# MERGE SORT

- Merge sort is a sorting technique based on divide and conquer technique.
- With worst-case time complexity being O(n log n), it is one of the most respected algorithms.
- Merge sort first divides the array into equal halves and then combines them in a sorted manner

## Algorithm

- Merge sort keeps on dividing the list into equal halves until it can no more be divided. By definition, if it is only one element in the list, it is sorted. Then, merge sort combines the smaller sorted lists keeping the new list sorted too

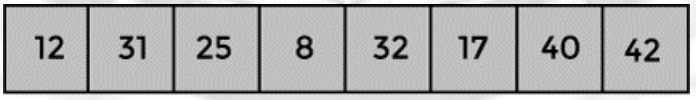
  Step 1 − if it is only one element in the list it is already sorted, return.

  Step 2 − divide the list recursively into two halves until it can no more be divided.Step 3 − merge the smaller lists into new list in sorted order.

## Working of Merge sort Algorithm

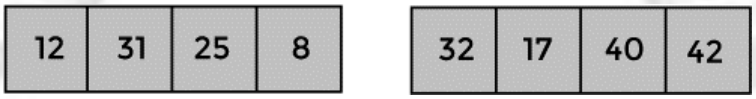
Consider an unsorted array elements12, 31, 25, 8, 32, 17, 40 and 42

Let the elements of array are

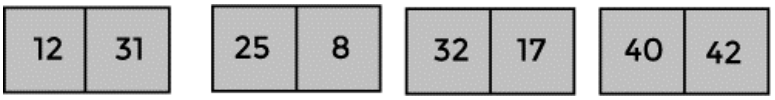



First divide the given array into two equal halves. Merge sort keeps dividing the list into equal parts until it cannot be further divided.

As there are eight elements in the given array, so it is divided into two arrays of size 4.



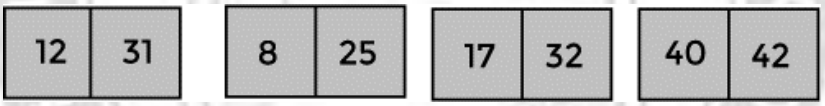

Now, again divide these two arrays into halves. As they are of size 4, so divide them into new arrays of size 2.



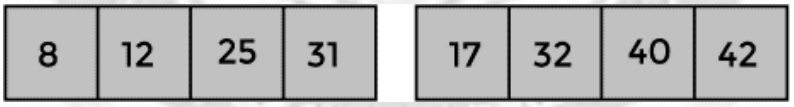

Now, again divide these arrays to get the atomic value that cannot be further divided.


Now, combine them in the same manner they were broken. First compare the element of each array and then combine them into another array in sorted order.

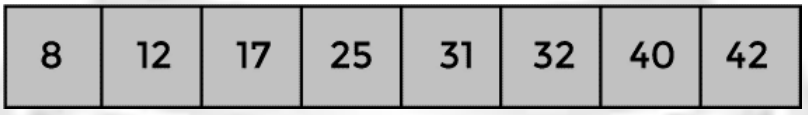
So, first compare 12 and 31, both are in sorted positions. Then compare 25 and 8, and in the list of two values, put 8 first followed by 25. Then compare 32 and 17, sort them and put 17 first followed by 32. After that, compare 40 and 42, and place them sequentially.




In the next iteration of combining, now compare the arrays with two data valuesand merge them into an array of found values in sorted order.




Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like




**Merge sort complexity**

| **Time Complexity** | |
|---|---|
| Best | O(n*log n) |
| Worst | O(n*logn) |
| Average | O(n*logn) |
| **Space Complexity** | O(n) |
| **Stability** | YES |

## Merge Sort Applications

- Inversion count problem
- External sorting
- E-commerce applications

**Example Program 5.4:** Program for implementing Merge Sort

```
#include <stdio.h>
/* Function to merge the subarrays of a[] */
void merge(int a[], int beg, int mid, int end)
{
   int i, j, k;
   int n1 = mid - beg + 1;
   int n2 = end - mid;
   int LeftArray[n1], RightArray[n2]; //temporary arrays
   /* copy data to temp arrays */
   for (int i = 0; i < n1; i++)
   LeftArray[i] = a[beg + i];
   for (int j = 0; j < n2; j++)
   RightArray[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(LeftArray[i] <= RightArray[j])
      {
         a[k] = LeftArray[i];
         i++;
      }
```

```
        else
        {
            a[k] = RightArray[j];
            j++;
        }
        k++;
    }
    while (i<n1)
    {
        a[k] = LeftArray[i];
        i++;
        k++;
    }
    while (j<n2)
    {
        a[k] = RightArray[j];
        j++;
        k++;
    }
}
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);
```

```
    }
}
/* Function to print the array */
void printArray(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
}
int main()
{
    int a[] = { 10, 35, 23, 5, 31, 19, 40, 43 };
    int n = sizeof(a) / sizeof(a[0]);
    printf("Before sorting array elements are -
    \n");printArray(a, n);
    mergeSort(a, 0, n - 1);
    printf("After sorting array elements are -
    \n");printArray(a, n);
    return 0;
    }
```

**Output**

```
Before sorting array elements are
10, 35, 23, 5, 31, 19, 40, 43
After sorting array elements are
5, 10, 19, 23, 31, 35, 40, 43
```