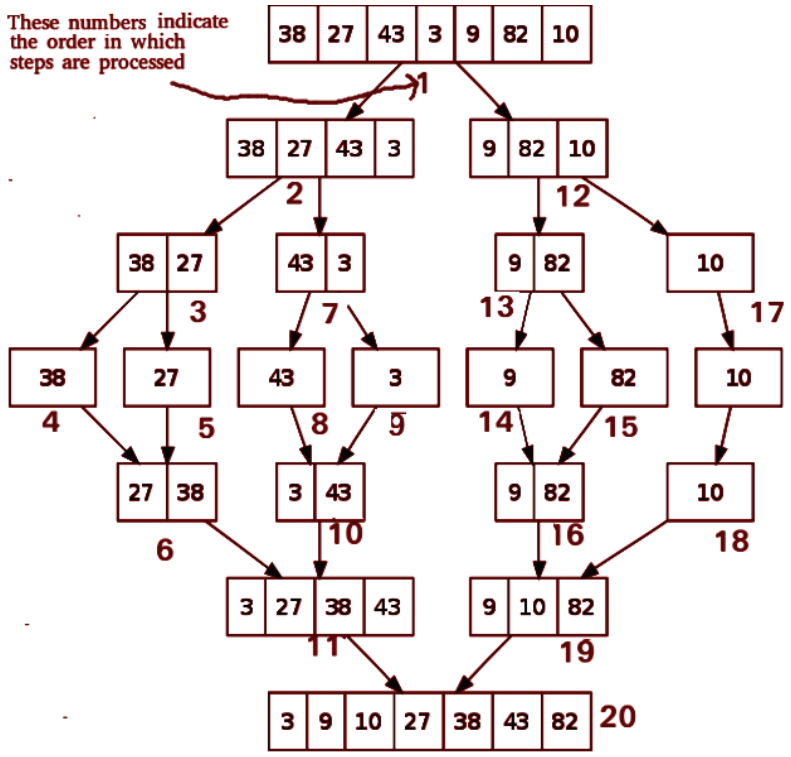
Merge Sort

Merge sort requires dividing a given list into equal halves until it can no longer be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines/merges the smaller sorted lists keeping the new list sorted too.

- If it is only one element in the list it is already sorted, return.
- Divide the list recursively into two halves until it can't be divided further.
- Merge the smaller lists into a new list in sorted order.

It has just one disadvantage that it creates a copy of the array and then works on that copy.

The following diagram 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 into two halves till the size becomes 1. Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged.



Pseudocode:

```
/*
      array from leftidx(0) to rightidx(arr.length-1) is considered
*/
function mergeSort(arr, leftidx , rightidx )

      //   base case : only 1 element to be sorted
```

```
        if  leftidx == rightidx
                return

         middle = (leftidx + rightidx) / 2

         //   smaller problems
         mergerSort(arr, leftidx, middle)
         mergeSort(arr, middle + 1, rightidx)


         //   selfwork
         merge(leftidx , middle, rightidx)


function merge(arr, leftidx , middle , rightidx)

        leftlo = leftidx
        rightlo = middle+1
        idx = 0
        //   create an array temp of size (rightidx - leftidx + 1)

        while leftlo <= middle AND rightlo <= rightidx
                if   arr[leftlo] < arr[rightlo]
                        temp[idx] = arr[leftlo]
                        leftlo++
                        idx++
                  else
                        temp[idx] = arr[rightlo]
                        rightlo++
                        idx++


        while leftlo <= middle
                temp[idx] = arr[leftlo]
                leftlo++
                idx++


        while rightlo <= rightidx
                temp[idx] = arr[rightlo]
                rightlo++
                idx++
```

```
//   copy temp to original array
count = 0
while count < temp.length AND leftidx <= rightidx
        arr[leftidx] = temp[count]
        leftidx++
        count++
```

Time complexity: O(N*logN), in the worst case.
N elements are divided recursively into 2 parts, this forms a tree with nodes as divided parts of the array representing the subproblems. The height of the tree will be $\log_2 N$ and at each level of the tree, the computation cost of all the subproblems will be N. At each level the merge operation will take O(N) time. So the overall complexity comes out to be O(N*logN).

Space complexity: O(N), as extra space is required to sort the array.