**ASSIGNMENT-10**

**Question-1**

**Problem Statement:** WAP in Java to design a class SortArray with suitable constructors, data members and member functions to do the following operations:

a) Initialize the array for n elements (to be sorted).

b) To input n numbers into the array (setter method).

c) To show the values stored in the array.

d) To arrange them in ascending order using:

i) Bubble sort

ii) Selection sort

iii) Insertion sort

iv) Merge sort.

v) Quick sort.

**Source Code**

// Class SortArray

import java.util.Scanner;

public class SortArray {

private int[] arr;

private int size;

// Constructor to initialize the array

public SortArray(int n) {

arr = new int[n];

size = n;

}

// Setter method to input n numbers into the array

public void inputNumbers() {

Scanner sc = new Scanner(System.in);

System.out.println("Enter " + size + " numbers:");

for (int i = 0; i < size; i++) {

arr[i] = sc.nextInt();

}

}

// Method to display the values stored in the array

public void display() {

System.out.println("Array elements:");

for (int i = 0; i < size; i++) {

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

}

System.out.println();

}

// Bubble sort

public void bubbleSort() {

for (int i = 0; i < size - 1; i++) {

for (int j = 0; j < size - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

// Swap arr[j] and arr[j+1]

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

// Selection sort

public void selectionSort() {

for (int i = 0; i < size - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < size; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Swap arr[i] and arr[minIndex]

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

}

}

// Insertion sort

public void insertionSort() {

for (int i = 1; i < size; ++i) {

int key = arr[i];

int j = i - 1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key) {

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

j = j - 1;

}

arr[j + 1] = key;

}

}

// Merge sort

public void mergeSort(int[] array, int left, int right) {

if (left < right) {

int middle = (left + right) / 2;

// Sort first and second halves

mergeSort(array, left, middle);

mergeSort(array, middle + 1, right);

// Merge the sorted halves

merge(array, left, middle, right);

}

}

private void merge(int[] array, int left, int middle, int right) {

int n1 = middle - left + 1;

int n2 = right - middle;

// Create temporary arrays

int L[] = new int[n1];

int R[] = new int[n2];

// Copy data to temporary arrays L[] and R[]

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

L[i] = array[left + i];

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

R[j] = array[middle + 1 + j];

// Merge the temporary arrays

int i = 0, j = 0;

// Initial index of merged subarray array

int k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

array[k] = L[i];

i++;

} else {

array[k] = R[j];

j++;

}

k++;

}

// Copy remaining elements of L[] if any

while (i < n1) {

array[k] = L[i];

i++;

k++;

}

// Copy remaining elements of R[] if any

while (j < n2) {

array[k] = R[j];

j++;

k++;

}

}

// Quick sort

public void quickSort(int[] array, int low, int high) {

if (low < high) {

/\* pi is partitioning index, array[pi] is

now at right place \*/

int pi = partition(array, low, high);

// Recursively sort elements before partition and after partition

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

private int partition(int[] array, int low, int high) {

int pivot = array[high];

int i = (low - 1); // Index of smaller element

for (int j = low; j < high; j++) {

// If current element is smaller than the pivot

if (array[j] < pivot) {

i++;

// Swap array[i] and array[j]

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

// Swap array[i+1] and array[high] (or pivot)

int temp = array[i + 1];

array[i + 1] = array[high];

array[high] = temp;

return i + 1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

SortArray sortArray = new SortArray(n);

sortArray.inputNumbers();

// Display unsorted array

System.out.println("Unsorted Array:");

sortArray.display();

System.out.println("Select sorting algorithm:");

System.out.println("1. Bubble Sort");

System.out.println("2. Selection Sort");

System.out.println("3. Insertion Sort");

System.out.println("4. Merge Sort");

System.out.println("5. Quick Sort");

int choice = scanner.nextInt();

switch (choice) {

case 1:

sortArray.bubbleSort();

break;

case 2:

sortArray.selectionSort();

break;

case 3:

sortArray.insertionSort();

break;

case 4:

int[] tempArrayMerge = new int[n];

System.arraycopy(sortArray.arr, 0, tempArrayMerge, 0, n);

sortArray.mergeSort(tempArrayMerge, 0, n - 1);

System.out.println("Array after Merge sort:");

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

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

}

System.out.println();

break;

case 5:

int[] tempArrayQuick = new int[n];

System.arraycopy(sortArray.arr, 0, tempArrayQuick, 0, n);

sortArray.quickSort(tempArrayQuick, 0, n - 1);

System.out.println("Array after Quick sort:");

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

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

}

System.out.println();

break;

default:

System.out.println("Invalid choice!");

}

if (choice >= 1 && choice <= 3) {

System.out.println("Array after sorting:");

sortArray.display();

}

}

}

**OUTPUT:**

Enter the number of elements in the array:

6

Enter 6 numbers:

3

7

4

2

9

5

Unsorted Array:

Array elements:

3 7 4 2 9 5

Select sorting algorithm:

1. Bubble Sort

2. Selection Sort

3. Insertion Sort

4. Merge Sort

5. Quick Sort

3

Array after sorting:

Array elements:

2 3 4 5 7 9