Fatima Khan

Did you know?

Merge Sort follows the Divide and Conquer approach. It works by recursively dividing the input array into two halves, recursively sorting the two halves and finally merging them back together to obtain the sorted array.

Explanation

Step by step explanation

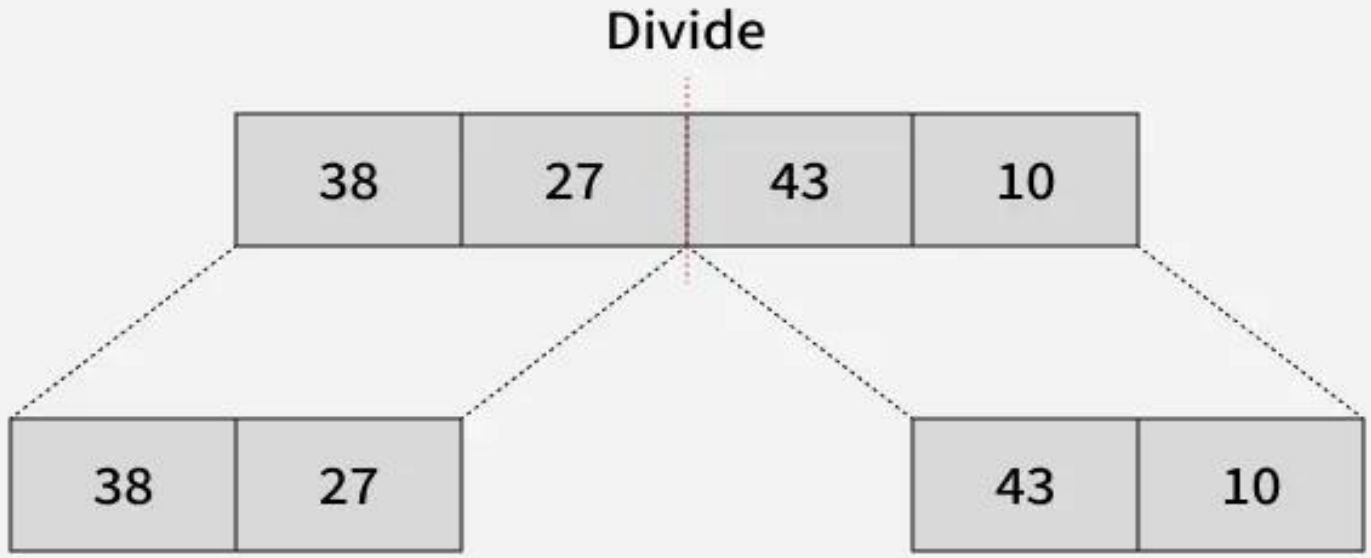
Divide: Divide the list or array recursively into two halves until it can no more be divided.

Conquer: Each subarray is sorted individually using the merge sort algorithm.

Merge: The sorted subarrays are merged back together in sorted order. The process continues until all elements from both subarrays have been merged.

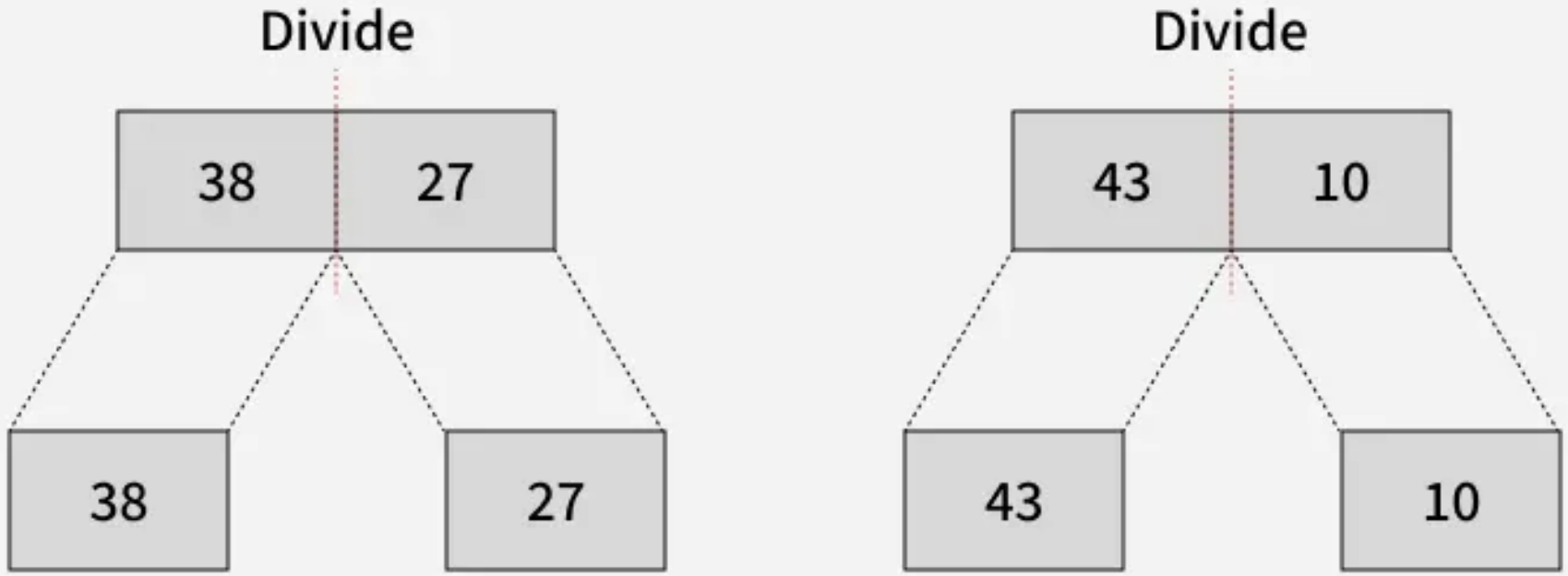
Step 1

01 Step | Splitting the Array into two equal halves



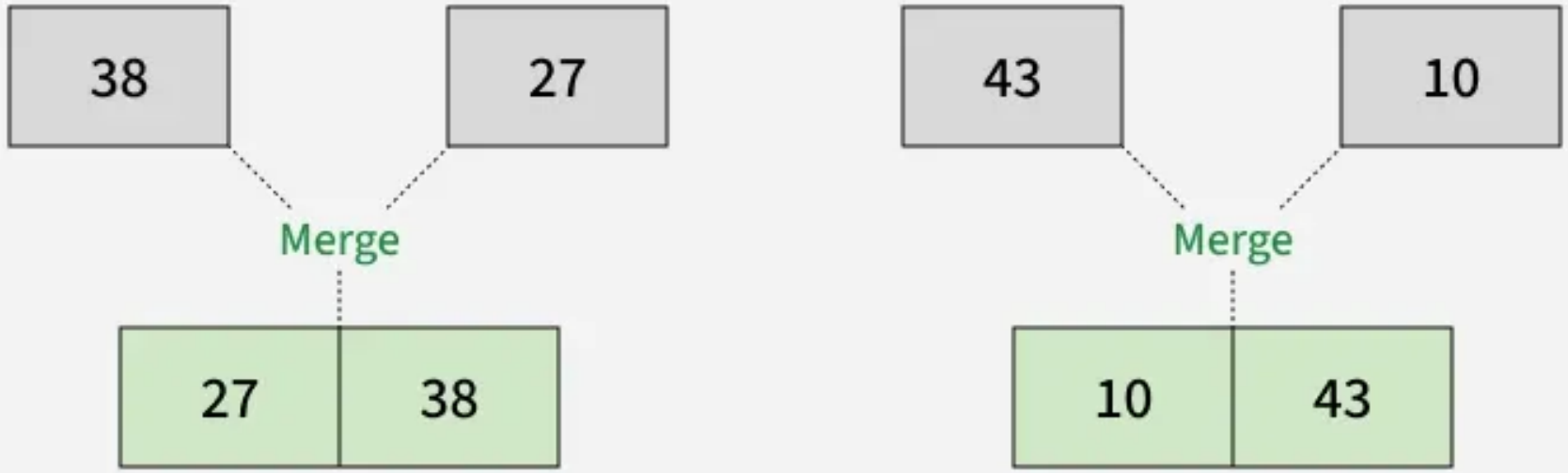
Step 2

02 Step | Splitting the subarrays into two halves



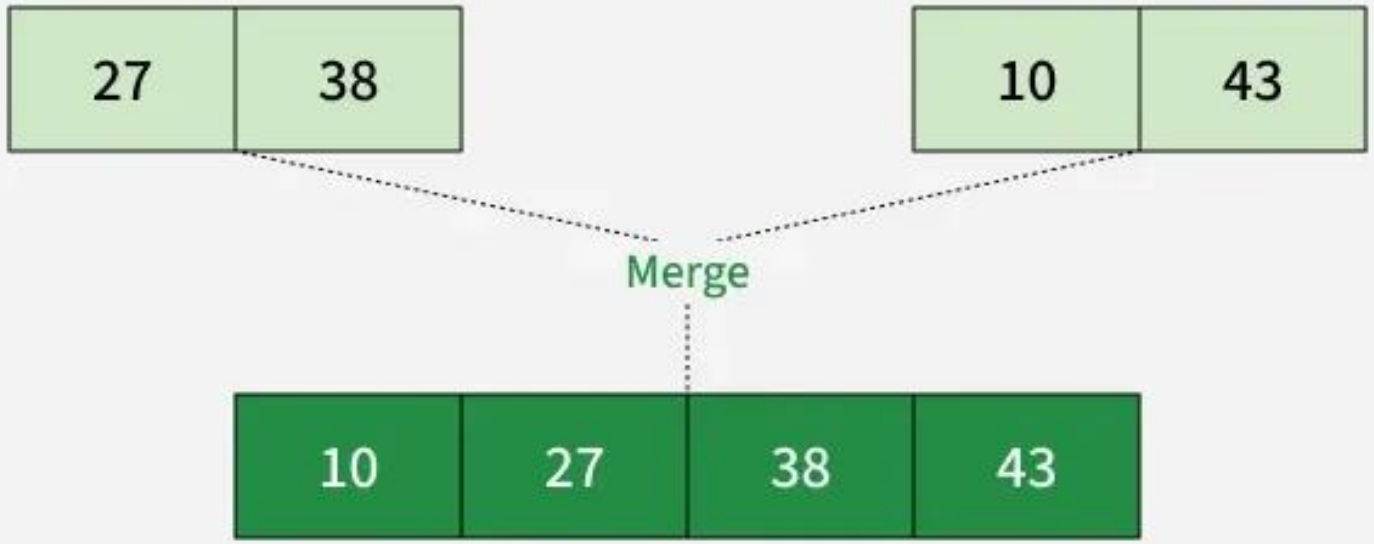
Step 3

03 Step | Merging unit length cells into sorted subarrays



Step 4

04 Step | Merging sorted subarrays into the sorted array



Advantages of Merge Sort

Stable Sorting Algorithm

Simple To Implement

Performs well on large datasets

Disadvantages of Merge Sort

Not an in-place Algorithm

Space Complexity

Slower than Quicksort


Kirby™
Quick
Sort

Did you know?

QuickSort is a sorting algorithm based on the Divide and Conquer that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array.
It works on the principle of divide and conquer, breaking down the problem into smaller sub-problems.

Explanation

Step by step explanation

Choose a Pivot: Select an element from the array as the pivot. The choice of pivot can vary (e.g., first element, last element, random element, or median).

Partition the Array: Re arrange the array around the pivot. After partitioning, all elements smaller than the pivot will be on its left, and all elements greater than the pivot will be on its right. The pivot is then in its correct position, and we obtain the index of the pivot.

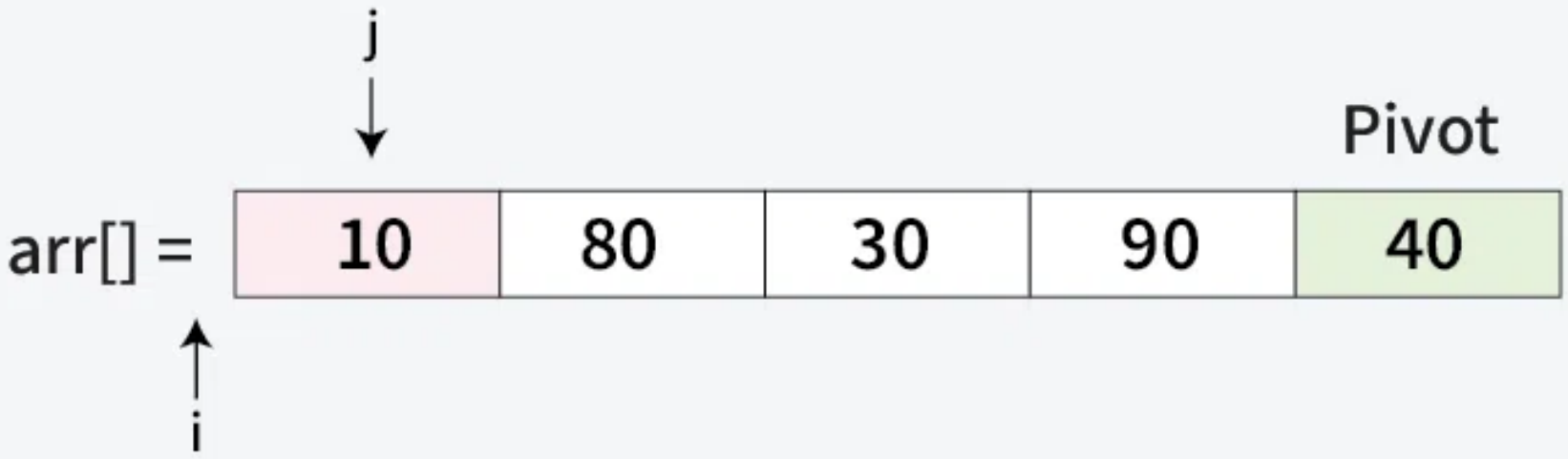
Recursively Call: Recursively apply the same process to the two partitioned sub-arrays (left and right of the pivot).

Base Case: The recursion stops when there is only one element left in the sub-array, as a single element is already sorted.

Step 1

01 Step

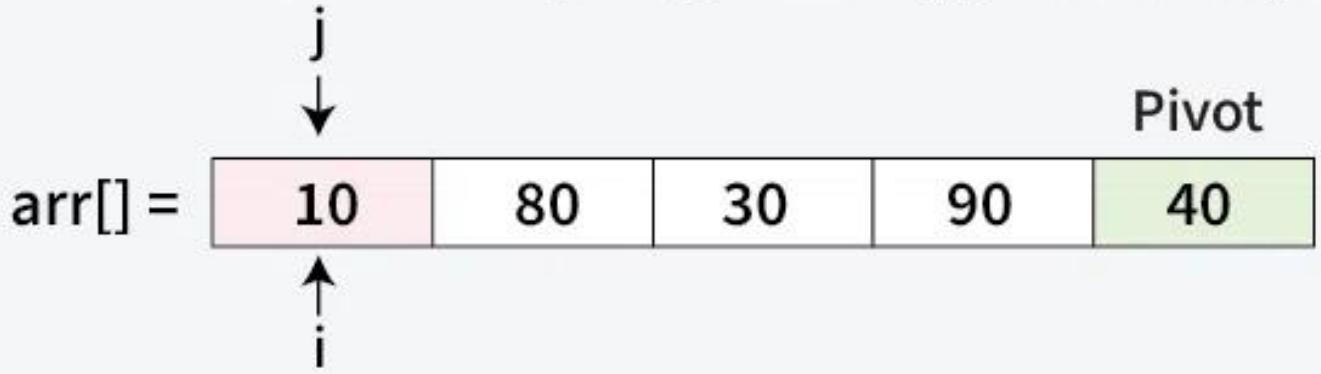
Pivot Selection: The last element arr[4] = 40 is chosen as the pivot.
Initial Pointers: i = -1 and j = 0.



Step 2

02 Step

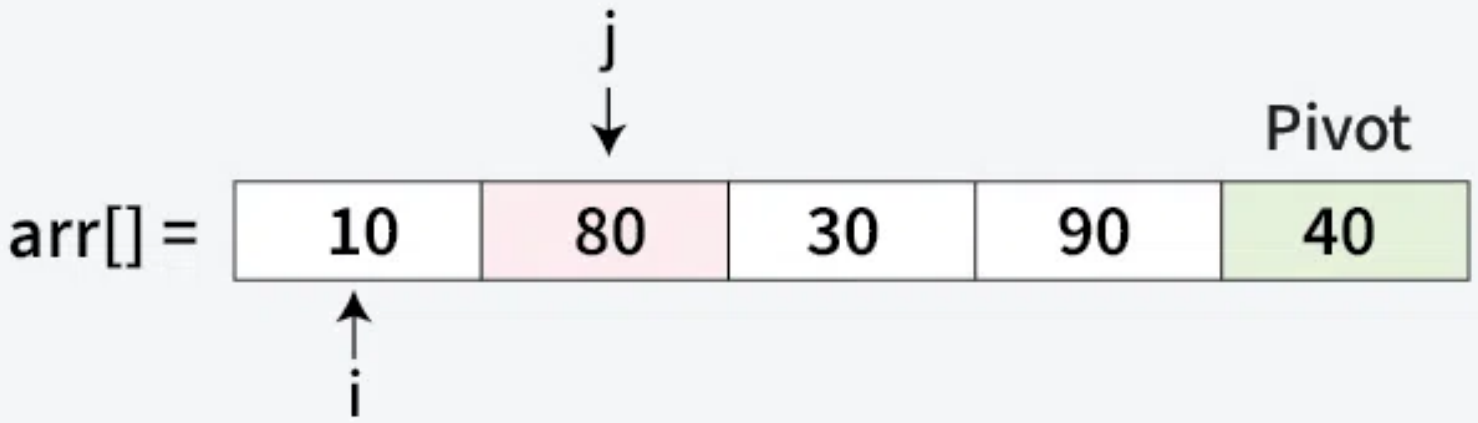
Since, arr[j] < pivot (10<40)
Increment i to 0 and swap arr[i] with arr[j]. Increment j by 1



Step 3

03 Step

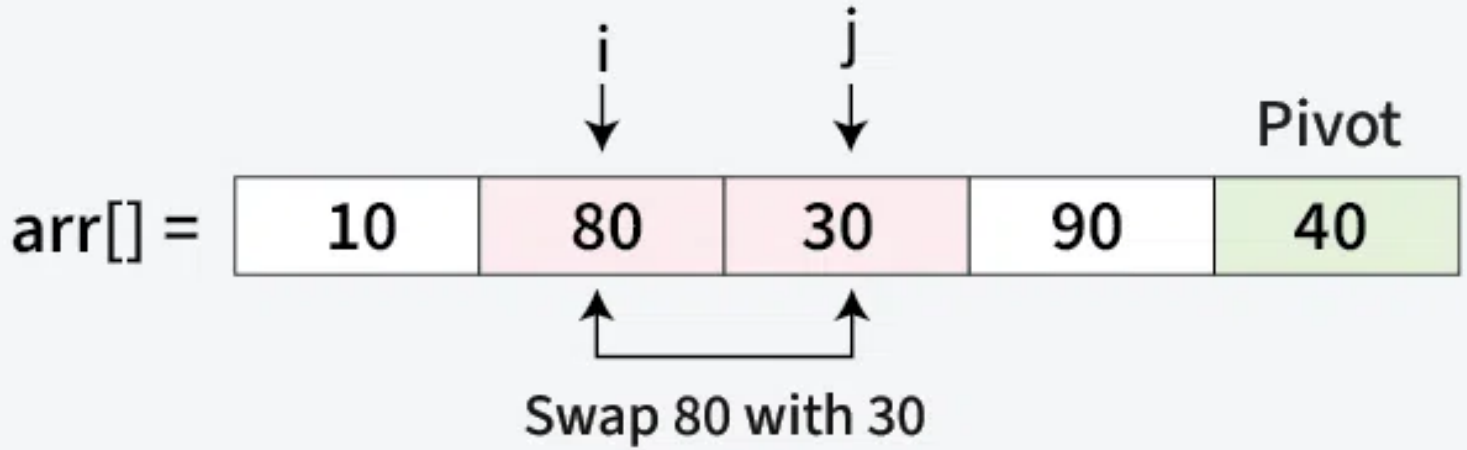
Since, arr[j] > pivot (80>40)
No swap needed. Increment j by 1



Step 4

04 Step

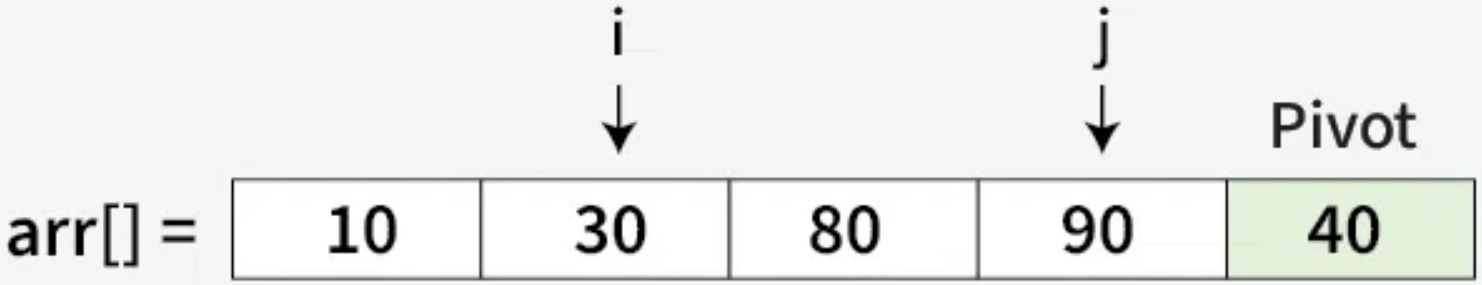
Since, arr[j] < pivot (30<40)
Increment i by 1 and swap arr[i] with arr[j]. Increment j by 1



Step 5

05 Step

Since, arr[j] > pivot (90>40)
No swap needed. Increment j by 1



Advantages of Quick Sort

Efficient on large dataset

Cache friendly

Doesn’t require a lot of memory

Disadvantages of Quick Sort

not a good choice for small data sets.

O(N^2) Complexity

Not a stable sort

Thank you!