Reg.No : 24BCE0554
Name : Partha Pratim Gogoi
Topic : Merge Sort

Algorithm:

```
1  array = [ … ]                                                   [T.C: O(1)]
2  mergeSort(arr)                                             [T.C: O(nlog(n))]
   2.1     if (arr.size() <= 1) then return
   2.2     split(arr, 0, arr.size()-1)                        [T.C: O(nlog(n))]
3  split(arr, left, right)                                    [T.C: O(nlog(n))]
   3.1     if (left >= right) then return
   3.2     int mid = left + (right-left)/2
   3.3     split(arr, left, mid)                         [T.C: T(n)=2T(n/2)+O(n)]
   3.4     split(arr, mid+1, right)                      [T.C: T(n)=2T(n/2)+O(n)]
   3.5     merge(arr, left, mid, right)                               [T.C: O(n)]
4  merge(arr, left, mid, right)                            [T.C: O(n1+n2)=O(n)]
   4.1     n1 = mid - left + 1
   4.2     n2 = right - mid
   4.3     L[n1] = []
   4.4     R[n2] = []
   4.5     for i in 0→n1                                             [T.C: O(n1)]
     4.5.1   L[i] = arr[left+i]
   4.6     for j in 0→n2                                             [T.C: O(n2)]
     4.6.1   R[j] = arr[right+j]
   4.7     i=0, j=0, k=left
   4.8     while (i<n1 && j<n2)                               [T.C: O(n1)|O(n2)]
     4.8.1   if (L[i] <= R[j]) then arr[k++] = L[i++]
     4.8.2   else arr[k++] = R[j++]
   4.9     while (i<n1)                                              [T.C: O(n1)]
     4.9.1   arr[k++] = L[i++]
   4.10    while (j<n2)                                              [T.C: O(n2)]
     4.10.1  arr[k++] = R[j++]
5  mergeSort(array)                                           [T.C: O(nlog(n))]
```

Time Complexity:

1. split()

split(arr, left, mid) → T(n/2) [implies ‘n/2 time’]
split(arr, mid+1, right) → T(n/2) [implies ‘n/2 time’]
merge(…) → O(n)

Hence,

T.C. = 2T(n/2) + O(n)

By Master’s Theorem:

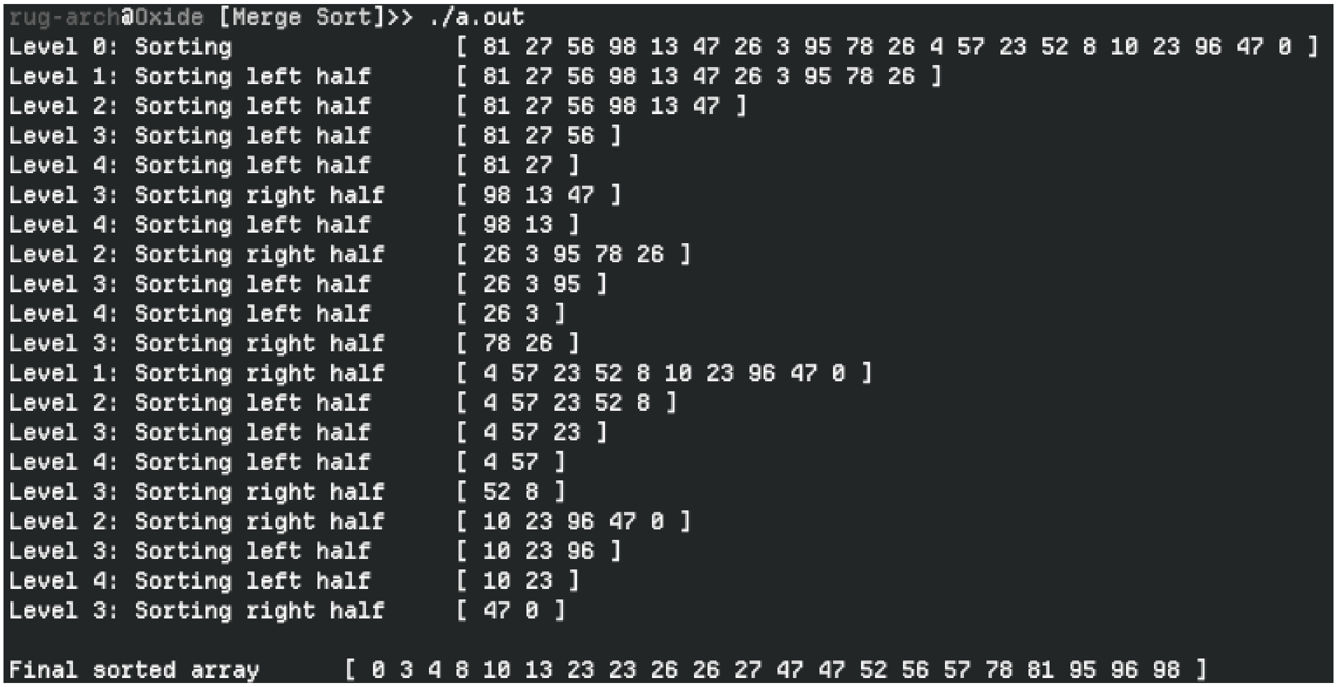
$$T(n) = O(n^{\log_2 2}.\log^{0+1}(n))$$
$$= O(n\log(n))$$

Total Time Complexity = O(1) + O(nlog(n)) + O(n)
= O(nlog(n))

Source Code:

```
#include <iostream>
#include <vector>
using namespace std;

///////////////////
/// Merge Logic ///
///////////////////
void merge(vector<int>& arr, int left, int mid, int right) {
    // ---------------------- //
    // Define sub-array sizes //
    // ---------------------- //
    int n1 = mid - left + 1;                          // Left sub-array
    int n2 = right - mid;                             // Right sub-array

    // ----------------------- //
    // Create temporary arrays //
    // ----------------------- //
    vector<int> L(n1);                                // Left sub-array
    vector<int> R(n2);                                // Right sub-array

    // -------------------- //
    // Populate temp arrays //
    // -------------------- //
    for (int i=0; i<n1; i++) {                        // Left sub-array
      L[i] = arr[left+i];
    }
    for (int j=0; j<n2; j++) {                        // Right sub-array
      R[j] = arr[mid+1+j];
    }

    // ----------------- //
    // Merge temp arrays //
    // ----------------- //
    int i=0;                                          // Initial index of left sub-array
    int j=0;                                          // Initial index of right sub-array
    int k=left;                                       // Initial index of merged sub-array
    while (i<n1 && j<n2) {
      if (L[i] <= R[j]) { arr[k++] = L[i++]; }
      else           { arr[k++] = R[j++]; }
    }

    // ----------------------- //
    // Copy remaining elements //                 // This process will occur only for
one array
    // ----------------------- //
    while (i<n1) { arr[k++] = L[i++]; }           // Case 1: Left sub-array has
elements remaining
    while (j<n2) { arr[k++] = R[j++]; }           // Case 2: Right sub-array has
elements remaining
}

//////////////////
/// Propagator ///
//////////////////
void split(vector<int>& arr, int left, int right, int level, bool isLeft) {
    if (left >= right) return;

    // ------------------------------------------------ //
    // Display current level and subarray being sorted //
    // ------------------------------------------------ //
    cout << "Level " << level << ": Sorting ";

    if (level == 0)      { cout << "        [ "; }
    else if (isLeft)     { cout << "left half     [ "; }
    else           { cout << "right half    [ "; }

    for (int i=left; i<=right; i++) {
      cout << arr[i] << " ";
    }
    cout << "]" << endl;



    int mid = left + (right - left)/2;

    // ------------------------ //
    // Create + Sort sub-arrays //                // Front Propagation
    // ------------------------ //
    split(arr, left, mid, level+1, true);         // Left sub-array
    split(arr, mid+1, right, level+1, false);     // Right sub-array

    // ---------------- //
    // Merge sub-arrays //                        // Back Propagation
    // ---------------- //
    merge(arr, left, mid, right);
}

/////////////////
/// Initiator ///
/////////////////
void mergeSort(vector<int>& arr) {
    if (arr.size() <= 1) return;

    split(arr, 0, arr.size()-1, 0, true);         // Either true/false works during
initiation
}

////////////////////
/// Driver Code ///
////////////////////
int main() {
    vector<int> array = {81,27,56,98,13, 47,26,3,95,78, 26,4,57,23,52, 8,10,23,96,47,
0};

    mergeSort(array);

    cout << endl << "Final sorted array    [ ";
    for (int val: array) {
      cout << val << " ";
    }
    cout << "]" << endl;

    return 0;
}
```

Sample Output:



```
rug-arch@Oxide [Merge Sort]>> ./a.out
Level 0: Sorting                [ 81 27 56 98 13 47 26 3 95 78 26 4 57 23 52 8 10 23 96 47 0 ]
Level 1: Sorting left half      [ 81 27 56 98 13 47 26 3 95 78 26 ]
Level 2: Sorting left half      [ 81 27 56 98 13 47 ]
Level 3: Sorting left half      [ 81 27 56 ]
Level 4: Sorting left half      [ 81 27 ]
Level 3: Sorting right half     [ 98 13 47 ]
Level 4: Sorting left half      [ 98 13 ]
Level 2: Sorting right half     [ 26 3 95 78 26 ]
Level 3: Sorting left half      [ 26 3 95 ]
Level 4: Sorting left half      [ 26 3 ]
Level 3: Sorting right half     [ 78 26 ]
Level 1: Sorting right half     [ 4 57 23 52 8 10 23 96 47 0 ]
Level 2: Sorting left half      [ 4 57 23 52 8 ]
Level 3: Sorting left half      [ 4 57 23 ]
Level 4: Sorting left half      [ 4 57 ]
Level 3: Sorting right half     [ 52 8 ]
Level 2: Sorting right half     [ 10 23 96 47 0 ]
Level 3: Sorting left half      [ 10 23 96 ]
Level 4: Sorting left half      [ 10 23 ]
Level 3: Sorting right half     [ 47 0 ]

Final sorted array      [ 0 3 4 8 10 13 23 23 26 26 27 47 47 52 56 57 78 81 95 96 98 ]
```