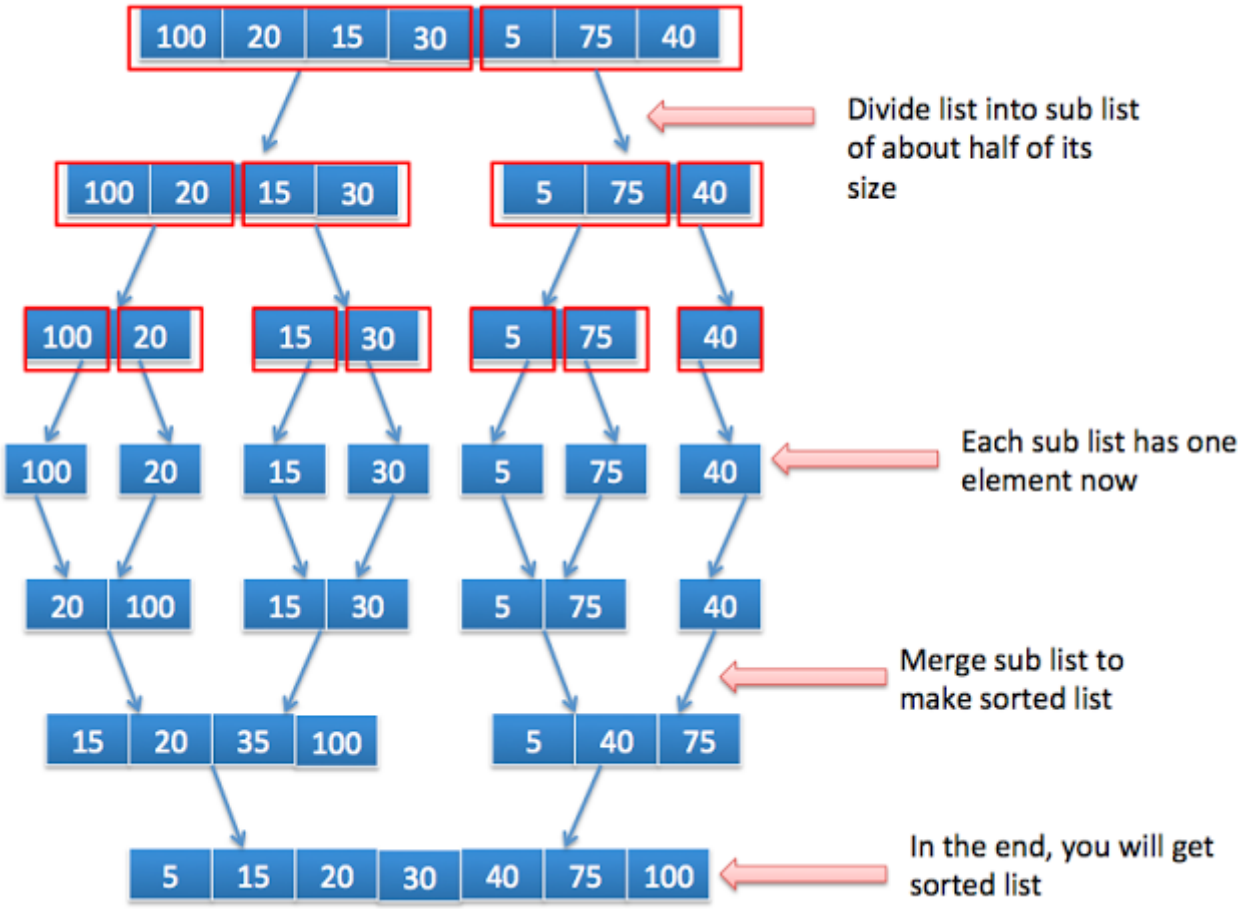
Merge sort is divide and conquer sorting algorithm. It is efficient, comparison based sorting algorithm.

It works on below principle:

- Divide list into sub list of about half size in each iteration until each sublist has only one element.
- Merge each sublist repeatedly to create sorted list. It will run until we have only 1 sorted list. This will be the sorted list.

Below diagram will make it clearer:



Example of Merge sort:

I have printed intermediate steps to understand algorithm better.

MergeSortMain.java

```
1
```

```
package org.arpit.java2blog;
public class MergeSortMain {

    /*
     * @author: Arpit Mandliya
     */
    static  int  arr[]={100,20,15,30,5,75,40};

    public static void main(String args[])
    {
        // Print array before merge sort
        System.out.println("Array before sorti
ng:");
        printArray(arr,0,arr.length-1);
        System.out.println("------------------
-----------");

        mergeSort(0,arr.length-1);

        System.out.println("------------------
-----------");

        // Print array after sorting
        System.out.println("Array After sortin
g:");
        printArray(arr,0,arr.length-1);


    }

    // Recursive algorithm for merge sort
    public static void mergeSort(int start,int
 end)
    {
        int mid=(start+end)/2;
        if(start<end)
        {
            // Sort left half
            mergeSort(start,mid);
            // Sort right half
            mergeSort(mid+1,end);
            // Merge left and right half
            merge(start,mid,end);
        }

    }


    private static void merge(int start, int m
id, int end) {
        // Initializing temp array and index
        int[] tempArray=new int[arr.length];
        int tempArrayIndex=start;

        System.out.print("Before Merging:  ");
        printArray(arr,start,end);

        int startIndex=start;
```

```
        int midIndex=mid+1;

        // It will iterate until smaller list
 reaches to the end
        while(startIndex<=mid && midIndex<=end
)
        {
            if(arr[startIndex]< arr[midIndex])
            {
                tempArray[tempArrayIndex++]=ar
r[startIndex++];
            }
            else
            {
                tempArray[tempArrayIndex++]=ar
r[midIndex++];
            }
        }

        // Copy remaining elements
        while(startIndex<=mid)
        {
            tempArray[tempArrayIndex++]=arr[st
artIndex++];
        }
        while(midIndex<=end)
        {
            tempArray[tempArrayIndex++]=arr[mi
dIndex++];
        }

        // Copy tempArray to actual array afte
r sorting
        for (int i = start; i <=end; i++) {
            arr[i]=tempArray[i];
        }

        System.out.print("After merging:   ");
        printArray(tempArray,start,end);
        System.out.println();
    }

    public static void printArray(int arr[],in
t start,int end)
    {
        for (int i = start; i <=end; i++) {
            System.out.print(arr[i]+" ");
        }
        System.out.println();
    }
}
```

When you run above program , you will get following output:

```
100 20 15 30 5 75 40
```

```
3  ----------------------------
4  Before Merging: 100 20
5  After merging: 20 100
6
7  Before Merging: 15 30
8  After merging: 15 30
9
10 Before Merging: 20 100 15 30
11 After merging: 15 20 30 100
12
13 Before Merging: 5 75
14 After merging: 5 75
15
16 Before Merging: 5 75 40
17 After merging: 5 40 75
18
19 Before Merging: 15 20 30 100 5 40 75
20 After merging: 5 15 20 30 40 75 100
21
22 ----------------------------
23 Array After sorting:
24 5 15 20 30 40 75 100
25
```

Complexity:

Best case: O(nlogn) or O(n)
Average case: O(nlogn)
Worst case: O(nlogn)
To understand more about complexity,please go through complexity of algorithm.