**Exercise 1: Find the sum of all elements in an array.**

public class SumArray {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int sum = 0;

for (int num : arr) {

sum += num;

}

System.out.println("Sum of array elements: " + sum);

}

}

**Exercise 2: Find the maximum element in an array.**

public class FindMax {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 7};

int max = arr[0];

for (int num : arr) {

if (num > max) {

max = num;

}

}

System.out.println("Maximum element in the array: " + max);

}

}

**Exercise 3: Find the minimum element in an array.**

public class FindMin {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 7};

int min = arr[0];

for (int num : arr) {

if (num < min) {

min = num;

}

}

System.out.println("Minimum element in the array: " + min);

}

}

**Exercise 4: Calculate the average of elements in an array.**

public class CalculateAverage {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int sum = 0;

for (int num : arr) {

sum += num;

}

double average = (double) sum / arr.length;

System.out.println("Average of array elements: " + average);

}

}

**Exercise 5: Check if an element exists in an array.**

public class ContainsElement {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int target = 3;

boolean found = false;

for (int num : arr) {

if (num == target) {

found = true;

break;

}

}

if (found) {

System.out.println(target + " exists in the array.");

} else {

System.out.println(target + " does not exist in the array.");

}

}

}

**Exercise 6: Reverse an array.**

public class ReverseArray {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int left = 0;

int right = arr.length - 1;

while (left < right) {

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

System.out.print("Reversed array: ");

for (int num : arr) {

System.out.print(num + " ");

}

}

}

**Exercise 7: Count the occurrences of a specific element in an array.**

public class CountOccurrences {

public static void main(String[] args) {

int[] arr = {1, 2, 2, 3, 4, 2, 5};

int target = 2;

int count = 0;

for (int num : arr) {

if (num == target) {

count++;

}

}

System.out.println(target + " occurs " + count + " times in the array.");

}

}

**Exercise 8: Find the second largest element in an array**.

public class FindSecondLargest {

public static void main(String[] args) {

int[] arr = {5, 2, 9, 1, 7};

int max = Integer.MIN\_VALUE;

int secondMax = Integer.MIN\_VALUE;

for (int num : arr) {

if (num > max) {

secondMax = max;

max = num;

} else if (num > secondMax && num != max) {

secondMax = num;

}

}

System.out.println("Second largest element in the array: " + secondMax);

}

}

**Exercise 9: Rotate an array to the right by a given number of positions.**

public class RotateArray {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int positions = 2;

positions %= arr.length; // In case positions is larger than the array size

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

for (int i = 0; i < arr.length; i++) {

result[(i + positions) % arr.length] = arr[i];

}

System.out.print("Array after rotating right by " + positions + " positions: ");

for (int num : result) {

System.out.print(num + " ");

}

}

}

**Exercise 10: Find the common elements between two arrays.**

public class CommonElements {

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4, 5};

int[] arr2 = {3, 4, 5, 6, 7};

System.out.print("Common elements between the two arrays: ");

for (int num1 : arr1) {

for (int num2 : arr2) {

if (num1 == num2) {

System.out.print(num1 + " ");

break; // Break the inner loop once a common element is found

}

}

}

}

}

**Exercise 11: Find the index of the first occurrence of a specific element in an array.**

public class FindIndexOfElement {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int target = 3;

int index = -1;

for (int i = 0; i < arr.length; i++) {

if (arr[i] == target) {

index = i;

break;

}

}

if (index != -1) {

System.out.println("Index of " + target + " in the array: " + index);

} else {

System.out.println(target + " is not found in the array.");

}

}

}

**Exercise 12: Find the index of the last occurrence of a specific element in an array.**

public class FindLastIndexOfElement {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 2, 5};

int target = 2;

int index = -1;

for (int i = arr.length - 1; i >= 0; i--) {

if (arr[i] == target) {

index = i;

break;

}

}

if (index != -1) {

System.out.println("Last index of " + target + " in the array: " + index);

} else {

System.out.println(target + " is not found in the array.");

}

}

}

**Exercise 13: Find the equilibrium index of an array (an index where the sum of elements at lower indexes is equal to the sum of elements at higher indexes).**

public class EquilibriumIndex {

public static void main(String[] args) {

int[] arr = {-7, 1, 5, 2, -4, 3, 0};

int totalSum = 0;

int leftSum = 0;

int equilibriumIndex = -1;

for (int num : arr) {

totalSum += num;

}

for (int i = 0; i < arr.length; i++) {

totalSum -= arr[i];

if (leftSum == totalSum) {

equilibriumIndex = i;

break;

}

leftSum += arr[i];

}

if (equilibriumIndex != -1) {

System.out.println("Equilibrium index: " + equilibriumIndex);

} else {

System.out.println("No equilibrium index found.");

}

}

}

**Exercise 14: Remove duplicates from a sorted array in-place (without using additional data structures).**

public class RemoveDuplicatesFromSortedArray {

public static void main(String[] args) {

int[] arr = {1, 1, 2, 2, 2, 3, 4, 4, 5};

int length = arr.length;

if (length <= 1) {

System.out.println("Array with no duplicates: " + Arrays.toString(arr));

return;

}

int currentIndex = 0;

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

if (arr[i] != arr[currentIndex]) {

currentIndex++;

arr[currentIndex] = arr[i];

}

}

int[] result = Arrays.copyOfRange(arr, 0, currentIndex + 1);

System.out.println("Array with duplicates removed: " + Arrays.toString(result));

}

}

**Exercise 15: Check if an array is sorted in ascending order.**

public class IsSortedAscending {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

boolean sorted = true;

for (int i = 1; i < arr.length; i++) {

if (arr[i] < arr[i - 1]) {

sorted = false;

break;

}

}

if (sorted) {

System.out.println("The array is sorted in ascending order.");

} else {

System.out.println("The array is not sorted in ascending order.");

}

}

}

**Exercise 16: Check if an array is sorted in descending order.**

public class IsSortedDescending {

public static void main(String[] args) {

int[] arr = {5, 4, 3, 2, 1};

boolean sorted = true;

for (int i = 1; i < arr.length; i++) {

if (arr[i] > arr[i - 1]) {

sorted = false;

break;

}

}

if (sorted) {

System.out.println("The array is sorted in descending order.");

} else {

System.out.println("The array is not sorted in descending order.");

}

}

}

**Exercise 17: Find the product of all elements in an array.**

public class ProductOfArrayElements {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int product = 1;

for (int num : arr) {

product \*= num;

}

System.out.println("Product of array elements: " + product);

}

}

**Exercise 18: Check if an array contains only even numbers.**

public class ContainsOnlyEven {

public static void main(String[] args) {

int[] arr = {2, 4, 6, 8, 10};

boolean containsOnlyEven = true;

for (int num : arr) {

if (num % 2 != 0) {

containsOnlyEven = false;

break;

}

}

if (containsOnlyEven) {

System.out.println("The array contains only even numbers.");

} else {

System.out.println("The array contains at least one odd number.");

}

}

}

**Exercise 19: Check if an array contains only prime numbers.**

public class ContainsOnlyPrimes {

public static void main(String[] args) {

int[] arr = {2, 3, 5, 7, 11};

boolean containsOnlyPrimes = true;

for (int num : arr) {

if (!isPrime(num)) {

containsOnlyPrimes = false;

break;

}

}

if (containsOnlyPrimes) {

System.out.println("The array contains only prime numbers.");

} else {

System.out.println("The array contains at least one non-prime number.");

}

}

private static boolean isPrime(int num) {

if (num <= 1) {

return false;

}

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

}

**Exercise 20: Find the kth smallest element in an array.**

import java.util.Arrays;

public class KthSmallestElement {

public static void main(String[] args) {

int[] arr = {3, 1, 4, 2, 5};

int k = 3;

Arrays.sort(arr);

if (k >= 1 && k <= arr.length) {

System.out.println("The " + k + "th smallest element in the array: " + arr[k - 1]);

} else {

System.out.println("Invalid value of k.");

}

}

}

**Exercise 21: Find the median of an array.**

import java.util.Arrays;

public class FindMedian {

public static void main(String[] args) {

int[] arr = {3, 1, 4, 2, 5};

Arrays.sort(arr);

double median;

if (arr.length % 2 == 0) {

int middle1 = arr.length / 2 - 1;

int middle2 = arr.length / 2;

median = (arr[middle1] + arr[middle2]) / 2.0;

} else {

int middle = arr.length / 2;

median = arr[middle];

}

System.out.println("Median of the array: " + median);

}

}

**Exercise 22: Find the subarray with the maximum sum, allowing for an empty subarray (Kadane's algorithm with a twist).**

public class MaxSumSubarrayWithEmptySubarray {

public static void main(String[] args) {

int[] arr = {1, 2, -5, 4, 2, -1, 6};

int maxSum = 0;

int currentSum = 0;

for (int num : arr) {

currentSum = Math.max(0, currentSum + num);

maxSum = Math.max(maxSum, currentSum);

}

System.out.println("Maximum sum of a subarray (allowing empty subarray): " + maxSum);

}

}

**Exercise 23: Check if an array is a palindrome (reads the same forwards and backwards).**

public class IsPalindromeArray {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 2, 1};

boolean isPalindrome = true;

int left = 0;

int right = arr.length - 1;

while (left < right) {

if (arr[left] != arr[right]) {

isPalindrome = false;

break;

}

left++;

right--;

}

if (isPalindrome) {

System.out.println("The array is a palindrome.");

} else {

System.out.println("The array is not a palindrome.");

}

}

}

**Exercise 24: Find the longest subarray with a specific sum.**

public class LongestSubarrayWithSum {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5, 6};

int targetSum = 9;

int maxLength = 0;

int currentSum = 0;

int start = 0;

for (int end = 0; end < arr.length; end++) {

currentSum += arr[end];

while (currentSum > targetSum) {

currentSum -= arr[start];

start++;

}

if (currentSum == targetSum) {

maxLength = Math.max(maxLength, end - start + 1);

}

}

System.out.println("Length of the longest subarray with sum " + targetSum + ": " + maxLength);

}

}

**Exercise 25: Merge two sorted arrays into a single sorted array.**

public class MergeSortedArrays {

public static void main(String[] args) {

int[] arr1 = {1, 3, 5, 7};

int[] arr2 = {2, 4, 6};

int[] merged = new int[arr1.length + arr2.length];

int i = 0, j = 0, k = 0;

while (i < arr1.length && j < arr2.length) {

if (arr1[i] < arr2[j]) {

merged[k++] = arr1[i++];

} else {

merged[k++] = arr2[j++];

}

}

while (i < arr1.length) {

merged[k++] = arr1[i++];

}

while (j < arr2.length) {

merged[k++] = arr2[j++];

}

System.out.print("Merged and sorted array: ");

for (int num : merged) {

System.out.print(num + " ");

}

}

}

**Exercise 26: Find the longest increasing subarray within an array.**

public class LongestIncreasingSubarray {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 1, 2, 3, 4, 5};

int maxLength = 1;

int currentLength = 1;

for (int i = 1; i < arr.length; i++) {

if (arr[i] > arr[i - 1]) {

currentLength++;

} else {

maxLength = Math.max(maxLength, currentLength);

currentLength = 1;

}

}

maxLength = Math.max(maxLength, currentLength);

System.out.println("Length of the longest increasing subarray: " + maxLength);

}

}

**Exercise 27: Find the missing number in an array containing consecutive integers.**

public class FindMissingNumber {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 5, 6};

int n = arr.length + 1;

int expectedSum = (n \* (n + 1)) / 2;

int actualSum = 0;

for (int num : arr) {

actualSum += num;

}

int missingNumber = expectedSum - actualSum;

System.out.println("Missing number in the array: " + missingNumber);

}

}

**Exercise 28: Rotate a square matrix (2D array) 90 degrees clockwise.**

public class RotateMatrix {

public static void main(String[] args) {

int[][] matrix = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int n = matrix.length;

int[][] rotatedMatrix = new int[n][n];

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

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

rotatedMatrix[i][j] = matrix[n - j - 1][i];

}

}

System.out.println("Rotated matrix 90 degrees clockwise:");

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

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

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

}

System.out.println();

}

}

}

**Exercise 29: Check if two arrays are equal (contain the same elements in the same order).**

public class AreArraysEqual {

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4, 5};

int[] arr2 = {1, 2, 3, 4, 5};

boolean areEqual = true;

if (arr1.length != arr2.length) {

areEqual = false;

} else {

for (int i = 0; i < arr1.length; i++) {

if (arr1[i] != arr2[i]) {

areEqual = false;

break;

}

}

}

if (areEqual) {

System.out.println("The arrays are equal.");

} else {

System.out.println("The arrays are not equal.");

}

}

}

**Exercise 30: Find the longest subarray with a sum less than or equal to a given target sum.**

public class LongestSubarraySum {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int targetSum = 8;

int maxLength = 0;

int currentSum = 0;

int start = 0;

for (int end = 0; end < arr.length; end++) {

currentSum += arr[end];

while (currentSum > targetSum) {

currentSum -= arr[start];

start++;

}

maxLength = Math.max(maxLength, end - start + 1);

}

System.out.println("Length of the longest subarray with sum <= " + targetSum + ": " + maxLength);

}

}

**Exercise 31: Find the second smallest element in an array.**

public class FindSecondSmallest {

public static void main(String[] args) {

int[] arr = {3, 1, 4, 2, 5};

int smallest = Integer.MAX\_VALUE;

int secondSmallest = Integer.MAX\_VALUE;

for (int num : arr) {

if (num < smallest) {

secondSmallest = smallest;

smallest = num;

} else if (num < secondSmallest && num != smallest) {

secondSmallest = num;

}

}

System.out.println("Second smallest element in the array: " + secondSmallest);

}

}

**Exercise 32: Find the longest subarray with equal number of even and odd elements.**

public class LongestSubarrayWithEqualEvenOdd {

public static void main(String[] args) {

int[] arr = {2, 2, 3, 4, 1, 5, 7, 8};

int maxLength = 0;

int currentCount = 0;

int[] oddEvenCounts = new int[2];

int left = 0;

for (int right = 0; right < arr.length; right++) {

if (arr[right] % 2 == 0) {

oddEvenCounts[0]++;

} else {

oddEvenCounts[1]++;

}

if (oddEvenCounts[0] == oddEvenCounts[1]) {

maxLength = Math.max(maxLength, right - left + 1);

}

while (oddEvenCounts[0] < oddEvenCounts[1]) {

if (arr[left] % 2 == 0) {

oddEvenCounts[0]--;

} else {

oddEvenCounts[1]--;

}

left++;

}

}

System.out.println("Length of the longest subarray with equal even and odd elements: " + maxLength);

}

}

**Exercise 33: Find the maximum subarray sum using Kadane's algorithm.**

public class MaximumSubarraySum {

public static void main(String[] args) {

int[] arr = {-2, 1, -3, 4, -1, 2, 1, -5, 4};

int maxSum = arr[0];

int currentSum = arr[0];

for (int i = 1; i < arr.length; i++) {

currentSum = Math.max(arr[i], currentSum + arr[i]);

maxSum = Math.max(maxSum, currentSum);

}

System.out.println("Maximum subarray sum: " + maxSum);

}

}

**Exercise 34: Find the smallest missing positive integer in an array.**

public class SmallestMissingPositive {

public static void main(String[] args) {

int[] arr = {3, 4, -1, 1};

int n = arr.length;

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

while (arr[i] > 0 && arr[i] <= n && arr[i] != arr[arr[i] - 1]) {

int temp = arr[i];

arr[i] = arr[temp - 1];

arr[temp - 1] = temp;

}

}

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

if (arr[i] != i + 1) {

System.out.println("Smallest missing positive integer: " + (i + 1));

return;

}

}

System.out.println("Smallest missing positive integer: " + (n + 1));

}

}

**Exercise 35: Find the majority element (element that appears more than n/2 times) in an array.**

public class MajorityElement {

public static void main(String[] args) {

int[] arr = {2, 2, 1, 1, 1, 2, 2};

int majority = arr[0];

int count = 1;

for (int i = 1; i < arr.length; i++) {

if (arr[i] == majority) {

count++;

} else {

count--;

if (count == 0) {

majority = arr[i];

count = 1;

}

}

}

System.out.println("Majority element in the array: " + majority);

}

}

**Exercise 36: Find the longest subarray with even sum.**

public class LongestSubarrayWithEvenSum {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int maxLength = 0;

int currentSum = 0;

int left = 0;

for (int right = 0; right < arr.length; right++) {

currentSum += arr[right];

if (currentSum % 2 == 0) {

maxLength = Math.max(maxLength, right - left + 1);

}

while (currentSum % 2 != 0) {

currentSum -= arr[left];

left++;

}

}

System.out.println("Length of the longest subarray with even sum: " + maxLength);

}

}

**Exercise 37: Find the longest subarray with odd sum.**

public class LongestSubarrayWithOddSum {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int maxLength = 0;

int currentSum = 0;

int left = 0;

for (int right = 0; right < arr.length; right++) {

currentSum += arr[right];

if (currentSum % 2 != 0) {

maxLength = Math.max(maxLength, right - left + 1);

}

while (currentSum % 2 == 0) {

currentSum -= arr[left];

left++;

}

}

System.out.println("Length of the longest subarray with odd sum: " + maxLength);

}

}

**Exercise 38: Find the maximum product of any two elements in an array.**

public class MaxProductOfTwoElements {

public static void main(String[] args) {

int[] arr = {3, 1, 4, 2, 5};

int maxProduct = Integer.MIN\_VALUE;

for (int i = 0; i < arr.length; i++) {

for (int j = i + 1; j < arr.length; j++) {

int product = arr[i] \* arr[j];

maxProduct = Math.max(maxProduct, product);

}

}

System.out.println("Maximum product of two elements in the array: " + maxProduct);

}

}

**Exercise 39: Reverse an array in-place (without using additional data structures).**

public class ReverseArrayInPlace {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int left = 0;

int right = arr.length - 1;

while (left < right) {

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

System.out.print("Reversed array: ");

for (int num : arr) {

System.out.print(num + " ");

}

}

}

**Exercise 40: Find the longest increasing subarray with contiguous elements**.

public class LongestIncreasingContiguousSubarray {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 1, 2, 3, 4, 5};

int maxLength = 1;

int currentLength = 1;

int start = 0;

for (int i = 1; i < arr.length; i++) {

if (arr[i] == arr[i - 1] + 1) {

currentLength++;

} else {

start = i;

currentLength = 1;

}

maxLength = Math.max(maxLength, currentLength);

}

System.out.println("Length of the longest increasing contiguous subarray: " + maxLength);

}

}

**Exercise 41: Check if an array is a rotation of another array.**

public class IsRotation {

public static void main(String[] args) {

int[] arr1 = {1, 2, 3, 4, 5};

int[] arr2 = {3, 4, 5, 1, 2};

if (arr1.length != arr2.length) {

System.out.println("Arrays are not rotations of each other.");

} else {

int n = arr1.length;

boolean isRotation = false;

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

boolean foundRotation = true;

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

if (arr1[j] != arr2[(i + j) % n]) {

foundRotation = false;

break;

}

}

if (foundRotation) {

isRotation = true;

break;

}

}

if (isRotation) {

System.out.println("Arrays are rotations of each other.");

} else {

System.out.println("Arrays are not rotations of each other.");

}

}

}

}

**Exercise 42: Find the smallest subarray with a sum greater than or equal to a given target sum.**

public class SmallestSubarrayWithSum {

public static void main(String[] args) {

int[] arr = {1, 4, 45, 6, 0, 19};

int targetSum = 51;

int minLength = Integer.MAX\_VALUE;

int currentSum = 0;

int start = 0;

for (int end = 0; end < arr.length; end++) {

currentSum += arr[end];

while (currentSum >= targetSum) {

minLength = Math.min(minLength, end - start + 1);

currentSum -= arr[start];

start++;

}

}

if (minLength == Integer.MAX\_VALUE) {

System.out.println("No subarray found.");

} else {

System.out.println("Length of the smallest subarray with sum >= " + targetSum + ": " + minLength);

}

}

}

**Exercise 43: Rotate an array to the right by k steps.**

public class RotateArrayRightWithoutMethod {

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

int k = 3;

k = k % arr.length; // Adjust k if it's larger than the array length

int n = arr.length;

// Reverse the entire array

for (int i = 0; i < n / 2; i++) {

int temp = arr[i];

arr[i] = arr[n - 1 - i];

arr[n - 1 - i] = temp;

}

// Reverse the first k elements

for (int i = 0; i < k / 2; i++) {

int temp = arr[i];

arr[i] = arr[k - 1 - i];

arr[k - 1 - i] = temp;

}

// Reverse the remaining elements

for (int i = k; i < (k + n) / 2; i++) {

int temp = arr[i];

arr[i] = arr[n - 1 - (i - k)];

arr[n - 1 - (i - k)] = temp;

}

System.out.print("Rotated array: ");

for (int num : arr) {

System.out.print(num + " ");

}

}

}

**Exercise 44: Find the maximum sum subarray of a circular array.**

public class MaxSumCircularSubarray {

public static void main(String[] args) {

int[] arr = {10, -3, -4, 7, 6, 5, -4, -1};

int maxKadane = kadane(arr);

int maxWrap = 0;

int totalSum = 0;

for (int i = 0; i < arr.length; i++) {

totalSum += arr[i];

arr[i] = -arr[i];

}

maxWrap = totalSum + kadane(arr);

int result = (maxWrap > maxKadane) ? maxWrap : maxKadane;

System.out.println("Maximum sum of circular subarray: " + result);

}

private static int kadane(int[] arr) {

int maxSum = arr[0];

int currentSum = arr[0];

for (int i = 1; i < arr.length; i++) {

currentSum = Math.max(arr[i], currentSum + arr[i]);

maxSum = Math.max(maxSum, currentSum);

}

return maxSum;

}

}

**Exercise 45: Find the kth largest element in an unsorted array (QuickSelect algorithm) without using methods:**

public class KthLargestElementWithoutMethods {

public static void main(String[] args) {

int[] arr = {3, 2, 1, 5, 6, 4};

int k = 2;

int left = 0;

int right = arr.length - 1;

k = arr.length - k; // Convert to find the kth smallest element

while (left < right) {

int pivotIndex = right;

int i = left - 1;

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

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

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

int temp = arr[i + 1];

arr[i + 1] = arr[pivotIndex];

arr[pivotIndex] = temp;

pivotIndex = i + 1;

if (pivotIndex == k) {

int kthLargest = arr[pivotIndex];

System.out.println("The " + (k + 1) + "th largest element is: " + kthLargest);

return;

} else if (pivotIndex < k) {

left = pivotIndex + 1;

} else {

right = pivotIndex - 1;

}

}

int kthLargest = arr[left];

System.out.println("The " + (k + 1) + "th largest element is: " + kthLargest);

}

}

**Exercise 46: Implement an ArrayList from scratch without using methods.**

public class MyArrayListWithoutMethods {

public static void main(String[] args) {

Object[] array = new Object[10];

int size = 0;

// Add elements

array[size++] = 1;

array[size++] = 2;

array[size++] = 3;

// Get an element

int element = (int) array[1];

System.out.println("Element at index 1: " + element);

// Remove an element (by shifting elements to the left)

int indexToRemove = 1;

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

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

}

size--;

// Print the ArrayList

System.out.print("ArrayList: [");

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

System.out.print(array[i]);

if (i < size - 1) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

**Exercise 47: Find the common elements in three sorted arrays without using methods.**

public class CommonElementsInSortedArraysWithoutMethods {

public static void main(String[] args) {

int[] arr1 = {1, 5, 10, 20, 40, 80};

int[] arr2 = {6, 7, 20, 80, 100};

int[] arr3 = {3, 4, 15, 20, 30, 70, 80, 120};

int i = 0, j = 0, k = 0;

System.out.print("Common elements: ");

while (i < arr1.length && j < arr2.length && k < arr3.length) {

if (arr1[i] == arr2[j] && arr2[j] == arr3[k]) {

System.out.print(arr1[i] + " ");

i++;

j++;

k++;

} else if (arr1[i] < arr2[j]) {

i++;

} else if (arr2[j] < arr3[k]) {

j++;

} else {

k++;

}

}

}

}

**Exercise 48: Merge two sorted arrays without using methods.**

public class MergeSortedArraysWithoutMethods {

public static void main(String[] args) {

int[] arr1 = {1, 3, 5, 7};

int[] arr2 = {2, 4, 6, 8, 9};

int[] mergedArray = new int[arr1.length + arr2.length];

int i = 0, j = 0, k = 0;

while (i < arr1.length && j < arr2.length) {

if (arr1[i] < arr2[j]) {

mergedArray[k++] = arr1[i++];

} else {

mergedArray[k++] = arr2[j++];

}

}

while (i < arr1.length) {

mergedArray[k++] = arr1[i++];

}

while (j < arr2.length) {

mergedArray[k++] = arr2[j++];

}

System.out.print("Merged array: [");

for (int m = 0; m < mergedArray.length; m++) {

System.out.print(mergedArray[m]);

if (m < mergedArray.length - 1) {

System.out.print(", ");

}

}

System.out.println("]");

}

}

**Exercise 49: Find the majority element in an array without using methods.**

public class MajorityElementWithoutMethods {

public static void main(String[] args) {

int[] arr = {2, 2, 3, 5, 2, 2, 6};

int candidate = -1;

int count = 0;

for (int num : arr) {

if (count == 0) {

candidate = num;

count = 1;

} else if (num == candidate) {

count++;

} else {

count--;

}

}

count = 0;

for (int num : arr) {

if (num == candidate) {

count++;

}

}

if (count > arr.length / 2) {

System.out.println("Majority element is: " + candidate);

} else {

System.out.println("No majority element found.");

}

}

}

**Exercise 50: Search in a rotated sorted array without using methods.**

public class SearchInRotatedSortedArrayWithoutMethods {

public static void main(String[] args) {

int[] arr = {4, 5, 6, 7, 0, 1, 2};

int target = 0;

int left = 0;

int right = arr.length - 1;

while (left <= right) {

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

if (arr[mid] == target) {

System.out.println("Target " + target + " found at index: " + mid);

return;

}

if (arr[left] <= arr[mid]) {

if (target >= arr[left] && target < arr[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

} else {

if (target > arr[mid] && target <= arr[right]) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

System.out.println("Target " + target + " not found in the array.");

}

}