1. Write a function named **foo** with signature  
   int foo (int n, int d)  
which returns how many times *d* divides *n* without a remainder. If *n* or *d* is 0, it should return -1.

Examples

* foo(24, 2) returns 3 because 24/2=12, 12/2=6, 6/2=3 and 3/2 has a remainder of 1
* foo(24, 3) returns 1 because 24/3=8 and 8/3 has a remainder of 2
* foo(36, 5) returns 0 because 36/5 has a remainder of 1
* foo(198, 199) returns 0 because 198/199 has a remainder of 198.
* foo(8, 2) returns 3 because 8/2=4, 4/2=2, 2/2=1 and 1/2 has a remainder of 1.
* foo(0, 5) returns -1
* foo(5, 0) returns -1

Note that the result is not dependent on whether *n* and *d* are negative or positive.

Answer1

public class Foo {

public static void main(String[] args) {

System.out.println(foo(24, 2));

System.out.println(foo(24, 3));

System.out.println(foo(36, 5));

System.out.println(foo(198, 199));

System.out.println(foo(8, 2));

System.out.println(foo(0, 5 ));

System.out.println(foo(5, 0 ));

}

public static int foo(int n,int d){

if (n==0 || d==0)

return -1;

int remainder=0;

int result;

int count=0;

while(remainder==0){

result=n/d;

remainder=n%d;

if (remainder==0){

n=result;

count++;

}

}

return count;

}

}

2. A **Daphne array** is defined to be an array that satisfies the following condition:

***If 5 is in the array then so is 13.***

The following arrays are Daphne arrays

* {5, 0, 13}
* {13, -1, 6, 5}
* {5, 5, 5, 3, 13, 13, 13}
* {13, 6, 13, 12} (there is no requirement that the array contains a 5)
* {1, 2, 3, 4} (the array can contain neither a 5 nor a 13).

The following arrays are **not** Daphne array

* {5, 3, 2} (no 13)
* {5, 5, 5} (no 13)

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

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

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

Answer2

public class IsDaphene {

public static void main(String[] args) {

int arr[]={13, -1, 6, 5};

int arr2[]={5, 5, 5, 3, 13, 13, 13};

int arr3[]={13, 6, 13, 12};

int arr4[]={1, 2, 3, 4};

int arr5[]={5, 3, 2};

int arr6[]={5, 5, 5};

System.out.println(isDaphneArray(arr));

System.out.println(isDaphneArray(arr2));

System.out.println(isDaphneArray(arr3));

System.out.println(isDaphneArray(arr4));

System.out.println(isDaphneArray(arr5));

System.out.println(isDaphneArray(arr6));

}

public static int isDaphneArray(int a[]){

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

if (a[i]==5)

{

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

if (a[j]==13)

return 1;

}

return 0;

}

}

return 1;

}

}

3. Write a function named *maxOccurringValue* that returns the value that occurs the most times in its array argument. If two or more different values occur the maximum number of times, return the one that occurs first in the array.

Examples

* maxOccurringValue(new int[]{5, 3, 1, 4, 5, 3, 5}) returns 5, because 5 occurs the most times.
* maxOccurringValue(new int[]{5, 3, 1, 3, 4, 5, 3, 5}) returns 5, because 5 and 3 both occur three times but 5 occurs before 3 in the array.
* maxOccurringValue(new int[]{8, 5, 3, 1, 4, 5, 3, 5}) returns 5, because 5 occurs the most times. (Be sure that your function does not return 8)
* maxOccurringValue(new int[]{4, 3, 2, 1}) returns 4 (it is okay if there are no duplicate values).

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

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

Hint: Use a nested loop.

Answer3

public class MaxOccurance {

public static void main(String[] args) {

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

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

int arr3[]={8, 5, 3, 1, 4, 5, 3, 5};

int arr4[]={4, 3, 2, 1};

System.out.println(maxOccuringValue(arr));

System.out.println(maxOccuringValue(arr2));

System.out.println(maxOccuringValue(arr3));

System.out.println(maxOccuringValue(arr4));

}

public static int maxOccuringValue(int a[]){

int maxValue=a[0];

int count;

int maxCount=1;

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

{

count=0;

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

{

if (a[i]==a[j]){

count++;

}

if (count> maxCount)

{

maxValue=a[i];

maxCount=count;

}

}

}

return maxValue;

}}