**Step 1: Initial Call**

* The initial call is made with an input array, e.g., **[12, 11, 13, 5, 6, 7]**.

**Step 2: Length Check**

* The length of the input array is calculated using **count($arr)**. In this case, the length is **6**.

**Step 3: Base Case Check**

* Since the length is greater than **1**, the base case condition is not met, and the array needs to be divided further.

**Step 4: Array Splitting**

* The midpoint index is calculated as **$mid = (int) ($length / 2)** which results in **3**.
* The input array **[12, 11, 13, 5, 6, 7]** is split into two halves:
  + Left: **[12, 11, 13]**
  + Right: **[5, 6, 7]**

**Step 5: Recursive Calls for Left and Right Halves**

* Two recursive calls are made to **mergeSort()**:
  + **mergeSort([12, 11, 13])**
  + **mergeSort([5, 6, 7])**

**Step 6: Recursive Call for [12, 11, 13]**

* This is a recursive call to sort the left half **[12, 11, 13]**.

**Step 7: Length Check**

* The length of the input array **[12, 11, 13]** is calculated as **3**.

**Step 8: Base Case Check**

* The length is still greater than **1**, so the base case is not met.

**Step 9: Array Splitting**

* The midpoint index is calculated as **$mid = (int) ($length / 2)** which results in **1**.
* The input array **[12, 11, 13]** is split into two halves:
  + Left: **[12]**
  + Right: **[11, 13]**

**Step 10: Recursive Calls for Left and Right Halves**

* Two recursive calls are made to **mergeSort()**:
  + **mergeSort([12])**
  + **mergeSort([11, 13])**

**Step 11: Base Case Check for [12]**

* The length of the input array **[12]** is **1**, which meets the base case condition.

**Step 12: Returning Sorted Result for [12]**

* Since the length is 1, the array **[12]** is returned as is.

**Step 13: Base Case Check for [11, 13]**

* The length of the input array **[11, 13]** is **2**, which does not meet the base case condition.

**Step 14: Array Splitting for [11, 13]**

* The midpoint index is calculated as **$mid = (int) ($length / 2)** which results in **1**.
* The input array **[11, 13]** is split into two halves:
  + Left: **[11]**
  + Right: **[13]**

**Step 15: Recursive Calls for Left and Right Halves**

* Two recursive calls are made to **mergeSort()**:
  + **mergeSort([11])**
  + **mergeSort([13])**

**Step 16: Base Case Check for [11] and [13]**

* Both **[11]** and **[13]** have a length of **1**, meeting the base case condition.

**Step 17: Returning Sorted Results for [11] and [13]**

* Since both arrays have only one element, they are returned as is.

**Step 18: Merging [11] and [13]**

* The results from the recursive calls for **[11]** and **[13]** are merged using the **merge()** function (not shown).

**Step 19: Returning Merged Result for [11, 13]**

* The merged and sorted array **[11, 13]** is returned.

**Step 20: Merging [12] and [11, 13]**

* The results from the recursive call for **[12]** and the merged result for **[11, 13]** are merged using the **merge()** function.

**Step 21: Returning Final Merged Result for [12, 11, 13]**

* The fully merged and sorted array **[11, 12, 13]** is returned.

**Step 22: Recursive Call for [5, 6, 7]**

* This is a recursive call to sort the right half **[5, 6, 7]**.

**Step 23: Length Check**

* The length of the input array **[5, 6, 7]** is calculated as **3**.

**Step 24: Base Case Check**

* The length is greater than **1**, so the base case is not met.

**Step 25: Array Splitting**

* The midpoint index is calculated as **$mid = (int) ($length / 2)** which results in **1**.
* The input array **[5, 6, 7]** is split into two halves:
  + Left: **[5]**
  + Right: **[6, 7]**

**Step 26: Recursive Calls for Left and Right Halves**

* Two recursive calls are made to **mergeSort()**:
  + **mergeSort([5])**
  + **mergeSort([6, 7])**

**Step 27: Base Case Check for [5]**

* The length of the input array **[5]** is **1**, meeting the base case condition.

**Step 28: Returning Sorted Result for [5]**

* Since the length is 1, the array **[5]** is returned as is.

**Step 29: Base Case Check for [6, 7]**

* The length of the input array **[6, 7]** is **2**, which meets the base case condition.

**Step 30: Returning Sorted Result for [6, 7]**

* Since both arrays have only one element, they are returned as is.

**Step 31: Merging [5] and [6, 7]**

* The results from the recursive call for **[5]** and the sorted result for **[6, 7]** are merged using the **merge()** function.

**Step 32: Returning Final Merged Result for [5, 6, 7]**

* The fully merged and sorted array **[5, 6, 7]** is returned.

**Step 33: Merging [11, 12, 13] and [5, 6, 7]**

* The results from the recursive calls for **[11, 12, 13]** and **[5, 6, 7]** are merged using the **merge()** function.

**Step 34: Returning Final Merged Result for [5, 6, 7, 11, 12, 13]**

* The fully merged and sorted array **[5, 6, 7, 11, 12, 13]** is returned.

The **mergeSort()** function uses recursion to divide the input array into smaller sub-arrays, sort them, and then merge them back together to achieve a fully sorted array.