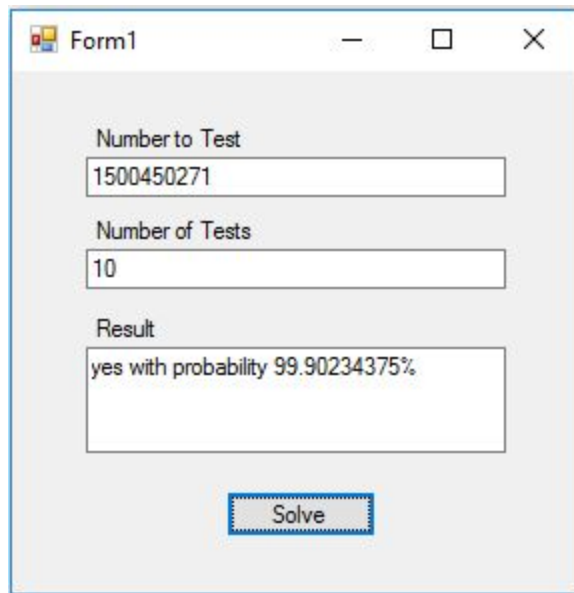


# Project 1 - Fermat Primality Test

Scott Leland Crossen

## 1. Screenshot



Form1

Number to Test  
1500450271

Number of Tests  
10

Result  
yes with probability 99.90234375%

Solve

## 2. Code

### 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 Fermat_Primality_Tester
{
    public partial class Form1 : Form
    {
        public Form1()
        {
```

```

InitializeComponent();
}

private void solve_Click(object sender, EventArgs e)
{ // Button was clicked. The following is now my code:
    int number = Convert.ToInt32(input.Text); // Takes the input number and stores it as int. O(c)
    int numtests = Convert.ToInt32(kvalue.Text); // Takes the inputted number of tests and stores. O(c)
    if (number < 0) // This case statement looks for bad input O(c)
    {
        output.Text = "Please select a positive integer";
    }
    else if (numtests > number - 1)
    {
        output.Text = "K value should be less than n-1";
    }
    else
    { // This block represents the algorithm that we use on valid input.
        Boolean prime = true;
        Random random = new Random(); // Generate random numbers for use as the base
        for (int iter = 1; iter <= numtests; iter++) // Iterate through the number of tests.
        { // Primality checker, if we checked all numbers its O(n) time
            // Use modular exponentiation as per requirement
            if (modexp(random.Next(2, number - 1), number - 1, number) != 1) prime = false;
        }
        if (prime == true) // The tests didn't show that the value was false.
        {
            double p = 1.0;
            for (int iter = 1; iter <= numtests; iter++)
            { // Calculates probability that it's prime. Better for more tests.
                p = (1.0 / 2.0) * p; // Make float
            }
            p = 100 * (1 - p);
            output.Text = "yes with probability " + p + "%";
        }
        else
        { // Yes and no output given in the if/else
            output.Text = "no";
        }
    }
}

private long modexp(int x, int y, int N)
{ // Modular Exponentiation O(n^3)
    if (y == 0) return 1; // Procedure described in book
    long z = modexp(x, Convert.ToInt32(Math.Floor(y / 2.0)), N);
    if (y % 2 == 0) return (z * z) % N;
    else return ((x % N) * ((z * z) % N)) % N;
}
}
}

```

### 3. Complexity

The time complexity of this code is not difficult to compute. The code is commented to include the complexity of specific procedures used. For example, the first section inputs the file prompts and stores them as variables. This and the output writing is  $O(c)$  time (as noted in comments). The next section is a loop that iterates twice through for every test amount specified by the user. This is  $O(n)$  time. The procedure that actually defines the complexity of the program is the modular exponentiation which runs in  $O(n^3)$  time as specified in class and in the manual. The modular exponentiation is the limiting complexity but if we wished to display a more approximate time step the equation  $f=A*n^3+A+7$  would be representative of this program where  $A$  is the number of times to test the modular exponentiation.

In case you want the short version: the complexity is limited at  $O(n^3)$

### 4. Discussion of Equation

The equation used to calculate the probability of correctness displayed in the output was

$$p = 100 \left( 1 - \frac{1}{2^k} \right)$$

The formula was chosen based on the lemma given in the book. It can be explained / represented in terms of the primality-test cases returned. For example, if  $N$  is likely prime then  $a^{N-1} \not\equiv 1$  for at least half the choices of  $a$  in the primality test. On the other hand if  $N$  is not likely prime then  $a^{N-1} \equiv 1 \pmod{N}$  for at most half the values of  $a < N$ . Thus our confidence-level is increased by a factor of  $\frac{1}{2}$  for each successive test we make. As discussed in class, the first test is the most informative and then each subsequent test raises our confidence level but with dimensioning returns.