MIU java

Sample Qualifying Exam for Direct Entry Track

Java Basic Program

```
class Main {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Basic Java Program with Class and function

```
class Main {
    public static void hello(String message) {
        System.out.println(message);
    }

    public static void main(String[] args) {
        String message = "Hello world!";
        hello(message);
    }
}
```

MIU Array Related Questions and Solutions

Question 1 :

1. **An array with an odd number of elements is said to be centered if all elements (except the middle one) are strictly greater than the value of the middle element. Note that only arrays with an odd number of elements have a middle element. Write a function that accepts an integer array and returns 1 if it is a centered array, otherwise it returns 0.**

An array is said to be centered if

- length of array must be odd and grater than 1
- middle element of an array must be less than other elements

Examples:

if the input array is	**return**
{1, 2, 3, 4, 5}	0 (the middle element 3 is not strictly less than all other elements)
{3, 2, 1, 4, 5}	1 (the middle element 1 is strictly less than all other elements)
{3, 2, 1, 4, 1}	0 (the middle element 1 is not strictly less than all other elements)
{1, 2, 3, 4}	0 (no middle element)
{}	0 (no middle element)
{10}	1 (the middle element 10 is strictly less than all other elements)

```
public class isCentered {
    public static int isCenteredArray(int[] array){
        int size = array.length;
        if (size == 0 || size % 2 == 0){
            return 0;

        }
        int mid = size / 2;
        for (int i = 0; i < size; i++){
            if (i != mid && array[i] <= array[mid]){
                return 0;
            }
        }
        return 1;

    }
    public static void main(String[] args) {
        // int[] array = {1, 2, 3, 4, 5};
        // int[] array = {3, 2, 1, 4, 5};
        // int[] array = {3, 2, 1, 4, 1};
        // int[] array = {1, 2, 3, 4};
        // int[] array = {};
        int[] array = {10};
```

```
        int isCentered = isCenteredArray(array);
        System.out.println(isCentered);
    }
}
```

Question 2:

Write a function that takes an array of integers as an argument and returns a value based on the sums of the even and odd numbers in the array. Let X = the sum of the odd numbers in the array and let Y = the sum of the even numbers. The function should return X – Y

The signature of the function is: **int f(int[ ] a)**

Examples

if input array is	**return**
{1}	1
{1, 2}	-1
{1, 2, 3}	2
{1, 2, 3, 4}	-2
{3, 3, 4, 4}	-2
{3, 2, 3, 4}	0
{4, 1, 2, 3}	-2
{1, 1}	2
{}	0

```
public class sumEvenDiffOdd {
    public static int sumEvenDiffOddNumber(int[] array){
        int sumEven = 0;
        int sumOdd = 0;
        for (int arrayElem:array){
            if (arrayElem % 2 == 0){
                sumEven += arrayElem;
            }else{
                sumOdd += arrayElem;
            }
        }
        return sumEven - sumOdd;
    }

    public static void main(String[] args) {
```

```
        // int[] array = {1};
        // int[] array = {1, 2, 3};
        // int[] array = {1, 2};
        // int[] array = {1, 2, 3, 4};
        // int[] array = {3, 3, 4, 4};
        // int[] array = {3, 2, 3, 4};
        // int[] array = {4, 1, 2, 3};
        // int[] array = {1, 1};
        int[] array = {};
        int sumEvenDiffOdd = sumEvenDiffOddNumber(array);
        System.out.println(sumEvenDiffOdd);
    }
}
```

Question 3:

Write a function that accepts a character array, a zero-based start position and a length. It should return a character array containing containing *length*characters starting with the *start*character of the input array. The function should do error checking on the start position and the length and return null if the either value is not legal.

The function signature is:**charf(chara, int start, int len)**

Examples

if input parameters are	**return**
{‘a’, ‘b’, ‘c’}, 0, 4	null
{‘a’, ‘b’, ‘c’}, 0, 3	{‘a’, ‘b’, ‘c’}
{‘a’, ‘b’, ‘c’}, 0, 2	{‘a’, ‘b’}
{‘a’, ‘b’, ‘c’}, 0, 1	{‘a’}
{‘a’, ‘b’, ‘c’}, 1, 3	null
{‘a’, ‘b’, ‘c’}, 1, 2	{‘b’, ‘c’}
{‘a’, ‘b’, ‘c’}, 1, 1	{‘b’}
{‘a’, ‘b’, ‘c’}, 2, 2	null
{‘a’, ‘b’, ‘c’}, 2, 1	{‘c’}
{‘a’, ‘b’, ‘c’}, 3, 1	null
{‘a’, ‘b’, ‘c’}, 1, 0	{}
{‘a’, ‘b’, ‘c’}, -1, 2	null
{‘a’, ‘b’, ‘c’}, -1, -2	null

{}, 0, 1	null

```
public class charFindingStartLen {
    public static char[] stringHandling(char[] charArray,int start, int len){
        if ( start <0 || start >= charArray.length){
            return null;
        }
        if (len < 0 || start + len > charArray.length){
            return null;
        }
        char[] result = new char[len];
        for (int i = 0; i < len; i++){
            result[i] = charArray[start+i];
        }
        return result;
    }
    public static void arrayPrint(char[] array) {
        System.out.print("{");
        for (int i = 0; i < array.length; i++) {
            if (i > 0) {
                System.out.print(", ");
            }
            System.out.print("'" + array[i] + "'");
        }
        System.out.println("}");
    }
    public static void main(String[] args) {
        // char[] charArray ={'a', 'b', 'c'};
        // int start = 0;
        // int len = 4;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 0;
        // int len = 3;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 0;
        // int len = 2;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 0;
        // int len = 1;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 1;
        // int len = 3;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 1;
        // int len = 2;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 1;
        // int len = 1;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 2;
        // int len = 2;
        // char[] charArray= {'a', 'b', 'c'};
```

```
        // int start= 2;
        // int len = 1;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 3;
        // int len = 1;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= 1;
        // int len = 0;
        // char[] charArray= {'a', 'b', 'c'};
        // int start= -1;
        // int len = 2
        // char[] charArray= {'a', 'b', 'c'};
        // int start= -1;
        // int len = -2
        char[] charArray= {};
        int start = 0;
        int len = 1;

        char[] array = stringHandling(charArray,start,len);
        if ( array != null){
            arrayPrint(array);

        }else{
            System.out.println("null");
        }
    }
}
```

Question 4:

Write a function to reverse an integer using numeric operators and without using any arrays or other data structures.

The signature of the function is:**int f(int n)**

Examples

if the input integer is	**return**
1234	4321
12005	50021
1	1
1000	1
0	0
-12345	-54321

```
public class reverseNumber {
    public static int reverseNum(int number) {
        int reversedNumber = 0;
        while (number != 0) {
            int digit = number % 10;
            reversedNumber = reversedNumber * 10 + digit;
            number /= 10;
        }
        return reversedNumber;
    }

    public static void main(String[] args) {
        // int number = 12324;
        // int number = 12005;
        // int number = 1;
        // int number = 1000;
        // int number = 0;
        int number = -12345;
        int reversed = reverseNum(number);
        System.out.println(reversed);
    }
}
```

Question 5:

Write a function to return an array containing all elements common to two given arrays containing distinct positive integers. You should not use any inbuilt methods. You are allowed to use any number of arrays.

The signature of the function is:**int[] f(int[] first, int[] second)**

Examples

if input parameters are	**return**
{1, 8, 3, 2}, {4, 2, 6, 1}	{1, 2}
{1, 8, 3, 2, 6}, {2, 6, 1}	{2, 6, 1}
{1, 3, 7, 9}, {7, 1, 9, 3}	{1, 3, 7, 9}
{1, 2}, {3, 4}	{}
{}, {1, 2, 3}	{}
{1, 2}, {}	{}
{1, 2}, null	null
null, {}	null

null, null	null

```
public class commonArrayElement {
    public static void arrayPrint(int[] array) {
        System.out.print("{");
        for (int i = 0; i < array.length; i++) {
            if (i > 0) {
                System.out.print(", ");
            }
            System.out.print(array[i]);
        }
        System.out.println("}");
    }

    public static int[] commonElement(int[] first, int[] second) {
        if ( first == null || second == null){
            return null;
        }else{

            int[] commonElem = new int[Math.min(first.length, second.length)];
            int resultIndex = 0;

            for (int i = 0; i < first.length; i++) {
                for (int j = 0; j < second.length; j++) {
                    if (first[i] == second[j]) {
                        commonElem[resultIndex] = first[i];
                        resultIndex++;
                        break;
                    }
                }
            }
            int[] commonElemArray = new int[resultIndex];
            System.arraycopy(commonElem, 0, commonElemArray, 0, resultIndex);
            return commonElemArray;
        }
    }

    public static void main(String[] args) {
        // int[] first = {1, 8, 3, 2};
        // int[] second = {4, 2, 6, 1};
        // int[] first = {1, 8, 3, 2, 6};
        // int[] second = {2, 6, 1};
        // int[] first = {1, 3, 7, 9};
        // int[] second = {7, 1, 9, 3};
        // int[] first = {1, 2};
        // int[] second = {3, 4};
        // int[] first = {};
        // int[] second = {1, 2, 3};
        // int[] first = {1, 2};
        // int[] second = {};
        // int[] first = {1, 2};
        // int[] second = null;
```

```
        // int[] first = null;
        // int[] second = {};
        int[] first = null;
        int[] second = null;

        int[] commonElem = commonElement(first, second);
        if (commonElem == null){
            System.out.println("null");
        }else{
            arrayPrint(commonElem);
        }
    }
}
```

Question 6:

Consider an array A with n of positive integers. An integer idx is called a POE (point of equilibrium) of A, if A[0] + A[1] + … + A[idx – 1] is equal to A[idx + 1] + A[idx + 2] + … + A[n – 1]. Write a function to return POE of an array, if it exists and -1 otherwise.

The signature of the function is:**int f(int[] a)**

Examples

if input arrays are	**return**
{1, 8, 3, 7, 10, 2}	3 Reason: a[0] + a[1] + a[2] is equal to a[4] + a[5]
{1, 5, 3, 1, 1, 1, 1, 1, 1}	2 Reason: a[0] + a[1] is equal to a[3] + a[4] + a[5] + a[6] + a[7] + a[8]
{2, 1, 1, 1, 2, 1, 7}	5 Reason: a[0] + a[1] + a[2] + a[3] + a[4] is equal to a[6]
{1, 2, 3}	-1 Reason: No POE.
{3, 4, 5, 10}	-1 Reason: No POE.
{1, 2, 10, 3, 4}	-1 Reason: No POE.

```
public class pointOfEquilibrium {

    public static int pointOfEqui(int[] array) {
        int len = array.length;
        if ( len < 3){
            return -1;
        }
        int totalSum = 0;
        for (int arrayElem:array){
            totalSum += arrayElem;
        }
```

```
        int leftSum = 0;
        for (int idx = 0; idx < len; idx++){
            int rightSum = totalSum -leftSum -array[idx];

            if(leftSum == rightSum){
                return idx;
            }
            leftSum += array[idx];
        }
        return -1;
    }

    public static void main(String[] args) {
        // int[] array = {1, 8, 3, 7, 10, 2};
        // int[] array = {1, 5, 3, 1, 1, 1, 1, 1, 1};
        // int[] array = {2, 1, 1, 1, 2, 1, 7};
        // int[] array = {1, 2, 3};
        // int[] array = {3, 4, 5, 10};
        int[] array = {1, 2, 10, 3, 4};
        int idx = pointOfEqui(array);
        System.out.println(idx);
    }
}
```

Question 7:

Smallest / Second Smallest / Largest / Second Largest

```
public class arraySorting {
    public static int[] arraySortAscending(int[] array) { // For Ascending Order
        int n = array.length;
        for (int i = 1; i < n; i++) {
            for (int j = i; j > 0; j--) {
                if (array[j] < array[j - 1]) { // For Ascending Order
                    // Swap elements if they are out of order
                    int temp = array[j];
                    array[j] = array[j - 1];
                    array[j - 1] = temp;
                } else {
                    break;
                }
            }
        }
        return array;
    }

    public static int[] arraySortDescending(int[] array) { // for Descending Order
        int n = array.length;
        for (int i = 1; i < n; i++) {
```

```
            for (int j = i; j > 0; j--) {
                if (array[j] > array[j - 1]) { // For Descending Order
                    // Swap elements if they are out of order
                    int temp = array[j];
                    array[j] = array[j - 1];
                    array[j - 1] = temp;
                } else {
                    break;
                }
            }
        }
        return array;
    }

    public static void arrayPrint(int[] array) {
        System.out.print("{");
        for (int i = 0; i < array.length; i++) {
            if (i > 0) {
                System.out.print(", ");
            }
            System.out.print(array[i]);
        }
        System.out.println("}");
    }

    public static void main(String[] args) {
        int[] array = { 5, 2, 9, 1, 5, 6, 8, 3, -3, 9 };

        // Call the methods with arguments for ascending order
        System.out.println("Array in Ascending Order :");
        int[] sortedArrayAsc = arraySortAscending(array);
        arrayPrint(sortedArrayAsc);
        // smallest Element in Array
        System.out.println(sortedArrayAsc[0]);
        // Second smallest Element in Array
        System.out.println(sortedArrayAsc[2]);

        // Call the methods with arguments for descending order
        System.out.println("Array in Descending Order :");
        int[] sortedArrayDes = arraySortDescending(array);
        arrayPrint(sortedArrayDes);
        // Largest Element in Array
        System.out.println(sortedArrayDes[0]);
        // Second Largest Element in Array
        System.out.println(sortedArrayDes[2]);
    }
}
```

Question 8:

Find:-

- Sum of Even number
- Sum of odd number
- Number of even numbers
- Number of odd numbers
- Even number array
- Odd number array
- Positive number array
- Negative number array
- Sum(Even numbers) - Sum(Odd numbers)
- Sum(Positive numbers) - Sum(Negative number)

```
public class evenOddInArray {
    public static void arrayPrint(int[] array) {
        System.out.print("{");
        for (int i = 0; i < array.length; i++) {
            if (i > 0) {
                System.out.print(", ");
            }
            System.out.print(array[i]);
        }
        System.out.println("}");
    }

    public static int[] evenOddSum(int[] array){
        int evenSum = 0;
        int oddSum = 0;
        int[] evenOddSumArray = {0, 0};
        for (int arrayElem : array) {
            if (arrayElem % 2 == 0) {
                evenSum += arrayElem;
            } else {
                oddSum += arrayElem;
            }
        }
        evenOddSumArray[0] = evenSum;
        evenOddSumArray[1] = oddSum;
        return evenOddSumArray;
    }

    public static int[] evenOddCount(int[] array){
        int evenCount = 0;
```

```
        int oddCount = 0;
        int[] evenOddCountArray = {0, 0};
        for (int arrayElem : array) {
            if (arrayElem % 2 == 0) {
                evenCount += 1;
            } else {
                oddCount += 1;
            }
        }
        evenOddCountArray[0] = evenCount;
        evenOddCountArray[1] = oddCount;
        return evenOddCountArray;
    }

    public static int[][] evenOdd(int[] array) {
        // Count the number of even and odd elements
        int evenCounts = 0;
        int oddCounts = 0;

        for (int arrayElem : array) {
            if (arrayElem % 2 == 0) {
                evenCounts++;
            } else {
                oddCounts++;
            }
        }
        // 2 rows for even and odd array
        int[][] evenOddArray = new int[2][];

        evenOddArray[0] = new int[evenCounts];
        evenOddArray[1] = new int[oddCounts];

        int evenIndex = 0;
        int oddIndex = 0;

        // Populate even and odd arrays
        for (int arrayElem:array) {
            if (arrayElem % 2 == 0) {
                evenOddArray[0][evenIndex] = arrayElem;
                evenIndex++;
            } else {
                evenOddArray[1][oddIndex] = arrayElem;
                oddIndex++;
            }
        }
        return evenOddArray;
    }
    public static int evenDiffOdd(int[] array){
        int evenSum = 0;
        int oddSum = 0;
        for(int arrayElem:array){
            if (arrayElem % 2 == 0){
                evenSum += arrayElem;
            } else {
```

```
                oddSum += arrayElem;
            }
        }
        return (evenSum - oddSum);
    }

    public static int[][] posNeg(int[] array) {
        // Count the number of even and odd elements
        int posCounts = 0;
        int negCounts = 0;

        for (int arrayElem : array) {
            if (arrayElem > 0) {
                posCounts++;
            } else {
                negCounts++;
            }
        }
        // 2 rows for even and odd array
        int[][] posNegArray = new int[2][];

        posNegArray[0] = new int[posCounts];
        posNegArray[1] = new int[negCounts];

        int posIndex = 0;
        int negIndex = 0;

        // Populate even and odd arrays
        for (int arrayElem:array) {
            if (arrayElem > 0) {
                posNegArray[0][posIndex] = arrayElem;
                posIndex++;
            } else {
                posNegArray[1][negIndex] = arrayElem;
                negIndex++;
            }
        }
        return posNegArray;
    }


    public static int posDiffNeg(int[] array){
        int posSum = 0;
        int negSum = 0;
        for(int arrayElem:array){
            if (arrayElem > 0){
                posSum += arrayElem;
            } else {
                negSum += arrayElem;
            }
        }
        return (posSum - negSum);
    }
    public static void main(String[] args) {
```

```
        int[] array = { 5, 2, 9, 1, 5, 6, 8, 7, -3, 9, -5 };

        int[] evenOddSumArray = evenOddSum(array);
        System.out.println("Sum of Even Number: " + evenOddSumArray[0]);
        System.out.println("Sum of Odd Number: " + evenOddSumArray[1]);

        int[] evenOddCountArray = evenOddCount(array);
        System.out.println("Number  of Even Number: " + evenOddCountArray[0]);
        System.out.println("Number of Odd Number: " + evenOddCountArray[1]);

        int[][] evenOddArray = evenOdd(array);

        int[] evenArray = evenOddArray[0];
        System.out.print("Even Array: ");
        arrayPrint(evenArray);

        int[] oddArray = evenOddArray[1];
        System.out.print("Odd Array: ");
        arrayPrint(oddArray);

        int diff = evenDiffOdd(array);
    System.out.println("Sum(even numbers) - Sum(odd numbers): " + diff);

        int[][] posNegArray = posNeg(array);

        int[] posArray = posNegArray[0];
        System.out.print("Positive Array: ");
        arrayPrint(posArray);

        int[] negArray = posNegArray[1];
        System.out.print("Negative Array: ");
        arrayPrint(negArray);

        int diff1 = posDiffNeg(array);
    System.out.println("Sum(positive numbers) - Sum(negatve numbers): " + diff1);
    }
}
```

Question 9:

Write a function to remove the duplicate form a given array of integer

Function signature: int[] f(int[] a)

Example:

if input arrays are	**return**
{1, 2, 3, 4, 5,6,7}	{1, 2, 3, 4, 5,6,7}

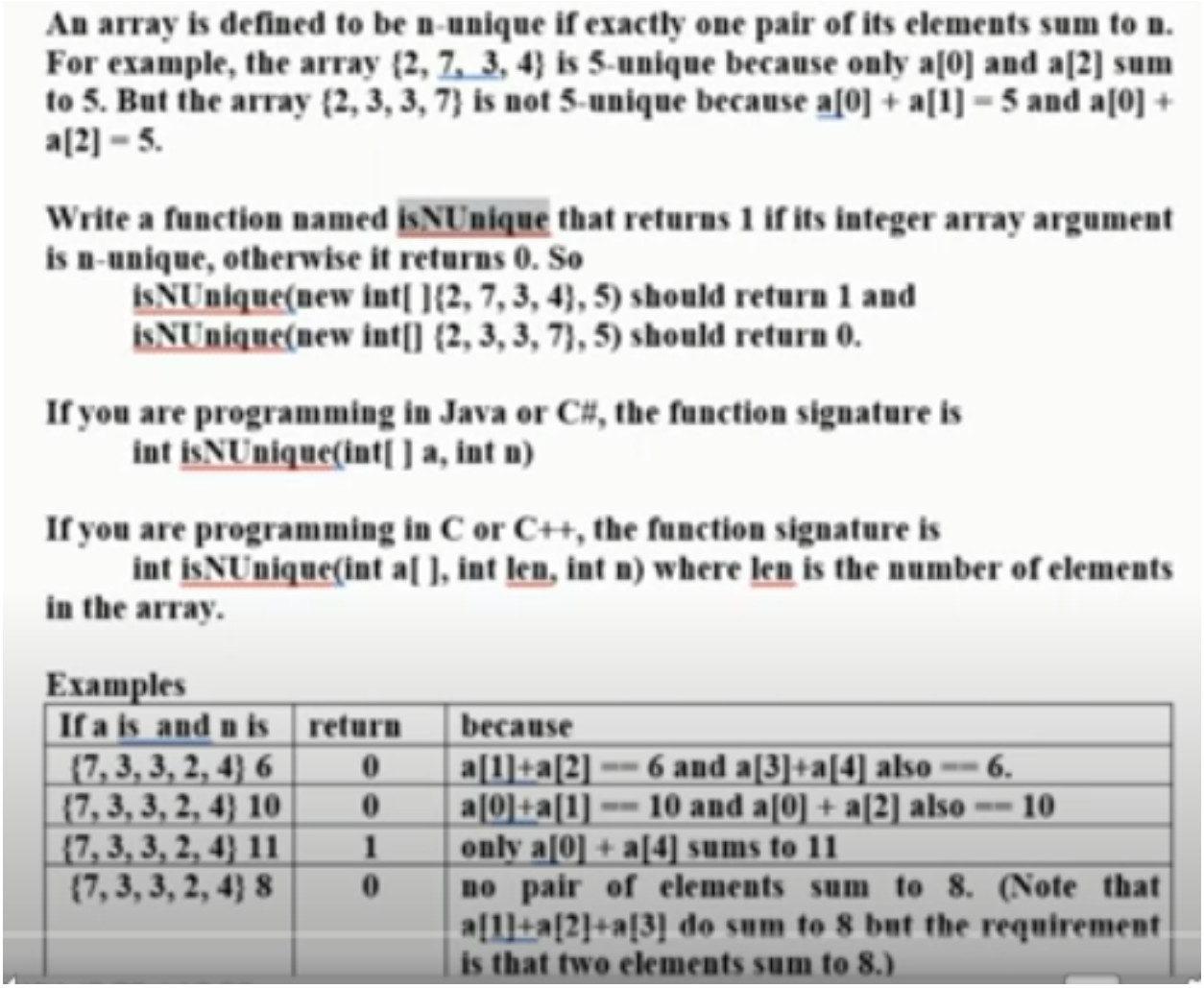
{1,2, 1, 2, 3, 4, 5,6,7}	{1, 2, 3, 4, 5,6,7}
{12, 31, 1, 3, 6, 12, 3, 1, 2}	{12, 31, 1, 3, 6, 2}

```
public class removeDuplicate {
    public static void arrayPrint(int[] array) {
        System.out.print("{");
        for (int i = 0; i < array.length; i++) {
            if (i > 0) {
                System.out.print(", ");
            }
            System.out.print(array[i]);
        }
        System.out.println("}");
    }

    public static int[] dropDuplicate(int[] array){
        int size = array.length;
        if (size == 1 || array == null){
            return array;
        }
        int[] cleanedArray = new int[array.length];
        int index = 0;
        for (int i = 0; i < array.length; i++){
            boolean isUnique = true;
            for (int j = 0; j < i; j++){
                if (array[i]== array[j]){
                    isUnique = false;
                    break;
                }
            }
            if (isUnique){
                cleanedArray[index++] = array[i];
            }
        }
        int[] resultArray = new int[index];
        System.arraycopy(cleanedArray, 0, resultArray, 0, index);
        return resultArray;

    }
    public static void main(String[] args) {
        // int[] array = {1, 2, 3, 4, 5,6,7};
        // int[] array = {1, 2, 1, 2, 3, 4, 5};
        int[] array = {12, 31, 1, 3, 6, 12, 3, 1, 2};
        int[] arrayCleaned = dropDuplicate(array);
        arrayPrint(arrayCleaned);
    }
}
```

Question 10:



An array is defined to be n-unique if exactly one pair of its elements sum to n. For example, the array {2, 7, 3, 4} is 5-unique because only a[0] and a[2] sum to 5. But the array {2, 3, 3, 7} is not 5-unique because a[0] + a[1] = 5 and a[0] + a[2] = 5.

Write a function named isNUnique that returns 1 if its integer array argument is n-unique, otherwise it returns 0. So
isNUnique(new int[ ]{2, 7, 3, 4}, 5) should return 1 and
isNUnique(new int[] {2, 3, 3, 7}, 5) should return 0.

If you are programming in Java or C#, the function signature is
int isNUnique(int[ ] a, int n)

If you are programming in C or C++, the function signature is
int isNUnique(int a[ ], int len, int n) where len is the number of elements in the array.

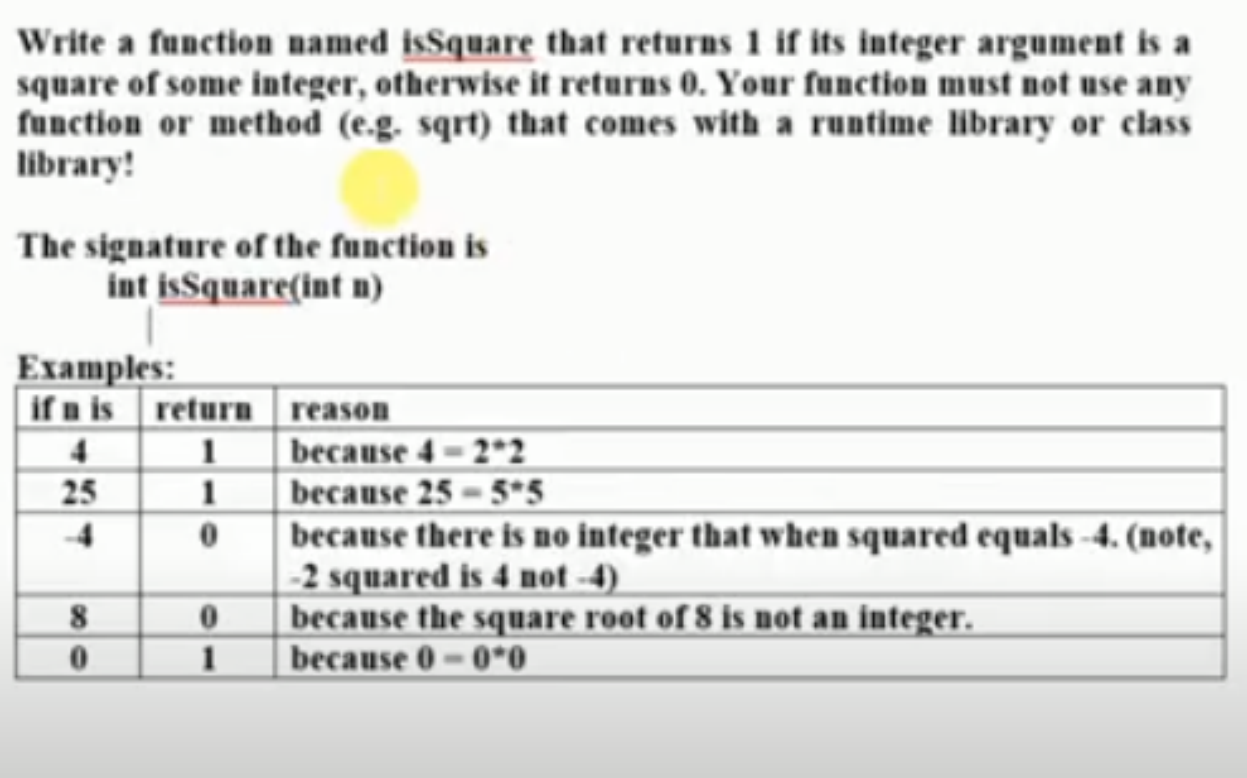
Examples

If a is and n is	return	because
{7, 3, 3, 2, 4} 6	0	a[1]+a[2] == 6 and a[3]+a[4] also == 6.
{7, 3, 3, 2, 4} 10	0	a[0]+a[1] == 10 and a[0] + a[2] also == 10
{7, 3, 3, 2, 4} 11	1	only a[0] + a[4] sums to 11
{7, 3, 3, 2, 4} 8	0	no pair of elements sum to 8. (Note that a[1]+a[2]+a[3] do sum to 8 but the requirement is that two elements sum to 8.)

```
public class isNUnique {
    public static int isNUniqueNumber(int[] array, int uniqueNumber) {
        int n = array.length;
        int uniqueCount = 0;
        for (int i = 0; i < n; i++) {
            for (int j = i+1; j < n; j++) {
                if (array[i] + array[j] == uniqueNumber) {
                    uniqueCount++;
                }
            }
        }
        if (uniqueCount == 1){
            return 1;
        }
        return 0;
    }
```

```
    public static void main(String[] args) {
        // int[] array = {7,3,3,2,4};
        // int uniqueNumber = 6;
        // int[] array = {7,3,3,2,4};
        // int uniqueNumber = 10;
        // int[] array = {7,3,3,2,4};
        // int uniqueNumber = 11;
        int[] array = {7,3,3,2,4};
        int uniqueNumber = 8;
        System.out.println(isNUniqueNumber(array,uniqueNumber));
    }
}
```

Question 11:



Write a function named isSquare that returns 1 if its integer argument is a square of some integer, otherwise it returns 0. Your function must not use any function or method (e.g. sqrt) that comes with a runtime library or class library!

The signature of the function is

int isSquare(int n)

Examples:

if n is	return	reason
4	1	because 4 = 2*2
25	1	because 25 = 5*5
-4	0	because there is no integer that when squared equals -4. (note, -2 squared is 4 not -4)
8	0	because the square root of 8 is not an integer.
0	1	because 0 = 0*0

```
public class isSquare {
    public static int isSquareNumber(int n){
        if (n < 0){
            return 0;
        }
        int sqrt = (int) Math.sqrt(n);
        if(sqrt * sqrt == n){
```

```
            return 1;
        }
        return 0;

        // int sqrt = 0;
        // while (sqrt * sqrt <= n) {
        //     if (sqrt * sqrt == n) {
        //         return 1;
        //     }
        //     sqrt++;
        // }
        // return 0;
    }
    public static void main(String[] e){
        // int number = 4;
        // int number = 25;
        // int number = -4;
        // int number = 8;
        int number = 0;
        System.out.println(isSquareNumber(number));
    }

}
```

Question 12:

Different Base to Decimal and vice versa

```
public class baseConversion {
    public static int baseToDecimal(String baseNumber,int base) {
        int decimal = 0;

        // Reverse the binary string for easier processing
        String reversedBaseNumber = new StringBuilder(baseNumber).reverse().toString();

        // Iterate through each digit of the reversed binary string
        for (int i = 0; i < reversedBaseNumber.length(); i++) {
            int digit = Character.getNumericValue(reversedBaseNumber.charAt(i));
            decimal += digit * Math.pow(base, i);
        }

        return decimal;
    }

    public static String decimalToBase(int decimal, int base) {
        if (decimal == 0) {
            return "0";
        }
```

```
        StringBuilder result = new StringBuilder();

        while (decimal > 0) {
            int remainder = decimal % base;
            result.append(Integer.toString(remainder, base));
            decimal /= base;
        }

        return result.reverse().toString();
    }

    public static void main(String[] args) {
        // String baseNumber = "11111"; // Binary number
        // int base = 2;
        // String baseNumber = "4444"; // Quinery number
        // int base = 5;
        // String baseNumber = "7777"; // Octal number
        // int base = 8;
        // String baseNumber = "FFFF"; // Hexadecimal number
        // int base = 16;
        // int decimalResult = baseToDecimal(baseNumber,base);
        // System.out.println(decimalResult);

        // int decimalNumber = 10;
        // int base = 16; // Hexadecimal number
        // int decimalNumber = 10;
        // int base = 8; // Octal number
        // int decimalNumber = 10;
        // int base = 5; // Hexadecimal number
        int decimalNumber = 10;
        int base = 2; // Binary number
        String baseResult = decimalToBase(decimalNumber, base);
        System.out.println(baseResult.toUpperCase());

    }
}
```

Question 13:

Write a method named isDivisible that takes an integer array and a divisor and returns 1 if all its elements are divided by the divisor with no remainder. Otherwise it returns 0.

If you are programming in Java or C#, the function signature is
int isDivisible(int [ ] a, int divisor)

If you are programming in C or C++, the function signature is
int isDivisible(int a[ ], int len, int divisor) where len is the number of elements in the array.

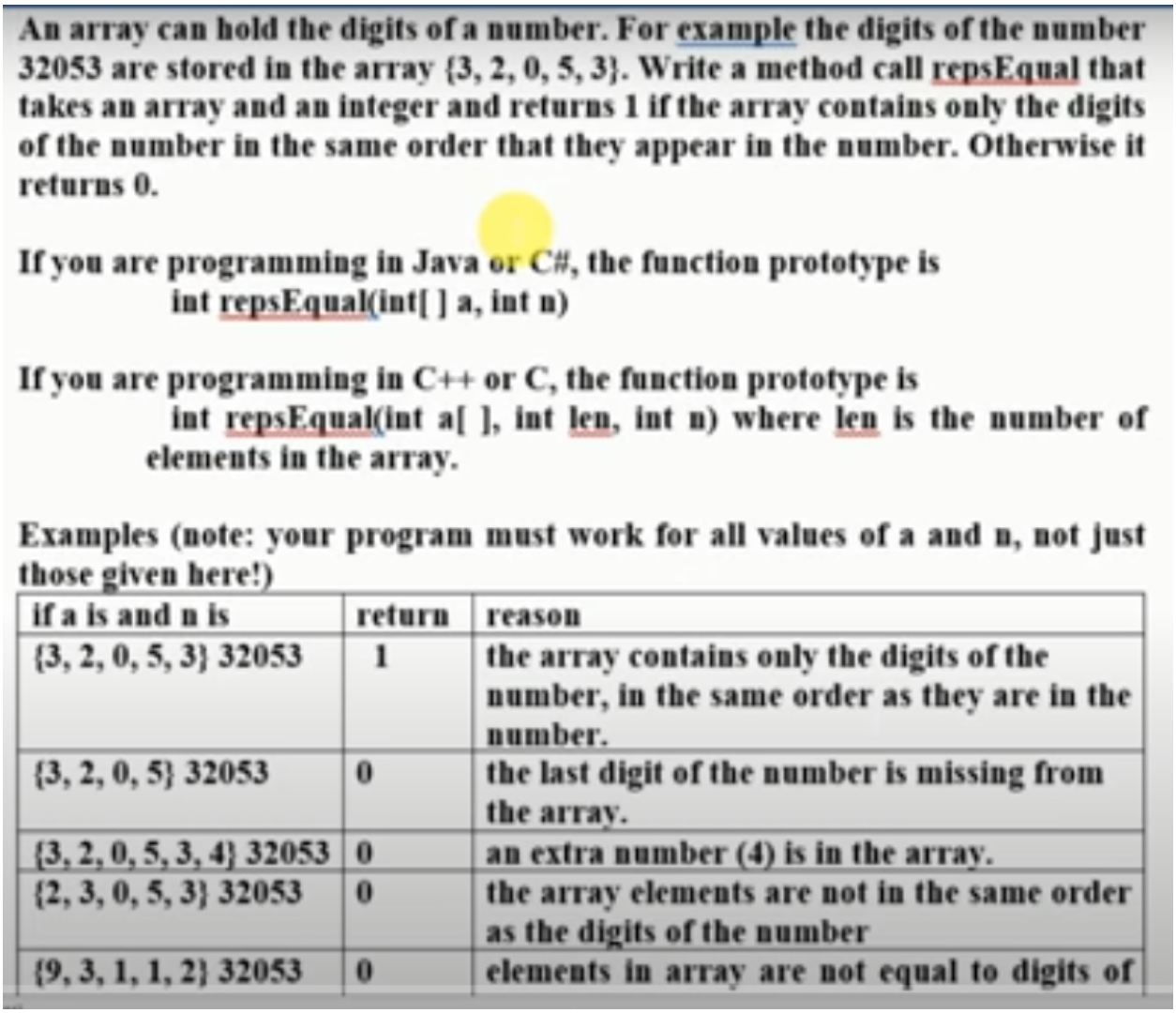
Examples

if a is and divisor is	return	because
{3, 3, 6, 36} 3	1	all elements of a are divisible by 3 with no remainder.
{4} 2	1	all elements of a are even
{3, 4, 3, 6, 36} 3	0	because when a[1] is divided by 3, it leaves a remainder of 1
{6, 12, 24, 36} 12	0	because when a[0] is divided by 12, it leaves a remainder of 6.
{} anything	1	because no element fails the division test.

```
public class isDivisible {
    public static int isDivisibleByNumber(int[] array, int number) {
        int len = array.length;
        if (len == 0){
            return 1;
        }
        for (int i= 0; i < len; i++) {
            if (array[i] % number == 0){
                continue;
            }else{
                return 0;
            }
        }
        return 1;
    }
    public static void main(String[] args){
        // int[] array = {3, 3, 6, 36};
        // int number = 3;
        // int[] array = {4};
        // int number = 2;
        // int[] array = {3, 4, 3, 6, 36};
        // int number = 3;
        // int[] array = {3, 4, 3, 6, 36};
```

```
        // int number = 12;
        int[] array = {};
        int number = 3;
        System.out.println(isDivisibleByNumber(array, number));
    }
}
```

Question 14:



An array can hold the digits of a number. For example the digits of the number 32053 are stored in the array {3, 2, 0, 5, 3}. Write a method call repsEqual that takes an array and an integer and returns 1 if the array contains only the digits of the number in the same order that they appear in the number. Otherwise it returns 0.

If you are programming in Java or C#, the function prototype is
int repsEqual(int[ ] a, int n)

If you are programming in C++ or C, the function prototype is
int repsEqual(int a[ ], int len, int n) where len is the number of elements in the array.

Examples (note: your program must work for all values of a and n, not just those given here!)

if a is and n is	return	reason
{3, 2, 0, 5, 3} 32053	1	the array contains only the digits of the number, in the same order as they are in the number.
{3, 2, 0, 5} 32053	0	the last digit of the number is missing from the array.
{3, 2, 0, 5, 3, 4} 32053	0	an extra number (4) is in the array.
{2, 3, 0, 5, 3} 32053	0	the array elements are not in the same order as the digits of the number
{9, 3, 1, 1, 2} 32053	0	elements in array are not equal to digits of

```
public class isRepsEqual {
    public static int repsEqual(int[] array, int len, int number) {
        if (len <= 0 || number == 0){
            return 0;
        }
```

```
        int arrayNumber = 0;
        for (int arrayElem : array) {
            arrayNumber = arrayNumber * 10 + arrayElem;
            }
        if (arrayNumber == number){
            return 1;
        }
        return 0;
    }

    public static void main(String[] args){
        // int[] array = {3, 2, 0, 5, 3};
        // int number = 32053;
        // int[] array = {3, 2, 0, 5};
        // int number = 32053;
        // int[] array = {3, 2, 0, 5, 3, 4};
        // int number = 32053;
        // int[] array = {2, 3, 0, 5, 3, 4};
        // int number = 32053;
        int[] array = {9, 3, 1, 1, 2};
        int number = 32053;
        int len = array.length;
        System.out.println(repsEqual(array,len, number));
    }
}
```

Question 15:

Write a function named primeCount with signature int primeCount(int start, int end);
The function returns the number of primes between start and end inclusive.
Recall that a prime is a positive integer greater than 1 whose only integer factors are 1 and itself.

Examples

if start is	and end is	return	reason
10	30	6	The primes between 10 and 30 inclusive are 11,13,17,19,23 and 29
11	29	6	The primes between 11 and 29 inclusive are 11,13,17,19,23 and 29
20	22	0	20,21, and 22 are all non-prime
1	1	0	By definition, 1 is not a prime number
5	5	1	5 is a prime number
6	2	0	start must be less than or equal to end

-10	6	3	primes are greater than 1. 2, 3 and 5 are prime

```
public class primeCountter {
    public static boolean isPrime(int num) {
        if (num <= 1) return false;
        if (num <= 3) return true;
        if (num % 2 == 0 || num % 3 == 0) return false;

        int i = 5;
        while (i * i <= num) {
            if (num % i == 0 || num % (i + 2) == 0) return false;
            i += 6;
        }

        return true;
    }

    public static int primeCount(int start, int end) {
        int count = 0;
        for (int i = start; i <= end; i++) {
            if (isPrime(i)) {
                count++;
            }
        }
        return count;
    }

    public static void main(String[] args) {
        // int start = 10;
        // int end = 30;
        // int start = 11;
        // int end = 29;
        // int start = 20;
        // int end = 22;
        // int start = 1;
        // int end = 1;
        // int start = 5;
        // int end = 5;
        // int start = 6;
        // int end = 2;
        int start = -10;
        int end = 6;
        System.out.println(primeCount(start, end));
    }
}
```

Question 16:

Define the n-upcount of an array to be the number of times the partial sum goes from less than or equal to n to greater than n during the calculation of the sum of elements of the array.
if you are programming in Java or C#, the function signature is int nUpCount(int[] a, int n)

For example, if n=5, the 5-upcount of the array {2,3,1,-6,8,-3,-1,2} is 3.

i	a[i]	partial sum	upcount	comment
0	2	2		
1	3	5		
2	1	6	1	partial sum goes from 5 to 6
3	-6	0		
4	8	8	2	partial sum goes from 0 to 8
5	-3	5		
6	-1	4		
7	2	6	3	partial sum goes from 4 to 6

Question 17:

Write a function nextPerfectSquare that returns the first perfect square that is greater than it's integer argument. A perfect square is an integer that is equal to some integer squared.

For example 16 is a perfect square because 16 = 4x*4. However 15 is not a perfect square because there is no integer n such that 15 = nx*n.

The signature of the function is
int isPerfectSquare(int n)
Examples

n	next perfect square
6	9
36	49
0	1

-5	0

Question 18:

An isSym (even/odd Symmetric) array is defined to be an array in which even numbers and odd numbers appear in the same order from "both directions".
You can assume array has at least one element.
{2,7,9,10,11,5,8} is a isSym array.
Note that from left to right or right to left we have even,odd,odd,even,odd,odd,even.

{9,8,7,13,14,17} is a isSym array.
Note from left to right or right to left we have {odd,even,odd,odd,even,odd}

However, {2,7,8,9,11,13,10} is NOT a isSym array.
From left to right we have {even,odd,even,odd,odd,odd,even}
From right to left we have {even,odd,odd,odd,even,odd,even} which is not the same.

Write a function named isSym that returns 1 if its array argument is a isSym array, otherwise it returns 0.

if you are programming in Java or C#, the function signature is:
int isSym(int[] a)

```
// public class IsSym {
//     public static boolean isEven(int num) {
//         return num % 2 == 0;
//     }

//     public static int isSym(int[] arr) {
//         int start = 0;
//         int end = arr.length - 1;

//         while (start <= end) {  // Change the loop condition to handle odd-sized arrays
//             if (isEven(arr[start]) != isEven(arr[end]))
//                 return 0;
//             start = start + 1;
//             end = end - 1;
//         }

//         return 1;
//     }

//     public static void main(String[] args) {
```

```
//         int[] arr = {2, 7, 9, 10, 11, 5, 8};
//         System.out.println(isSym(arr));
//     }
// }

public class IsSym {
    public static int isSym(int[] arr) {
        int start = 0;
        int end = arr.length - 1;

        while (start <= end) {
            boolean startIsEven = arr[start] % 2 == 0;
            boolean endIsEven = arr[end] % 2 == 0;

            if (startIsEven != endIsEven) {
                return 0;
            }

            start++;
            end--;
        }

        return 1;
    }

    public static void main(String[] args) {
        // int[] array = {2, 7, 9, 10, 11, 5, 8};
        // int[] array = {9, 8, 7, 13, 14, 17};
        int[] array = {2, 7, 8, 9, 11, 13, 10};
        System.out.println(isSym(array)); // Output: 0
    }
}
```

Question 19:

A Sub Array is defined to be an array in which each element is greater than sum of all elements after that. see example below:
{13,6,3,2} is a sub array. Note that 13 > 6+3+2, 6 > 3+2, 3>2.
{11,5,3,2} is NOT a sub array. Note that 5 is not greater than 3+2.
Write a function named isSub that returns 1 if its array argument is a Sub array. otherwise it returns 0.
if you are programming in Java or C#, the function signature is: int isSub(int[] a)

Question 20:

An integer array is said to be evenSpaced, if the difference between the largest value and the smallest value is an even number.

Write a function isEvenSpaced(int[] a) that will return 1 if it is evenSpaced and 0 otherwise.
if array has less than two elements, function will return 0.

Array	Largest value	Smallest value	Difference	Return value
{100,19,131,140}	140	19	140 - 19 = 121	0
{200,1,151,160}	200	1	200 - 1 = 199	0
{200,10,151,160}	200	10	200 - 10 = 190	1
{100,19,-131,-140}	100	-140	100-(-140)= 240	1
{80,-56,11,-81}	80	-81	80 -(-81) = 161	0

Question 21:

The sum factor of an array is defined to be the number of times that the sum of the array appears as an element of the array. So the sum factor of {1,-1,1,-1,1,-1,1} is 4 because the sum of the elements
of the array is 1 and 1 appears four times in the array.
And the sum factor of {1,2,3,4} is 0 because the sum of the elements of the array is 10 and 10 does not occur as an element of the array. The sum factor of the empty array is defined to be 0.

Write a function named sumFactor that returns the sum factor of its array argument.
if you are programming in Java or C#, the function signature is
int sumFactor(int[] a)

if a is	return	reason
{3,0,2,-5,0}	2	The sum of array is 0 and 0 occurs 2 times
{9,-3,-3,-1,-1}	0	The sum of the array is 1 and 1 does not occur in array
{1}	1	The sum of the array is 1 and 1 occurs once in the array
{0,0,0}	3	The sum of the array is 0 and 0 occurs 3 times in the array

Question 22:

The Stanton measure of an array is computed as follows. Count the number of 1s in the array.
Let this count be n. The Stanton measure is the number of times that n appears in the array.
For example, the Stanton measure of {1,4,3,2,1,2,3,2} is 3 because 1 occurs 2 times in the array and 2 occurs 3 times.
Write a function named stantonMeasure that returns the Stanton measure of its array argument.
If you are programming in Java or C#, the function prototype is
int stantonMeasure(int[] a)
Examples

if a is	return	reason
{1}	1	1 occurs 1 time, 1 occurs 1 time
{0}	1	1 occurs 0 times, O occurs 1 time
{3,1,1,4}	0	1 occurs 2 times, 2 occurs 0 times
{1,3,1,1,3,3,2,3,3,3,4}	6	1 occurs 3 times, 3 occurs 6 times
{}	0	1 occurs 0 times, 0 occurs 0 times

Question 23:

A Madhav array has the following property.
a[0] = a[1] + a[2] = a[3] + a[4] + a[5] = a[6] + a[7] + a[8] + a[9] = ...
The length of a Madhav array must be n*(n+1)/2 for some n.
Write a method named isMadhavArray that returns 1 if its array argument is a Madhav array, otherwise it returns 0.

If you are programming in Java or C# the function signature is
int isMadhavArray(int[] a)

Examples

if a is	return	reason
{2,1,1}	1	2 = 1 + 1
{2,1,1,4,-1,-1}	1	2 = 1 + 1, 2 = 4 + -1 + -1
{6,2,4,2,2,2,1,5,0,0}	1	6 = 2 + 4, 6 = 2 + 2 + 2, 6 = 1 + 5 + 0 + 0
{18,9,10,6,6,6}	0	18 != 9 + 10

{-6,-3,-3,8,-5,-4}	0	-6 != 8 + -5 + -4
{0,0,0,0,0,0,0,0,0,0,1,1,1,-2,-1}	1	0 = 0+0, 0 = 0+0+0, 0 = 0+0+0+0, 0 = 1+1+1+-2+-1
{3,1,2,3,0}	0	The length of the array is 5, but 5 != n*(n+1)/2

```
public class IsMadhavArray{
    public static int isMadhavArray(int[] arr) {
        int n = 1;  // Initialize n to 1
        int index = 0;

        while (index < arr.length) {
            int sum = 0;
            for (int i = 0; i < n; i++) {
                if (index >= arr.length) {
                    return 0;  // Array length is not valid for Madhav array
                }
                sum += arr[index];
                index++;
            }

            if (sum != arr[0]) {
                return 0;  // The sums are not equal
            }

            n++;
        }

        return 1;
    }

    public static void main(String[] args) {
        int[] arr1 = {2, 1, 1};
        int[] arr2 = {2, 1, 1, 4, -1, -1};
        int[] arr3 = {6, 2, 4, 2, 2, 2, 1, 5, 0, 0};
        int[] arr4 = {18, 9, 10, 6, 6, 6};

        System.out.println(isMadhavArray(arr1)); // Output: 1
        System.out.println(isMadhavArray(arr2)); // Output: 1
        System.out.println(isMadhavArray(arr3)); // Output: 1
        System.out.println(isMadhavArray(arr4)); // Output: 0
    }
}
```

Question 24:

Write a function that will return 1 if an integer array satisfies the following conditions and returns 0 otherwise.

1. it has even numbers of elements
2. Sum of all the numbers in the first half of the array is equal to the sum of all the numbers in the second half of the array.

If you are programming in Java, the function Signature is
int answerThree(int[] a)
Examples

a	return	Explanation
{5,4,3,2,3,4,6,1}	1	*There are 8 (even) number of elements in the array. Thus condition 1 satisfied. *The sum of all the numbers in the first half = 5+4+3+2 = 14

```
public class AnswerThree {
    static int answerThree(int[] arr) {
        if (arr.length % 2 != 0)
            return 0;

        int midInd = arr.length / 2;
        int leftSum = 0, rightSum = 0;

        for (int i = 0; i < arr.length; i++) {
            if (i < midInd) {
                leftSum = leftSum + arr[i];
            } else {
                rightSum = rightSum + arr[i];
            }

        }
        if (leftSum == rightSum)
            return 1;
        return 0;
    }

    public static void main(String args[]) {
        int[] arr = { 5, 4, 3, 2, 3, 4, 6, 1 };
        System.out.println(answerThree(arr));
    }
}
```

Question 25:

Write a function that will return the most occurring number in an array. If there is more than one such number, the function may return any one of them.

If you are programming in Java or C#, the function signature is int answerTwo(int[] a)

Examples

a	return	Explanation
{6,8,1,8,2}	8	8 occurs two times. No other number occurs 3 or more times
{6,8,1,8,6}	8 or 6	8, 6. The Function may return either 8 or 6

```
public class AnserTwo {
    static int answerTwo(int[] array) {
        if (array == null || array.length == 0){
            return 0;
        }
        int len = array.length;
        for (int i = 0; i < len - 1; i++) {
            for (int j = i + 1; j < len; j++) {
                if (array[i] == array[j]) {
                    return array[i];
                }
            }
        }
        return 0;
    }

    public static void main(String args[]) {
        int[] array = {6,8,1,8,2};
        // int[] array = {6, 8, 1, 8, 6};
        System.out.println(answerTwo(array));
    }
}
```

```
public class OccuringNumber {
    static int occuranceNumber(int[] arr) {
        int maxNum = 0, maxCount = 0;

        for (int i = 0; i < arr.length; i++) {
            int num = arr[i];
            int count = 0;

            for (int j = 0; j < arr.length; j++) {
                if (num == arr[j]) {
                    count++;
                }
            }

            if (count > maxCount) {
                maxCount = count;
                maxNum = num;
```

```
            }
        }

        return maxNum;
    }

    public static void main(String args[]) {
        int[] arr = { 6, 8, 1, 8,9,8,6 };
        System.out.println(occuranceNumber(arr));
    }
}
```

Question 26:

Consider the positive integer 50.

Note that 50 = 25 + 25 = 5^2 + 5^2 and 50 = 1 + 49 = 1^2 + 7^2,
Thus 50 can be expressed as a sum of the two squares in two different ways.
Write a method whether or not a positive integer n can be expressed as a sum of two squares in exactly two different ways.
The signature of the function is
int answerOne(int n)

```
public class AnswerOne {
    static String answerOne(int n) {
        for (int i = 1; i < n; i++) {
            for (int j = 1; j < n; j++) {
                if (n == i * i + j * j)
                    return n + " = " + i + "^2" + " + " + j + "^2";
            }
        }
        return n + "cannot be expressed as a sum of two squares.";
    }

    public static void main(String args[]) {
        System.out.println(answerOne(50));
    }
}
```

Question 27:

A Bean array is defined to be an integer array where for every value n in the array,
there is also an element 2n, 2n+1 or n/2 in the array.

for example {4,9,8} is a Bean array because for 4,8 is present, for 9, 4 is present, for 8, 4 is present.
Other Bean arrays include {2,2,5,11,23}, {7,7,3,6} and {0}

The array {3,8,4} is not a Bean array because of the value 3 which requires that the array contains either the value 6, 7 or 1 and none of these values are in the array.
Write a function named isBean that returns 1 if it's array argument is a Bean array,
otherwise it returns a 0.
if you are programming in Java or C#, the function signature is int isBean(int[] a)

```
public class IsBean {
    static int isBean(int[] array) {
        int len = array.length;
        int beanFlag = 0;
        for (int i = 0; i < len; i++) {
            for (int j = 0; j < len; j++) {
                int num = array[i];
                int temp = array[j];
                if (num == 2 * temp || num == 2 * temp + 1 || num == temp / 2) {
                    beanFlag = 1;
                    break;
                } else {
                    beanFlag = 0;
                }
            }
            if (beanFlag == 0)
                return beanFlag;

        }
        return beanFlag;
    }

    public static void main(String args[]) {
        int[] array = { 2, 2, 5, 11, 2 };
        System.out.println(isBean(array));
    }
}
```

Question 28:

A Meera array is any array that contains the value 1 if and only if it contains 9.
The array {7,9,0,10,1} is a Meera array because it contains 1 and 9.
The array {6,10,8} is a Meera array because it contains no 1 and no 9.
The array {7,6,1} is not a Meera array because it contains 1 but does not contain a 9.
The array {9,10,0} is not a Meera array because it contains a 9 but does not contain 1.
It is okay if a Meera array contains more than one value 1 and more than one 9.
So the array {1,1,0,8,0,9,9,1} is a Meera array.
Write a function named isMeera that returns 1 if it's array argument is a Meera array

and returns 0 otherwise.
If you are programming in Java or C#, the function signature is int isMeera(int[] a)

```
public class IsMeera {
    static int isMeera(int[] array) {
        boolean bool_1 = false;
        boolean bool_9 = false;
        int len = array.length;
        for (int i = 0; i < len; i++) {
            if (array[i] == 1)
                bool_1 = true;
            if (array[i] == 9)
                bool_9 = true;
        }
        if ((!bool_1 && bool_9) || (bool_1 && !bool_9))
            return 0;

        return 1;
    }

    public static void main(String args[]) {
        // int[] array = { 7, 0, 1, 8, 9, 10 };
        // int[] array = {6,10,8};
        // int[] array = {7,6,1};
        // int[] array = {9,10,0};
        int[] array = {1,1,0,8,0,9,9,1};
        System.out.println(isMeera(array));
    }
}
```

Question 29:

A fancy number is a number in the sequence 1,1,5,17,61,... Note the first two fancy numbers are 1 and any fancy number other than the first two is sum of three times previous one and two times the one before that. see below
1,
1,
3*1* + *2*1 = 5,
5*3* + *2*1 = 17,
17*3* + *5*2 = 61
Write a function named isFancy that returns 1, if it's integer argument is a Fancy number, otherwise it returns 0.

```
public class IsFancy {
    public static int isFancy(int number) {
```

```
        if (number == 1) {
            return 1;
        }

        int firstFancy = 1;
        int secondFancy = 1;
        int currentFancy = 0;

        while (currentFancy < number) {
            currentFancy = 3 * secondFancy + 2 * firstFancy;

            if (currentFancy == number) {
                return 1;
            }

            firstFancy = secondFancy;
            secondFancy = currentFancy;
        }

        return 0;
    }

    public static void main(String[] args) {
        System.out.println(isFancy(1));
        System.out.println(isFancy(1));
        System.out.println(isFancy(5));
        System.out.println(isFancy(17));
        System.out.println(isFancy(61));
    }
}
```

Question 30:

A prime number is an integer that is divisible only by 1 and itself. A porcupine number is a prime number whose last digit is 9 and the next prime number that follows it also ends with the digit 9. For example 139 is a porcupine number because:
a - it is prime
b - it ends in a 9
c - The next prime number after it is 149 which also ends in 9.
Note that 140, 141, 142, 143, 144, 145, 146, 147, and 148 are not prime so 149 is the next prime number after 139.

Write a method named findPorcupineNumber which takes an integer argument n and returns the first porcupine number that is greater than n. So findPorcupineNumber(0) would return 139 (because 139 happens to be the first porcupine number) and so would findPorcupineNumber(138).
But findPorcupineNumber(139) would return 409 which is the second porcupine number.

The function signature is
int findPorcupineNumber(int n)
You may assume that a porcupine number greater than n exists.
You may assume that a function isPrime exists that returns 1 if its argument is prime, otherwise it returns 0. e.g isPrime(7) returns 1 and isPrime(8) returns 0.
Hint: Use modulo base 10 arithmetic to get last digit of a number.

Question 31:

Consider the following algorithm

- Start with a positive number n
- if n is even then divide by 2
- if n is odd then multiply by 3 and add 1
  -* continue this until n becomes 1

The Guthrie sequence of a positive number n is defined to be the numbers generated by the above algorithm.

For example, the Guthrie sequence of the number 7 is
7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
It is easy to see that this sequence was generated from number 7 by the above algorithm.
Since 7 is odd, multiply by 3 and add 1 to get 22 which is the second number of the sequence.
Since 22 is even, divide by 2 to get 11 which is the third number of the sequence.
11 is odd, multiply by 3 and add 1 to get 34 which is the fourth number of the sequence and so on.

Note: the first number of a Guthrie sequence is always the number that generated the sequence and the last number is always 1.

Write a function named isGuthrieSequence which returns 1 if the elements of the array form a Guthrie sequence. Otherwise, it returns 0.
If you are programming in Java or c#, the function signature is
int isGuthrieSequence(int[] a)
Examples

if a is	return	reason
{8,4,2,1}	1	This is the Guthrie sequence for 8
{8,17,4,1}	0	This is not the Guthrie sequence for 8

{8,4,1}	0	Missing the 2
{8,4,2}	0	A Guthrie sequence must end with 1

Question 32:

Define a square pair to be the tuple <x,y> where x and y are positive, non-zero integers, x<y and x+y is a perfect square. A perfect square is an integer whose square root is also an integer, e.g 4,9,16 are perfect squares but 3,10,and 17 are not.

Write a function named countSquarePairs that takes an array and returns the number of square pairs that can be constructed from the elements in the array.
For example, if the array is {11,5,4,20} the function would return 3 because the only square pairs that can be constructed from those numbers are <5,11>,<5,20> and <4,5>.
You may assume that there exists a function named isPerfectSquare that returns 1 if its argument is a perfect square and 0 otherwise. isPerfectSquare(4) returns 1 and isPerfectSquare(8) returns 0.
If you are programming in Java or C#, the function signature is
int countSquarePairs(int[] a)
You may assume that there are no duplicate values in the array. ie you don't have to deal with an array like {2,7,2,2}.

examples:

if a is	return	reason
{9,0,2,-5,7}	2	The square pairs are <2,7> and <7,9>. Note that <-5,9> and <0,9> are not square pairs, even though they sum to perfect squares, because both members of a square pair have to be greater than 0. Also <7,2> and <9,7> are not square pairs because the first number has to be less than the second number.
{9}	0	The array must have at least 2 elements

```
import java.lang.Math;

public class CountSquarePairs {

    public static boolean isPerfectSquare(int sum){
        double sqrtValue = Math.sqrt(sum);

        if(sqrtValue - Math.floor(sqrtValue) == 0) return true;
```

```
        return false;
    }
    public static int countSquarePairs(int [] arr){
        int count = 0;

        for(int i=0; i<arr.length ; i++){
            int x = arr[i];
            for(int j=0; j<arr.length; j++){
                if(i!=j) {
                    int y = arr[j];
                    // check the conditions
                    if (x > 0 && y > 0 && x < y) {

                        if (isPerfectSquare(x + y)) {
                            count = count + 1;
                        }
                    }
                }
            }
        }

        return count;
    }
    public static void main(String[] args) {
        // -- Return how many pairs in an array satisfies the condition:
                    // x and y are both non-zero and positive integers
                    // x<y
                    // x+y is a perfect square
        int[] arr = {9,0,2,-5,7};
        System.out.println(countSquarePairs(arr));
    }
}
```

Question 33:
An array is defined to be inertial if the following conditions hold:
a. it contains at least one odd value
b. the maximum value in the array is even
c. every odd value is greater than every even value that is not the maximum value.
so {11,4,20,9,2,8} is inertial because

- it contains at least one odd value
- the maximum value in the array is 20 which is even
- the two odd values (11 and 9) are greater than all the even values that are not equal to 20 (the maximum), i.e {4,2,8}.
  However, {12,11,4,9,2,3,10} is not inertial because it fails condition (c), i.e.

10 (which is even) is greater than 9 (which is odd), and 10 is not the maximum value in the array.

Write a function called isInertial that accepts an integer array and returns 1 if the array is inertial; otherwise it returns 0.
If you are programing in Java or C#, the function signature is
int isInertial(int[] a)

Some other examples:

if the input array is	return	reason
{1}	0	fails condition (a) - the maximum value must be even
{2}	0	fails condition (b) - the array must contain at least one odd value
{1,2,3,4}	0	fails condition (c) - 1(which is odd) is not greater than all even
		values other than the maximum (1 is less than 2 which is not the max)
{1,1,1,1,1,1,2}	1	there is no even number other than the maximum. Hence, there can be no
		other even values that are greater than 1.
{2,12,4,6,8,11}	1	11, the only odd value is greater than all even values except 12 which
		is the maximum value in the array.
{2,12,12,4,6,8,11}	1	same as previous, ie, it is ok if maximum value occurs more than once
{-2,-4,-6,-8,-11}	0	-8, which is even, is not the maximum value but is greater than -11
{2,3,5,7}	0	the maximum value is odd
{2,4,6,8,10}	0	there is no odd value in the array

NOTE: To ease debugging, i will return a string

```
import java.util.ArrayList;
import java.util.Collections;

public class IsInertia {
    static int isInertial(int[] arr) {
        ArrayList<Integer> oddArray = new ArrayList<>();
        ArrayList<Integer> evenArray = new ArrayList<>();

        for (int i = 0; i < arr.length; i++) {
            // add to odd array and even array
            if (arr[i] % 2 == 0) {
                evenArray.add(arr[i]);
```

```
            } else {
                oddArray.add(arr[i]);
            }
        }

        // check the length of odd array
        // sort the arrays
        // save the max and second max value of the even array
        // compare the min value of odd array with all items in even array
        // so that it proves each odd value is greater than even value other than max even value
        // check if the max value in the whole array is even

        if (oddArray.size() == 0) {
            return 0;
        }

        // sorting odd and even array
        Collections.sort(evenArray);
        Collections.sort(oddArray);

        // finding max and second max value in the even array
        int maxEvenValue = evenArray.get(evenArray.size() - 1); // max even value in the array

        // finding min and max odd value in the odd array
        int minOddValue = oddArray.get(0); // min odd value
        int maxOddValue = oddArray.get(oddArray.size() - 1); //max odd value

        // compare each item in even array except max value with the least odd item
        for( int i = 0; i < evenArray.size()-2; i++) {
            if (evenArray.get(i) > minOddValue) {
                return 0;
            }
        }

        // check if the even max value is the greater than max odd value
        if (maxOddValue > maxEvenValue){
            return 0;
        }

        return 1;
    }

    public static void main(String args[]) {
        int[] arr = {2,12,12,4,6,8,11}  ;
        System.out.println(isInertial(arr));
    }

}
```