Merge Sort Algorithm

In this article, we will discuss the merge sort Algorithm. Merge sort is the sorting technique that follows the divide and conquer approach. This article will be very helpful and interesting to students as they might face merge sort as a question in their examinations. In coding or technical interviews for software engineers, sorting algorithms are widely asked. So, it is important to discuss the topic.

Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithm. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the **merge()** function to perform the merging.

The sub-lists are divided again and again into halves until the list cannot be divided further. Then we combine the pair of one element lists into two-element lists, sorting them in the process. The sorted two-element pairs is merged into the four-element lists, and so on until we get the sorted list.

Now, let's see the algorithm of merge sort.

Java

Algorithm

In the following algorithm, **arr** is the given array, **beg** is the starting element, and **end** is the last element of the array.

```
1. MERGE_SORT(arr, beg, end)
2.
3. if beg < end
4. set mid = (beg + end)/2
5. MERGE_SORT(arr, beg, mid)
6. MERGE_SORT(arr, mid + 1, end)
7. MERGE (arr, beg, mid, end)
8. end of if
9.
10. END MERGE_SORT
```

The important part of the merge sort is the **MERGE** function. This function performs the merging of two sorted sub-arrays that are **A[beg…mid]** and **A[mid+1…end]**, to

build one sorted array **A[beg...end]**. So, the inputs of the **MERGE** function are **A[], beg, mid,** and **end**.

The implementation of the **MERGE** function is given as follows -

```
1.  /* Function to merge the subarrays of a[] */
2.  void merge(int a[], int beg, int mid, int end)
3.  {
4.      int i, j, k;
5.      int n1 = mid - beg + 1;
6.      int n2 = end - mid;
7.
8.      int LeftArray[n1], RightArray[n2]; //temporary arrays
9.
10.     /* copy data to temp arrays */
11.     for (int i = 0; i < n1; i++)
12.     LeftArray[i] = a[beg + i];
13.     for (int j = 0; j < n2; j++)
14.     RightArray[j] = a[mid + 1 + j];
15.
16.     i = 0, /* initial index of first sub-array */
17.     j = 0; /* initial index of second sub-array */
18.     k = beg;  /* initial index of merged sub-array */
19.
20.     while (i < n1 && j < n2)
21.     {
22.         if(LeftArray[i] <= RightArray[j])
23.         {
24.             a[k] = LeftArray[i];
25.             i++;
26.         }
27.         else
28.         {
29.             a[k] = RightArray[j];
30.             j++;
31.         }
32.         k++;
33.     }
```

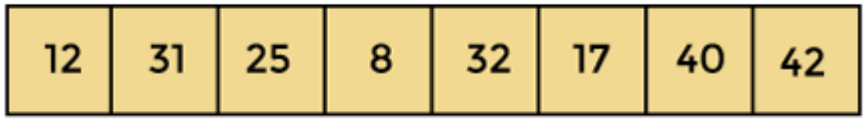
```
34.    while (i<n1)
35.    {
36.        a[k] = LeftArray[i];
37.        i++;
38.        k++;
39.    }
40.
41.    while (j<n2)
42.    {
43.        a[k] = RightArray[j];
44.        j++;
45.        k++;
46.    }
47. }
```

Working of Merge sort Algorithm

Now, let's see the working of merge sort Algorithm.

To understand the working of the merge sort algorithm, let's take an unsorted array. It will be easier to understand the merge sort via an example.

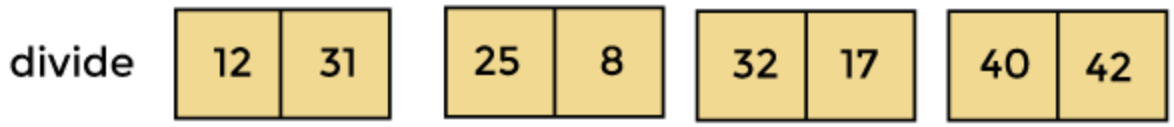
Let the elements of array are -



According to the merge sort, first divide the given array into two equal halves. Merge sort keeps dividing the list into equal parts until it cannot be further divided.

As there are eight elements in the given array, so it is divided into two arrays of size 4.

Now, again divide these two arrays into halves. As they are of size 4, so divide them into new arrays of size 2.

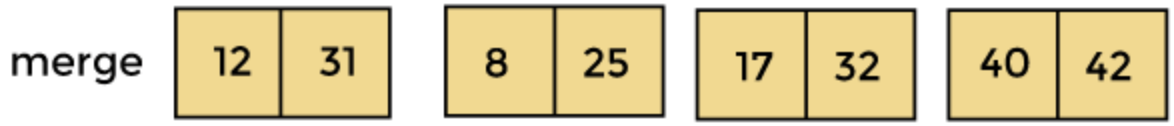


Now, again divide these arrays to get the atomic value that cannot be further divided.

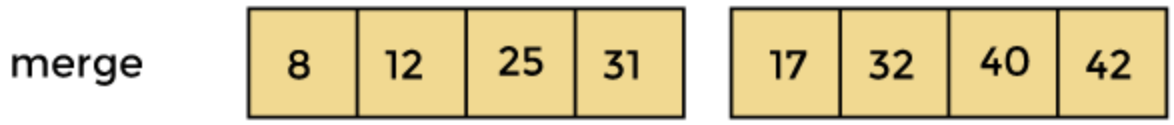
Now, combine them in the same manner they were broken.

In combining, first compare the element of each array and then combine them into another array in sorted order.

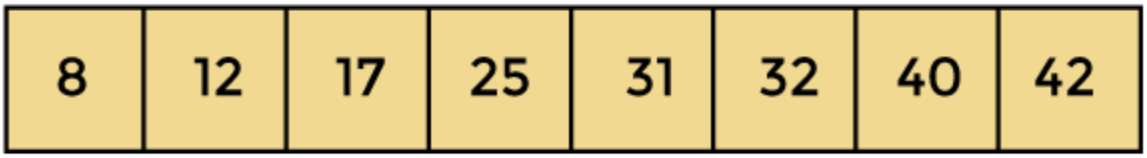
So, first compare 12 and 31, both are in sorted positions. Then compare 25 and 8, and in the list of two values, put 8 first followed by 25. Then compare 32 and 17, sort them and put 17 first followed by 32. After that, compare 40 and 42, and place them sequentially.



In the next iteration of combining, now compare the arrays with two data values and merge them into an array of found values in sorted order.



Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like -



Now, the array is completely sorted.

Merge sort complexity

Now, let's see the time complexity of merge sort in best case, average case, and in worst case. We will also see the space complexity of the merge sort.

1. Time Complexity

Case	Time Complexity
Best Case	O(n*logn)
Average Case	O(n*logn)
Worst Case	O(n*logn)

- **Best Case Complexity -** It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of merge sort is **O(n*logn)**.
- **Average Case Complexity -** It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of merge sort is **O(n*logn)**.
- **Worst Case Complexity -** It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of merge sort is **O(n*logn)**.

2. Space Complexity

Space Complexity	O(
Stable	YE

- The space complexity of merge sort is O(n). It is because, in merge sort, an extra variable is required for swapping.

Implementation of merge sort

Now, let's see the programs of merge sort in different programming languages.

Program: Write a program to implement merge sort in C language.

```c
#include <stdio.h>

/* Function to merge the subarrays of a[] */
void merge(int a[], int beg, int mid, int end)
{
    int i, j, k;
    int n1 = mid - beg + 1;
    int n2 = end - mid;

    int LeftArray[n1], RightArray[n2]; //temporary arrays

    /* copy data to temp arrays */
    for (int i = 0; i < n1; i++)
    LeftArray[i] = a[beg + i];
    for (int j = 0; j < n2; j++)
    RightArray[j] = a[mid + 1 + j];

    i = 0; /* initial index of first sub-array */
    j = 0; /* initial index of second sub-array */
    k = beg;  /* initial index of merged sub-array */

    while (i < n1 && j < n2)
    {
        if(LeftArray[i] <= RightArray[j])
        {
            a[k] = LeftArray[i];
            i++;
        }
        else
        {
            a[k] = RightArray[j];
            j++;
        }
        k++;
    }
    while (i<n1)
    {
```

```
38.         a[k] = LeftArray[i];
39.         i++;
40.         k++;
41.     }
42.
43.     while (j<n2)
44.     {
45.         a[k] = RightArray[j];
46.         j++;
47.         k++;
48.     }
49. }
50.
51. void mergeSort(int a[], int beg, int end)
52. {
53.     if (beg < end)
54.     {
55.         int mid = (beg + end) / 2;
56.         mergeSort(a, beg, mid);
57.         mergeSort(a, mid + 1, end);
58.         merge(a, beg, mid, end);
59.     }
60. }
61.
62. /* Function to print the array */
63. void printArray(int a[], int n)
64. {
65.     int i;
66.     for (i = 0; i < n; i++)
67.         printf("%d ", a[i]);
68.     printf("\n");
69. }
70.
71. int main()
72. {
73.     int a[] = { 12, 31, 25, 8, 32, 17, 40, 42 };
74.     int n = sizeof(a) / sizeof(a[0]);
```

```
75.     printf("Before sorting array elements are - \n");
76.     printArray(a, n);
77.     mergeSort(a, 0, n - 1);
78.     printf("After sorting array elements are - \n");
79.     printArray(a, n);
80.     return 0;
81. }
```

Output:

```
Before sorting array elements are -
12 31 25 8 32 17 40 42
After sorting array elements are -
8 12 17 25 31 32 40 42
```

Program: Write a program to implement merge sort in C++ language.

```
1.  #include <iostream>
2.
3.  using namespace std;
4.
5.  /* Function to merge the subarrays of a[] */
6.  void merge(int a[], int beg, int mid, int end)
7.  {
8.      int i, j, k;
9.      int n1 = mid - beg + 1;
10.     int n2 = end - mid;
11.
12.     int LeftArray[n1], RightArray[n2]; //temporary arrays
13.
14.     /* copy data to temp arrays */
15.     for (int i = 0; i < n1; i++)
16.     LeftArray[i] = a[beg + i];
17.     for (int j = 0; j < n2; j++)
18.     RightArray[j] = a[mid + 1 + j];
19.
20.     i = 0; /* initial index of first sub-array */
21.     j = 0; /* initial index of second sub-array */
22.     k = beg;  /* initial index of merged sub-array */
```

```
23.
24.     while (i < n1 && j < n2)
25.     {
26.         if(LeftArray[i] <= RightArray[j])
27.         {
28.             a[k] = LeftArray[i];
29.             i++;
30.         }
31.         else
32.         {
33.             a[k] = RightArray[j];
34.             j++;
35.         }
36.         k++;
37.     }
38.     while (i<n1)
39.     {
40.         a[k] = LeftArray[i];
41.         i++;
42.         k++;
43.     }
44.
45.     while (j<n2)
46.     {
47.         a[k] = RightArray[j];
48.         j++;
49.         k++;
50.     }
51. }
52.
53. void mergeSort(int a[], int beg, int end)
54. {
55.     if (beg < end)
56.     {
57.         int mid = (beg + end) / 2;
58.         mergeSort(a, beg, mid);
59.         mergeSort(a, mid + 1, end);
```

```
60.         merge(a, beg, mid, end);
61.     }
62. }
63.
64. /* Function to print the array */
65. void printArray(int a[], int n)
66. {
67.     int i;
68.     for (i = 0; i < n; i++)
69.         cout<<a[i]<<" ";
70. }
71.
72. int main()
73. {
74.     int a[] = { 11, 30, 24, 7, 31, 16, 39, 41 };
75.     int n = sizeof(a) / sizeof(a[0]);
76.     cout<<"Before sorting array elements are - \n";
77.     printArray(a, n);
78.     mergeSort(a, 0, n - 1);
79.     cout<<"\nAfter sorting array elements are - \n";
80.     printArray(a, n);
81.     return 0;
82. }
```

Output:

```
Before sorting array elements are -
11 30 24 7 31 16 39 41
After sorting array elements are -
7 11 16 24 30 31 39 41
```

Program: Write a program to implement merge sort in Java.

```
1.  class Merge {
2.
3.  /* Function to merge the subarrays of a[] */
4.  void merge(int a[], int beg, int mid, int end)
5.  {
6.      int i, j, k;
```

```
7.      int n1 = mid - beg + 1;
8.      int n2 = end - mid;
9.
10.   /* temporary Arrays */
11.       int LeftArray[] = new int[n1];
12.       int RightArray[] = new int[n2];
13.
14.   /* copy data to temp arrays */
15.   for (i = 0; i < n1; i++)
16.   LeftArray[i] = a[beg + i];
17.   for (j = 0; j < n2; j++)
18.   RightArray[j] = a[mid + 1 + j];
19.
20.   i = 0; /* initial index of first sub-array */
21.   j = 0; /* initial index of second sub-array */
22.   k = beg;  /* initial index of merged sub-array */
23.
24.   while (i < n1 && j < n2)
25.   {
26.       if(LeftArray[i] <= RightArray[j])
27.       {
28.           a[k] = LeftArray[i];
29.           i++;
30.       }
31.       else
32.       {
33.           a[k] = RightArray[j];
34.           j++;
35.       }
36.       k++;
37.   }
38.   while (i<n1)
39.   {
40.       a[k] = LeftArray[i];
41.       i++;
42.       k++;
43.   }
```

```
44.
45.     while (j<n2)
46.     {
47.         a[k] = RightArray[j];
48.         j++;
49.         k++;
50.     }
51. }
52.
53. void mergeSort(int a[], int beg, int end)
54. {
55.     if (beg < end)
56.     {
57.         int mid = (beg + end) / 2;
58.         mergeSort(a, beg, mid);
59.         mergeSort(a, mid + 1, end);
60.         merge(a, beg, mid, end);
61.     }
62. }
63.
64. /* Function to print the array */
65. void printArray(int a[], int n)
66. {
67.     int i;
68.     for (i = 0; i < n; i++)
69.         System.out.print(a[i] + " ");
70. }
71.
72. public static void main(String args[])
73. {
74.     int a[] = { 11, 30, 24, 7, 31, 16, 39, 41 };
75.     int n = a.length;
76.     Merge m1 = new Merge();
77.     System.out.println("\nBefore sorting array elements are - ");
78.     m1.printArray(a, n);
79.     m1.mergeSort(a, 0, n - 1);
80.     System.out.println("\nAfter sorting array elements are - ");
```

```
81.     m1.printArray(a, n);
82.     System.out.println("");
83. }
84.
85.  }
```

Output:

```
D:\JTP>javac Merge.java

D:\JTP>java Merge

Before sorting array elements are -
11 30 24 7 31 16 39 41
After sorting array elements are -
7 11 16 24 30 31 39 41
```

Program: Write a program to implement merge sort in C#.

```
1.  using System;
2.  class Merge {
3.
4.  /* Function to merge the subarrays of a[] */
5.  static void merge(int[] a, int beg, int mid, int end)
6.  {
7.      int i, j, k;
8.      int n1 = mid - beg + 1;
9.      int n2 = end - mid;
10.
11.   //temporary arrays
12.     int[] LeftArray = new int [n1];
13.     int[] RightArray = new int [n2];
14.
15.     /* copy data to temp arrays */
16.     for (i = 0; i < n1; i++)
17.     LeftArray[i] = a[beg + i];
18.     for (j = 0; j < n2; j++)
19.     RightArray[j] = a[mid + 1 + j];
20.
21.     i = 0; /* initial index of first sub-array */
22.     j = 0; /* initial index of second sub-array */
```

```
23.     k = beg;  /* initial index of merged sub-array */
24.
25.     while (i < n1 && j < n2)
26.     {
27.         if(LeftArray[i] <= RightArray[j])
28.         {
29.             a[k] = LeftArray[i];
30.             i++;
31.         }
32.         else
33.         {
34.             a[k] = RightArray[j];
35.             j++;
36.         }
37.         k++;
38.     }
39.     while (i<n1)
40.     {
41.         a[k] = LeftArray[i];
42.         i++;
43.         k++;
44.     }
45.
46.     while (j<n2)
47.     {
48.         a[k] = RightArray[j];
49.         j++;
50.         k++;
51.     }
52. }
53.
54. static void mergeSort(int[] a, int beg, int end)
55. {
56.     if (beg < end)
57.     {
58.         int mid = (beg + end) / 2;
59.         mergeSort(a, beg, mid);
```

```
60.         mergeSort(a, mid + 1, end);
61.         merge(a, beg, mid, end);
62.     }
63. }
64.
65. /* Function to print the array */
66. static void printArray(int[] a, int n)
67. {
68.     int i;
69.     for (i = 0; i < n; i++)
70.         Console.Write(a[i] + " ");
71. }
72.
73. static void Main()
74. {
75.     int[] a = { 10, 29, 23, 6, 30, 15, 38, 40 };
76.     int n = a.Length;
77.     Console.Write("Before sorting array elements are - ");
78.     printArray(a, n);
79.     mergeSort(a, 0, n - 1);
80.     Console.Write("\nAfter sorting array elements are - ");
81.     printArray(a, n);
82. }
83.
84.  }
```

Output:

```
Before sorting array elements are - 10 29 23 6 30 15 38 40
After sorting array elements are - 6 10 15 23 29 30 38 40
```

Program: Write a program to implement merge sort in PHP.

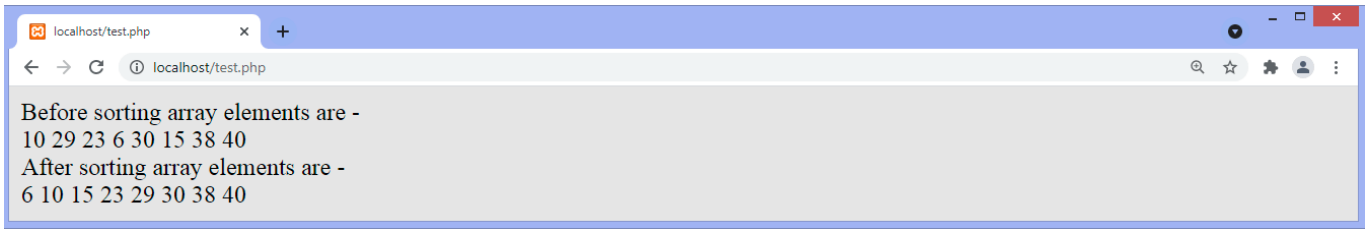
```
1.  <?php
2.
3.  /* Function to merge the subarrays of a[] */
4.  function merge(&$a, $beg, $mid, $end)
5.  {
```

```
6.      $n1 = ($mid - $beg) + 1;
7.      $n2 = $end - $mid;
8.
9.     /* temporary Arrays */
10.        $LeftArray = array($n1);
11.        $RightArray = array($n2);
12.
13.    /* copy data to temp arrays */
14.    for ($i = 0; $i < $n1; $i++)
15.    $LeftArray[$i] = $a[$beg + $i];
16.    for ($j = 0; $j < $n2; $j++)
17.    $RightArray[$j] = $a[$mid + 1 + $j];
18.
19.    $i = 0; /* initial index of first sub-array */
20.    $j = 0; /* initial index of second sub-array */
21.    $k = $beg;  /* initial index of merged sub-array */
22.
23.    while ($i<$n1 && $j<$n2)
24.    {
25.        if($LeftArray[$i] <= $RightArray[$j])
26.        {
27.            $a[$k] = $LeftArray[$i];
28.            $i++;
29.        }
30.        else
31.        {
32.            $a[$k] = $RightArray[$j];
33.            $j++;
34.        }
35.        $k++;
36.    }
37.    while ($i<$n1)
38.    {
39.        $a[$k] = $LeftArray[$i];
40.        $i++;
41.        $k++;
42.    }
```

```
43.
44.     while ($j<$n2)
45.     {
46.         $a[$k] = $RightArray[$j];
47.         $j++;
48.         $k++;
49.     }
50. }
51.
52. function mergeSort(&$a, $beg, $end)
53. {
54.     if ($beg < $end)
55.     {
56.         $mid = (int)(($beg + $end) / 2);
57.         mergeSort($a, $beg, $mid);
58.         mergeSort($a, $mid + 1, $end);
59.         merge($a, $beg, $mid, $end);
60.     }
61. }
62.
63. /* Function to print array elements */
64.     function printArray($a, $n)
65.     {
66.     for($i = 0; $i < $n; $i++)
67.     {
68.         print_r($a[$i]);
69.         echo " ";
70.     }
71.     }
72.
73.     $a = array( 10, 29, 23, 6, 30, 15, 38, 40 );
74.     $n = count($a);
75.     echo "Before sorting array elements are - <br>";
76.     printArray($a, $n);
77.     mergeSort($a, 0, $n - 1);
78.     echo "<br> After sorting array elements are - <br>";
79.     printArray($a, $n);
```

80. ?>

Output:


localhost/test.php
localhost/test.php
Before sorting array elements are -
10 29 23 6 30 15 38 40
After sorting array elements are -
6 10 15 23 29 30 38 40