//Assignment no 12

import java.util.Scanner;

class MergeSort

{

// Merge two subarrays L and M into arr

void merge(int arr[], int p, int q, int r)

{

int n1 = q - p + 1;

int n2 = r - q;

int L[] = new int[n1];

int M[] = new int[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[p + i];

for (int j = 0; j < n2; j++)

M[j] = arr[q + 1 + j];

// Maintain current index of sub-arrays and main array

int i, j, k;

i = 0;

j = 0;

k = p;

// Until we reach either end of either L or M, pick larger among

// elements L and M and place them in the correct position at A[p..r]

while (i < n1 && j < n2)

{

if (L[i] <= M[j])

{

arr[k] = L[i];

i++;

} else

{

arr[k] = M[j];

j++;

}

k++;

}

// When we run out of elements in either L or M,

// pick up the remaining elements and put in A[p..r]

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = M[j];

j++;

k++;

}

}

// Divide the array into two subarrays, sort them and merge them

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// m is the point where the array is divided into two subarrays

int m = (l + r) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Merge the sorted subarrays

merge(arr, l, m, r);

}

}

// Print the array

static void printArray(int arr[])

{

int n = arr.length;

for (int i = 0; i < n; ++i)

System.out.print(arr[i] + " ");

System.out.println();

}

// Driver program

public static void main(String args[]) {

int n;

Scanner sc=new Scanner(System.in);

System.out.print("Enter the number of elements you want to store: ");

//reading the number of elements from the that we want to enter

n=sc.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array: ");

for(int i=0; i<n; i++)

{

//reading array elements from the user

arr[i]=sc.nextInt();

}

System.out.println("Enter Array before sorting");

System.out.println("Array before sorting");

printArray(arr);

MergeSort ob = new MergeSort();

ob.mergeSort(arr, 0, arr.length - 1);

System.out.println("Sorted array:");

printArray(arr);

}

}

output:

gescoe@gescoe-OptiPlex-3010:~/Desktop/SE-A-55$ javac MergeSort.java

gescoe@gescoe-OptiPlex-3010:~/Desktop/SE-A-55$ java MergeSort

Enter the number of elements you want to store: 5

Enter the elements of the array:

6

3

7

2

5

Enter Array before sorting

Array before sorting

6 3 7 2 5

Sorted array:

2 3 5 6 7

[gescoe@gescoe-OptiPlex-3010](mailto:gescoe@gescoe-OptiPlex-3010):~/Desktop/SE-A-55$

### Theory for Merge Sort Code

Merge Sort is a **divide-and-conquer** algorithm that divides an array into smaller subarrays, recursively sorts them, and then merges them back together to produce a sorted array. It is a highly efficient sorting algorithm with a time complexity of O(nlog⁡n)O(n \log n), making it one of the most preferred algorithms for large datasets.

#### ****Merge Sort Explanation:****

1. **Divide**: The array is repeatedly divided into two halves until each subarray contains only one element (which is trivially sorted).
2. **Conquer**: The subarrays are recursively sorted.
3. **Combine**: The sorted subarrays are merged back together to produce a final sorted array.

The merge process works by comparing elements from the two subarrays and placing the smaller element into the original array, ensuring the final array is sorted.

### Code Walkthrough

#### Classes and Methods:

1. MergeSort **Class**:
   * Contains the mergeSort and merge functions that implement the core algorithm.
   * mergeSort: Recursively divides the array into two halves until subarrays contain one element.
   * merge: Merges two sorted subarrays into a single sorted array.
2. merge(int arr[], int p, int q, int r):
   * Merges two subarrays of the array arr[] into a sorted order.
   * The first subarray runs from index p to q, and the second runs from index q+1 to r.
   * It uses two temporary arrays L[] and M[] to hold these subarrays.
   * The elements from L[] and M[] are compared and placed back into the original array arr[].
3. mergeSort(int arr[], int l, int r):
   * This is the recursive function that splits the array into two halves.
   * It then calls itself recursively for both halves and finally calls merge to merge them back together.
4. printArray(int arr[]):
   * A helper function to print the array elements.

#### Steps in Merge Sort:

1. **Divide**: The array is split into two halves by calculating the middle point m = (l + r) / 2 in the mergeSort function.
2. **Recurse**: The function is recursively called on each half, dividing the array further until each subarray has only one element.
3. **Merge**: The merge function is called to merge two sorted subarrays into one sorted array.

### Algorithm:

1. **Base Case**: If l >= r, return (the subarray has only one element).
2. **Divide**: Otherwise, find the middle point m = (l + r) / 2.
3. Recursively sort the left half: mergeSort(arr, l, m).
4. Recursively sort the right half: mergeSort(arr, m + 1, r).
5. **Merge**: Finally, merge the two sorted halves by calling merge(arr, l, m, r).

### Example Execution:

Enter the number of elements you want to store: 5

Enter the elements of the array:

6

3

7

2

5

Enter Array before sorting

Array before sorting

6 3 7 2 5

Sorted array:

2 3 5 6 7

### Code Explanation:

1. The program first asks for the number of elements and the elements themselves.
2. The mergeSort method recursively divides the array into smaller subarrays.
3. The merge method is called to merge sorted subarrays back into a sorted array.
4. The final sorted array is printed.

### Code Summary:

This program implements the **Merge Sort** algorithm, which is an efficient sorting algorithm based on the **divide-and-conquer** strategy. It works by dividing the input array into two halves, recursively sorting the two halves, and merging the sorted halves back together.

#### ****Time Complexity****:

* **Best case**: O(nlog⁡n)O(n \log n)
* **Average case**: O(nlog⁡n)O(n \log n)
* **Worst case**: O(nlog⁡n)O(n \log n)

Merge Sort is stable and works well for large datasets, especially when the data is stored externally (e.g., on disk), because it minimizes data movement.