package com.iimtiaz.day\_04;  
  
import java.util.Arrays;  
import java.util.List;  
  
public class SortColors {  
 public static void main(String[] args) {  
 int[] nums = {2, 0, 2, 1, 1, 0};  
 Solution\_1 solution\_1 = new Solution\_1();  
 solution\_1.sortColors(nums);  
 System.*out*.println(Arrays.*toString*(Arrays.*stream*(nums).toArray()));  
 }  
}  
  
*/\*\*  
 \* Time Complexity: O(n)  
 \* O(1): Initializing variables low, mid, and high.  
 \* O(n): The while loop iterates at most n times, as mid moves towards high in each iteration.  
 \* O(1): Operations within the loop (comparisons, swaps) are constant time.  
  
 \* Space Complexity: O(1)  
 \* Uses only a few fixed-size variables, regardless of the input array size.  
 \* No additional data structures are created, ensuring constant space usage.  
 \*/*class Solution\_1 {  
 public void sortColors(int[] array) {  
 int low = 0, mid = 0, high = array.length - 1;  
 while (mid <= high) {  
 if (array[mid] == 0) {  
 swap(array, low, mid);  
 low++;  
 mid++;  
 } else if (array[mid] == 1) {  
 mid++;  
 } else {  
 swap(array, mid, high);  
 high--;  
 }  
 }  
 }  
  
 private void swap(int[] array, int i, int j) {  
 int temp = array[i];  
 array[i] = array[j];  
 array[j] = temp;  
 }  
}  
  
*/\*\*  
 \* Time Complexity: O(n)  
 \* O(n): The first loop iterates through the entire array, counting occurrences of each color (0, 1, 2).  
 \* O(n): The three subsequent loops also iterate through the array, overwriting elements with their respective colors.  
 \* O(1): Other operations (initialization, counting, assignments) are constant time.  
  
 \* Space Complexity: O(1)  
 \* Uses only three integer variables (c0, c1, c2) to store color counts, regardless of input size.  
 \* No additional data structures are created, maintaining constant space usage.  
 \*/*class Solution\_2 {  
 public void sortColors(int[] nums) {  
 int c0 = 0;  
 int c1 = 0;  
 int c2 = 0;  
 for (int i = 0; i < nums.length; i++) {  
 if (nums[i] == 0) c0++;  
 else if (nums[i] == 1) c1++;  
 else if (nums[i] == 2) c2++;  
 }  
 for (int i = 0; i < c0; i++) {  
 nums[i] = 0;  
 }  
 for (int i = c0; i < c0 + c1; i++) {  
 nums[i] = 1;  
 }  
 for (int i = c0 + c1; i < nums.length; i++) {  
 nums[i] = 2;  
 }  
 }  
}  
  
*/\*\*  
 \* Time Complexity: O(n^2)  
 \* O(n): The outer loop iterates through the array n times.  
 \* O(n): The inner loop iterates through the remaining elements of the array for each outer loop iteration, resulting  
 \* in a total of approximately n^2 / 2 comparisons.  
 \* O(1): Operations within the loops (comparisons, swaps) are constant time.  
  
 \* Space Complexity: O(1)  
 \* Uses a few fixed-size variables (n, temp), irrespective of the input array size.  
 \* No additional data structures are created, ensuring constant space usage.  
 \*/*class Solution\_3 {  
 public void sortColors(int[] nums) {  
 int n = nums.length;  
 for (int i = 0; i < n; i++) {  
 for (int j = 0; j < n - 1; j++) {  
 if (nums[i] < nums[j]) {  
 int temp = nums[j];  
 nums[j] = nums[i];  
 nums[i] = temp;  
 }  
 }  
 }  
 }  
}