European Journal of Academic Essays Special Issue (1): 49-56, 2014
ISSN: 2183-1904
www.euroessays.org

Recent Trends in Analysis of Algorithms and Complexity Theory

Performance Analysis Of Sorting Algorithm

< Dixit P. K.> [1] , < Sahoo Archana > [2] , < Sahoo Bikash Kumar > [3]

1 Director, Interface Software, Bhubaneswar

pk_dixit@yahoo.com

2 Passed MCA from OUAT, Bhubaneswar
archana.sahoo93@gmail.com

3 Passed B.Tech from ITER , Bhubaneswar
bikash020@gmail.com

Abstract: Sorting is a classic subject in computer science. There are mainly 3 reasons for studying sorting. First sorting algorithm gives a creative way for problem solving. Second sorting algorithm goods to practicing programming technique by using method, loop, array, statement. Third It is best to demonstrate the algorithm performance. This paper introduce bubble sort, merge sort, quick sort, bucket sort, radix sort, external sort.

Key words : Sorting, bubble sort, merge sort, radix sort, quick sort

---

1. Introduction

Bubble sort also called as sinking sort. It is a simple algorithm which working principle is comparing each pair of adjacent items and swapping them if the data item are in wrong order. The swapping is done until no swap is needed. Then only we can say the data item or array is in sorted. Merge sort is a divide and conquer algorithm and it was invented by john von Neumann in 1945.The algorithm divides the array into two halves and applies merge sort on each half recursively. After two half are sorted again merge to get the sorted array. T Quick sort is developed by C.A.R. Hoare. In this sorting the algorithm selects an element from an array called as pivot element. Heap data structure is an array of element that looks like a complete binary tree. The value of root node is greater than from the child node. Radix sort is based on bucket sort but it is more efficient than bucket sort. This sort is based on the value of actual digit in the positional representation of the numbers being sorted. To sort data from external file first you have to bring the data from memory then sort them internally.

2. Review of Literature

[1] Sorting algorithm is frequently used in many application or in research area. More depth research is going on to make the algorithm more efficient. Y.Danielliang has published introduction to java programming. He has attended to explain the design, implement the sorting algorithm by using java classes and interfaces and also gave a visualization of time complexity.

[2] Data structure using c Second edition by Amiya kumar Ratha and alok jagdev has given a brief explanation of sorting in example and programming in c language. And also given brief explanation of example.

[3] In wikipidia of bubble sort has given depth knowledge . In this , Bubble sort has explained by taking example which is good to understand.And also explained about time comlexity algorithm etc.

[4] *The Design and Analysis of Computer Algorithms* by Addison-Wesley has explained the fundamental concept of algorithm which is heart of the computer science. It introduce the basic data structure and programming technique of efficient algorithm.

[5] Mehlhorn, K. Data Structures and Algorithms 1: Sorting and Searching has derived the time compleity of insertion sort with the input length of n.

3. Problem Analysis :

3.1 Bubble Sort :

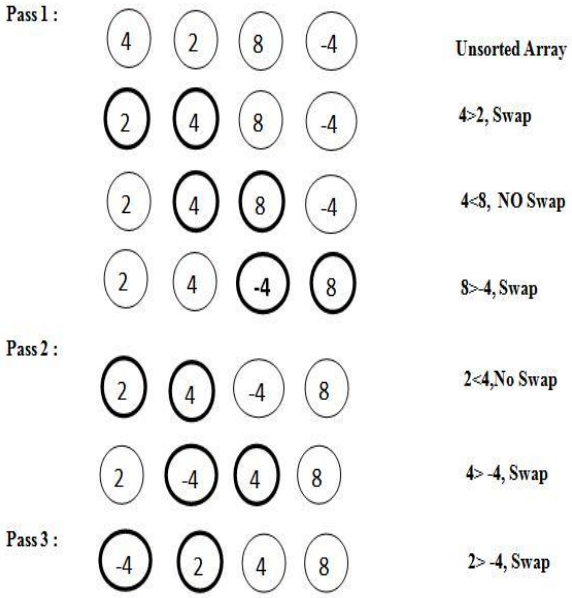
Bubble sort also called as sinking sort. It is a simple algorithm which working principle is comparing each pair of adjacent items and swapping them if the data item are in wrong order. The swapping is done until no swap is needed. Then only we can say the data item or array is in sorted. So

Corresponding Author: P. K. Dixit

why this is called bubble because all swapping time the smaller values gradually "**bubble**" their way to the top and largest value comes to bottom.

3.1.1 Step by step Explanation:

Let us take the array of numbers [4 2 8 -4] and sort the array in increasing order by using bubble sort. In each step element written in bold bubble is compared. In each step adjacent items item are compared if they are unsorted then swap tem, if they are in sorted order keep them as they are.



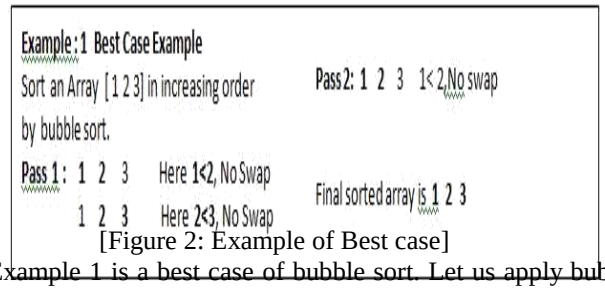
[Figure 1:Example of Bubble sort]

3.1.2 Bubble Sort Algorithm :

```
1.    for (int k=1 ; k <  list.length; k++)  {
2.     //perform the kth pass
3.          for (int i=0 ; I < list.length-k ; i++)
{
4.               if (list[ i ] >  list[ i +1])
5.                    swap  list[  i  ]  with
      list[ i +1];
6.               }
7.          }
8.    }
```

3.1.3 Performance Analysis :

Best Case : In the best case Bubble sort just need one pass to find out the array in sorted order. That means number are in sorted order.So the best case time is ***O(n)*** because first pass need n-1 no. of comparison.



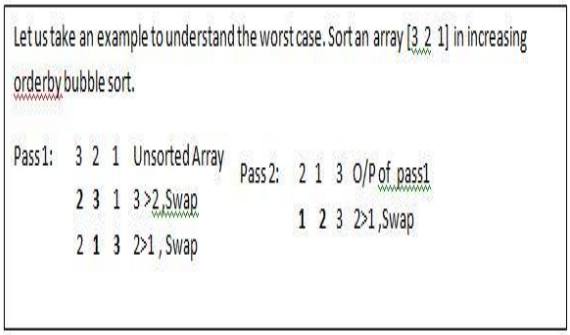
[Figure 2: Example of Best case]

Example 1 is a best case of bubble sort. Let us apply bubble sort algorithm upon Example 1. First for loop use for number of pass and second for loop uses for traverse adjacent pair of number. In each pass number are sorted from the end.

Worst case : In the worst case Bubble sort need n-1 no. of pass to find out the array in sorted order. In one pass we need n-1 no. of comparison. So the total no. of comparison is

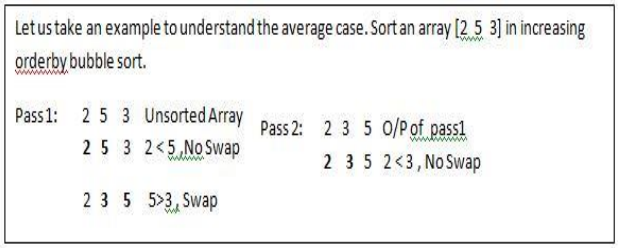
$(n-1) + (n-2) + \ldots\ldots\ldots\ldots +2+1 = ((n-1)n) / 2 = O(n^2)$

So the time complexity is $\boldsymbol{O(n^2)}$ **.**



[Figure 3: Example of Worst Case]

Average Case: Time complexity for average case is same as worst case. Because the input element are in random so algorithm need n-1 no. of pass and each pass need n-1 no. of comparison. So the time complexity is $\boldsymbol{O(n^2)}$ **.**



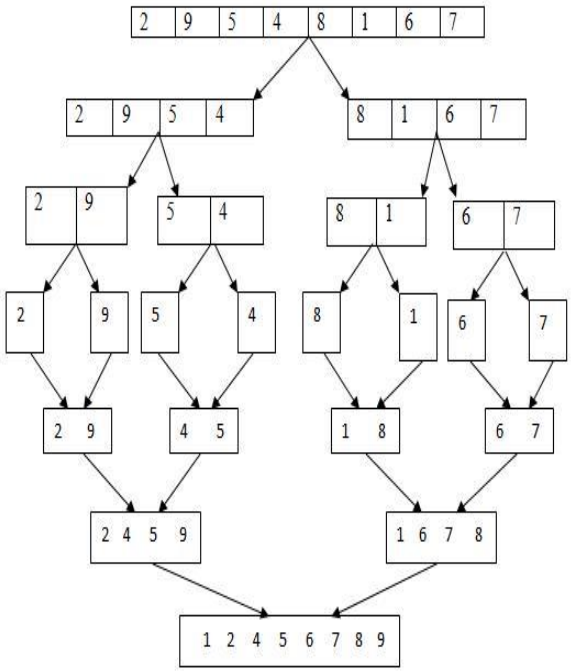
[Figure 4 : Example of average case]

3.2 Merge Sort :

Merge sort is a divide and conquer algorithm and it was invented by john von Neumann in 1945.The algorithm divides the array into two halves and applies merge sort on each half recursively. After two half are sorted again merge to get the sorted array.

3.2.1 Step By Step Example :

Let us take the array of numbers [2 9 5 4 8 1 6 7] and sort the array in increasing order by using merge sort.The original array divide into (2 9 5 4 and [8 1 6 7).apply merge sort on these two subarrays recursively to split [2 9 5 4) into [2 9) and [5 4) and [8 1 6 7) into [8 1) and [6 7).This process will continues until the subarrays contains one element. Since array [2] contains single element so it can not divide further.Now merge [2] with [9] into a new sorted array [2 9], merge [5] with [4] into a new sorted array [4 5] , merge [8] with [1] into a new sorted array [1 8] , merge [6] with [7] into a new sorted array [6 7].Now merge [2 9]with [4 5] into a new sorted array [2 4 5 9] and [1 8] with [6 7] into a new sorted array [1 6 7 8] and finally merge [2 4 5 9] and [1 6 7 8] into a new sorted array [1 2 4 5 6 7 8 9].



[Figure 5 : Example of Merge sort]

3.2.2 Merge sort Algorithm :

```
1.  public void mergeSort(int[] list)  {
2.          if (list.length > 1)    {
3.
        mergeSort(list[0……..list.length / 2]);
4.                  mergeSort(list[list.length /
2 + 1……..list.length]);
5.                  merge  list[0……
                list.length / 2 ] with list[list.length /
                2 +1……. List.length] ;
6.          }
7.      }
```

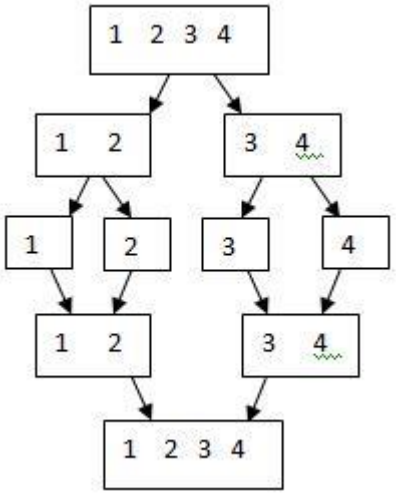
Algorithm for merging two Array :

```
1. public int[] merge(int[] list1,int[] list2)  {
2.          int[]  temp  =  new
            int[list1.length+list2.length];
3.          int current1=0;
4           int current2=0;
5.          int current3=0;
6.          while(current1  <  list1.length &&
            current2 < list2.length)  {
7.                  if  (list1[current1]  <
            list2[current2])
8.                          temp[current3++]
            = list1[current1++];
9.                  else
10.                         temp[current3++]
            = list2[current2++];
11.         }
12.         while(current1 < list1.length)
13.                 temp[current3++]  =
            list1[current1++];
14.         while(current2   < list1.length)
15.                 temp[current3++]  =
            list2[current2++];
16.         return temp;
17.  }
```

3.2.3 Performance Analysis :

Best Case:

Let T(n) be the time required to sort n element using merge sort. The algorithm splits the array into two sub arrays. Then sort the algorithm using same algorithm recursively. So to sort first half of array required T(n/2) times and same time for second half array. After that to merge two subarrays required n-1 compersions and n no. of moves to move the element to a temperary array.So total time required for merging and moves the element to a temperary array is 2n-1. T(n) = 2T(n/2) + 2n-1 which is same as O(n logn).

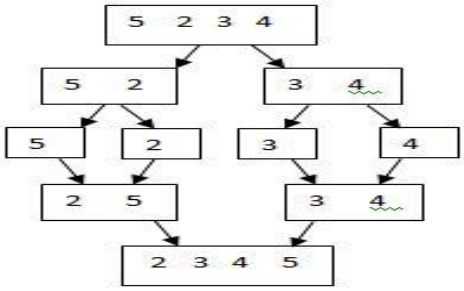


[Figure : Example of best case]

In merge sort when all the element are in sorted order that is best case of merge sort.

Worst Case :

When all the element are in random order then that is worst case of merge sort. But the time complexity is same as best case because all time algorithm split the array and then merge the array. So the time complexity is O(n logn).



[Figure -7 : Example of worst case]

3.3 Quick Sort

Quick sort is developed by C.A.R. Hoare. In this sorting the algorithm selects an element from an array called as pivot element. Then the array is divided into two parts. First part contains all the element less than or equal to the pivot element and second part contain greatest element from pivot element. Recursively apply quick sort to the first part and then second part to obtain sorted array.

3.3.1 Quick Sort Algorithm:

```
1. public void quickSort(int[] list)  {
2.   if( list.length>1 )   {
3.          partition list into list1 and list2 such that all
                elements in list1 <= pivot And all elements
                in list2>pivot
4.          quickSort(list1);
5.          quickSort(list2);
6. }
7.   }
```

Partition Method :

```
1.   /** partition the array list [first …. last] **/
2.  Private int partition(int[]  list,int first,int  last)
    {
3.       Int pivot = list[first];   // choose the
    firstelement as pivot
4.       Int low = first+1;              // index for
    forward search
5.       Int high=last;      // index for backword
    search
6.       While( high > low)     {
7.            // search forward from left
8.               While( low <= high && list[low]
    <= pivot )
9.                    Low++;
10.           // search Backward from right
11.  While( low <= high && list[high] > pivot )

12.                   High--;
13.          // swap two element in the list
14.  If( high > low)   {
15.                   Int temp=list[ high ];
16.                        List[high]      =
    list[low];
17.                   List[low]=temp;
18.           }
19.    }
20.    While( high > first && list[high] >= pivot )
21.                   High--;
22.    // swap pivot with list[high]
23.   If( pivot > list[ high ])   {
24.           List[ first ] = list[ high ];
25.           List[high ] = pivot;
26.           Return high;
27.       }
28.      Else {
29.       Return first;
30.       }
31.  }
```

3.3.2 Performance Analysis :

Best case: In best case the pivot divides an array each time into two part of equal size. Let T(n) donates the time required for sorting an array of n size.

So T(n)=T(n/2)+ T(n/2)+ n

T(n/2) = Recursive quick sort on subarrays

n = Partition Time

So finally T(n) = O(nlogn)

Worst Case : In worst case the pivot divides an array each times into one big size with the other empty. So

the algorithm requires (n-1)+(n-2)+……..2+1 =$O(n^2)$.

Average Case: In average case the pivot will not divides the array into two parts of same size nor one empty part. Size of the two part are very close. So average time is O(nlogn).

3.4 Heap Sort

Heap data structure is an array of element that looks like a complete binary tree. The value of root node is greater than from the child node. The tree is completely filled on all levels except possibly the lowest which is filled from the left. To sort an array using heap first create an object using the Heap class. Add all the element by using add method and remove all other element by using remove method. The element are removed in descending order.

3.4.1 Algorithm :

```
1.  Public class HeapSort  {
2.     Public static void main(String[] args)   {
3.           Integer[] list = {2,3,2,5,6,1,3,14,12};
4.             heapSort(list);
5.           For(int i=0; i<=list.length; i++)
6.                        System.out.println(list[i] +
    " ");
7.      }
8.  Public  static  void  heapSort(Object[]  list)
    {
9.          Heap heap=new Heap();
10.         // Add element to the heap
11.         For(int i=0; i< list.length ; i++)
12.             Heap.add(list[i]);
13.         // Remoove elements from the heap
14.               For(int i= list.length – 1;i>=0;i++)
15.                 List[i] = heap.remove();
16.     }
17.  }
```

OUTPUT: 1 2 2 3 3 5 6 12 14

3.4.2 Performance Analysis :

The algorithm first inserts n elements into heap. For inserting time complexity is O(logn). Total time for constructing heap is O(nlogn).Total time required for removing all the element is O(nlogn).

Both merge sort and heap sort requires O(nlogn) time. But in merge sort extra temporary array is required for merging two subarrays . But in Heap sort no addition space required. So it is better than merge sort.

3.5 BUCKET SORT AND RADIX SORT :

All the sort algorithm we have discussed that work for any type of element (e. g integer , string ,object)

.These algorithm sort the element by comparing their key. But bucket sort the element with out comparing the keys. he Principle of bucket sort is , assume that there are 0 to N-1 number of element so we need N no, of bucket starts from 0 1 2 …….N-1. If an elements key is I , then the element is put in I bucket. Each bucket holds the element with same key value.

3.5.1 Algorithm

```
1.   Void bucketsort( E[]  list)  {
2.         E[] buckets = (E[]) new java.util.ArrayList();
3.         // Distribute the element from list to bucket
4.         For( int i=0 ; i<list.length ; i++ )  {
5.              Int key= list[i].getKey();
6.              If( bucket[keys] == null )
7. bucket[keys] = new java.util.ArrayList() ;

8.              bucket[keys].add(list[i]) ;
9.            }
10.  // Now move the element from the buckets back to
     list
11.  Int k = 0;  // k is index for list
12.      For(int i=0 ; i<bucket.length ; i++)  {
13.           If( bucket[i] != null )  {
14. For( iint j=0, j< bucket[i]. size ; j++ )

15. List[k++] = bucket[i].get[j];

16.           }
17.       }
18.   }
```

3.5.2 Performance analysis

It takes O(n+N) times to sort the list and uses O(n+N)

space , where n is list size. If n is too large bucket

sort is not desirable.You can use Radix sort.

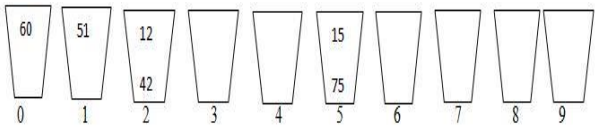
Radix sort is based on bucket sort but it is more efficient than bucket sort. For example the number 367 in decimal notation is written with a 3 in hundred place, a 6 in ten Place and 7 in one place. Let us take an example which is better understand. Arrange (51,60,75,42,12,15) in sorted order musing radix sort. To sort decimal we need 10 bucket because the base is 10. These buckets are numbered as 0,1,2,3,4,5,6,7,8,9.The elements are stored in an array as follows.

51 60 75 42 12 15

In this case 75 is largest number having 2 digit so 2 pass is required to complete the sort.

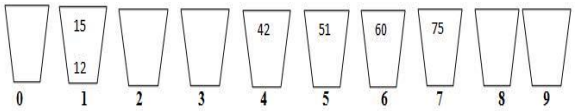
Pass 1:

In this step we need to place the number in corresponding bucket depend upon the least Significant digit.



[Figure 8: Pass 1 of Radix Sort]

Pass 2 : In this step we have to place the number in corresponding bucket depends upon the second digit.

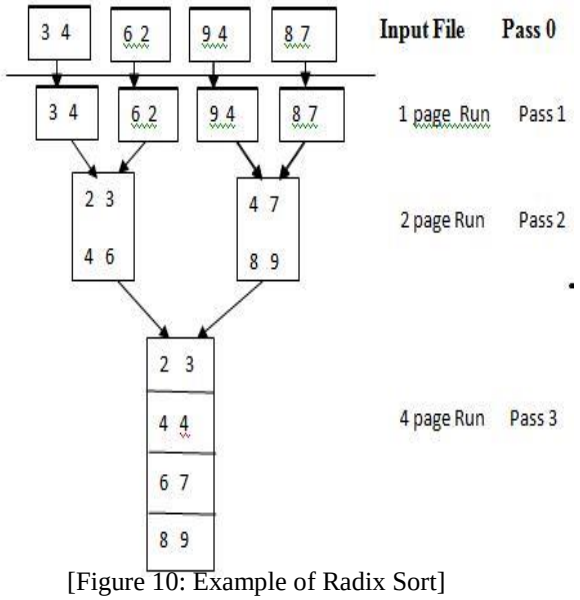


[Figure 9: Pass 1 of Radix Sort]

After pass 2 the final sorted number are 12 15 42 51 60 75.

3.6 External Sort :

All the sort algorithm we have discussed till now all the data to be sorted are available at one time in internal memory. To sort data from external file first you have to bring the data from memory then sort them internally. If the file size is too large then all the data can not brought to the memory at a time. So data comes to the memory in segment. After that each segment is merged and finally we get sorted segment. The time complexity is O(nlogn).



[Figure 10: Example of Radix Sort]

In External sort a large file is divided into segment which are sorted by algorithm. So in pass 0 a large file is divided in to segment and segment size in depends upon the programmer. Programmer need to pass the segment size to the algorithm. After that file came to primary memory. Like divide and conquer approach the segment are sorted and merge to a single segment. The final Single segment (pass 3) contains the sorted element.

3.6.1 Algorithm for Creating A Large file

```
1.  import java.io.*;
2.  public class External {
3.  public static void main(String[] args) {
4.  DataOutputStream output=new DataOutputStream(ne
5.  BufferedOutputStream(new FileOutputStream("large.dat")));
6.          for(int i=0;i<2000000;i++){
7.
            output.writeInt((int)Math.random() * 1000000);
8.                  output.close();
9.          }
10. }
```

Algorithm for creating initial sorted Segment

```
1.  private static int segment(int segmentsize, String originalfile, String f1) {
2.  int[] list = new int[segmentsize];
3.  DataInputStream input=new DataInputStream(new BufferedInputStream(new FileInputStream(originalfile)));
4.  DataOutputStream output=new DataOutputStream(new BufferedOutputStream(new FileOutputStream(f1)));
5.  int numberofsegment=0;
6.  while(input.available()>0){
7.          numberofsegment++;
8.          int i=0;
9.          for(     ;     input.available()>0     &&  i<segmentsize; i++)
10. {
11.     list[i]=input.readInt();
12. }
13. // sort an array list
14. java.util.Arrays.sort(list,0,i);
15. // write the array to f1.dat
16. for(int j=0;j<i;j++) {
17.         output.writeInt(list[j]);
18. }
19. }
20. input.close();
21. output.close();
22. return numberofsegment;
23. }
```

Algorithm for copying First half segment

```
1.  private static void copyHalfToF2(int numberofsegment,int segmentsize,DataInputStream f1,DataOutputStream f2) {
2.    for(int i=0;i<(numberofsegment/2)*segmentsize;i++) {
3.    f2.writeInt(f1.readInt());
4.     }
5.  }
```

Algorithm for mergeing all segments

```
1.  private static void mergeSegment(int numberOfSegment,int size,DataInputStream
```

```
    f1,DataInputStream f2,DataOutputStream f3) {
2.      for(int i=0;i<numberOfSegment;i++)  {
3.              mergeTwoSegment(size,f1,f2,f3);
4.      }
5.      while(f1.readInt()>0)
6.      {
7.          f3.writeInt(f1.readInt());
8.      }
9.  }
```

Algorithm for mergeing two segment

```
1.  private static void mergeTwoSegment(int size,DataInputStream f1,DataInputStream f2,DataOutputStream f3) {
2.  int intFronf1=f1.readInt();
3.  int intFronf2=f2.readInt();
4.  int f1count=1;
5.  int f2count=1;
6.  while(true)  {
7.          if(intFronf1<intFronf2)  {
8.                  f3.writeInt(intFronf2);
9.                          if(f1.available() == 0 || f1count++>=size) {
10.
                    f3.writeInt(intFronf2);
11.                         break;
12.                 }
13.                 else {
14.
                    intFronf1=f1.readInt();
15.                 }
16.         }
17.         else {
18.                 f3.writeInt(intFronf2);
19.                 if(f2.available()==0        ||  f2count++>=size)  {
20.
                    f3.writeInt(intFronf1);
21.                         break;
22.                 }
23.                 else {
24.
                    intFronf2=f2.readInt();
25.                 }
26.         }
27.  }

28. while(f1.available() > 0 && f1count++ <size)
29.         f3.writeInt(f1.readInt());
30. while(f2.available() > 0 && f2count++<size)
31.         f3.writeInt(f2.read());
32. }

33.         }
```

4. Compersion Of All sorting algorithm :

In this table n is the number of element to be sorted. The column Best case, Average case, worst case gives you the time complexity in each case.

[Table 1 : Comparison of sorting Algorithm]

Sorting Name	Best Case	Average Case	Worst Case
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Merge Sort	O(n logn)	O(n logn)	O(n logn)
Quick Sort	O(n logn)	O(n logn)	$O(n^2)$
Heap Sort	O(n logn)	O(n logn)	O(n logn)
Bucket and Radix Sort	O(n+N)	O(n+N)	$O(n^2)$

5. Conclusion : We can conclude that from all the sorting algorithm Quick sort is Best. Because Its time complexity is O(n logn).Merge sort time complexity is same as Quick sort but when the input size is large that time quick sort is better to use.

6. Future Work: In today mainly oops concept are using in computer science. what we have discussed in this paper we have tried to give a oops concept but that is not totally oops concept. We have use object of ArrayList, List in programming Who are a pre defined class in java. If we take each element as an object of a class and that object has some variable .We have to sort both the object and that variable. Here we can get the total oops concept. Here we have to sort both the object and that variable which is a good concept.This concept will come in next paper.

7. References

[1] Introduction to java programming Seventh Edition By Y.Danielliang.

[2] Data Structure using C Second Edition by Amiya Kumar Ratha and Alok Kumar Jagadev.

[3] Hoare, C.A.R. Quicksort. Computer Journal 5, 1 (April 1962), 10-15.

[4] Mehlhorn, K. Data Structures and Algorithms 1: Sorting and Searching. Springer-Verlag, Berlin, 1984.

[5] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[6] http://en.wikipedia.org/wiki/Bubble_sort

[7] Knuth D.E The art of programming-sorting and searching.2nd edition addison wesley.

Authors' Profile

Er.P.K.Dixit is the Director of Interface Software. He completed his M.Tech in Computer Sc.& Engg. From S.O.A. University, Bhubaneswar.

Ms. Archana Sahoo completed her MCA from OUAT , Bhubaneswar.

Bikash Kumar Sahoo has completed his B.Tech. from ITER, Bhubaneswar.