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]

Solution :
```java
import java.util.*;

class Main {
    public static void sortArr(int[] arr, int n) {
        if (n <= 1) {
            return;
        }

        int x = -1;
        int y = -1;
        int prev = arr[0];

        for (int i = 1; i < n; i++) {
            if (prev > arr[i]) {

                if (x == -1) {
                    x = i - 1;
                    y = i;
                }
                else {
                    y = i;
                }
            }
                prev = arr[i];
        }

        //swapping those two elements
        int temp = arr[x];
        arr[x] = arr[y];
        arr[y] = temp;
    }

    public static void main(String[] args) {

        int[] arr = {1, 2, 4, 9, 8, 7, 12};

        int n = arr.length;

        sortArr(arr, n);

        for(int i=0;i<n;i++){
            System.out.print(arr[i] + " ");
        }
    }
}
```

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}

Solutions

```
import java.util.*;
public class Main{
    public static void partition(int[] arr, int s, int e){
        int pIndex= s;
        for (int j = s; j <= e; j++) {
            if (arr[j] < 0){    // pivot is 0
                int temp = arr[j];
                arr[j] = arr[pIndex];
                arr[pIndex] = temp;
                pIndex++;
            }
        }
    }
    public static void main(String[] args){
        int[] arr = { 9, 20, -7, -4, 13, -11, -15, 3 };
        int n = arr.length;
        partition(arr, 0, n - 1);
        for (int i : arr) {
              System.out.print(i + " ");
        }
    }
}
```

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}

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

```
Solution :
```

```java
import java.util.*;
public class Main{
    public static void partition(int[] num, int[] temp, int l, int h){
        if (h <= l) {
            return;
        }
        // find midpoint
        int mid = (l + ((h - l) >> 1));

        partition(num, temp, l, mid);          // split/merge left half
        partition(num, temp, mid + 1, h);     // split/merge right half

        merge(num, temp , l, mid, h);        // join the two half runs
        }

        public static void merge(int[] num, int[] temp, int l, int mid, int
h){
            int k = l;
            // copy negative elements from the left subarray
            for (int i = l; i <= mid; i++) {
                    if (num[i] < 0) {
                            temp[k++] = num[i];
                    }
            }
            // copy negative elements from the right subarray
            for (int j = mid + 1; j <= h; j++) {
                    if (num[j] < 0) {
                            temp[k++] = num[j];
                    }
            }
            // copy positive elements from the left subarray
            for (int i = l; i <= mid; i++) {
                    if (num[i] >= 0) {
                            temp[k++] = num[i];
                    }
            }
            // copy positive elements from the right subarray
            for (int j = mid + 1; j <= h; j++) {
                    if (num[j] >= 0) {
                            temp[k++] = num[j];
                    }
            }
            // copy back to the original array to reflect the relative order
            for (int i = l; i <= h; i++) {
                    num[i] = temp[i];
            }
        }
        public static void main(String[] args){
            int[] num = { 19, -20, 7, -4, -13 , 11 , -5 , 3};
            int n = num.length;
            int[] temp = new int[n];
            for (int i = 0; i < n; i++) {
                    temp[i] = num[i];
            }
            partition(num, temp, 0, n - 1);
            for (int i = 0; i < n; i++) {
                    System.out.print(num[i] + " ");
            }
        }
}
```

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},

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

k = 10.

Yes

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

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

k = 5.

No

Solution :

```
import java.util.*;

class Main{
    static boolean isPossible(Integer a[], int b[], int n, int k){
        // Sort the array a[] in decreasing order.
        Arrays.sort(a, Collections.reverseOrder());

        // Sort the array b[] in increasing order.
        Arrays.sort(b);

        // Checking condition on each index.
        for (int i = 0; i < n; i++)
        if (a[i] + b[i] < k)
            return false;

        return true;
    }
    public static void main(String[] args) {
        Integer a[] = {2, 1, 3, 5, 1};
        int b[] = {7, 8, 9 , 2 , 6};
        int k = 10;
        int n = a.length;

        if (isPossible(a, b, n, k))
        System.out.print("Yes");
        else
        System.out.print("No");
    }
}
```

Q5.Anintervalisrepresentedasacombinationofstarttimeandendtime.Givenasetofintervals,

checkifanytwointervalsintersect.

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

Solution :

```
class Main{

    // An interval has start time and end time
    static class Interval{
        int start;
        int end;
        public Interval(int start, int end){
            super();
            this.start = start;
            this.end = end;
        }
    };

    static boolean isIntersect(Interval arr[], int n){

        int max_ele = 0;

        // Find the overall maximum element
        for (int i = 0; i < n; i++)
        {
            if (max_ele < arr[i].end)
                max_ele = arr[i].end;
        }
        // Initialize an array of size max_ele
        int []aux = new int[max_ele + 1];
        for (int i = 0; i < n; i++)
        {

            // starting point of the interval
            int x = arr[i].start;

            // end point of the interval
            int y = arr[i].end;
            aux[x]++;
            aux[y ]--;
        }
        for (int i = 1; i <= max_ele; i++)
        {
            // Calculating the prefix Sum
            aux[i] += aux[i - 1];

            // Overlap
            if (aux[i] > 1)
                return true;
        }

        // If we reach here, then no Overlap
        return false;
    }

    public static void main(String[] args){
        Interval arr1[] = { new Interval(1, 3), new Interval(7, 9), new
Interval(4, 6), new Interval(10, 13) };
        int n1 = arr1.length;

        if(isIntersect(arr1, n1))
            System.out.print("Yes\n");
        else
            System.out.print("No\n");
        }
    }
```