

Merge Sort

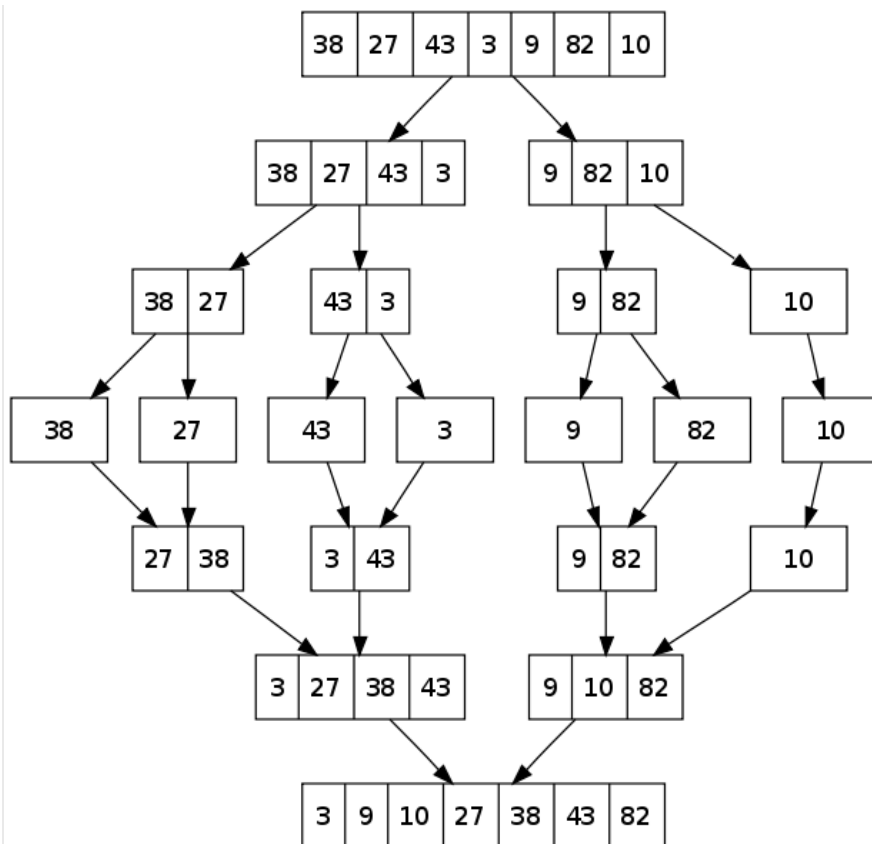
Like **QuickSort**, Merge Sort is a **Divide and Conquer** algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

```
MergeSort(arr[], l, r)
```

```
If r > l
```

1. Find the middle point to divide the array into two halves:
middle m = (l+r)/2
2. Call mergeSort for first half:
Call mergeSort(arr, l, m)
3. Call mergeSort for second half:
Call mergeSort(arr, m+1, r)
4. Merge the two halves sorted in step 2 and 3:
Call merge(arr, l, m, r)

The following diagram from [wikipedia](#) shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.



We strongly recommend that you click [here](#) and practice it, before moving on to the solution.

C/C++

```

/* C program for Merge Sort */
#include<stdlib.h>
#include<stdio.h>

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
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; // Initial index of first subarray
    j = 0; // Initial index of second subarray
    k = l; // Initial index of merged subarray
    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++;
}
}

/* 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)
    {
        // Same as (l+r)/2, but avoids overflow for
        // large l and h
        int m = l+(r-l)/2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);

        merge(arr, l, m, r);
    }
}

/* UTILITY FUNCTIONS */
/* 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

```
/* Java program for Merge Sort */
class MergeSort
{
    // Merges two subarrays of arr[].
    // First subarray is arr[l..m]
    // Second subarray is arr[m+1..r]
    void merge(int arr[], int l, int m, int r)
    {
        // Find sizes of two subarrays to be merged
        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*/
        for (int i=0; i<n1; ++i)
            L[i] = arr[l + i];
        for (int j=0; j<n2; ++j)
            R[j] = arr[m + 1+ j];

        /* Merge the temp arrays */

        // Initial indexes of first and second subarrays
        int i = 0, j = 0;

        // Initial index of merged subarray array
        int 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 remaining elements of L[] if any */
        while (i < n1)
        {
            arr[k] = L[i];
            i++;
            k++;
        }

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

    // Main function that sorts arr[l..r] using
    // merge()
    void sort(int arr[], int l, int r)
    {
        if (l < r)
        {
            // Find the middle point
            int m = (l+r)/2;

```

```

        // Sort first and second halves
        sort(arr, l, m);
        sort(arr , m+1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
    int n = arr.length;
    for (int i=0; i<n; ++i)
        System.out.print(arr[i] + " ");
    System.out.println();
}

// Driver method
public static void main(String args[])
{
    int arr[] = {12, 11, 13, 5, 6, 7};

    System.out.println("Given Array");
    printArray(arr);

    MergeSort ob = new MergeSort();
    ob.sort(arr, 0, arr.length-1);

    System.out.println("\nSorted array");
    printArray(arr);
}
}
/* This code is contributed by Rajat Mishra */

```

Run on IDE

Python

```

# Python program for implementation of MergeSort

# Merges two subarrays of arr[].
# First subarray is arr[l..m]
# Second subarray is arr[m+1..r]
def merge(arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    # create temp arrays
    L = [0] * (n1)
    R = [0] * (n2)

    # Copy data to temp arrays L[] and R[]
    for i in range(0 , n1):
        L[i] = arr[l + i]

    for j in range(0 , n2):
        R[j] = arr[m + 1 + j]

    # Merge the temp arrays back into arr[l..r]
    i = 0      # Initial index of first subarray
    j = 0      # Initial index of second subarray
    k = l      # Initial index of merged subarray

    while i < n1 and j < n2 :
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]

```

```

        j += 1
        k += 1

# Copy the remaining elements of L[], if there
# are any
while i < n1:
    arr[k] = L[i]
    i += 1
    k += 1

# Copy the remaining elements of R[], if there
# are any
while j < n2:
    arr[k] = R[j]
    j += 1
    k += 1

# l is for left index and r is right index of the
# sub-array of arr to be sorted
def mergeSort(arr, l, r):
    if l < r:

        # Same as (l+r)/2, but avoids overflow for
        # large l and h
        m = (l+(r-1))/2

        # Sort first and second halves
        mergeSort(arr, l, m)
        mergeSort(arr, m+1, r)
        merge(arr, l, m, r)

# Driver code to test above
arr = [12, 11, 13, 5, 6, 7]
n = len(arr)
print ("Given array is")
for i in range(n):
    print ("%d" %arr[i]),

mergeSort(arr,0,n-1)
print ("\n\nSorted array is")
for i in range(n):
    print ("%d" %arr[i]),

# This code is contributed by Mohit Kumra

```

[Run on IDE](#)

Output:

```

Given array is
12 11 13 5 6 7

```

```

Sorted array is
5 6 7 11 12 13

```

Time Complexity: Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

$$T(n) = 2T(n/2) + \Theta(n)$$

The above recurrence can be solved either using Recurrence Tree method or Master method. It falls in case II of Master Method and solution of the recurrence is $\Theta(n \log n)$.

Time complexity of Merge Sort is $\Theta(n \log n)$ in all 3 cases (worst, average and best) as merge sort always divides the array in two halves and take linear time to merge two halves.

Auxiliary Space: $O(n)$

Algorithmic Paradigm: Divide and Conquer

Sorting In Place: No in a typical implementation

Stable: Yes

Applications of Merge Sort

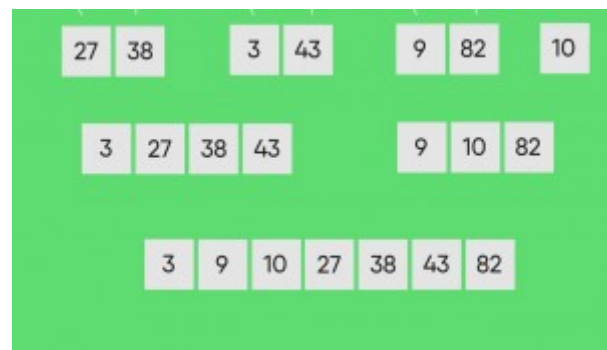
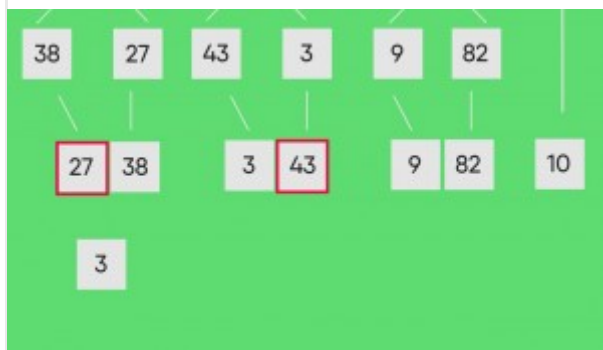
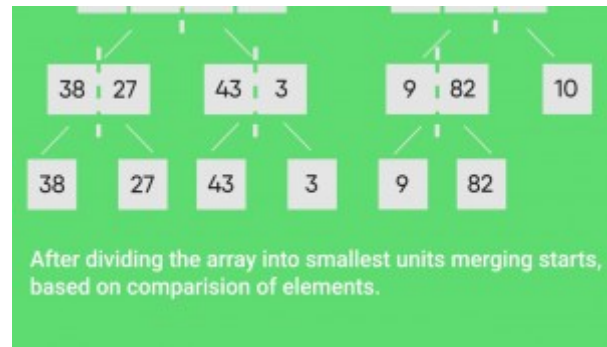
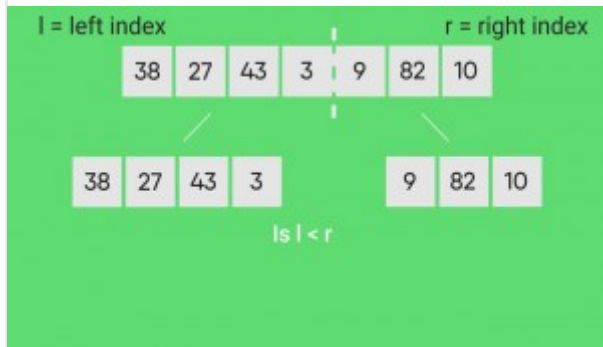
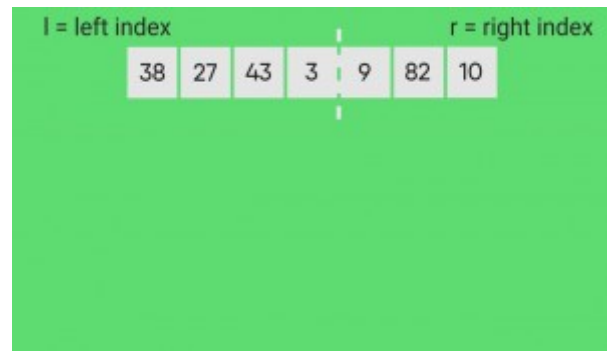
1. **Merge Sort is useful for sorting linked lists in $O(n \log n)$ time.** In case of linked lists the case is different mainly due to difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike array, in linked list, we can insert items in the middle in $O(1)$ extra space and $O(1)$ time. Therefore merge operation of merge sort can be implemented without extra space for linked lists.

In arrays, we can do random access as elements are continuous in memory. Let us say we have an integer (4-byte) array A and let the address of A[0] be x then to access A[i], we can directly access the memory at $(x + i * 4)$. Unlike arrays, we can not do random access in linked list. Quick Sort requires a lot of this kind of access. In linked list to access i'th index, we have to travel each and every node from the head to i'th node as we don't have continuous block of memory. Therefore, the overhead increases for quick sort. Merge sort accesses data sequentially and the need of random access is low.

2. **Inversion Count Problem**
3. Used in **External Sorting**

Merge Sort | GeeksforGeeks



Snapshots:**Quiz on Merge Sort****Other Sorting Algorithms on GeeksforGeeks/GeeksQuiz:**

3-way Merge Sort, Selection Sort, [Bubble Sort](#), [Insertion Sort](#), [Merge Sort](#), [Heap Sort](#), [QuickSort](#), Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort

Coding practice for sorting.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

GATE CS Corner Company Wise Coding Practice

Divide and Conquer Sorting

Recommended Posts:

Heap Sort

QuickSort

Count Inversions in an array I Set 1 (Using Merge Sort)

Bubble Sort

Insertion Sort

(Login to Rate and Mark)

3

Average Difficulty : **3/5.0**
Based on 1 vote(s)

Add to TODO List

Mark as DONE

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

Share this post!

1 Comment

GeeksforGeeks

 Aymeric Flaisler ▾

 Recommend

 Share

Sort by Newest ▾



Join the discussion...



Ashish Sharma · 18 hours ago

"Auxiliary Space: $O(n)$




Algorithmic Paradigm: Divide and Conquer

Sorting In Place: No in a typical implementation

Stable: Yes"

I like way you use short points like this. Please use these type of short points in all your videos.

^ | ▾ · Reply · Share ›

 Subscribe  Add Disqus to your siteAdd DisqusAdd  Privacy

@geeksforgeeks, Some rights reserved

Contact Us!

About Us!

Advertise with us!

[Privacy Policy](#)

