**Merge Sort Algorithm**

The Merge Sort algorithm is a sorting algorithm that is based on the Divide and Conquer paradigm. In this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner.

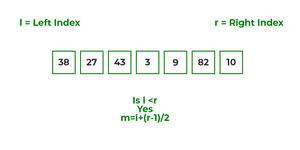
**Merge Sort Working Process**:

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:

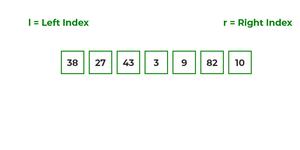
To know the functioning of merge sort, lets consider an array arr[] = {38, 27, 43, 3, 9, 82, 10}

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

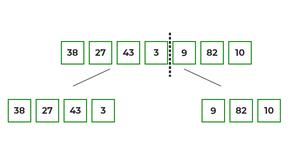


Now, as we already know that merge sort first divides the whole array iteratively into equal halves, unless the atomic values are achieved.

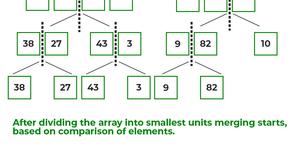
Here, we see that an array of 7 items is divided into two arrays of size 4 and 3 respectively.



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.

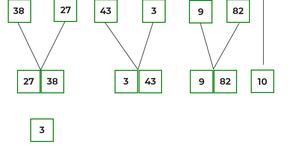


Now, further divide these two arrays into further halves, until the atomic units of the array is reached and further division is not possible.

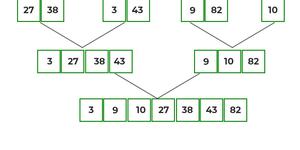


After dividing the array into smallest units, start merging the elements again based on comparison of size of elements

Firstly, compare the element for each list and then combine them into another list in a sorted manner.

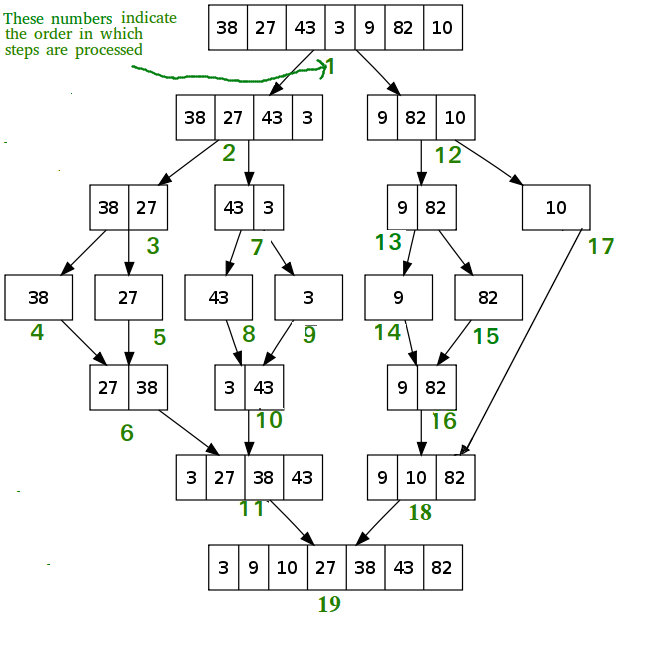


After the final merging, the list looks like this:



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

**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

**Pseudo code:**

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)

**Time Complexity**: O(N log(N)),  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) + θ(n)***

The above recurrence can be solved either using the Recurrence Tree method or the Master method. It falls in case II of the Master Method and the solution of the recurrence is θ(Nlog(N)). The time complexity of Merge Sort isθ(Nlog(N)) in all 3 cases (worst, average, and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.

**Auxiliary Space**: O(n), In merge sort all elements are copied into an auxiliary array. So N auxiliary space is required for merge sort.

**Is Merge sort In Place?**

No, In merge sort the merging step requires extra space to store the elements.

**Is Merge sort Stable?**

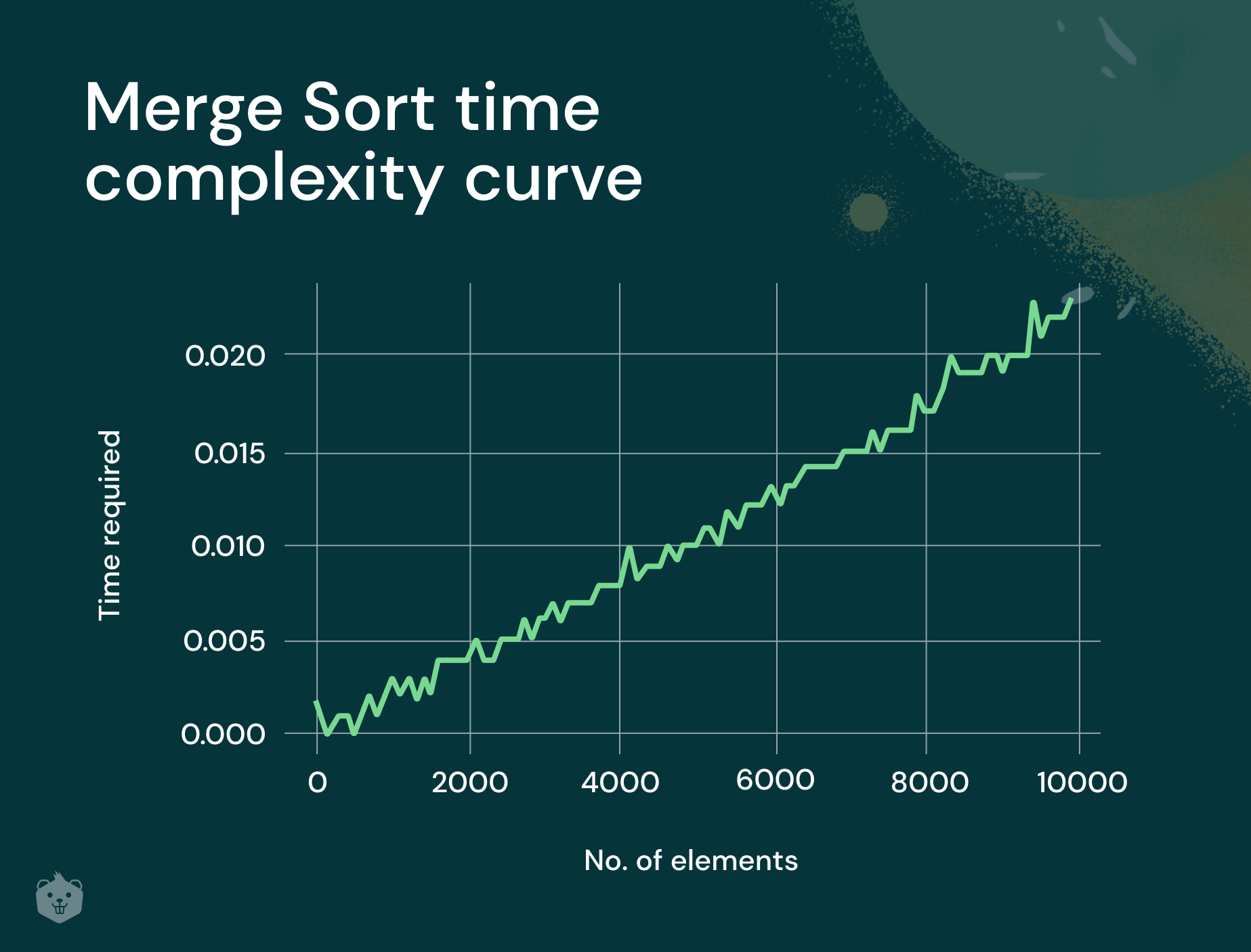
Yes, merge sort is stable.

**How can we make Merge sort more efficient?**

Merge sort can be made more efficient by replacing recursive calls with Insertion sort for smaller array sizes, where the size of the remaining array is less or equal to 43 as the number of operations required to sort an array of max size 43 will be less in Insertion sort as compared to the number of operations required in Merge sort.

**Analysis of Merge Sort:**

A merge sort consists of several passes over the input. The first pass merges segments of size 1, the second merges segments of size 2, and the Rendered by QuickLaTeX.com pass merges segments of size 2i-1. Thus, the total number of passes is [log2n]. As merge showed, we can merge two sorted segments in linear time, which means that each pass takes O(n) time. Since there are [log2n] passes, the total computing time is O(nlogn).



**Applications of Merge Sort:**

1. [Merge Sort is useful for sorting linked lists in O(N log N) time](https://www.geeksforgeeks.org/merge-sort-for-linked-list/). 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 contiguous 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](https://www.geeksforgeeks.org/counting-inversions/)
3. Used in [External Sorting](http://en.wikipedia.org/wiki/External_sorting" \t "https://www.geeksforgeeks.org/merge-sort/_blank)

**Drawbacks of Merge Sort:**

1. Slower compared to the other sort algorithms for smaller tasks.
2. The merge sort algorithm requires an additional memory space of 0(n) for the temporary array.
3. It goes through the whole process even if the array is sorted.
4. [Recent Articles on Merge Sort](https://www.geeksforgeeks.org/tag/merge-sort/)
5. [Coding practice for sorting.](https://practice.geeksforgeeks.org/tag-page.php?tag=sorting&isCmp=0)
6. [Quiz on Merge Sort](https://www.geeksforgeeks.org/quiz-mergesort-gq/)
7. Solution of the drawback for additional storage:
8. Use linked list.

**Merge Sort using Multi-threading**

Merge Sort is a popular sorting technique which divides an array or list into two halves and then start merging them when sufficient depth is reached. Threads are lightweight processes and threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.  
Multi-threading is way to improve parallelism by running the threads simultaneously in different cores of your processor. In this program, we’ll use 4 threads but you may change it according to the number of cores your processor has.  
Examples: 

Input : 83, 86, 77, 15, 93, 35, 86, 92, 49, 21,

62, 27, 90, 59, 63, 26, 40, 26, 72, 36

Output : 15, 21, 26, 26, 27, 35, 36, 40, 49, 59,

62, 63, 72, 77, 83, 86, 86, 90, 92, 93

Input : 6, 5, 4, 3, 2, 1

Output : 1, 2, 3, 4, 5, 6

**Time complexity** of merge sort is O(nlogn).