**Assignment Divide and Conquer**

Q1. Given an array where all its elements are sorted in increasing order except two swapped

elements, sort it in linear time. Assume there are no duplicates in the array.

Input: arr[] = [3, 8, 6, 7, 5, 9, 10]

Output: arr[] = [3, 5, 6, 7, 8, 9, 10]

public class FixSwappedElements {

public void sortArray(int[] arr) {

int n = arr.length;

int first = -1, second = -1;

// Find the first element that is out of order

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

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

first = i;

break;

}

}

// Find the second element that is out of order

for (int i = n - 1; i > 0; i--) {

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

second = i;

break;

}

}

// Swap the two elements

if (first != -1 && second != -1) {

int temp = arr[first];

arr[first] = arr[second];

arr[second] = temp;

}

}

public static void main(String[] args) {

FixSwappedElements solution = new FixSwappedElements();

int[] arr = {3, 8, 6, 7, 5, 9, 10};

solution.sortArray(arr);

for (int num : arr) {

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

}

// Output should be: 3 5 6 7 8 9 10

}

}

Q2. Given an array of positive and negative integers, segregate them in linear time and constant

space. The output should print all negative numbers, followed by all positive numbers.

Input: arr[] = {19, -20, 7, -4, -13, 11, -5, 3}

Output: arr[] = {-20 ,-4, -13, -5, 19 ,11 ,3, 7}

public class SegregatePositiveNegative {

public void segregate(int[] arr) {

int i = 0;

int j = arr.length - 1;

while (i <= j) {

// Increment i until you find a positive number

while (i <= j && arr[i] < 0) {

i++;

}

// Decrement j until you find a negative number

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

j--;

}

// Swap the positive number at i with the negative number at j

if (i < j) {

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

i++;

j--;

}

}

}

public static void main(String[] args) {

SegregatePositiveNegative solution = new SegregatePositiveNegative();

int[] arr = {19, -20, 7, -4, -13, 11, -5, 3};

solution.segregate(arr);

for (int num : arr) {

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

}

// Output should be: -20 -4 -13 -5 19 11 7 3

}

}

Q3. Given an array of positive and negative integers, segregate them in linear time and constant

space. The output should print all negative numbers, followed by all positive numbers. The

relative order of elements must remain the same.

Input: arr[] = {19, -20, 7, -4, -13, 11, -5, 3}

Input: arr[] = {-20 ,-4, -13, -5, 19 ,7 ,11, 3}

import java.util.Arrays;

public class StableSegregation {

public void segregate(int[] arr) {

int n = arr.length;

int j = 0; // Index to place the next negative number

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

if (arr[i] < 0) {

// If a negative number is found, place it at the j-th index

int temp = arr[i];

// Shift positive numbers to the right

int k = i;

while (k > j) {

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

k--;

}

arr[j] = temp;

j++;

}

}

}

public static void main(String[] args) {

StableSegregation solution = new StableSegregation();

int[] arr = {19, -20, 7, -4, -13, 11, -5, 3};

solution.segregate(arr);

System.out.println(Arrays.toString(arr));

// Output should be: [-20, -4, -13, -5, 19, 7, 11, 3]

}

}

Q4. Given two arrays of equal size n and an integer k. The task is to permute both arrays such

that the sum of their corresponding element is greater than or equal to k i.e a[i] + b[i] >= k. The

task is to print “Yes” if any such permutation exists, otherwise print “No”.

input: a[] = {2, 1, 3},

b[] = { 7, 8, 9 },

k = 10.

Output: Yes

input: a[] = {1, 2, 2, 1},

b[] = { 3, 3, 3, 4 },

k = 5.

Output: No

import java.util.Arrays;

public class ArrayPermutation {

public static String canPermuteToSatisfyCondition(int[] a, int[] b, int k) {

// Sort array `a` in ascending order

Arrays.sort(a);

// Sort array `b` in descending order

Arrays.sort(b);

for (int i = 0; i < b.length / 2; i++) {

int temp = b[i];

b[i] = b[b.length - 1 - i];

b[b.length - 1 - i] = temp;

}

// Check if all pairs satisfy the condition

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

if (a[i] + b[i] < k) {

return "No";

}

}

return "Yes";

}

public static void main(String[] args) {

int[] a1 = {2, 1, 3};

int[] b1 = {7, 8, 9};

int k1 = 10;

System.out.println(canPermuteToSatisfyCondition(a1, b1, k1)); // Output: Yes

int[] a2 = {1, 2, 2, 1};

int[] b2 = {3, 3, 3, 4};

int k2 = 5;

System.out.println(canPermuteToSatisfyCondition(a2, b2, k2)); // Output: No

}

}

Q5. An interval is represented as a combination of start time and end time. Given a set of

intervals, check if any two intervals intersect.

Input:

arr[] = {{1, 3}, {5, 7}, {2, 4}, {6, 8}}

output: Yes

The intervals {1, 3} and {2, 4} overlap

Input:

arr[] = {{1, 3}, {7, 9}, {4, 6}, {10, 13}}

output:No

import java.util.Arrays;

import java.util.Comparator;

public class IntervalIntersection {

public static boolean doIntervalsIntersect(int[][] intervals) {

// Sort intervals based on the start time

Arrays.sort(intervals, Comparator.comparingInt(interval -> interval[0]));

// Check for intersections

for (int i = 0; i < intervals.length - 1; i++) {

// If the end time of the current interval is greater than the start time of the next interval

if (intervals[i][1] > intervals[i + 1][0]) {

return true;

}

}

return false;

}

public static void main(String[] args) {

int[][] intervals1 = {{1, 3}, {5, 7}, {2, 4}, {6, 8}};

int[][] intervals2 = {{1, 3}, {7, 9}, {4, 6}, {10, 13}};

System.out.println(doIntervalsIntersect(intervals1) ? "Yes" : "No"); // Output: Yes

System.out.println(doIntervalsIntersect(intervals2) ? "Yes" : "No"); // Output: No

}

}