GeeksforGeeks
A computer science portal for geeks

Custom Search

Geeks Classes Login

Write an Article

# Iterative Merge Sort

Following is a typical recursive implementation of Merge Sort that uses last element as pivot.

## C/C++

```
/* Recursive C program for merge sort */
#include<stdlib.h>
#include<stdio.h>

/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r);

/* l is for left index and r is right index of the sub-array
  of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{
   if (l < r)
   {
      int m = l+(r-l)/2; //Same as (l+r)/2 but avoids overflow for large l & h
      mergeSort(arr, l, m);
      mergeSort(arr, m+1, r);
      merge(arr, l, m, r);
   }
}

/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there are any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int arr_size = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    printf("\nSorted array is \n");
    printArray(arr, arr_size);
    return 0;
}
```

Run on IDE

## Java

```
// Recursive Java Program for merge sort

import java.util.Arrays;
public class GFG
{
    public static void mergeSort(int[] array)
    {
        if(array == null)
        {
            return;
        }

        if(array.length > 1)
        {
            int mid = array.length / 2;

            // Split left part
            int[] left = new int[mid];
            for(int i = 0; i < mid; i++)
            {
                left[i] = array[i];
            }

            // Split right part
            int[] right = new int[array.length - mid];
            for(int i = mid; i < array.length; i++)
            {
                right[i - mid] = array[i];
            }
            mergeSort(left);
            mergeSort(right);

            int i = 0;
            int j = 0;
            int k = 0;

            // Merge left and right arrays
            while(i < left.length && j < right.length)
            {
                if(left[i] < right[j])
                {
                    array[k] = left[i];
                    i++;
                }
                else
                {
                    array[k] = right[j];
                    j++;
                }
                k++;
            }
            // Collect remaining elements
            while(i < left.length)
            {
                array[k] = left[i];
                i++;
                k++;
            }
            while(j < right.length)
            {
                array[k] = right[j];
                j++;
                k++;
            }
        }
    }

    // Driver program to test above functions.
    public static void main(String[] args)
    {
        int arr[] = {12, 11, 13, 5, 6, 7};
        int i=0;
        System.out.println("Given array is");

        for(i=0; i<arr.length; i++)
            System.out.print(arr[i]+" ");
        mergeSort(arr);

        System.out.println("\n");
        System.out.println("Sorted array is");

        for(i=0; i<arr.length; i++)
            System.out.print(arr[i]+" ");
    }
}

// Code Contributed by Mohit Gupta_OMG
```

Run on IDE

## Python

```python
# Recursive Python Program for merge sort

def merge(left, right):
    if not len(left) or not len(right):
        return left or right

    result = []
    i, j = 0, 0
    while (len(result) < len(left) + len(right)):
        if left[i] < right[j]:
            result.append(left[i])
            i+= 1
        else:
            result.append(right[j])
            j+= 1
        if i == len(left) or j == len(right):
            result.extend(left[i:] or right[j:])
            break

    return result

def mergesort(list):
    if len(list) < 2:
        return list

    middle = len(list)/2
    left = mergesort(list[:middle])
    right = mergesort(list[middle:])

    return merge(left, right)

seq = [12, 11, 13, 5, 6, 7]
print("Given array is")
print(seq);
print("\n")
print("Sorted array is")
print(mergesort(seq))

# Code Contributed by Mohit Gupta_OMG
```

Run on IDE

## C#

```
/* Iterative C# program for merge
sort */
using System;
```

```
class GFG {

    /* l is for left index and r
    is right index of the sub-array
    of arr to be sorted */
    static void mergeSort(int[] arr,
                               int l, int r)
    {
        if (l < r)
        {

            // Same as (l+r)/2 but avoids
            // overflow for large l & h
            int m = l + (r - l) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m+1, r);
            merge(arr, l, m, r);
        }
    }

    /* Function to merge the two haves
    arr[l..m] and arr[m+1..r] of array
    arr[] */
    static void merge(int[] arr, int l,
                               int m, int r)
    {
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;

        /* create temp arrays */
        int []L = new int[n1];
        int []R = new int[n2];

        /* Copy data to temp arrays
        L[] and R[] */
        for (i = 0; i < n1; i++)
            L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
            R[j] = arr[m + 1+ j];

        /* Merge the temp arrays back
        into arr[l..r]*/
        i = 0;
        j = 0;
        k = l;
        while (i < n1 && j < n2)
        {
            if (L[i] <= R[j])
            {
                arr[k] = L[i];
                i++;
            }
            else
            {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        /* Copy the remaining elements of
        L[], if there are any */
        while (i < n1)
        {
            arr[k] = L[i];
            i++;
            k++;
        }
        /* Copy the remaining elements of
        R[], if there are any */
        while (j < n2)
        {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    /* Function to print an array */
    static void printArray(int []A, int size)
    {
        int i;
        for (i=0; i < size; i++)
            Console.Write(A[i]+" ");
        Console.Write("\n");
    }

    /* Driver program to test above functions */
    public static void Main()
    {
        int []arr = {12, 11, 13, 5, 6, 7};
        int arr_size = arr.Length;

        Console.Write("Given array is \n");
        printArray(arr, arr_size);

        mergeSort(arr, 0, arr_size - 1);

        Console.Write("\nSorted array is \n");
        printArray(arr, arr_size);
    }
}

// This code is contributed by Smitha
```

Run on IDE

**Output:**

```
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13
```

**Iterative Merge Sort:**

The above function is recursive, so uses function call stack to store intermediate values of l and h. The function call stack stores other bookkeeping information together with parameters. Also, function calls involve overheads like storing activation record of the caller function and then resuming execution. Unlike Iterative QuickSort, the iterative MergeSort doesn't require explicit auxiliary stack.

The above function can be easily converted to iterative version. Following is iterative Merge Sort.

C

```
/* Iterative C program for merge sort */
#include<stdlib.h>
#include<stdio.h>

/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r);

// Utility function to find minimum of two integers
int min(int x, int y) { return (x<y)? x :y; }


/* Iterative mergesort function to sort arr[0...n-1] */
void mergeSort(int arr[], int n)
{
   int curr_size;  // For current size of subarrays to be merged
                   // curr_size varies from 1 to n/2
   int left_start; // For picking starting index of left subarray
                   // to be merged

   // Merge subarrays in bottom up manner.  First merge subarrays of
   // size 1 to create sorted subarrays of size 2, then merge subarrays
   // of size 2 to create sorted subarrays of size 4, and so on.
   for (curr_size=1; curr_size<=n-1; curr_size = 2*curr_size)
   {
       // Pick starting point of different subarrays of current size
       for (left_start=0; left_start<n-1; left_start += 2*curr_size)
       {
           // Find ending point of left subarray. mid+1 is starting
           // point of right
           int mid = left_start + curr_size - 1;

           int right_end = min(left_start + 2*curr_size - 1, n-1);

           // Merge Subarrays arr[left_start...mid] & arr[mid+1...right_end]
           merge(arr, left_start, mid, right_end);
       }
   }
}

/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there are any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Given array is \n");
    printArray(arr, n);

    mergeSort(arr, n);

    printf("\nSorted array is \n");
    printArray(arr, n);
    return 0;
}
```

Run on IDE

## Java

```
/* Iterative Java program for merge sort */
import java.lang.Math.*;

class GFG {

    /* Iterative mergesort function to sor
    t arr[0...n-1] */
    static void mergeSort(int arr[], int n)
    {

        // For current size of subarrays to
        // be merged curr_size varies from
        // 1 to n/2
        int curr_size;

        // For picking starting index of
        // left subarray to be merged
        int left_start;


        // Merge subarrays in bottom up
        // manner. First merge subarrays
        // of size 1 to create sorted
        // subarrays of size 2, then merge
        // subarrays of size 2 to create
        // sorted subarrays of size 4, and
        // so on.
        for (curr_size = 1; curr_size <= n-1;
                      curr_size = 2*curr_size)
        {

            // Pick starting point of different
            // subarrays of current size
            for (left_start = 0; left_start < n-1;
                        left_start += 2*curr_size)
            {
                // Find ending point of left
                // subarray. mid+1 is starting
                // point of right
                int mid = left_start + curr_size - 1;

                int right_end = Math.min(left_start
                             + 2*curr_size - 1, n-1);

                // Merge Subarrays arr[left_start...mid]
                // & arr[mid+1...right_end]
                merge(arr, left_start, mid, right_end);
            }
        }
    }

    /* Function to merge the two haves arr[l..m] and
    arr[m+1..r] of array arr[] */
    static void merge(int arr[], int l, int m, int r)
    {
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;

        /* create temp arrays */
        int L[] = new int[n1];
        int R[] = new int[n2];

        /* Copy data to temp arrays L[]
        and R[] */
        for (i = 0; i < n1; i++)
            L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
            R[j] = arr[m + 1+ j];

        /* Merge the temp arrays back into
        arr[l..r]*/
        i = 0;
        j = 0;
        k = l;
        while (i < n1 && j < n2)
        {
            if (L[i] <= R[j])
            {
                arr[k] = L[i];
                i++;
            }
            else
            {
                arr[k] = R[j];
                j++;
            }
            k++;
        }

        /* Copy the remaining elements of
        L[], if there are any */
        while (i < n1)
        {
            arr[k] = L[i];
            i++;
            k++;
        }

        /* Copy the remaining elements of
        R[], if there are any */
        while (j < n2)
        {
            arr[k] = R[j];
            j++;
            k++;
        }
    }

    /* Function to print an array */
    static void printArray(int A[], int size)
    {
        int i;
        for (i=0; i < size; i++)
            System.out.printf("%d ", A[i]);
        System.out.printf("\n");
    }

    /* Driver program to test above functions */
    public static void main(String[] args)
    {
        int arr[] = {12, 11, 13, 5, 6, 7};
        int n = arr.length;

        System.out.printf("Given array is \n");
        printArray(arr, n);

        mergeSort(arr, n);

        System.out.printf("\nSorted array is \n");
        printArray(arr, n);
    }
}

// This code is contributed by Smitha
```

Run on IDE

## Python 3

```
# Iterative Merge sort (Bottom Up)

# Iterative mergesort function to
# sort arr[0...n-1]
def mergeSort(a):

    current_size = 1

    # Outer loop for traversing Each
    # sub array of current_size
    while current_size < len(a) - 1:

        left = 0
        # Inner loop for merge call
        # in a sub array
        # Each complete Iteration sorts
        # the iterating sub array
        while left < len(a)-1:

            # mid index = left index of
            # sub array + current sub
            # array size - 1
            mid = left + current_size - 1

            # (False result,True result)
            # [Condition] Can use current_size
            # if 2 * current_size < len(a)-1
            # else len(a)-1
            right = ((2 * current_size + left - 1,
                    len(a) - 1)[2 * current_size
                          + left - 1 > len(a)-1])

            # Merge call for each sub array
            merge(a, left, mid, right)
            left = left + current_size*2

        # Increasing sub array size by
        # multiple of 2
        current_size = 2 * current_size

# Merge Function
def merge(a, l, m, r):
    n1 = m - l + 1
    n2 = r - m
    L = [0] * n1
    R = [0] * n2
    for i in range(0, n1):
        L[i] = a[l + i]
    for i in range(0, n2):
        R[i] = a[m + i + 1]

    i, j, k = 0, 0, l
    while i < n1 and j < n2:
        if L[i] > R[j]:
            a[k] = R[j]
            j += 1
        else:
            a[k] = L[i]
            i += 1
        k += 1

    while i < n1:
        a[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        a[k] = R[j]
        j += 1
        k += 1


# Driver code
a = [12, 11, 13, 5, 6, 7]
print("Given array is ")
print(a)

mergeSort(a)

print("Sorted array is ")
print(a)

# Contributed by Madhur Chhangani [RCOEM]
```

Run on IDE

Output:

```
Given array is
12 11 13 5 6 7

Sorted array is
5 6 7 11 12 13
```

Time complexity of above iterative function is same as recursive, i.e., Θ(nLogn).

**References:**

http://csg.sph.umich.edu/abecasis/class/2006/615.09.pdf

This article is contributed by **Shivam Agrawal**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

**Practice Tags :** Sorting Merge Sort

**Article Tags :** Sorting Merge Sort

Login to Improve this Article

## Recommended Posts:

QuickSort
Merge Sort
Iterative Quick Sort
Merge Sort for Linked Lists
Why Quick Sort preferred for Arrays and Merge Sort for Linked Lists?
Multiset Equivalence Problem
Print all triplets with given sum
Maximise the number of toys that can be purchased with amount K
Minimum swaps so that binary search can be applied
Closest product pair in an array

▲

(Login to Rate)

3 Average Difficulty : **3/5.0**
Based on **23** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!

▲