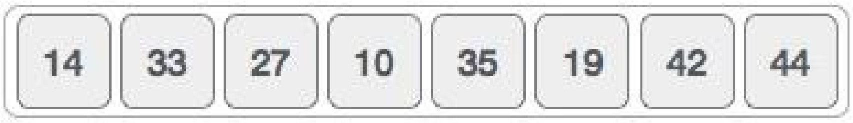
Merge Sort Algorithm

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 used and approached algorithms.

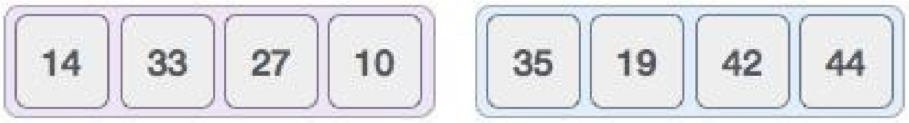
Merge sort first divides the array into equal halves and then combines them in a sorted manner.

How Merge Sort Works?

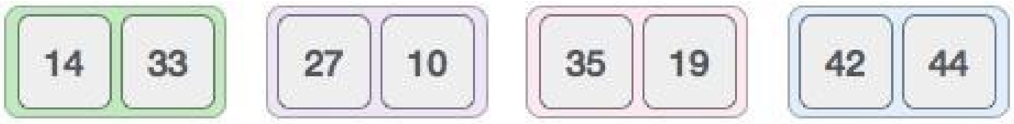
To understand merge sort, we take an unsorted array as the following −



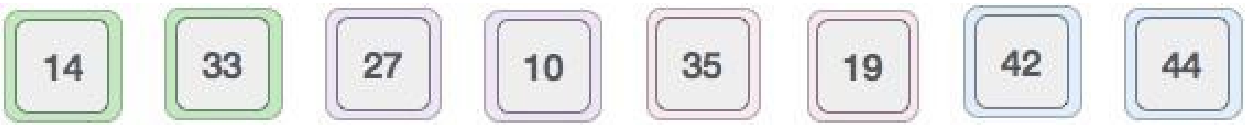
We know that merge sort first divides the whole array iteratively into equal halves unless the atomic values are achieved. We see here that an array of 8 items is divided into two arrays of size 4.



This does not change the sequence of appearance of items in the original. Now we divide these two arrays into halves.

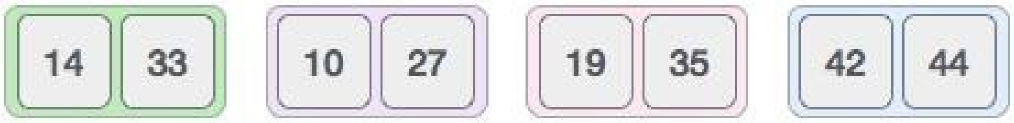


We further divide these arrays and we achieve atomic value which can no more be divided.

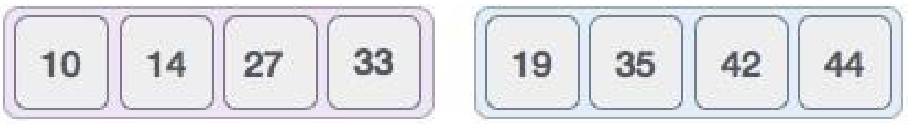


Now, we combine them in exactly the same manner as they were broken down. Please note the color codes given to these lists.

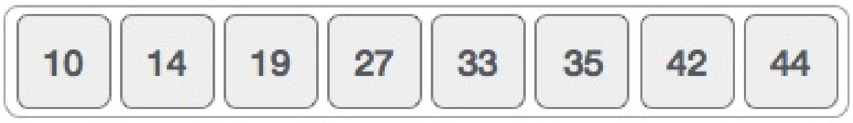
We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the target list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.



In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.



After the final merging, the list becomes sorted and is considered the final solution.



Merge Sort 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 considered 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, consider it already
sorted, so 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.
```

Pseudocode

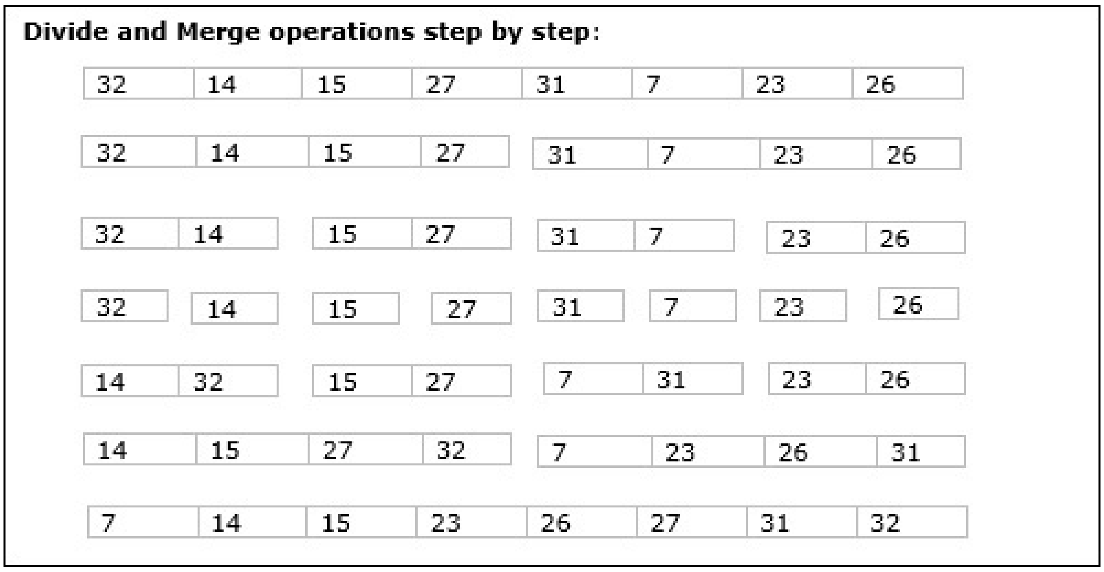
We shall now see the pseudocodes for merge sort functions. As our algorithms point out two main functions − divide & merge.

Merge sort works with recursion and we shall see our implementation in the same way.

```
procedure mergesort( var a as array )
   if ( n == 1 ) return a
      var l1 as array = a[0] ... a[n/2]
      var l2 as array = a[n/2+1] ... a[n]
      l1 = mergesort( l1 )
      l2 = mergesort( l2 )
      return merge( l1, l2 )
end procedure
procedure merge( var a as array, var b as array )
   var c as array
   while ( a and b have elements )
      if ( a[0] > b[0] )
         add b[0] to the end of c
         remove b[0] from b
      else
         add a[0] to the end of c
         remove a[0] from a
      end if
   end while
   while ( a has elements )
      add a[0] to the end of c
      remove a[0] from a
   end while
   while ( b has elements )
      add b[0] to the end of c
      remove b[0] from b
   end while
   return c
end procedure
```

Example

In the following example, we have shown Merge-Sort algorithm step by step. First, every iteration array is divided into two sub-arrays, until the sub-array contains only one element. When these sub-arrays cannot be divided further, then merge operations are performed.



Analysis

Let us consider, the running time of Merge-Sort as **T(n)**. Hence,

$$\mathrm{T}(\mathrm{n}) = \begin{cases} c & if\, n \leq 1 \\ 2\, xT\left(\frac{n}{2}\right) + dxn & otherwise \end{cases} \; where\, c\, and\, d\, are\, constants$$

Therefore, using this recurrence relation,

$$T(n) = 2^i\, T\left(n/2^i\right) + i \cdot d \cdot n$$

$$As,\ i = log\, n,\, T(n) = 2^{log\, n} T\left(n/2^{log\, n}\right) + log\, n \cdot d \cdot n$$

$$= c \cdot n + d \cdot n \cdot log\, n$$

$$Therefore,\ T(n) = O(n\, log\, n).$$

Example

Following are the implementations of the above approach in various programming languages −

C | C++ | Java | Python

```
#define max 10
int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
int b[10];
void merging(int low, int mid, int high){
   int l1, l2, i;
   for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high;
      if(a[l1] <= a[l2])
         b[i] = a[l1++];
      else
         b[i] = a[l2++];
   }
   while(l1 <= mid)
      b[i++] = a[l1++];
   while(l2 <= high)
      b[i++] = a[l2++];
   for(i = low; i <= high; i++)
      a[i] = b[i];
}
void sort(int low, int high){
   int mid;
   if(low < high) {
      mid = (low + high) / 2;
      sort(low, mid);
      sort(mid+1, high);
      merging(low, mid, high);
   } else {
      return;
   }
}
int main(){
   int i;
   printf("Array before sorting\n");
   for(i = 0; i <= max; i++)
      printf("%d ", a[i]);
   sort(0, max);
   for(i = 0; i <= max; i++)
      printf("%d ", a[i]);
}
```

Output

```
Array before sorting
10 14 19 26 27 31 33 35 42 44 0
Array after sorting
0 10 14 19 26 27 31 33 35 42 44
```