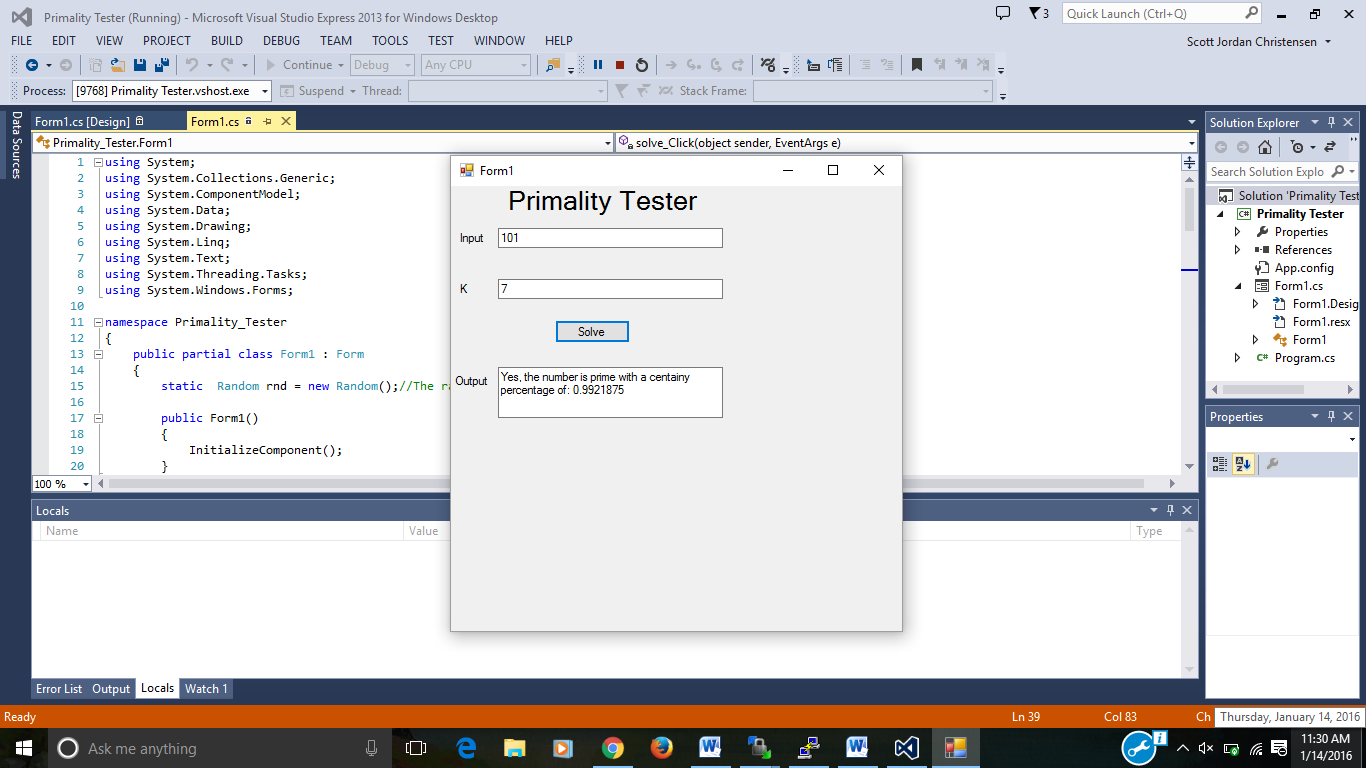
Scott Christensen

01/14/16

CS 312

Project 1 Submission



The program’s Big O is O(N\*M^3). In the main function, solve\_Click, all if does is call the primalityTester to run N times. Each loop in the primality tester calls the modExponential function when computing the proper number to mod, which takes O(M^3) time. Thus, multiplying these together preforms this whole Primality Tester algorithm in O(N\*M^3) time.

The function to compute the probability of correctness is pretty straightforward. It is computed to be 1-(1/2)^N, where N is the number of times you’d run the Primality test using a different A. Should the number fail the Primality Test, it would drop the correctness percentage to less than 50% and fail.

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Primality\_Tester

{

public partial class Form1 : Form

{

static Random rnd = new Random();//The random number used in the primaility testing.

public Form1()

{

InitializeComponent();

}

private void solve\_Click(object sender, EventArgs e)

{

long num = Convert.ToInt64(input.Text);

int timesTester = Convert.ToInt32(k.Text);

Console.WriteLine(num + " " + timesTester);

//Take number through Feridai's Little Theorem primality tester

double answer = primalityTester(num, timesTester);

Console.WriteLine("Answer is " + answer);

if(answer < 0.5)

{

output.Text = "No, the number is composite.";

}

else

{

output.Text = "Yes, the number is prime with a centainy percentage of: " + answer;

}

}

private double primalityTester(long candidate, int timesTester)

{

//We will run the primalityTester operation k number of times in order to increase our confidence in our operations

double error = 1;

for (int i = 0; i < timesTester; i++)

{

long randomNumber = 1;

int num = rnd.Next(1, (int)candidate);

randomNumber = (long)num;

Console.WriteLine("rand num is " + randomNumber);

Console.WriteLine(Math.Pow(randomNumber, candidate - 1));

//Tests to see if the numbers really are relatively prime while using a function to find a^n-1

if(modExpo(randomNumber, candidate - 1, candidate) == 1)

{

Console.WriteLine("yes, error is " + error + "\n");

error = error / 2;

}

else

{

return 0;

}

}

return 1 - error;

}

//This function will find what x^y is using the modular exponentiation algorithm.

long modExpo(long x, long y, long N)

{

Console.WriteLine(x + " " + y + " " + N);

if (y == 0) return 1;

long z = modExpo(x, y / 2, N);

if(y%2 == 0)//Sees if the number is even of not.

{

return ((long)Math.Pow(z, 2)) % N;

}

else

{

return (x \* ((long)Math.Pow(z, 2))) % N;

}

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void label3\_Click(object sender, EventArgs e)

{

}

private void label4\_Click(object sender, EventArgs e)

{

}

}

}