```java
public class MergeSort
{

    public static <E> E[] merge(E[] leftSide, E[] rightSide)
    {
        int totalLength = leftSide.length + rightSide.length;
        E[] result = SortUtils.arrayAs(leftSide, totalLength);

        int indexLeft = 0, indexRight = 0, indexResult = 0;

        while (indexLeft < leftSide.length && indexRight < rightSide.length) {
            if (SortUtils.compare(leftSide[indexLeft], rightSide[indexRight]) <= 0) {
                result[indexResult++] = leftSide[indexLeft++];
            } else {
                result[indexResult++] = rightSide[indexRight++];
            }
        }

        while (indexLeft < leftSide.length) {
            result[indexResult++] = leftSide[indexLeft++];
        }

        while (indexRight < rightSide.length) {
            result[indexResult++] = rightSide[indexRight++];
        }

        return result;
    }

    public static <E> E[] sort(E[] items, int i, int j)
    {
        //      if ( i > j) {
            //          return SortUtils.arrayAs(items, 0);
            //      }
        //
        if (i == j) {  // Base case for recursion: no elements to sort
```

```
37              return SortUtils.copyCell(items, i);
38          }
39
40          int mid = (i + j) / 2;
41
42          // Recursive case: sort the two halves
43          E[] sortedLeft = sort(items, i, mid);
44          E[] sortedRight = sort(items, mid + 1, j);
45
46
47          // Merge the sorted halves
48          return merge(sortedLeft, sortedRight);
49      }
50
51      public static <E> E[] sort(E[] items)
52      {
53          return sort(items, 0, items.length - 1);
54      }
55
56  }
57
```