International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
International Conference & workshop on Advanced Computing 2014 (ICWAC 2014) – www.ijais.org

Adaptive Merge Sort

Nenwani Kamlesh
Ramrao Adik Institute of Technology
Nerul, Navi-Mumbai.

Vanita Mane
Ramrao Adik Institute of Technology
Nerul, Navi-Mumbai.

Smita Bharne
Ramrao Adik Institute of Technology
Nerul, Navi-Mumbai.

ABSTRACT
Merge Sort is a comparison based sorting algorithm with O(n log n) computational complexity. It is not adaptive to existence of ordering among the elements. Thus, has the same computational complexity in any case. In this paper, we propose Adaptive Merge Sort algorithm which is adaptive to existence of ordering among the elements in the list. Adaptive Merge sort has the complexity of O(n) for best case instead of O(n log n).Thus improvement requires additional space of O(n).The improvement in the performance is justified with an experimental analysis of the algorithm.

General Terms
Algorithm Optimization.

Keywords
Sorting, Merge Sort, Adaptive.

1. INTRODUCTION
Merge Sort is a comparison based algorithm invented by John von Neumann in 1945 [1]. It uses divide and conquer strategy to sort the list of elements. The algorithmic efficiency of merge sort is O(n log n). Merge sort has two approaches for implementation 1) Top-Down and 2) Bottom-Up. In top-down approach the list is divided into sub-lists until sub-list has only one element in it and then performs the merging process. Whereas in bottom-up approach each element of list is considered as sub-list and directly merging process starts with n sub-lists of size 1 [1], [2].

Although the merge sort computational complexity has lower order of growth than many sorting algorithms such as insertion, bubble etc, yet it is not suitable for list of smaller size. As the operation of dividing a list and then merging the list by placing it in temporary space and then putting back to its original location takes time [3]. To reduce the time the merging procedure can be improved using techniques described in [4]. However merge sort performs well for larger data set because it has lower order of growth and it can also use other sorting algorithm in conjunction to perform faster.

In this paper we focus on Bottom-Up approach of merge sort which is iterative and starts the merge procedure by considering each element of list as sub-list to be merged and propose Adaptive Merge Sort which is adaptive to existence of order (required or reverse) among the list of elements. The number of merging steps is reduced by locating sub-lists which are already sorted instead of starting with sub-list of size 1. Adaptive merge sort required additional storage space of O(n) for making Adaptive merge sort adaptive.

Further the paper is organized as follows: Section 2 describes the working of Merge Sort, Section 3 gives the design and implementation of Adaptive Merge Sort, Section 4 gives comparative analysis of Merge Sort and Adaptive Merge Sort, Section 5 gives the experimental analysis of Merge Sort and Adaptive Sort and the paper concludes in section 6.

2. MERGE SORT
Merge sort uses divide and conquer strategy to sort the list of elements. It starts merging process by taking each element of list as sub-list to be merged. The figure 1 shows the working of merge sort.

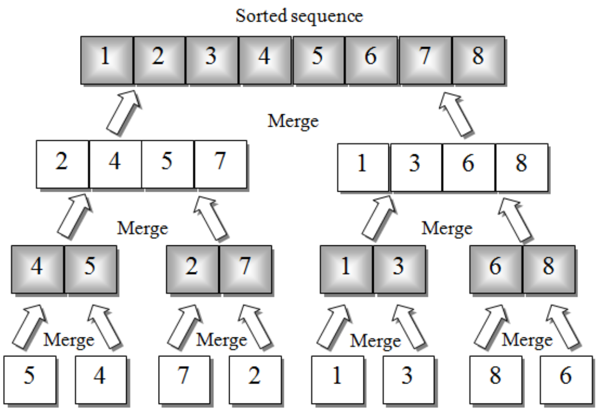

Fig. 1. Working of Merge Sort

2.1 Design Idea
1) Start with sub-list of size 1 (list with 1 element is considered sorted).

2) Keep on merging sub-lists to produce new sub-lists until there is only 1 list containing all the elements remaining.

2.2 Implementation in C
```
BottomUpSort( int n , int A[] , int B[])
{
  int w, i ;
  for (w=1;w<n; width=2*w)
  {
   for ( i =0;i<n; i=i+2*w)
   {
    merging (A, i ,min( i+w,n) ,min( i+2*w,n) ,B);
   }
   CopyArray(A,B,n );
  }
}
merging (int A[], int iLeft, int iRight, int iEnd, int B[])
{
  int i0=iLeft ; int i1=iRight ; int j ;
   for ( j=iLeft ; j<iEnd ; j ++)
```

International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868
Foundation of Computer Science FCS, New York, USA
International Conference & workshop on Advanced Computing 2014 (ICWAC 2014) – www.ijais.org

```
  {
    if (i0<iRight&&(i1>=iEnd | | A[ i0]<=A[ i1 ]))
    {
        B[ j ] = A[ i0 ];
        i0 = i0 + 1;
    }
    else
    {
       B[ j ] = A[ i1 ];
       i1 = i1 + 1;
    }
  }
}
```

Listing. 1. Merge Sort [5].

3. ADAPTIVE MERGE SORT

Adaptive Merge Sort performs the merging of sorted sub-list merge sort does. However, the size of initial sub-list depends upon the existence of ordering among the list of elements rather than having sub-list of size 1. For example consider list in the figure 2.

8	7	6	5	1	2	3	4

Fig. 2. List of Elements

It contains 2 sorted sub-lists.

- sub-list 1 with elements 8,7,6,5.
- sub-list 2 with elements 1,2,3,4.

8	7	6	5		1	2	3	4

Fig. 3. Sub-list of sorted Elements

The sub-list 1 is sorted but in reverse order. Thus, the sub-list 1 is reversed as shown in the figure 4.

5	6	7	8		1	2	3	4

Fig. 4. Sub-list of sorted elements in required order

Once the sub-lists are found merging process starts. Adaptive merge sort starts merging the sub-lists. Adaptive merge sort will require only one merging step as there are only 2 sub-lists. The result of merging is shown in figure 5

1	2	3	4	5	6	7	8

Fig. 5. Merging of sub-lists in figure 4.

3.1 Design Idea

1) Start by finding the sub-lists which are already sorted in required or reverse order

2) If there is any sub-list with elements in reverse order, then reverse the sub-list by exchanging 1st element with last, 2nd element with 2nd last and so on.

3) Keep on merging sub-lists to produce new sub-lists until there is only 1 sub-list remaining.

3.2 Implementation in C

```
void AdaptiveMerge ( int a [] , int b[] , int alength )
{
  int i =0, j =0,temp , lb = -1, ub = -1;
  int lb1 = -1,ub1 = -1,track = 0,p = 0;
  int as [ alength ] , prev track , ,k = 0;
  for ( j =0;j<alength -1;j ++)
  {
    if (a[ j]>a[ j +1])
    {
      if (lb1>-1)
      {
        b[ track ++] = lb1 ;
        b[ track ++] = ub1;
        lb1 = ub1 = -1;
        continue ;
      }
      else if ( lb == -1)
        lb = ub = j ;
      ub++;
    }
    else
    {
      if (lb>-1)
      {
        b[ track ++] = lb ;
        b[ track ++] = ub;
        while (lb<ub)
        {
          temp = a[ lb ];
          a[ lb ] = a[ub ];
          a[ub] = temp ;
          lb++;
          ub--;
        }
        lb = ub = -1;
        continue ;
      }
      else if ( lb1 == -1)
        lb1 = ub1 = j ;
      ub1++;
    }
  }
  if(lb>-1)
  {
    b[ track ++] = lb ;
    b[ track ++] = ub;
    while (lb<ub)
    {
      temp = a[ lb ];
      a[ lb ] = a[ub ];
      a[ub] = temp ;
      lb++; ub--;
    }
  }
 if (lb1>-1)
 {
   b[ track ++] = lb1 ;
   b[ track ++] = ub1;
 }
```

```
if (b[ track −1]<(alength −1))
{
   b[ track ++] = alength −1;
   b[ track ++] = alength −1;
 }
 prev track = track ;
 track = 0;
 while ( prev track >2)
 {
    int zl = b[k ];
    int zu = b[k+1];
    int xl = b[k+2];
    int xu = b[k+3];
    if ( zl>−1 && xl>−1)
    {
       for ( i=zl , j=xl ; i<=zu && j<=xu ;)
      {
         if (a[ i]>a[ j ])
         {
            as [p++] = a[ j ];
            j ++;
         }
        else
         {
            as [p++] = a[ i ];
            i ++;
         }
      }
      while (i<=zu)
      {
          as [p++] = a[ i ];
          i ++;
       }
       while (j<=xu)
       {
           as [p++] = a[ j ];
           j ++;
        }
        b[ track ++] = zl ;
        b[ track ++] = xu;
        k = k+4;
    }
    if (k+4>prev track )
    {
         if ((k+4−prev track)%4>0)
         {
              b[ track ++] = b[ prev track −2];
              b[ track ++] = b[ prev track −1];
         }
        k = 0;
        prev track = track ;
        track = 0;
        int y = 0;
       while (y<p)
       {
           a[y] = as [y ];
           y++;
        }
       p = 0;
    }
 }
}
```

Listing. 2. Merge Sort

4. ANALYSIS

4.1 Merge Sort

It works as follows:

suppose a list is of size 2n.

1) Starts with sub-list of size 1, sub-lists of size 1 are sorted.

2) Merge sub-lists of size1, results in sorted sub-lists of size 2.

3) Merge sub-lists of size2, results in sorted sub-list of size 4.

4)…...

5) The process of merging goes on until 2k<n. where k is the k^{th} merging step.

Each merging process requires a linear time of O(n) to merge n elements and 2k merging i.e.(log n) trips takes place. Thus, the Time Complexity of merge sort is O(n log n) [1], [2], [6]. Thus, it is clear that merge sort is not adaptive to existence of partial or total ordering in required or reverse order among the list to be sorted.

4.2 Adaptive Merge sort

Adaptive merge sort instead of starting with sub-list of size 1, finds a sub-list which are already in sorted in required or reverse order. The size of sub-lists found initially would be minimum 2 and maximum n (n is the number of elements). However, if the sub-list contains elements in reverse order, then it reverses the list before starting a merge operation. The reversal of list requires (n/2) exchange operations.

4.2.1 Best Case

If list is already in sorted order or in reverse order then the Adaptive merge sort will have only one list and will not require any merge operation. However, finding that the list is already sorted will require O(n) comparison operation and (n/2) exchange operation if the list is sorted in reverse order. This makes the Adaptive Merge sort adaptive even when the list sorted in reverse order.

Thus the Time complexity for best case is calculated as follows:

$$T(n) = (n-1)+(n/2)$$
$$T(n) = (2n-2+n)/2$$
$$T(n) = O(n).$$

However to Adaptive merge sort uses additional space of O(n) in comparison of merge sort

4.2.2 Worst Case

Adaptive merge sort will find sub-list which is already sorted in required or reverse order. However, in worst case there are no partial or total ordering elements. Thus, the sub-list found initially would be of size 2. Once the sub-lists are found the merging process starts.

• merging sub-lists of size 2 results in sorted sub-list of size 4.
• merging sub-lists of size 4 results in sorted sub-list of size 8.
• ...
• The process of merging goes on until 2k < n. where k is the k^{th} merging step.

Since the merging steps in worst case of Adaptive merge sort is same as merge sort. Thus, the Time Complexity for worst case of Adaptive merge sort is same as merge sort:

$$T(n) = O(n \log n).$$

4.3 Comparison of Merge and Adaptive Merge Sort

Table 1. Complexity Of Merge Sort

	Required order	**Random order**	**Reverse order**
No. of Merging steps	log n	log n	log n
Space Complexity	n	N	n
Time Complexity	n log n	n log n	n log n

Table 2. Complexity Of Adaptive Merge Sort

	Required order	**Random order**	**Reverse order**
No. of Merging steps	1	log n	1
Space Complexity	2n	2n	2n
Time Complexity	n	n log n	n

5. EXPERIMENTAL ANALYSIS

The efficiency, performance and correctness of Adaptive Merge Sort is checked and compared with Merge Sort. The result of comparison is shown below.

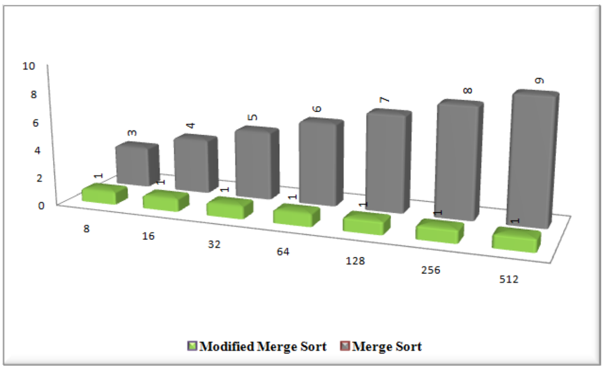

Fig. 6. Elements in Reverse Order

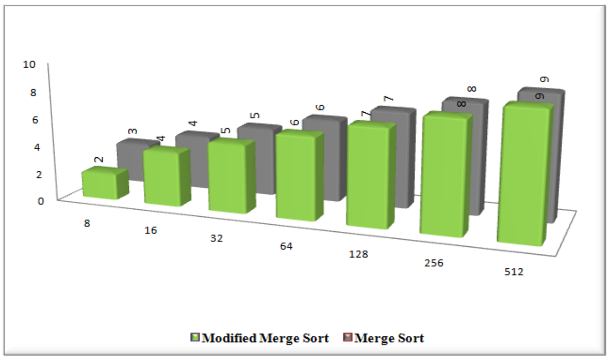

Fig. 7. Elements in Random Order

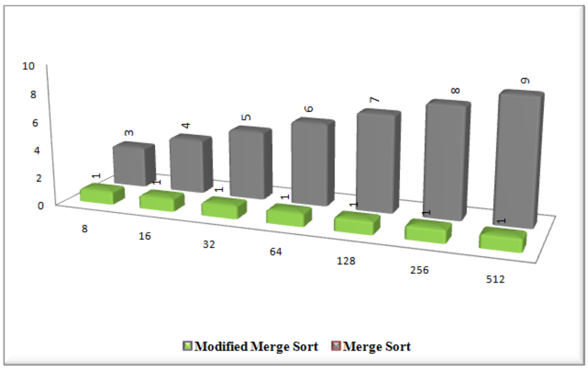

Fig. 8. Elements in Required Order

6. CONCLUSION

Thus Adaptive Merge Sort algorithm is adaptive to existence of order and has computational complexity of O(n) when the list is sorted in required or reverse order i.e.(best case) and O(n log n) in other cases.

Also, it can be concluded from experimental analysis that Adaptive merge sort out performs better than merge sort whenever the list is nearly sorted. However, the worst case complexity still O(n log n) same as merge sort. The Adaptive merge sort provides better performance at the cost additional storage of O(n). Thus, the space requirement for Adaptive merge sort is 2n whereas merge sort requires n.

7. REFERENCES

[1] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, Introduction to algorithms. The MIT press, 2001.

[2] A. Tenenbaum, Data Structures Using C. Pearson Education, 1990. [Online]. Available: http://books.google.co.in/books?id=X0Cd1Pr2W0gC.

[3] D. E. Knuth, “The art of computer programming, vol. 3: Sorting and searching,” 1973.”

[4] Estivill-Castro, Vladmir, and Derick Wood. "A survey of adaptive sorting algorithms." *ACM Computing Surveys (CSUR)* 24.4 (1992): 441-476.

[5] Lipschutz, Data Structures With C. McGraw-Hill Education (India) Pvt Limited, 2011. [Online]. Available: http://books.google.co.in/books?id=YJQIOLgFnnYC.

[6] Symvonis, Antonios. "Optimal stable merging." *The Computer Journal* 38.8 (1995): 681-690.

[7] Brown, Mark R., and Robert E. Tarjan. "A fast merging algorithm." *Journal of the ACM (JACM)* 26.2 (1979): 211-226.