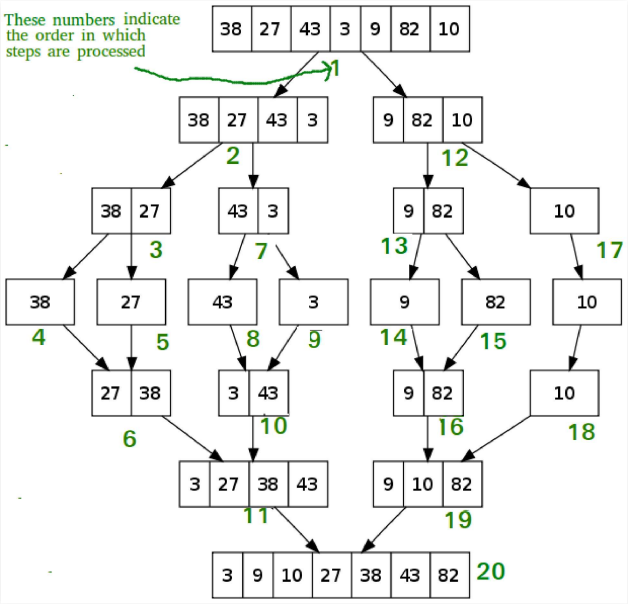
Merge Sort

Difficulty Level : Medium • Last Updated : 18 May, 2021

Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides the input array into 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 a 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 the 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-l)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, 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 into two halves till the size becomes 1. Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged.



Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

C++

```
// C++ program for Merge Sort
```

```
// 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 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 (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 back into arr[l..r]

    // Initial index of first subarray
    int i = 0;

    // Initial index of second subarray
    int j = 0;

    // Initial index of merged subarray
    int k = l;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
```

```
            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){
        return;//returns recursively
    }
    int m =l+ (r-l)/2;
    mergeSort(arr,l,m);
```

```
// UTILITY FUNCTIONS
// Function to print an array
void printArray(int A[], int size)
{
    for (int i = 0; i < size; i++)
        cout << A[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 12, 11, 13, 5, 6, 7 };
    int arr_size = sizeof(arr) / sizeof(arr[0]);

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

    mergeSort(arr, 0, arr_size - 1);

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

// This code is contributed by Mayank Tyagi
```

C

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

```
// 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++;
    }
```

```
        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);
        }
    }
```

```
{
    int i;
    for (i = 0; i < size; i++)
        printf("%d ", A[i]);
    printf("\n");
}

/* Driver code */
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;
}
```

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]
```

```
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 subarry 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++;
}
```

```
            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-l)/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[])
    {
```

```
        System.out.println();
    }

    // Driver code
    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 */
```

Python3

```
# Python program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) > 1:

         # Finding the mid of the array
        mid = len(arr)//2

        # Dividing the array elements
        L = arr[:mid]
```

```
        # Sorting the first half
        mergeSort(L)

        # Sorting the second half
        mergeSort(R)

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

# Code to print the list
```

```
        print(arr[i], end=" ")
    print()


# Driver Code
if __name__ == '__main__':
    arr = [12, 11, 13, 5, 6, 7]
    print("Given array is", end="\n")
    printList(arr)
    mergeSort(arr)
    print("Sorted array is: ", end="\n")
    printList(arr)

# This code is contributed by Mayank Khanna
```

C#

```
// C# program for Merge Sort
using System;
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;
```

```
    int i, j;

    // Copy data to temp arrays
    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

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

    // Initial index of merged
    // subarry 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
```

```
            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-l)/2;

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

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

```
    {
        int n = arr.Length;
        for (int i = 0; i < n; ++i)
            Console.Write(arr[i] + " ");
        Console.WriteLine();
    }

    // Driver code
    public static void Main(String[] args)
    {
        int[] arr = { 12, 11, 13, 5, 6, 7 };
        Console.WriteLine("Given Array");
        printArray(arr);
        MergeSort ob = new MergeSort();
        ob.sort(arr, 0, arr.Length - 1);
        Console.WriteLine("\nSorted array");
        printArray(arr);
    }
}

// This code is contributed by Princi Singh
```

Related Articles Save for later

```
// JavaScript program for Merge Sort

// Merges two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
```

```
var n2 = r - m;

// Create temp arrays
var L = new Array(n1);
var R = new Array(n2);

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

// Merge the temp arrays back into arr[l..r]

// Initial index of first subarray
var i = 0;

// Initial index of second subarray
var j = 0;

// Initial index of merged subarray
var k = l;

while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    }
    else {
        arr[k] = R[j];
        j++;
    }
```

```
        // 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 */
    function mergeSort(arr,l, r){
        if(l>=r){
            return;//returns recursively
        }
        var m =l+ parseInt((r-l)/2);
        mergeSort(arr,l,m);
        mergeSort(arr,m+1,r);
        merge(arr,l,m,r);
    }

    // UTILITY FUNCTIONS
    // Function to print an array
```

```
        document.write(  A[i] + " ");
}


var arr = [ 12, 11, 13, 5, 6, 7 ];
    var arr_size = arr.length;

    document.write(  "Given array is <br>");
    printArray(arr, arr_size);

    mergeSort(arr, 0, arr_size - 1);

    document.write( "<br>Sorted array is <br>");
    printArray(arr, arr_size);

// This code is contributed by SoumikMondal

</script>
```

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 the Recurrence Tree method or the Master method. It falls in case II of Master Method and the solution of the recurrence is $\theta(nLogn)$. Time complexity of Merge Sort is $\theta(nLogn)$ in all 3 cases (worst, average and best) as merge sort always divides the array

Algorithmic Paradigm: Divide and Conquer

Sorting In Place: No in a typical implementation

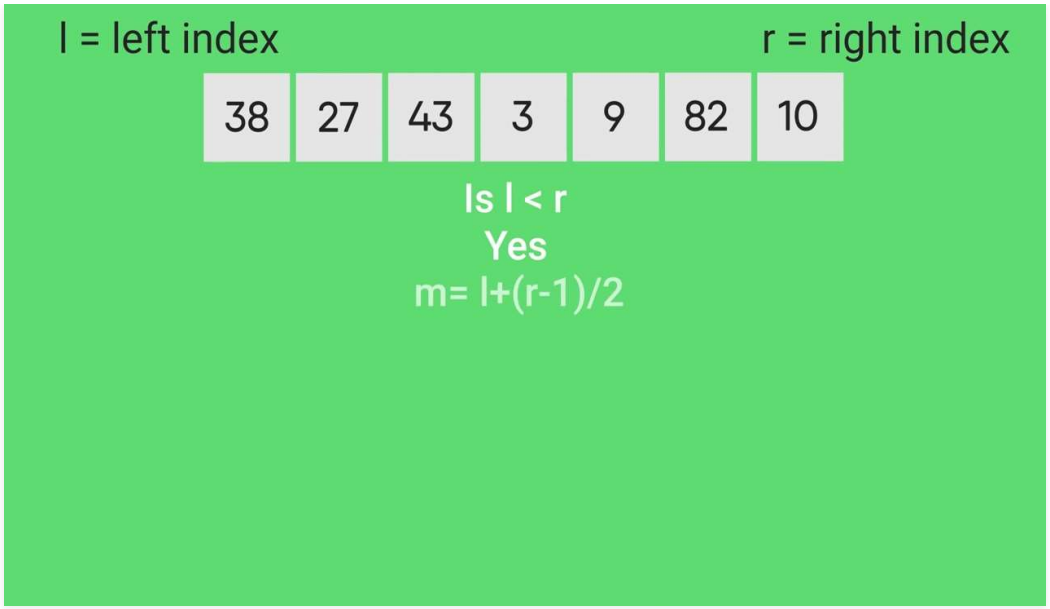
Stable: Yes

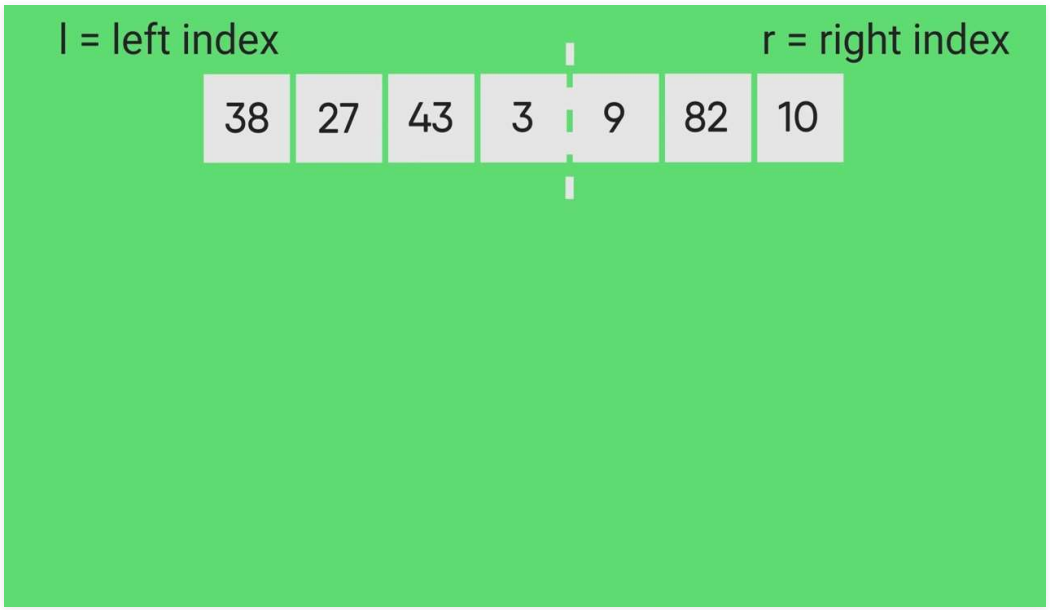
Applications of Merge Sort

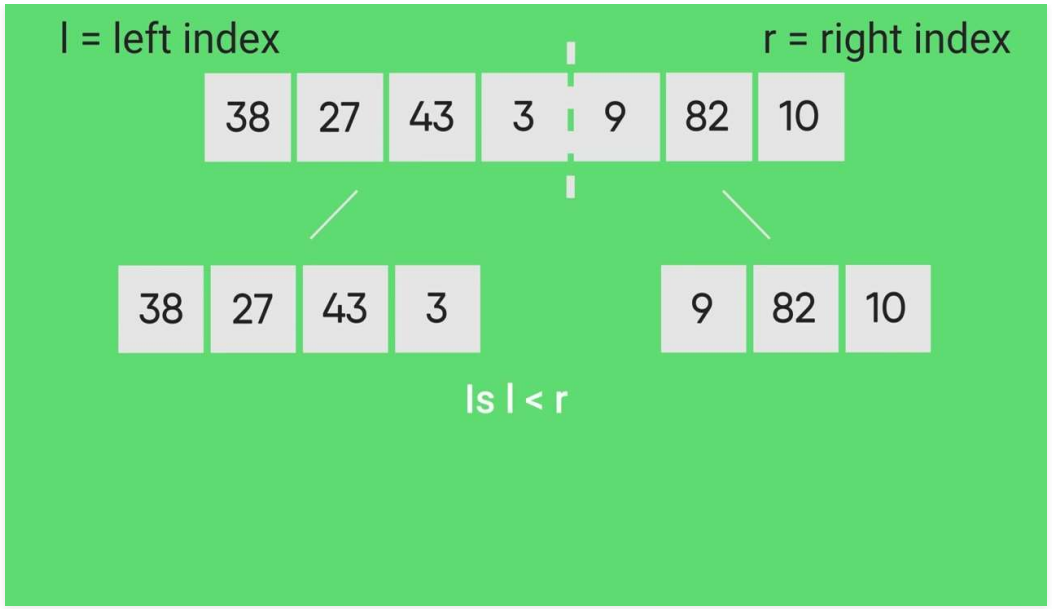
1. Merge Sort is useful for sorting linked lists in O(nLogn) time. In the case of linked lists, the case is different mainly due to the difference in memory allocation of arrays and linked lists. Unlike arrays, linked list nodes may not be adjacent in memory. Unlike an array, in the linked list, we can insert items in the middle in O(1) extra space and O(1) time. Therefore, the merge operation of merge sort can be implemented without extra space for linked lists.
   In arrays, we can do random access as elements are contiguous 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 the linked list. Quick Sort requires a lot of this kind of access. In a 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 a continuous block of memory. Therefore, the overhead increases for quicksort. Merge sort accesses data sequentially and the need of random access is low.
2. Inversion Count Problem
3. Used in External Sorting

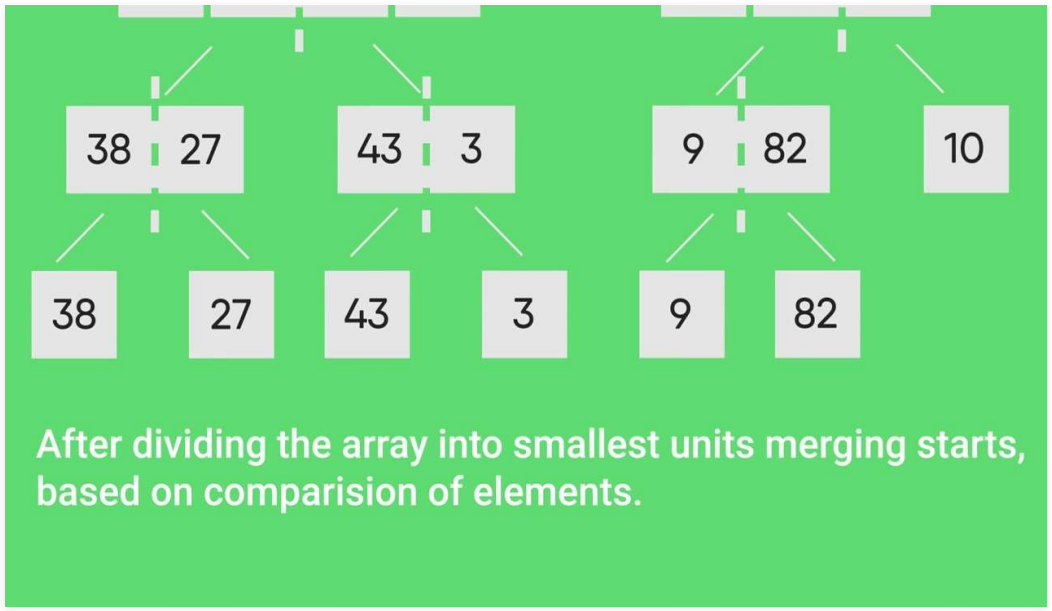
Drawbacks of Merge Sort

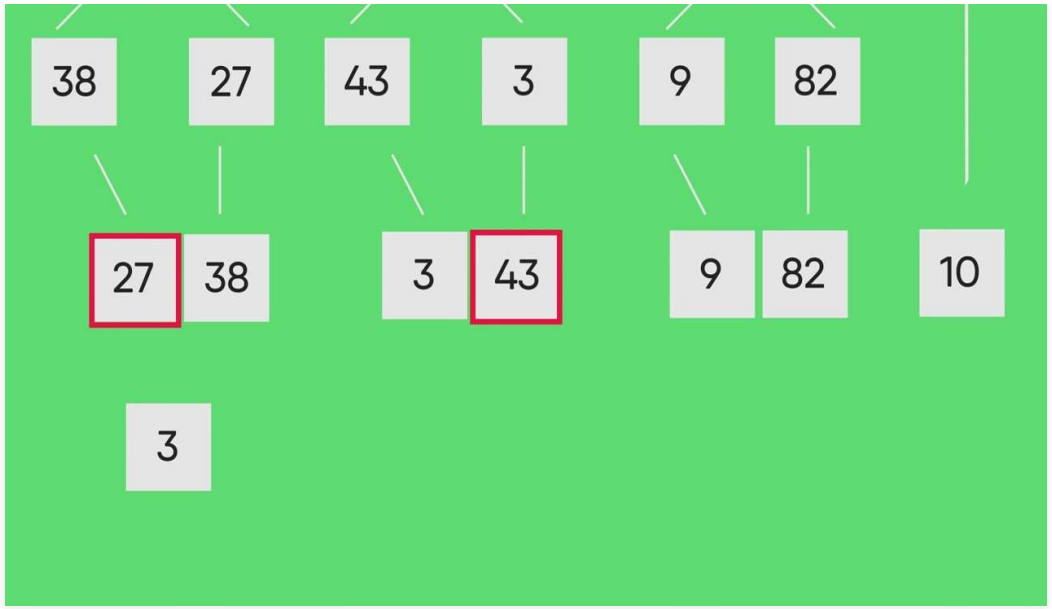
- Slower comparative to the other sort algorithms for smaller tasks.
- Merge sort algorithm requires an additional memory space of O(n) for the temporary array.

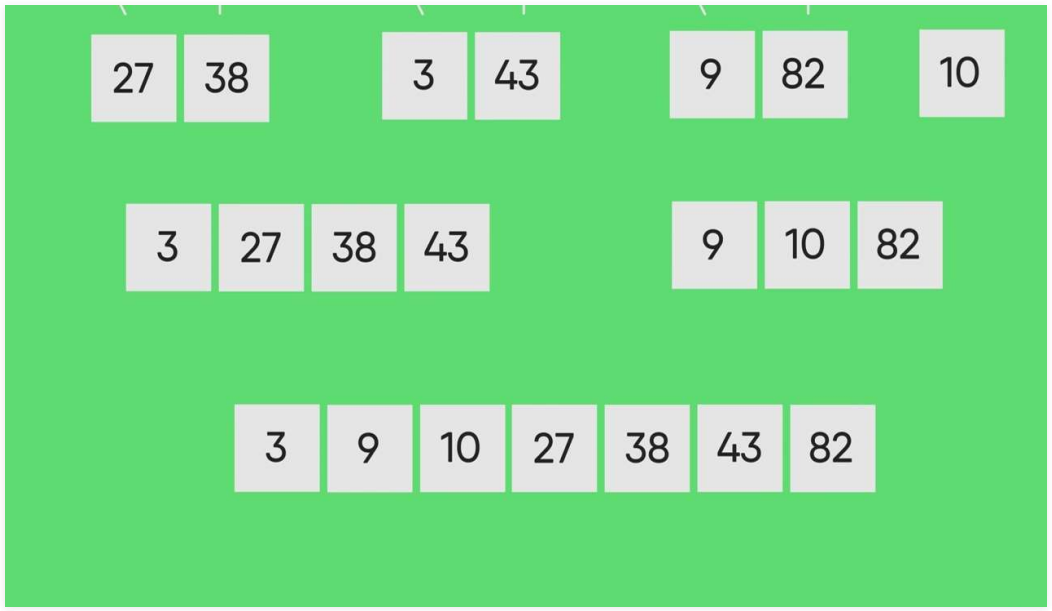












- Recent Articles on Merge Sort

- Quiz on Merge Sort

Other Sorting Algorithms on GeeksforGeeks:

3-way Merge Sort, Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort

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

Next

QuickSort

RECOMMENDED ARTICLES

Page : 1 2 3

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : ukasp, Mayank Khanna 2, pineconelam, jnjomnsn, mayanktyagi1709, princi singh, naveenkuma150, vishalg2, akkkkk, sidhijain, SoumikMondal

Article Tags : Amazon, Boomerang Commerce, Goldman Sachs, Grofers, Microsoft, Oracle, Paytm, Qualcomm, Snapdeal, Target Corporation, Divide and Conquer, Sorting

Practice Tags : Paytm, Amazon, Microsoft, Snapdeal, Oracle, Goldman Sachs, Qualcomm, Boomerang Commerce, Grofers, Target Corporation, Divide and Conquer, Sorting

Improve Article | Report Issue

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

Load Comments

GeeksforGeeks
5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Company-wise
Topic-wise
How to begin?
Write Interview Experience
Internships
Videos