to check time complexity of this algorithm in all three cases. So, that we want to do contrast proposed algorithm with traditional algorithms in terms of better time complexity.

6. Performance Analysis

Afterwards checking the execution of traditional sorting techniques .Finally conclude that, in terms of dynamic memory utilization using pointer, proposed algorithm is better than traditional sorting algorithms. We have already discussed the utilization of memory dynamically in this algorithm. Existing algorithm does not discuss the proper usage of dynamic memory. In this method, it is used arbitrary access technique to arrange the entries at correct positions in the array list .This access technique is faster than sequential access technique of traditional sorting algorithms. Also studied this algorithm in contrast with traditional algorithm is easy to know and affable to users. This algorithm is running efficacy when differentiate with existing algorithms, with less number of input values. More dynamic memory allocations are required when larger number is taken as size. Already discussed distinct inputs are taken to sort. Finally conclude this algorithm will be used for applications where ever no. of inputs are less number as well as no problem with memory for large no of inputs with larger value, and then a better performance can be achieved for particular applications.

References

[1] "Data Structures and Algorithms software developed by Department of Computer Science", The University of Auckland.
[2] R. Singh, V. Kumar, A. Kumar, Shrivastava, S. Kumar, A. Tiwari, "RVA Sorting Based on Bubble & Quick Sort Technique", 2014 International Conference on Information and Communication Technology for Competitive Strategies, **(2014)**.
[3] nptel.ac.in/courses/106103069/9 available during 26th , November, 2016.
[4] Sorting and Searching Algorithms by Thomas Niemann,epaperpress.com
[5] http://lcm.csa.iisc.ernet.in/dsa/node196.html available during 26th November, 2016.
[6] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, Robert E. Tarjan "Time bounds for selection", Journal of Computer and System Sciences, vol.7, no.4, **(1973)**, pp.448-461.
[7] E. Horowitz, S. Sahni, S. Rajasekaran, "Computer Algorithms", Galgotia publications, New Delhi.
[8] S. Palui, S. Gupta "An Unique Sorting Algorithm With Linear Time Complexity", IJCSN, vol.3, no.6, **(2014)**.
[9] http://lcm.csa.iisc.ernet.in/dsa/node193.html available on 26th November, 2016.
[10] Martin Aumuller, Martin Dietzfelbinger, Pascal Klaue "How Good Is Multi-Pivot Quicksort?",ACM Transactions on Algorithms (TALG), vol.13, no.1, **(2016)**.
[11] S. M. Merritt, "An inverted taxonomy of Sorting Algorithms", Communications of ACM, vol. 28, no.1.
[12] J.I Munro, M.S. Paterson "Selection and sorting with limited storage", Elsevier, Theoretical Computer Science, Vol. 12, No.3,**(1980)**, pp.315-323.
[13] E. W.Dijkstra, "An introduction to three Algorithms for sorting in situ", Elsevier,Information Processing Letters, vol.15, no.3,**(1982)**,pp.129-134.
[14] P. Yu, Y. Yang, Y. Gan "Experimental Analysis on the Bubble Sort Algorithm and Its Improved Algorithms", SpringerLink,Chapter:Information Engineering and Applications, vol.154, pp. 95-102.
[15] Enrico Nardelli,Guido Proietti,"Efficient unbalanced merge-sort", Elsevier, Information Sciences, vol. 176, (2006), pp. 1321-1337

