**Day 31**

**Merge Sort Algorithm**

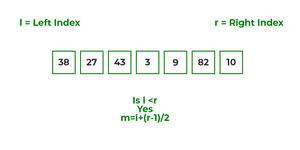
**Merge Sort Working Process:**

Think of it as a recursive algorithm continuously splits the array in half until it cannot be further divided. This means that if the array becomes empty or has only one element left, the dividing will stop, i.e. it is the base case to stop the recursion. If the array has multiple elements, split the array into halves and recursively invoke the merge sort on each of the halves. Finally, when both halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.

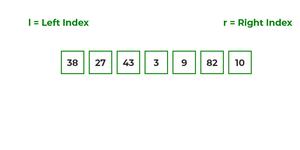
**Illustration:**

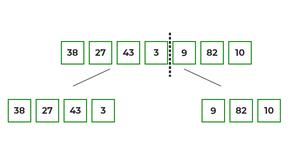
To know the functioning of merge sort, lets consider an array arr[] = {38, 27, 43, 3, 9, 82, 10}

* *At first, check if the left index of array is less than the right index, if yes then calculate its mid point*

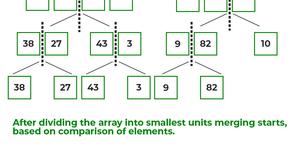
**

* *Now, as we already know that merge sort first divides the whole array iteratively into equal halves, unless the atomic values are achieved.*
* *Here, we see that an array of 7 items is divided into two arrays of size 4 and 3 respectively.*

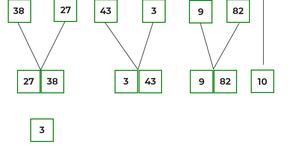
**

* *Now, again find that is left index is less than the right index for both arrays, if found yes, then again calculate mid points for both the arrays.*

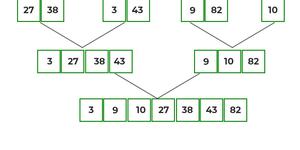
* *Now, further divide these two arrays into further halves, until the atomic units of the array is reached and further division is not possible.*

**

* *After dividing the array into smallest units, start merging the elements again based on comparison of size of elements*
* *Firstly, compare the element for each list and then combine them into another list in a sorted manner.*

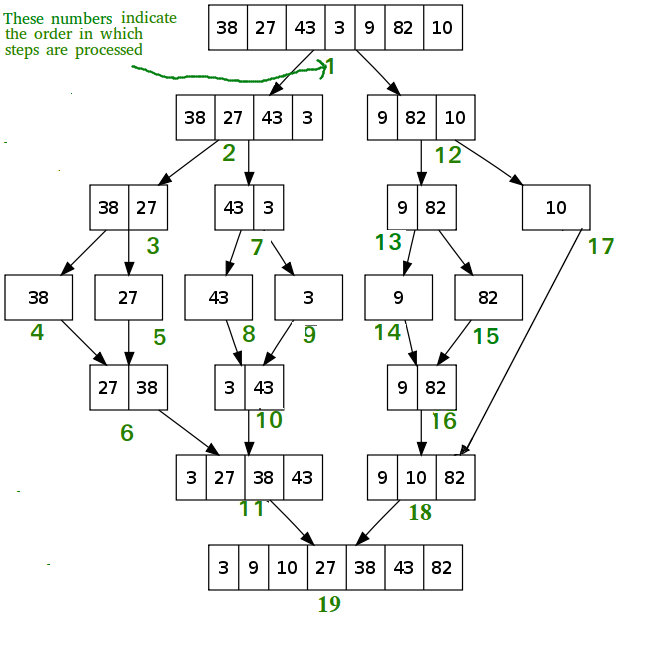
**

* *After the final merging, the list looks like this:*

**

The following diagram 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.



*Recursive steps of merge sort*

Recommended Problem

Merge Sort

[Divide and Conquer](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Divide%20and%20Conquer&sortBy=submissions)

[Sorting](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Sorting&sortBy=submissions)

+1 more

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[Boomerang Commerce](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Boomerang%20Commerce&sortBy=submissions)

+10 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/merge-sort/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article" \o "Permalink to Merge Sort)

Submission count: 1L

**Algorithm:**

*step 1: start*

*step 2: declare array and left, right, mid variable*

*step 3: perform merge function.  
    if left > right  
        return  
    mid= (left+right)/2  
    mergesort(array, left, mid)  
    mergesort(array, mid+1, right)  
    merge(array, left, mid, right)*

*step 4: Stop*

Follow the steps below the solve the problem:

MergeSort(arr[], l,  r)  
If r > l

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