**6 kyu**

**Pure odd digits primes**

19088% of 6120 of213[raulbc777](https://www.codewars.com/users/raulbc777)

C#

* [TRAIN](https://www.codewars.com/kata/pure-odd-digits-primes/train/csharp)
* [NEXT KATA](https://www.codewars.com/trainer/csharp)

Details

[Solutions](https://www.codewars.com/kata/pure-odd-digits-primes/solutions/csharp)

[Forks (3)](https://www.codewars.com/kata/pure-odd-digits-primes/forks/csharp)

[Discourse (16)](https://www.codewars.com/kata/pure-odd-digits-primes/discuss/csharp)

* |
* Share this kata:

Primes that have only odd digits are pure odd digits primes, obvious but necessary definition. Examples of pure odd digit primes are: 11, 13, 17, 19, 31... If a prime has only one even digit does not belong to pure odd digits prime, no matter the amount of odd digits that may have.

Create a function, only\_oddDigPrimes(), that receive any positive integer n, and output a list like the one bellow:

[number pure odd digit primes bellow n, largest pure odd digit prime smaller than n, smallest pure odd digit prime higher than n]

Let's see some cases:

only\_oddDigPrimes(20) ----> [7, 19, 31]

///7, beacause we have seven pure odd digit primes bellow 20 and are 3, 5, 7, 11, 13, 17, 19

19, because is the nearest prime of this type to 20

31, is the first pure odd digit that we encounter after 20///

only\_oddDigPrimes(40) ----> [9, 37, 53]

In the case that n, the given value, is a pure odd prime, should be counted it with the found primes and search for the inmediately bellow and the inmediately after.

Happy coding!!

<https://www.codewars.com/kata/pure-odd-digits-primes/csharp>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

// static int MAX\_SIZE = 1000;

// isPrime[] : isPrime[i] is true if number is prime

// prime[] : stores all prime number less than N

// SPF[] that store smallest prime factor of number

// [for Exp : smallest prime factor of '8' and '16'

// is '2' so we put SPF[8] = 2 , SPF[16] = 2 ]

//static List<bool > isprime = new List<bool> (MAX\_SIZE);

//static List<int > prime = new List<int> (MAX\_SIZE);

//static List<int > SPF = new List<int> (MAX\_SIZE);

// function generate all prime number less then N in O(n)

static List<long> manipulated\_seive(long N)

{

// 0 and 1 are not prime

// isprime[0] = isprime[1] = false;

List<bool> isprime = new List<bool>((int)(N + 1));

List<long> prime = new List<long>((int)(N + 1));

List<long> SPF = new List<long>((int)(N + 1));

isprime.Add(false);

isprime.Add(false);

SPF.Add(0);

SPF.Add(0);

for (int i = 2; i < N + 1; i++)

{

isprime.Add(true);

SPF.Add(0);

}

// Fill rest of the entries

for (int i = 2; i < N; i++)

{

// If isPrime[i] == True then i is

// prime number

if (isprime[i])

{

// put i into prime[] vector

prime.Add(i);

// A prime number is its own smallest

// prime factor

SPF[i] = i;

}

// Remove all multiples of i\*prime[j] which are

// not prime by making isPrime[i\*prime[j]] = false

// and put smallest prime factor of i\*Prime[j] as prime[j]

// [ for exp :let i = 5 , j = 0 , prime[j] = 2 [ i\*prime[j] = 10 ]

// so smallest prime factor of '10' is '2' that is prime[j] ]

// this loop run only one time for number which are not prime

for (int j = 0;

j < (int)prime.Count &&

i \* prime[j] < N && prime[j] <= SPF[i];

j++)

{

isprime[(int)(i \* prime[j])] = false;

// put smallest prime factor of i\*prime[j]

SPF[(int)(i \* prime[j])] = prime[j];

}

}

return prime;

}

static bool TodosImpares(long n)

{

while (n > 0)

{

int d = (int)n % 10;

if (d % 2 == 0) return false;

n /= 10;

}

return true;

}

static bool EsPrimo(long n)

{

if (n < 2) return false;

if (n == 2) return true;

if (n % 2 == 0) return false;

for (int i = 3; i \* i <= n; i += 2)

{

if (n % i == 0) return false;

}

return true;

}

public static long[] OnlyOddDigPrimes(long n)

{

List<long> sieve = manipulated\_seive(n);

int todosImp = 0;

for (int i = 0; i < sieve.Count; i++)

if (TodosImpares(sieve[i]) && sieve[i] < n)

todosImp++;

long anteriorPrimo = sieve[sieve.Count-1];

int j = sieve.Count - 1;

//while (TodosImpares(sieve[j]) == false || sieve[j] >= n)

//{

// anteriorPrimo = sieve[j];

// if (j >= 0) j--;

//}

while (j >= 0)

{

if (sieve[j] < n && TodosImpares(sieve[j]))

{

anteriorPrimo = sieve[j];

break;

}

j--;

}

long proximoPrimo = n + 1;

while (!EsPrimo(proximoPrimo) || !TodosImpares(proximoPrimo))

{

proximoPrimo++;

}

//Console.WriteLine("todos impares: " + todosImp);

//Console.WriteLine(sieve[sieve.Count - 1]);

//Console.WriteLine("proximo Primo: " + proximoPrimo);

return new long[3] { todosImp, anteriorPrimo, proximoPrimo };

}

static void Main(string[] args)

{

//long N = 10000; // Must be less than MAX\_SIZE

//List<long> prime = manipulated\_seive(N);

//// pint all prime number less then N

//for (int i = 0; i < prime.Count && prime[i] <= N; i++)

// //cout << prime[i] << " ";

// Console.Write(prime[i] + " ");

//Console.WriteLine("Basic Tests OnlyOddDigPrimes");

//[number pure odd digit primes bellow n,

//largest pure odd digit prime smaller than n,

//smallest pure odd digit prime higher than n]

//long[] l = new long[] { 20, 40, 55, 60, 100 };

//long[][] r = new long[][] { new long[] {7, 19, 31}, new long[] {9, 37, 53}, new long[] {10, 53, 59},

//new long[] {11, 59, 71}, new long[] {15, 97, 113} };

//tests(l, r);

//only\_oddDigPrimes(20)---- > [7, 19, 31]

/////7, beacause we have seven pure odd digit primes bellow 20 and are 3, 5, 7, 11, 13, 17, 19

//19, because is the nearest prime of this type to 20

//31, is the first pure odd digit that we encounter after 20///

//only\_oddDigPrimes(40)---- > [9, 37, 53]

long[] r = OnlyOddDigPrimes(40);

Console.WriteLine(r[0] + " " + r[1] + " " + r[2]);

Console.WriteLine(TodosImpares(211));

Console.ReadLine();

}

}

}