**Merge Sort**

***Merge sort****is defined as a*[*sorting algorithm*](https://www.geeksforgeeks.org/sorting-algorithms/)*that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.*

***Advantages of Merge Sort:***

* ***Stability****: Merge sort is a stable sorting algorithm, which means it maintains the relative order of equal elements in the input array.*
* ***Guaranteed worst-case performance:****Merge sort has a worst-case time complexity of O(N logN), which means it performs well even on large datasets.*
* ***Parallelizable****: Merge sort is a naturally parallelizable algorithm, which means it can be easily parallelized to take advantage of multiple processors or threads.*

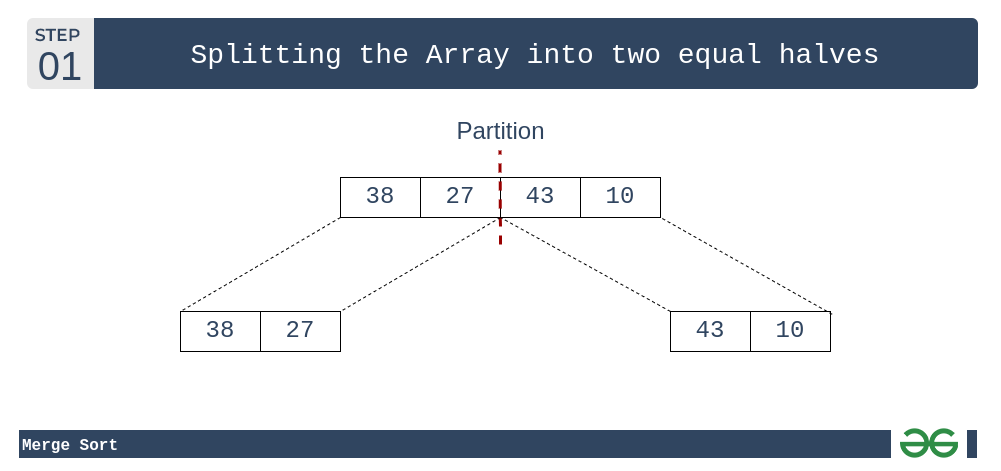
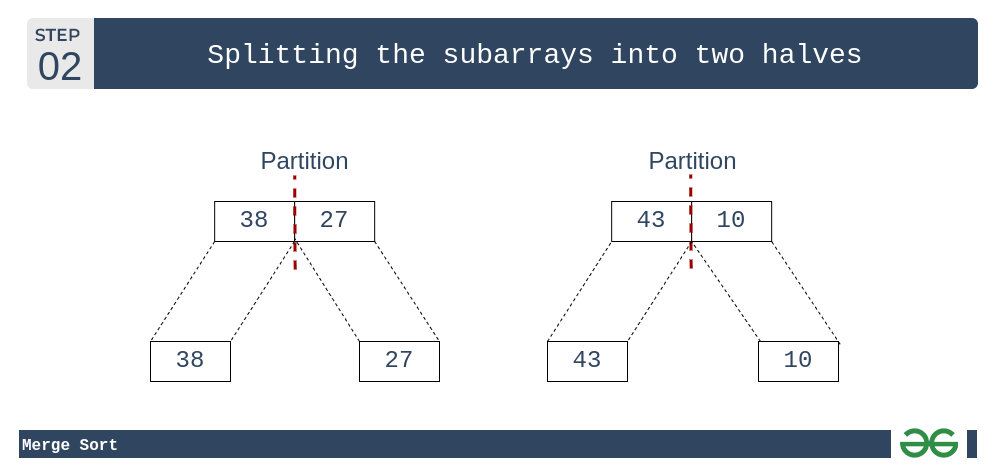
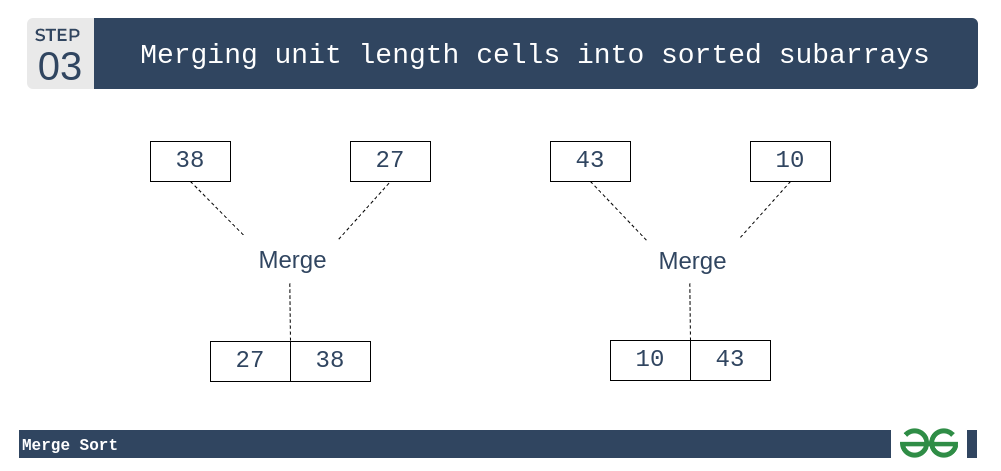
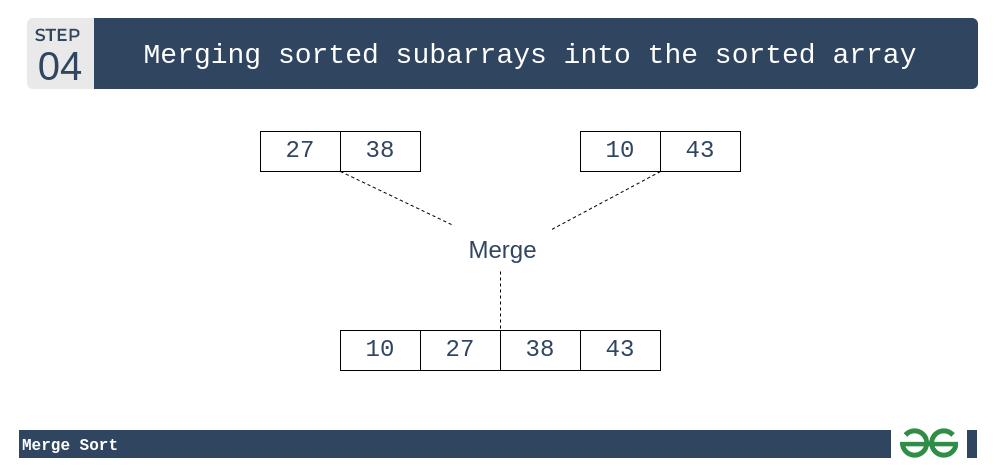
***Drawbacks of Merge Sort:***

* ***Space complexity:****Merge sort requires additional memory to store the merged sub-arrays during the sorting process.*
* ***Not in-place:****Merge sort is not an in-place sorting algorithm, which means it requires additional memory to store the sorted data. This can be a disadvantage in applications where memory usage is a concern.*
* ***Not always optimal for small datasets:****For small datasets, Merge sort has a higher time complexity than some other sorting algorithms, such as insertion sort. This can result in slower performance for very small datasets.*

**How does Merge Sort work?**

*Merge sort is a recursive algorithm that continuously splits the array in half until it cannot be further divided i.e., the array has only one element left (an array with one element is always sorted). Then the sorted subarrays are merged into one sorted array.*

*Lets consider an array****arr[] = {38, 27, 43, 10}***

* *Initially divide the array into two equal halves:*
* *These subarrays are further divided into two halves. Now they become array of unit length that can no longer be divided and array of unit length are always sorted.*
* *These sorted subarrays are merged together, and we get bigger sorted subarrays.*
* *This merging process is continued until the sorted array is built from the smaller subarrays.*

**Applications of Merge Sort:**

* **Sorting large datasets:** Merge sort is particularly well-suited for sorting large datasets due to its guaranteed worst-case time complexity of O(n log n).
* **External sorting:** Merge sort is commonly used in external sorting, where the data to be sorted is too large to fit into memory.
* **Custom sorting:**Merge sort can be adapted to handle different input distributions, such as partially sorted, nearly sorted, or completely unsorted data.

Done By: Anas Abdullah.

Teacher: Zaid Mansor.

**Source Code.**

**# 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]

# Into 2 halves

R = 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**

def printList(arr):

for i in range(len(arr)):

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

print()

**# Driver Code**

if \_\_name\_\_ == '\_\_main\_\_':

arr = [12, 11, 13, 5, 6, 7]

print("Given array is")

printList(arr)

mergeSort(arr)

print("\nSorted array is ")

printList(arr)

**Output**

Given array is

12 11 13 5 6 7

Sorted array is

5 6 7 11 12 13