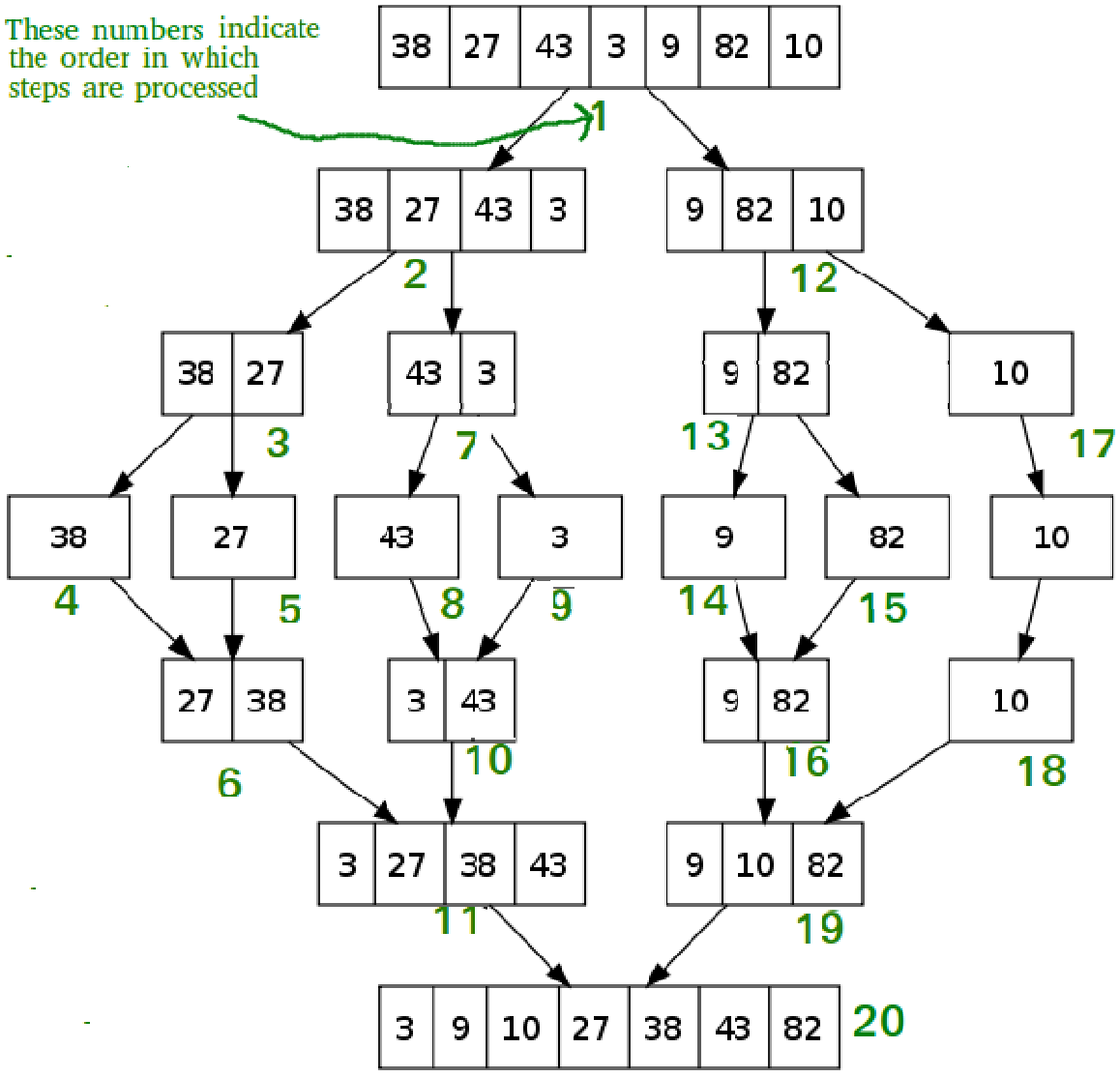
Merge Sort

Like QuickSort, Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves. The merge(arr, l, m, r) is key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one. See following C implementation for details.

```
MergeSort(arr[], l,  r)
If r > l
     1. Find the middle point to divide the array into two halves:
             middle m = (l+r)/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)
```

The following diagram from wikipedia shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}. If we take a closer look at the diagram, we can see that the array is recursively divided in two halves till the size becomes 1. Once the size becomes 1, the merge processes comes into action and starts merging arrays back till the complete array is merged.



```
/* Java program for Merge Sort */
class MergeSort
{
   // Merges two subarrays of arr[].
   // First subarray is arr[l..m]
   // Second subarray is arr[m+1..r]
   void merge(int arr[], int l, int m, int r)
   {
      // Find sizes of two subarrays to be merged
      int n1 = m - l + 1;
      int n2 = r - m;

      /* Create temp arrays */
      int L[] = new int [n1];
      int R[] = new int [n2];

      /*Copy data to temp arrays*/
      for (int i=0; i<n1; ++i)
         L[i] = arr[l + i];
      for (int j=0; j<n2; ++j)
         R[j] = arr[m + 1+ j];


      /* Merge the temp arrays */

      // Initial indexes of first and second subarrays
      int i = 0, j = 0;

      // Initial index of merged subarry array
      int k = l;
      while (i < n1 && j < n2)
      {
         if (L[i] <= R[j])
         {
            arr[k] = L[i];
            i++;
         }
         else
         {
            arr[k] = R[j];
            j++;
         }
         k++;
      }

      /* Copy remaining elements of L[] if any */
      while (i < n1)
      {
         arr[k] = L[i];
         i++;
         k++;
      }

      /* Copy remaining elements of R[] if any */
      while (j < n2)
      {
         arr[k] = R[j];
         j++;
         k++;
      }
```

```
    }

    // Main function that sorts arr[l..r] using
    // merge()
    void sort(int arr[], int l, int r)
    {
        if (l < r)
        {
            // Find the middle point
            int m = (l+r)/2;

            // Sort first and second halves
            sort(arr, l, m);
            sort(arr , m+1, r);

            // Merge the sorted halves
            merge(arr, l, m, r);
        }
    }

    /* A utility function to print array of size n */
    static void printArray(int arr[])
    {
        int n = arr.length;
        for (int i=0; i<n; ++i)
            System.out.print(arr[i] + " ");
        System.out.println();
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = {12, 11, 13, 5, 6, 7};

        System.out.println("Given Array");
        printArray(arr);

        MergeSort ob = new MergeSort();
        ob.sort(arr, 0, arr.length-1);

        System.out.println("\nSorted array");
        printArray(arr);
    }
}
```

Output:

```
Given array is

12 11 13 5 6 7


Sorted array is

5 6 7 11 12 13
```