Inplace Mergesort

In-place implementation of Merge Sort

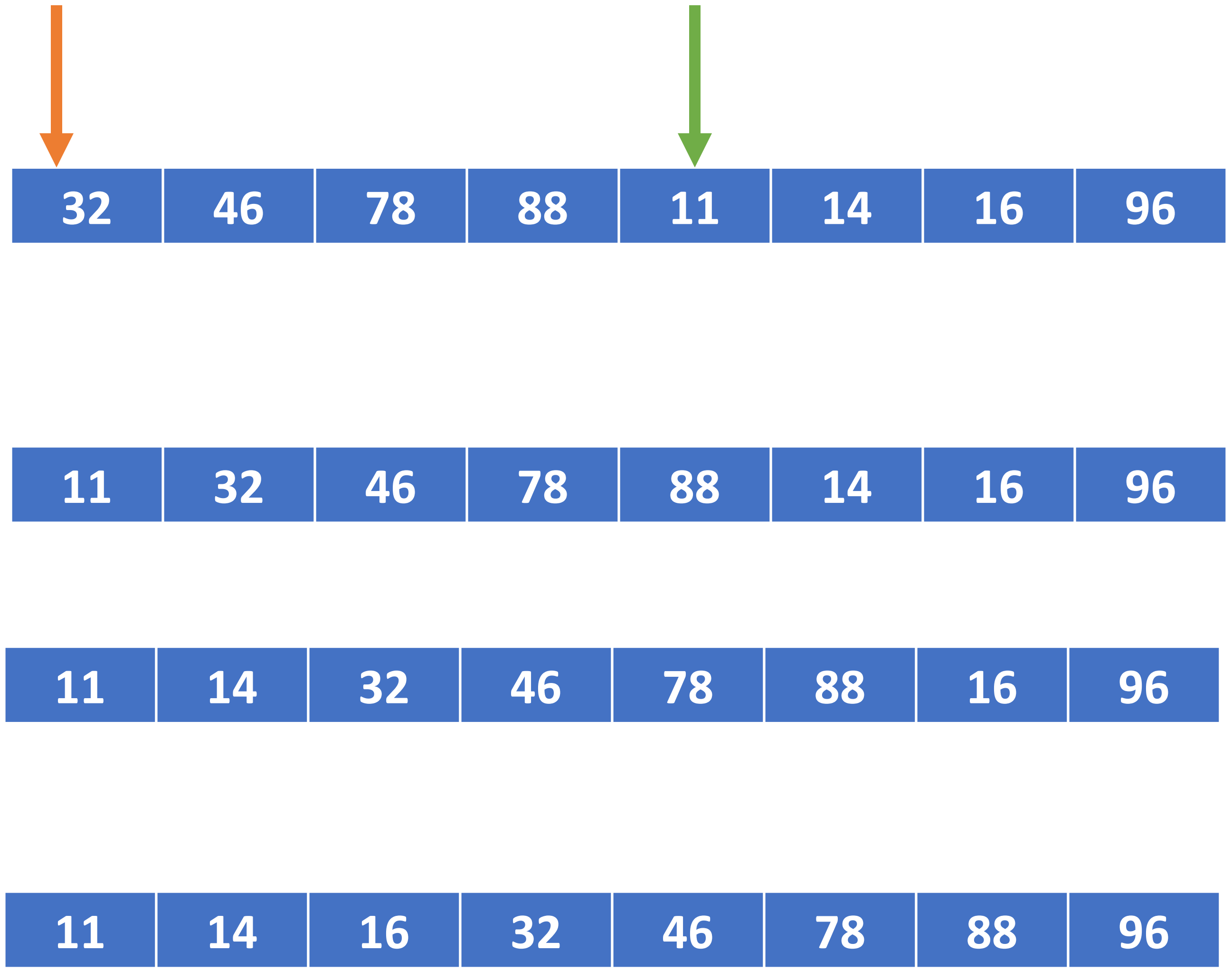
1. Maintain two pointers that point to the start of the segments which have to be merged.
2. Compare the elements at which the pointers are present.
3. If *element1 < element2*

   then *element1* is at right position,

   So, increase *pointer1*.

   Else shift **all the elements** between element1 and *element2(including element1 but excluding element2)* right by 1 and then place the element2 in the previous place*(i.e. before shifting right)* of element1. Increment **all the pointers by *1***.

In-place implementation of Merge Sort



```
// In-place Merge Sort
// Merges two subarrays of arr[].
// First subarray is arr[l..m].
// Second subarray is arr[m+1..r]
// Two pointers to maintain start of both arrays to merge
#include <stdio.h>
void merge(int arr[], int start, int mid, int end)
{       int start2 = mid + 1;
        // If already sorted
        if (arr[mid] <= arr[start2]) return;
```

```
    while (start <= mid && start2 <= end)
    {      // If element1 is in right place
         if (arr[start] <= arr[start2])     start++;
         else {   int value = arr[start2];
               int index = start2;
          // Shift all the elements between element1 and element2, right by 1.
               while (index != start)
               {   arr[index] = arr[index - 1];
                   index--;
                }
                arr[start] = value;
               start++;   mid++;    start2++;   // Update all the pointers
             }
      }
  }
```

```
/* l is for left index and r is right index of the sub-array of arr to be sorted */
void mergeSort(int arr[], int l, int r)
{  if (l < r)
   {   int m = (l + r) / 2;
       mergeSort(arr, l, m);
       printf("\nArray After sorting First half of l=%d,r=%d, m=%d : ",l,r,m);
       printArray(arr, l, m);
       mergeSort(arr, m + 1, r);
       printf("\nArray After sorting Second half of l=%d,r=%d, m=%d : ",l,r,m);
       printArray(arr, m+1, r);
       printf("\nArray Before Merging : ");
       printArray(arr, l, r);
       merge(arr, l, m, r);
       printf("\nArray After Merging : ");
       printArray(arr, l, r);
   }
}
```

```
/* Function to print an array */
void printArray(int A[], int l, int r)
{           int i;
            for (i = l; i <= r; i++)
                        printf("%d ", A[i]);
            printf("\n");
}
int main()
{           int arr[] = { 44,22};//,5,13,11,6,7,98,56,27,67,6,54,33,55,43,27,65 };
            int arr_size = sizeof(arr) / sizeof(arr[0]);
            printf("Given Array is :");
            printArray(arr, 0,arr_size-1);
            mergeSort(arr, 0, arr_size - 1);
            printf("Sorted Array is :");
            printArray(arr, 0,arr_size-1);
            return 0;
}
```

```
Given Array is :44 22
Array After sorting First half of l=0,r=1, m=0 : 44
Array After sorting Second half of l=0,r=1, m=0 : 22
Array Before Merging : 44 22
Array After Merging : 22 44
Sorted Array is :22 44
```

```
Given Array is :44 22 5 13 11 6
Array After sorting First half of l=0,r=1, m=0 : 44
Array After sorting Second half of l=0,r=1, m=0 : 22
Array Before Merging : 44 22
Array After Merging : 22 44
Array After sorting First half of l=0,r=2, m=1 : 22 44
Array After sorting Second half of l=0,r=2, m=1 : 5
Array Before Merging : 22 44 5
Array After Merging : 5 22 44
Array After sorting First half of l=0,r=5, m=2 : 5 22 44
Array After sorting First half of l=3,r=4, m=3 : 13
Array After sorting Second half of l=3,r=4, m=3 : 11
Array Before Merging : 13 11
Array After Merging : 11 13
Array After sorting First half of l=3,r=5, m=4 : 11 13
Array After sorting Second half of l=3,r=5, m=4 : 6
Array Before Merging : 11 13 6
Array After Merging : 6 11 13
Array After sorting Second half of l=0,r=5, m=2 : 6 11 13
Array Before Merging : 5 22 44 6 11 13
Array After Merging : 5 6 11 13 22 44
Sorted Array is :5 6 11 13 22 44
```

```
Given Array is :44 22 5 13 11 6 7 98 56 27 67 6 54 33 55 43 27 65
Array After sorting First half of l=0,r=1, m=0 : 44
Array After sorting Second half of l=0,r=1, m=0 : 22
Array Before Merging : 44 22
Array After Merging : 22 44
Array After sorting First half of l=0,r=2, m=1 : 22 44
Array After sorting Second half of l=0,r=2, m=1 : 5
Array Before Merging : 22 44 5
Array After Merging : 5 22 44
Array After sorting First half of l=0,r=4, m=2 : 5 22 44
Array After sorting First half of l=3,r=4, m=3 : 13
Array After sorting Second half of l=3,r=4, m=3 : 11
Array Before Merging : 13 11
Array After Merging : 11 13
Array After sorting Second half of l=0,r=4, m=2 : 11 13
Array Before Merging : 5 22 44 11 13
Array After Merging : 5 11 13 22 44
Array After sorting First half of l=0,r=8, m=4 : 5 11 13 22 44
Array After sorting First half of l=5,r=6, m=5 : 6
Array After sorting Second half of l=5,r=6, m=5 : 7
Array Before Merging : 6 7
Array After Merging : 6 7
Array After sorting First half of l=5,r=8, m=6 : 6 7
Array After sorting First half of l=7,r=8, m=7 : 98
Array After sorting Second half of l=7,r=8, m=7 : 56
Array Before Merging : 98 56
Array After Merging : 56 98
Array After sorting Second half of l=5,r=8, m=6 : 56 98
Array Before Merging : 6 7 56 98
Array After Merging : 6 7 56 98
Array After sorting Second half of l=0,r=8, m=4 : 6 7 56 98
Array Before Merging : 5 11 13 22 44 6 7 56 98
Array After Merging : 5 6 7 11 13 22 44 56 98
Array After sorting First half of l=0,r=17, m=8 : 5 6 7 11 13 22 44 56 98
Array After sorting First half of l=9,r=10, m=9 : 27
Array After sorting Second half of l=9,r=10, m=9 : 67
Array Before Merging : 27 67
Array After Merging : 27 67
Array After sorting First half of l=9,r=11, m=10 : 27 67
Array After sorting Second half of l=9,r=11, m=10 : 6
Array Before Merging : 27 67 6
Array After Merging : 6 27 67
Array After sorting First half of l=9,r=13, m=11 : 6 27 67
Array After sorting First half of l=12,r=13, m=12 : 54
Array After sorting Second half of l=12,r=13, m=12 : 33
Array Before Merging : 54 33
Array After Merging : 33 54
Array After sorting Second half of l=9,r=13, m=11 : 33 54
Array Before Merging : 6 27 67 33 54
Array After Merging : 6 27 33 54 67
Array After sorting First half of l=9,r=17, m=13 : 6 27 33 54 67
Array After sorting First half of l=14,r=15, m=14 : 55
Array After sorting Second half of l=14,r=15, m=14 : 43
Array Before Merging : 55 43
Array After Merging : 43 55
Array After sorting First half of l=14,r=17, m=15 : 43 55
Array After sorting First half of l=16,r=17, m=16 : 27
Array After sorting Second half of l=16,r=17, m=16 : 65
Array Before Merging : 27 65
Array After Merging : 27 65
Array After sorting Second half of l=14,r=17, m=15 : 27 65
Array Before Merging : 43 55 27 65
Array After Merging : 27 43 55 65
Array After sorting Second half of l=9,r=17, m=13 : 27 43 55 65
Array Before Merging : 6 27 33 54 67 27 43 55 65
Array After Merging : 6 27 27 33 43 54 55 65 67
Array After sorting Second half of l=0,r=17, m=8 : 6 27 27 33 43 54 55 65 67
Array Before Merging : 5 6 7 11 13 22 44 56 98 6 27 27 33 43 54 55 65 67
Array After Merging : 5 6 6 7 11 13 22 27 27 33 43 44 54 55 56 65 67 98
Sorted Array is :5 6 6 7 11 13 22 27 27 33 43 44 54 55 56 65 67 98
```

Complexity of the In-place Merge

- Complexity of the In-place Merge sort discussed earlier is
  - $O(n^2 \log n)$ as the in-place merge takes $O(n^2)$
- Compared the standard merge sort complexity $O(n \log n)$ in-place merge sort takes more time.
- But the space complexity is 0(1) compared to the $O(n)$ of standard merge sort