**What is Sorting?**

**56 89 67 34 21 92**

**56 67 89 34 21 92**

**56 67 34 89 21 92**

**56 34 67 89 21 92**

**34 56 67 89 21 92**

**34 56 67 21 89 92**

**34 56 21 67 89 92**

**34 21 56 67 89 92**

**21 34 56 67 89 92**

**21 34 56 67 89 92**

**A-> Z**

A Sorting Algorithm is used to rearrange a given array or list of elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of elements in the respective data structure.

**Merge Sort Algorithm**

Merge sort is defined as a sorting algorithm 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.

In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

**Need for Merge Sort**

One thing that you might wonder is what is the specialty of this algorithm. We already have a number of sorting algorithms then why do we need this algorithm? One of the main advantages of merge sort is that it has a time complexity of O(n log n), which means it can sort large arrays relatively quickly. It is also a stable sort, which means that the order of elements with equal values is preserved during the sort.

Merge sort is a popular choice for sorting large datasets because it is relatively efficient and easy to implement. It is often used in conjunction with other algorithms, such as quicksort, to improve the overall performance of a sorting routine.

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.



**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

->

4 1 56 7 89 93 43 2

41 56 7 89 93 43 2

41 56 7 89 93 43 2

41 56 7 89 43 93 2

41 56 7 89 43 2 93

7 41 56 89 2 43 93

2 7 41 43 56 89 93