**MERGE SORT ALGORITHM:**

Merge Sort is a Divide and Conquer algorithm. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sublists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.

[Worst complexity](https://www.google.com/search?q=merge+sort+worst+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3MDWIL9Yyzk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItSrPLyouiU_Ozy3ISa3ILKmML85ILEpNWcQqnZtalJ6qANKhAFakgFAEAEktqo9fAAAA&sa=X&ved=2ahUKEwiY1d33tbX0AhUISmwGHf1kA-YQ6BMoAHoECEcQAg):  n\*log(n)

[Average complexity](https://www.google.com/search?q=merge+sort+average+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3MDWIL9YyzU620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItUosSy1KTE-NT87PLchJrcgsqYwvzkgsSk1ZxCqbm1qUnqoA0qMAVaaAUAYABCtt3GMAAAA&sa=X&ved=2ahUKEwiY1d33tbX0AhUISmwGHf1kA-YQ6BMoAHoECDgQAg): n\*log(n)

[Best complexity](https://www.google.com/search?q=merge+sort+best+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3MDWIL9Yyyk620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItUpKLS6JT87PLchJrcgsqYwvzkgsSk1ZxCqVm1qUnqoA0qAAUqOAUAMAh6oj0l0AAAA&sa=X&ved=2ahUKEwiY1d33tbX0AhUISmwGHf1kA-YQ6BMoAHoECEgQAg): n\*log(n)

[Space complexity](https://www.google.com/search?q=merge+sort+space+complexity&stick=H4sIAAAAAAAAAOPgE-LQz9U3MDWIL9bSzE620s8uiM8p1y_OLyrJzEuPT8xJzy_KLMnItcpNzc0vqowvzkgsSk1ZxCqdm1qUnqoAUqdQXJCYnKqQnJ9bkJNakVlSCQAKRctpVQAAAA&sa=X&ved=2ahUKEwiY1d33tbX0AhUISmwGHf1kA-YQ6BMoAHoECEkQAg)**:**n

**ALOGORITHM:**

MergeSort(arr[], l, r)

If r > l

**1.** Find the middle point to divide the array into two halves:

middle m = l+ (r-l)/2

**2.** Call mergeSort for first half:

Call mergeSort(arr, l, m)

**3.** Call mergeSort for second half:

Call mergeSort(arr, m+1, r)

**4.** Merge the two halves sorted in step 2 and 3:

Call merge(arr, l, m, r)

**WORKING:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **8** | **12** | **22** | **20** | **40** | **68** | **90** | **77** |

As there are eight elements in the given array, so it is divided into two arrays of size 4.

|  |  |  |  |
| --- | --- | --- | --- |
| **8** | **12** | **22** | **20** |

|  |  |  |  |
| --- | --- | --- | --- |
| **40** | **68** | **90** | **77** |

Again divide these two arrays into halves. As they are of size 4, so divide them into new arrays of size 2.

|  |  |
| --- | --- |
| **8** | **12** |

|  |  |
| --- | --- |
| **40** | **68** |

|  |  |
| --- | --- |
| **22** | **20** |

|  |  |
| --- | --- |
| **90** | **77** |

Again divide these arrays to get the atomic value that cannot be further divided.

Compare the element of each array and then combine them into another array in sorted order.

|  |  |
| --- | --- |
| **8** | **12** |

|  |  |
| --- | --- |
| **20** | **22** |

|  |  |
| --- | --- |
| **40** | **68** |

|  |  |
| --- | --- |
| **77** | **90** |

Now compare the arrays with two data values and merge them into an array of found values in sorted order.

|  |  |  |  |
| --- | --- | --- | --- |
| **8** | **12** | **20** | **22** |

|  |  |  |  |
| --- | --- | --- | --- |
| **40** | **68** | **77** | **90** |

Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **8** | **12** | **20** | **22** | **40** | **68** | **77** | **90** |

Now, the array is completely sorted.