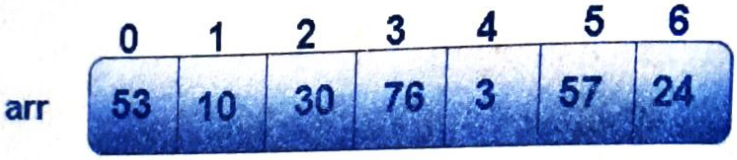
Sorting Data by Using Merge Sort

Another sorting algorithm that is efficient for sorting large lists of data is merge sort. Li quick sort algorithm, it uses the divide and conquer approach to sort the list. Using this algorithm, the list to be sorted is divided into two sublists of sizes as nearly equal as possible, and then the two sublists are sorted separately by using merge sort. The two sorted sublists are then merged into a single sorted list.

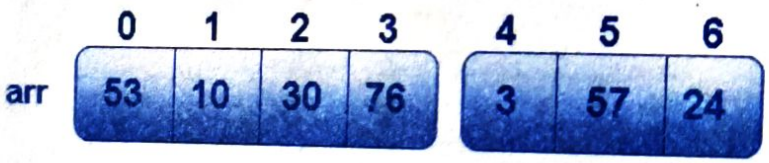
Implementing Merge Sort Algorithm

To understand the implementation of merge sort algorithm, consider the following unsorted list that needs to be sorted.



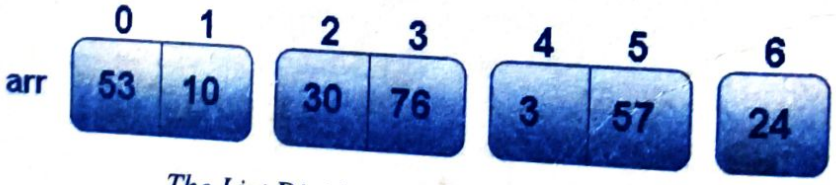
The Unsorted List

The first step to sort data by using merge sort is to split the list into parts. Therefore, you first divide the list into two equal halves. If the list has odd number of elements, then the left sublist is longer than the right sublist by one entry, as shown in the following figure.



The List Divided into Two Halves

The two sublists are further divided into sublists, as shown in the following figure.



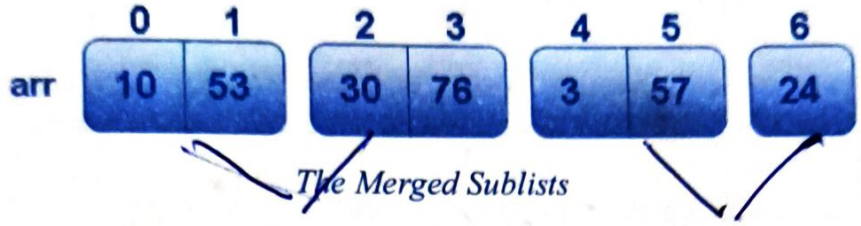
The List Divided into Four Sublists

The sublists obtained are further divided, as shown in the following figure.

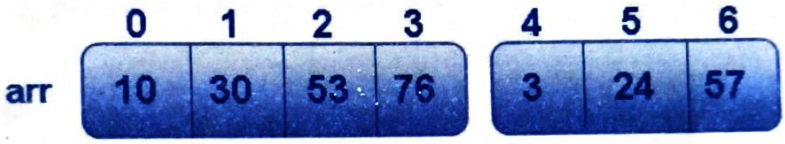
The List Divided into Seven Sublists

At this stage, there is a single element left in each sublist. Sublists with one element require no sorting.

Now, you need to start merging the sublists in such a way that the resultant lists are sorted. The sublists [53] and [10] are merged to form the sorted list [10 53]. Similarly, the sublists [30] and [76] are merged to form the sorted sublist [30 76]. The other pairs of lists are also sorted in a similar way to obtain the following sorted sublists.



Now, the first two sublists are merged to form a four element sorted list, and the remaining two sublists are merged to form a three element sorted list, as shown in the following figure.



The Merged Sublists

Then, these sorted lists are merged to form a complete sorted list, as shown in the following figure.

The original list is now sorted.

The following is the algorithm for merge sort:

Algorithm: MergeSort(low, high)

1. If low >= high:
   a. Return
2. Set mid = (low + high)/2
3. Divide the list into two sublists of nearly equal lengths, and sort each sublist by using merge sort. The steps to do this are as follows:
   a. MergeSort(low, mid)
   b. MergeSort(mid + 1, high)

4. Merge the two sorted sublists:
   a. Set i = low
   b. Set j = mid + 1
   c. Set k = low
   d. Repeat until i > mid or j > high: **// This loop will terminate when you reach // the end of one of the two sublists.**

      i. If (arr[i] <= arr[j])
         Store arr[i] at index k in array B
         Increment i by 1
         Else
         Store arr[j] at index k in array B
         Increment j by 1
      ii. Increment k by 1

   e. Repeat until j > high: **// If there are still some elements in the second // sublist append them to the new list**

      i. Store arr[j] at index k in array B
      ii. Increment j by 1
      iii. Increment k by 1

   f. Repeat until i > mid: **// If there are still some elements in the first sublist // append them to the new list**

      i. Store arr[i] at index k in array B
      ii. Increment i by 1
      iii. Increment k by 1

5. Copy all elements from the sorted array B into the original array arr

Determining the Efficiency of Merge Sort Algorithm

Consider an example of a list of size n. To sort the list, you need to recursively divide the list into two nearly equal sublists until each sublist contains only one element. To divide the list into sublists of size one requires log n passes.

In each pass, a maximum of n comparisons are performed. Therefore, the total number of comparisons will be a maximum of n × log n. Consequently, the efficiency of merge sort is equal to O(n log n).

There is no distinction between best, average, and worst case efficiencies of merge sort because all of them require the same amount of time.