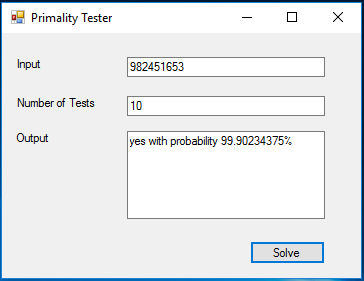
Joshua Van Steeter

Project 1: Primality Tester

1. Here is a screen shot of my working project



1. The code that I wrote:

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

// retrieve the inputs from the GUI

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

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

Random random = new Random();

int testsPassed = 0;

// run the number of tests specified in the input

// O(c)

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

{

// generate a random number between 2 and N

long a = random.Next(2, Convert.ToInt32(N));

// run the modular exponentiation function a^(N-1) % N

long result = modular\_exp(a, Convert.ToInt64(N - 1), N);

// if the result of modular\_exp is 1 then it passed the test, you may run another test,

// it may be prime

if (result == 1)

{

testsPassed++;

}

// if the result of modular\_exp is not 1 then N is not prime, display "no" and quit testing

else

{

output.Text = "no";

return;

}

}

// if all tests have passed, calculate the percent accuracy as 100 - (100/2^k) and display results

double percent = 100 - (100 / (Math.Pow(2, testsPassed)));

output.Text = "yes with probability " + Convert.ToString(percent) + "%";

}

private long modular\_exp(long value, long exponent, long N)

{

// the function for modular exponentiation

// for every test of solve\_Click() this function will fun log2(n) times which will halt after at most

// n recursive calls, at each call it multiplies n-bit numbers

// giving us a total run time of O(n^3)

if (exponent == 0)

{

// base case, when an exponent is 0 the result will always be 1

return 1;

}

// the recursive call

long z = modular\_exp(value, exponent / 2, N);

if (exponent % 2 == 0)

{

// if the exponent is even return z^2 mod N

long result = ((z \* z) % N);

return result;

}

else

{

// if the exponent is odd return x\*z^2 mod N

long result = ((value % N) \* ((z \* z) % N) % N);

return result;

}

}

1. The Fermat tests run in constant times, merely checking if the result of the modular exponentiation was equal to 1 or not, and doing this a constant number of times equal to the input number of tests. The modular exponentiation function runs in O(n3) time which I elaborate a little more on in comments in the code. Bringing our total run time to O(n3).
2. As each test is passed the likelihood for error decreases by 50%. Therefore the error to calculate percent probability of success out of 100% is 100 – (100/2^k)