1. Code

```
package medianv10;

import java.util.Arrays;
import java.util.Scanner;

public class Medianv10 {

    private static float median = 0;
    private static int xlength = 0;
    private static int ylength = 0;
    private static int[] x;
    private static int[] y;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter x array length: ");
        xlength = sc.nextInt();
        x = new int[xlength];
        System.out.print("Enter x array elements: ");
        for (int i = 0; i < xlength; i++) {
            x[i] = sc.nextInt();
        }
        System.out.print("Enter y array length: ");
        ylength = sc.nextInt();
        y = new int[ylength];
        System.out.print("Enter y array elements: ");
```

```
    for (int i = 0; i < ylength; i++) {
        y[i] = sc.nextInt();
    }
    Arrays.sort(x);
    Arrays.sort(y);
    if (xlength == ylength) {
        findMedianOfSameSizeSortedArrays();
    } else {
        findMedianOfDifferentSizeSortedArrays();
    }
    System.out.println(median);
}

public static void findMedianOfDifferentSizeSortedArrays() {
    int start = 0;
    int end = xlength - 1;
    boolean medianNotFound = true;
    int partitionx, partitiony;
    int[] leftx, lefty, rightx, righty;
    while (medianNotFound) {
        partitionx = (start + end) / 2;
        partitiony = (xlength + ylength + 1) / 2 - partitionx;
        if (partitionx == 0) {
            if (partitiony == ylength) {
                partitionx = 1;
                leftx = new int[partitionx];
                lefty = new int[partitiony];
                rightx = new int[xlength];
                righty = new int[1];
```

```
            leftx[0] = -5000;
            System.arraycopy(y, 0, lefty, 0, partitiony);
            System.arraycopy(x, 0, rightx, 0, xlength);
            righty[0] = 5000;
        } else {
            partitionx = 1;
            leftx = new int[partitionx];
            lefty = new int[partitiony];
            rightx = new int[xlength];
            righty = new int[ylength - partitiony];
            leftx[0] = -5000;
            System.arraycopy(y, 0, lefty, 0, partitiony);
            System.arraycopy(x, 0, rightx, 0, xlength);
            System.arraycopy(y, partitiony, righty, 0, ylength - partitiony);
        }
    } else if (partitiony == 0) {
        if (partitionx == xlength) {
            partitiony = 1;
            leftx = new int[partitionx];
            lefty = new int[partitiony];
            rightx = new int[1];
            righty = new int[ylength];
            System.arraycopy(x, 0, leftx, 0, xlength);
            lefty[0] = -5000;
            rightx[0] = 5000;
            System.arraycopy(y, 0, righty, 0, ylength);
        } else {
            partitiony = 1;
            leftx = new int[partitionx];
```

```
        lefty = new int[partitiony];
        rightx = new int[xlength - partitionx];
        righty = new int[ylength];
        System.arraycopy(x, 0, leftx, 0, partitionx);
        lefty[0] = -5000;
        System.arraycopy(x, partitionx, rightx, 0, xlength - partitionx);
        System.arraycopy(y, 0, righty, 0, ylength);
    }
} else if (partitionx == xlength) {
    if (partitiony == 0) {
        partitiony = 1;
        leftx = new int[partitionx];
        lefty = new int[partitiony];
        rightx = new int[1];
        righty = new int[ylength];
        System.arraycopy(x, 0, leftx, 0, xlength);
        lefty[0] = -5000;
        rightx[0] = 5000;
        System.arraycopy(y, 0, righty, 0, ylength);
    } else {
        leftx = new int[partitionx];
        lefty = new int[partitiony];
        rightx = new int[1];
        righty = new int[ylength - partitiony];
        System.arraycopy(x, 0, leftx, 0, xlength);
        System.arraycopy(y, 0, lefty, 0, partitionx);
        rightx[0] = 5000;
        System.arraycopy(y, partitiony, righty, 0, ylength - partitiony);
    }
```

```
} else if (partitiony == ylength) {
    if (partitionx == 0) {
        partitionx = 1;
        leftx = new int[partitionx];
        lefty = new int[partitiony];
        rightx = new int[xlength];
        righty = new int[1];
        leftx[0] = -5000;
        System.arraycopy(y, 0, lefty, 0, partitiony);
        System.arraycopy(x, 0, rightx, 0, xlength);
        righty[0] = 5000;
    } else {
        leftx = new int[partitionx];
        lefty = new int[partitiony];
        rightx = new int[xlength - partitionx];
        righty = new int[1];
        System.arraycopy(x, 0, leftx, 0, partitionx);
        System.arraycopy(y, 0, lefty, 0, ylength);
        System.arraycopy(x, partitionx, rightx, 0, xlength - partitionx);
        righty[0] = 5000;
    }
} else {
    leftx = new int[partitionx];
    lefty = new int[partitiony];
    rightx = new int[xlength - partitionx];
    righty = new int[ylength - partitiony];
    System.arraycopy(x, 0, leftx, 0, partitionx);
    System.arraycopy(y, 0, lefty, 0, partitiony);
    System.arraycopy(x, partitionx, rightx, 0, xlength - partitionx);
```

```
            System.arraycopy(y, partitiony, righty, 0, ylength - partitiony);
        }
        if (leftx[partitionx - 1] <= righty[0] && lefty[partitiony - 1] <= rightx[0]) {
            if ((xlength + ylength) % 2 == 1) {
                median = (leftx[partitionx - 1] > lefty[partitiony - 1]) ? leftx[partitionx - 1]
                        : lefty[partitiony - 1];
                medianNotFound = false;
            } else {
                int max = (leftx[partitionx - 1] > lefty[partitiony - 1]) ? leftx[partitionx - 1]
                        : lefty[partitiony - 1];
                int min = (rightx[0] > righty[0]) ? righty[0] : rightx[0];
                median = (max + min) / 2.0f;
                medianNotFound = false;
            }
        } else {
            if (leftx[partitionx - 1] > righty[0]) {
                end = partitionx - 1;
            } else {
                start = partitionx + 1;
            }
        }
    }
}

public static float findMedianOfSameSizeSortedArrays() {
    float median1, median2;
    if(x.length == 2){
        int max = (x[0] > y[0])? x[0] : y[0];
        int min = (x[1] > y[1])? y[1] : x[1];
```

```
    median = (max + min) / 2;
    return median;
} else {
if (x.length % 2 == 1) {
    median1 = x[x.length / 2];
} else {
    median1 = (x[x.length / 2] + x[x.length / 2 - 1]) / 2.0f;
}
if (y.length % 2 == 1) {
    median2 = y[y.length / 2];
} else {
    median2 = (y[y.length / 2] + y[y.length / 2 - 1]) / 2.0f;
}

if (median1 == median2) {
    return median1;

} else if (median1 > median2) {
    int x1[] = new int[x.length];
    System.arraycopy(x, 0, x1, 0, x.length);
    if (x.length % 2 == 1) {
        x = new int[x.length / 2 + 1];
    } else {
        x = new int[x.length / 2];
    }
    System.arraycopy(x1, 0, x, 0, x.length);
    int y1[] = new int[y.length];
    System.arraycopy(y, 0, y1, 0, y.length);
    if (y.length % 2 == 1) {
```

```
            y = new int[y.length / 2 + 1];
            if(y1.length == 3){
                System.arraycopy(y1, y1.length / 2, y, 0, y.length);
            } else {
                System.arraycopy(y1, y1.length / 2 + 1, y, 0, y.length);
            }
        } else {
            y = new int[y.length / 2];
            System.arraycopy(y1, y1.length / 2, y, 0, y.length);
        }
        return findMedianOfSameSizeSortedArrays();
    } else {
        int x1[] = new int[x.length];
        System.arraycopy(x, 0, x1, 0, x.length);
        if (x.length % 2 == 1) {
            x = new int[x.length / 2 + 1];
            if(x1.length == 3){
                System.arraycopy(x1, x.length / 2, x, 0, x.length);
            } else {
                System.arraycopy(x1, x.length / 2 + 1, x, 0, x.length);
            }
        } else {
            x = new int[x.length / 2];
            System.arraycopy(x1, x1.length / 2, x, 0, x.length);
        }

        int y1[] = new int[y.length];
        System.arraycopy(y, 0, y1, 0, y.length);
        if (y.length % 2 == 1) {
```

```
            y = new int[y.length / 2 + 1];
        } else {
            y = new int[y.length / 2];
        }
        System.arraycopy(y1, 0, y, 0, y.length);
        return findMedianOfSameSizeSortedArrays();
    }
    }
  }
}
```

2. Test Cases

i) Arrays of same size:

a) Enter x array length: 4

Enter x array elements: 1 2 3 6

Enter y array length: 4

Enter y array elements: 4 6 8 10

5.0

b) Enter x array length: 5

Enter x array elements: 1 15 12 26 38

Enter y array length: 5

Enter y array elements: 13 2 17 48 30

16.0

ii) Arrays of different sizes:

a) Enter x array length: 5

Enter x array elements: 1 3 8 9 15

Enter y array length: 6

Enter y array elements: 7 11 18 19 21 25

11.0

b) Enter x array length: 4

Enter x array elements: 35 26 31 23

Enter y array length: 6

Enter y array elements: 3 5 9 7 16 11

13.5

3. Code Algorithm

First, the algorithm relies on the arrays being sorted before starting. That's taken care of by invoking the sort method after entering the two arrays from the input.

Second, the arrays sizes will be checked. If they are the same and their common size equals 2, then by finding the maximum of the first two elements in the two arrays and also the minimum of the second two elements in the two arrays, adding the maximum and minimum and dividing by 2 we get the median that we're looking for. If their common size isn't 2, then we should perform the following:

1) Calculate the medians m1 and m2 of the input arrays ar1[] and ar2[] respectively.

2) If m1 and m2 both are equal then we are done. Return m1 (or m2)

3) If m1 is greater than m2, then median is present in one of the below two subarrays.

   a) From first element of ar1 to m1 (ar1[0...|_n/2_|])

   b) From m2 to last element of ar2 (ar2[|_n/2_|...n-1])

4) If m2 is greater than m1, then median is present in one of the below two subarrays.

   a) From m1 to last element of ar1 (ar1[|_n/2_|...n-1])

   b) From first element of ar2 to m2 (ar2[0...|_n/2_|])

5) Repeat the above process until size of both the subarrays becomes 2.

If the arrays sizes aren't equal, then we should make 2 partitions: left of 1st array and 2nd array, right of 1st array and 2nd array. The condition that should be checked when performing the partitioning is that both partitions should be equal in size (or have a unity difference in case of a total odd number of the two arrays sizes combined). To proceed further, we need to make sure that the maximum of the left of the 1st array is less than the minimum of the right of the 2nd array, and the maximum of the left of the 2nd array is less than the minimum of the right of the 1st array. This is done to ensure that every element on the left partition is less than any element on the right partition. Doing so, we can calculate the median by getting the maximum on the left side and the minimum on the right side, add and divide by 2 and we should get the correct median. However, in case of an odd number of adding the arrays sizes, we can calculate the median by choosing the maximum of the left partition only.

In the end, any of these two functions (same size, different sizes) return the median to display in the console.

References

1) https://www.geeksforgeeks.org/median-of-two-sorted-arrays/