**Given Code:**

**Sorting**

import java.util.\*;

public class Algorithm6 {

// Function to perform Merge Sort

static void mySort(int[] arr) {

if (arr == null || arr.length <= 1)

return;

int[] temp = new int[arr.length];

myTechnique(arr, temp, 0, arr.length - 1);

}

static void myTechnique(int[] arr, int[] temp, int left, int right) {

if (left < right) {

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

myTechnique(arr, temp, left, mid);

myTechnique(arr, temp, mid + 1, right);

together(arr, temp, left, mid, right);

}

}

static void together(int[] arr, int[] temp, int left, int mid, int right) {

for (int i = left; i <= right; i++) {

temp[i] = arr[i];

}

int i = left;

int j = mid + 1;

int k = left;

while (i <= mid && j <= right) {

if (temp[i] <= temp[j]) {

arr[k++] = temp[i++];

} else {

arr[k++] = temp[j++];

}

}

while (i <= mid) {

arr[k++] = temp[i++];

}

}

public static void main(String[] args) {

int[] arr = {64, 34, 25, 12, 22, 11, 90};

System.out.println("Original array: " + Arrays.toString(arr));

mySort(arr);

System.out.println("Sorted array: " + Arrays.toString(arr));

}

}

**Answer:** The program follows the **Merge Sort**

**Report on Merge Sort Implementation in Java**

**1. Program Overview**

The objective of the program is to sort an array of integers using the Merge Sort algorithm. The program takes an unsorted array as input and outputs a sorted version of that array. The functionality of the program can be summarized as follows:

- Input: An unsorted array of integers.

- Output: A sorted array of integers in ascending order.

**2. Analysis of Constructs and Logic**

The program utilizes several key constructs and logic patterns, including:

- Conditionals: To check if the array is null or contains one or fewer elements

(if (arr == null || arr.length <= 1) return;)

- Recursion: The myTechnique method is recursively called to divide the array into smaller subarrays until each subarray contains a single element.

- Loops: To copy elements to a temporary array and to merge the sorted subarrays back into the original array.

- Merge Pattern: This is a classic example of the Divide and Conquer algorithm, specifically the Merge Sort technique, which involves recursively splitting the array and then merging the sorted halves.

**3. Data Structures Used**

- Array: The primary data structure used is the array. The original array arr is used to store the input and output, and a temporary array temp is used to facilitate the merging process.

- Contribution to Goal: The array structure allows for efficient access and manipulation of elements, which is crucial for implementing the sorting algorithm.

**4. Program Flow and Logic**

1. Initial Call: The mySort function is called with the unsorted array. If the array is empty or has one element, it is already sorted, and the function returns immediately.

2. Recursive Division:

- The myTechnique function is called to recursively divide the array into two halves until subarrays of size one are achieved.

- The midpoint mid is calculated as left + (right - left) / 2**.**

- Recursive calls are made for the left (myTechnique(arr, temp, left, mid)) and

right (myTechnique(arr, temp, mid + 1, right)) halves of the array.

**3. Merge Step:**

- The together function is called to merge the two sorted halves back into the original array.

- Elements from the temporary array temp are compared and merged back into the original array arr in sorted order.

- Remaining elements from either half, if any, are copied to the original array.

**5. Test Input/Output Results**

**Sample Input:**

int[] arr = {64, 34, 25, 12, 22, 11, 90};

**Sample Output:**

Original array: [64, 34, 25, 12, 22, 11, 90]

Sorted array: [11, 12, 22, 25, 34, 64, 90]

**Discussion:**

- The Merge Sort algorithm correctly sorts the array in ascending order.

- The algorithm's time complexity is \(O(n \log n)\), which matches the expected outcome for Merge Sort. This efficiency is achieved by repeatedly dividing the array and merging the sorted subarrays.

**6. Conclusion**

Summary of Findings:

- Algorithm Identified: The program implements the Merge Sort algorithm.

- Constructs Used: Recursion, loops, and conditionals are effectively used to divide the array and merge it back in sorted order.

- Data Structures Employed: Arrays are utilized for both input/output and temporary storage during the merge process.

- Program Flow: The program follows a clear Divide and Conquer approach, splitting the array into smaller parts and merging them in sorted order.

**Challenges and Ambiguities:**

- There were no significant challenges in identifying the algorithm or understanding the program's flow. The code is straightforward and adheres to the standard implementation of Merge Sort.

Overall, the program demonstrates a robust implementation of the Merge Sort algorithm, effectively sorting an array of integers using well-defined recursive and merging techniques.