**Crypto Hmwk 3**

**Due Wed Sept. 24th**

**Instructions:**

**\*Print out these pages.**

**\*Write your answers below.**

**\*Staple your computer programs and printouts to it.**

**Homework Assignment 3**

**1) Find the number of Pythagorean primes not exceeding 500. π1,4(500) = \_\_\_\_\_\_\_44\_\_\_\_\_\_\_\_\_\_.**

**2) Find π3,4(500) = \_\_\_\_\_50\_\_\_\_\_\_\_\_\_**

**3) Find π(500) = \_\_\_\_\_\_95\_\_\_\_\_\_\_\_\_**

**4) Verify that π1,4(500) + π3,4(500) = π(500).**

**The difference comes from the pi function counting 2 as a prime number. If we exclude counting 2, pi\_1,4(500) + pi\_3,4(500) = pi(500) = 94**

**5) Find πsafe(10,000), the number of safe primes not exceeding 10,000\_\_\_\_\_\_\_\_\_\_115\_\_\_\_\_\_\_\_\_\_\_.**

Code output:

**The number of Pythagorean primes not exceeding 500 is 44**

**pi\_3,4(500) is 50**

**pi(500) is 95**

**pi\_1,4(500) + pi\_3,4(500) is 94**

**pi(500) is 95**

**The difference comes from the pi function counting 2 as a prime number. If we exclude counting 2, pi\_1,4(500) + pi\_3,4(500) = pi(500) = 94**

**pi\_safe(10,000) is 115**

#include <stdio.h>

#include <iostream>

#include <math.h>

using namespace std;

//The following is a deterministic algorithm to determine primality. It is not practical for large numbers n.

int isPrime(int n) {

//returns the value 1 if n is a prime, the value 0 otherwise.

for(int d = 2; d\*d <= n; ++d) if(n%d == 0) return 0;

return 1;

}

//The following is an algorithm for pi(x) that uses the function isPrime(n).

int pi(double x) {

//cout << "Entered pi(x)" << endl;

//Returns the value of the number of primes not exceeding x.

int count = 1; //2 is the only even prime.

for(int j = 3; j <= x; j = j + 2) {

if (isPrime(j)) {

//cout << j << " is prime." << endl;

++count;

}

}

return count;

}

//The following algorithm uses isPrime(n) to determine whether a number is safe prime. The first few safe primes are 5, 7, 11, ...

int isSafePrime(int n) {

int flag = isPrime(n);

if(!flag) return 0; //if n isn't a prime, then it isn't a safe prime.

flag = isPrime((n-1)/2);

if(!flag) return 0; //if (n-1)/2 isn't prime, then n is an unsafe prime.

return 1;

}

//The following algorithm counts the number of Pythagorean primes. These are the primes that are congruent to 1 modulo 4. The first few are 5, 13, 17, 29, ...

int p14(double x) {

//cout << "Entered p14(x)" << endl;

int count = 0;

for(int j = 5; j <= x; j = j + 4)

if(isPrime(j)) {

//cout << j << " is 14 Prime." << endl;

++count;

}

return count;

}

//The following algorithm counts the number of Pythagorean primes. These are the primes that are congruent to 3 modulo 4. The first few are 3, 7, 11, ...

int p34(double x) {

//cout << "Entered p34(x)" << endl;

int count = 0;

for(int j = 3; j <= x+1; j = j + 4)

if(isPrime(j)) {

//cout << j << " is 34 Prime." << endl;

++count;

}

return count;

}

//The following algorithm finds the number of safe primes not exceeding x by using the function isSafePrime(n).

int piSafe(double x) {

//cout << "Entered piSafe(x)" << endl;

//This will count the number of safe primes, beginning with 7.

int count = 0;

for(int j = 5; j <= x; j = j + 2)

if(isSafePrime(j)) ++count;

return count;

}

int main() {

//HW3 Problem 1

//Find the number of Pythagorean primes not exceeding 500.

int n = 500;

int np14 = p14(n);

cout << "The number of Pythagorean primes not exceeding 500 is " << np14 << endl;

//HW3 Problem 2

//Find pi\_3,4(500)

int np34 = p34(n);

cout << "pi\_3,4(500) is " << np34 << endl;

//HW3 Problem 3

//Find pi(500)

int np = pi(n);

cout << "pi(500) is " << np << endl;

//HW3 Problem 4

//Verify that pi\_1,4(500) + pi\_3,4(500) = pi(500)

cout << "pi\_1,4(500) + pi\_3,4(500) is " << np14+np34 << endl;

cout << "pi(500) is " << np << endl;

cout << "The difference comes from the pi function counting 2 as a prime number. If we exclude counting 2, pi\_1,4(500) + pi\_3,4(500) = pi(500) = 94" << endl;

//HW4 Problem 5

//Find pi\_safe(10,000)

int npsafe = piSafe(10000);

cout << "pi\_safe(10,000) is " << npsafe << endl;

return 0;

}