5.2.4 MERGE SORT

- Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm.
- Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually. Finally, sub-problems are combined to form the final solution.

Merge Sort Algorithm

- MergeSort is a recursive sorting procedure that uses at most **O(n lg(n))** comparisons.
- To sort an array of **n** elements, we perform the following steps in sequence:
- If **n < 2** then the array is already sorted.
- Otherwise, **n > 1**, and we perform the following three steps in sequence:
    1. **Sort** the **left half** of the the array using MergeSort.
    2. **Sort** the **right half** of the the array using MergeSort.
    3. **Merge** the sorted left and right halves.

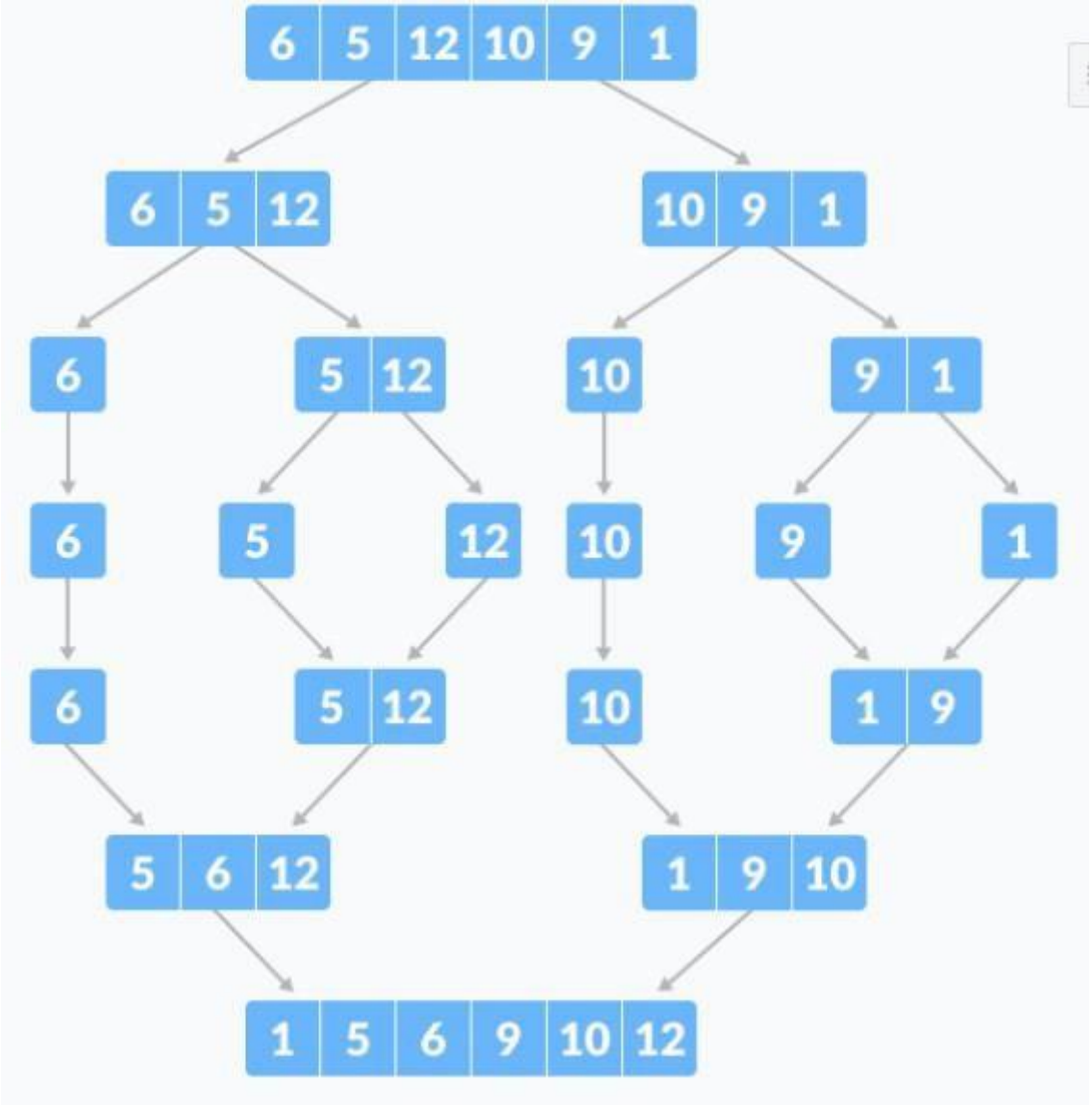
Algorithm

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



How to Merge

Here are two lists to be merged:

First: **(12, 16, 17, 20, 21, 27)**

Second: **(9, 10, 11, 12, 19)**

Compare**12** and **9**

First: **(12, 16, 17, 20, 21, 27)**

Second: **(10, 11, 12, 19)**

New: (9)

Compare **12** and **10**

First: (12, 16, 17, 20, 21, 27)

Second: (11, 12, 19)

New: (9, 10)

Compare **12** and **11**

First: (12, 16, 17, 20, 21, 27)

Second: (12, 19)

New: (9, 10, 11)

Compare **12** and **12**

First: (16, 17, 20, 21, 27)

Second: (12, 19)

New: (9, 10, 11, 12)

Compare **16** and **12**

First: (16, 17, 20, 21, 27)

Second: (19)

New: (9, 10, 11, 12, 12)

Compare **16** and **19**

First: (17, 20, 21, 27)

Second: (19)

New: (9, 10, 11, 12, 12, 16)

Compare **17** and **19**

First: (20, 21, 27)

Second: (19)

New: (9, 10, 11, 12, 12, 16, 17)

Compare **20** and **19**

First: (20, 21, 27)

Second: ( )

New: (9, 10, 11, 12, 12, 16, 17, 19)

Checkout **20** and **empty list**

First: ( )

Second: ( )

New: (9, 10, 11, 12, 12, 16, 17, 19, 20, 21, 27)

Program

```
void merge(int arr[], int low, int mid, int high)
{
   int i, j, k;
   int temp[50];

   i = low;
   j = mid + 1;
   k = low;

   while (i <= mid && j <= high)
   {
      if (arr[i] <= arr[j])
         temp[k++] = arr[i++];
      else
         temp[k++] = arr[j++];
   }

   while (i <= mid)
```

```
        temp[k++] = arr[i++];

    while (j <= high)
        temp[k++] = arr[j++];

    for (i = low; i <= high; i++)
        arr[i] = temp[i];
}

void mergeSort(int arr[], int low, int high)
{
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
        mergeSort(arr, low, mid);
        mergeSort(arr, mid + 1, high);
        merge(arr, low, mid, high);
    }
}
```

Advantages

- Sorts elements **faster even for large data**
- **Same speed in all cases** (best, average, worst)
- Keeps **same order for equal elements**
- Easy to understand using **divide and conquer**
- Works well for **linked lists**

Disadvantages

- Needs **extra memory** for temporary arrays
- Cannot sort in the **same array**
- Takes more time for **small number of elements**
- Uses **recursion**, which needs more memory

Applications

- Used to sort **large amounts of data**
- Used in **database systems**
- Used when **stable sorting** is needed
- Useful in **file sorting**
- Used in **parallel processing**