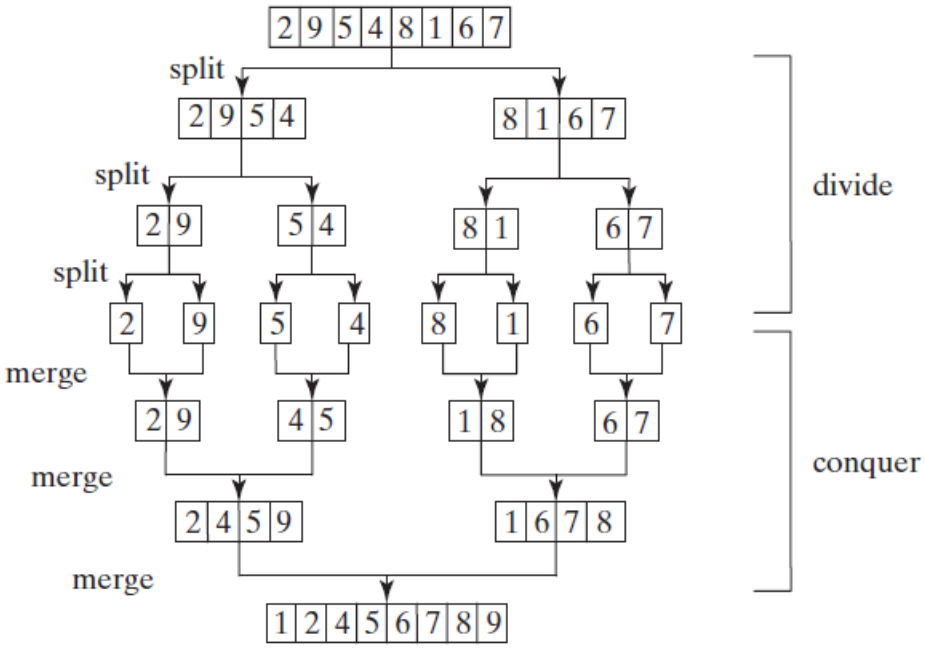
MERGE-SORT ALGORITHM

The merge-sort algorithm can be described recursively as follows:
The algorithm divides the array into two halves and applies a merge sort on each half recursively. After the two halves are sorted, the algorithm then merges them.



Let T(n) denote the time required for sorting an array of n elements using a merge sort. Without loss of generality, assume n is a power of 2. The merge-sort algorithm splits the array into two subarrays, sorts the subarrays using the same algorithm recursively, then merges the subarrays.
Therefore,

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + mergetime$$

The first $T(\frac{n}{2})$ is the time for sorting the first half of the array and the second $T(\frac{n}{2})$ is the time for sorting the second half. To merge two subarrays, it takes at most n - 1 comparisons to compare the elements from the two subarrays, and n moves to move elements to the temporary array. Thus, the total time is 2n - 1. Therefore,

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + 2n - 1 = O(n \log n)$$

The complexity of a merge sort is O(n logn). This algorithm is better than selection sort, insertion sort, and bubble sort because the time complexity of these algorithms is $O(n^2)$

```
class MS{
/* Function to merge the subarrays of a[] */
static void merge(int a[ ], int beg, int mid, int end)
{
   int i, j, k;
   int n1 = mid - beg + 1;
   int n2 = end - mid;
  /* temporary Arrays */
      int L[ ] = new int[n1];
      int R[ ] = new int[n2];
   /* copy data to temp arrays */
   for (i = 0; i < n1; i++)
   L[i] = a[beg + i];
   for (j = 0; j < n2; j++)
   R[j] = a[mid + 1 + j];
   i = 0; /* initial index of first sub-array */
   j = 0; /* initial index of second sub-array */
   k = beg;  /* initial index of merged sub-array */
    while (i < n1 && j < n2)
   {
      if(L[i] <= R[j])
      {
         a[k] = L[i];
         i++;
      }
      else
      {
         a[k] = R[j];
         j++;
      }
      k++;
   }
   while (i<n1)
   {
      a[k] = L[i];
      i++;
      k++;
   }
```

```
   while (j<n2)
   {
      a[k] = R[j];
      j++;
      k++;
   }
}
static void mergeSort(int a[ ], int beg, int end)
{
   if (beg < end)
   {
      int mid = (beg + end) / 2;
      mergeSort(a, beg, mid);
      mergeSort(a, mid + 1, end);
      merge(a, beg, mid, end);
   }
}
public static void main(String args[])
{
   int a[ ] = { 2, 9, 5, 4, 8, 1, 6, 7 };
   int n = a.length;
   mergeSort(a, 0, n - 1);
   System.out.println("\nAfter sorting array elements are - ");
   for (int i = 0; i < n; i++)
      System.out.print(a[i] + " ");
}
}
```