Full name: Ariel Ya'acobi

I.D: 318727187

Assignment 0 - Goldbach Conjecture

In this first assignment, the main task is to define a detailed algorithm (using free text and pseudocode) for testing the Goldbach Conjecture efficiently.

The "Goldbach Conjecture" states that every even natural number greater than 2 is the sum of two prime numbers.

Given a natural (even) number n (n>4), required for us to do the following:

A. Find two prime numbers (p1,p2) such that p1+p2==n, and print it according to the example below.

B. Compute (and print) how many prime numbers are between [2…n).

Meaning(The actual assignment):

1)Write in your own words the required functionality (both requirements 1,2)

2)Write a related pseudo-code for both requirements 1,2.

In addition the following notes are given:

***The IsPrime pseudo-code that is given to you below - can be improved significantly**
***You need to count all the primes in the range [2,n). Yet, you may print only the first 100 primes (case n>540).**

Example: for writing pseudocode for the **is-Prime** Algorithm**:**
This algorithm gets a natural number (n) and tests if n is a prime number (or not). The algorithm works by testing all the natural numbers in the range [2,n-1] if any of them divides n, if so, n is not a prime number.

```
Input(n >1)          // assuming n is a natural number n>0
ans =  true          // ans is the variable representing “is n a prime”
i = 2                 // init value of I is 2 (the divider)
while (i<n) {         // loop over all number 2<=i<n
  t = n%i            // t is the value of n mod i.
  if (t=0) {       // if t is zero, n is NOT a prime number (I divide it)
    ans = false       //change the value of ans to false (not prime).
  } // if          // the end of the “if” block
  i = i+1         // increment i by 1.
} // while            // the end of the “while” block
return (ans)          // returns the answer of the algorithm
```

solution to the assignment:

1. In order to describe the required functionality i will have to explain each one separately step by step.

A) To Find two prime numbers (p1, p2) such that p1 + p2 = n:

The algorithm can start with the smallest possible prime numbers and work upwards(meaning i=2)

I will set up a loop that will iterate from 2 up to n/2, as the largest prime factor of n cannot exceed n/2. So, the loop can run till n/2(this way the code is more effective).

Within the loop, for each number i, check if both i and n - i are prime numbers – By calling the isPrime function that was given to us.

If both i and n - i are primes, print these numbers as the pair of primes that sum up to n.

B) Counting prime numbers between [2...n]:

The code counts the total number of prime numbers within the range from 2 up to a given value n(by a while loop).

It iterates through each number from 2 to n and checks if the number is prime – calling the "is-Prime" function.

For every prime number found within this range, it increments a counter variable (count).

If the value of n exceeds 540, it also tracks the count of the first 100 prime numbers encountered within the range from 2 to n.

This is done concurrently while counting all prime numbers within the range.

It ensures that if n surpasses 540, the code stops counting after identifying the first 100 prime numbers and proceeds with counting all primes up to n.

2. related pseudo-code for both requirements 1,2:

A) for requirements 1(Find two prime numbers (p1,p2) such that p1+p2==n, and print it):

```
Input(n > 4 and even)            // Ensure n is a natural number greater than 4 and even
 i = 2                     // Initialize i to 2 (the smallest prime number)
 found = False                // Initialize found flag (Variable) to False
 while i <= n/2 and not found:      // Loop until i reaches n/2 or a solution is found. More efficient this way than i<=n
    if is-Prime(i) and is-Prime(n - i):  // Check if both i and (n - i) are primes by calling the is-Prime function.
       print(n + " = " + i + " + " + (n - i))  // Print the pair of primes
       found = True             // Set found flag to True as a solution is found
    i = i + 1                 // Increment i for the next iteration
 if not found:
    print("No such pair exists")     // If no pair of primes found, print a message
```

B) for requirements 2(Compute (and print) how many prime numbers are between [2…n):

```
count = 0                  // Initialize count of primes to 0
prime_count = 0               // Initialize count for the first 100 primes
print_max = 100               // Set threshold for counting the first 100 primes
num = 2                    // Start with the first number 2
while num <= n:               // Loop through numbers from 2 to n (inclusive)
   if is-Prime(num):           // Check if num is a prime using is-prime function given to us
      count = count + 1         // Increment count if num is prime
      if n > 540 and prime_count < print_max:  // If n > 540, count the first 100 primes
         prime_count = prime_count + 1 // Increment count for the first 100 primes
         if prime_count = print_max:  // Break when 100 primes are counted
            break*
   num = num + 1              // Move to the next number
return count                // Return the count of primes up to n
```

Extra – improving the "IsPrime" pseudo-code:

```
is_prime(n):
   Input(n > 1)               // Ensure n is a natural number greater than 1
   is_prime = true              // Initialize the variable to assume n is prime
   if (n <= 1) {              // Check if n is less than or equal to 1
        is_prime = false          // If so, it's not a prime number
        } else if (n <= 3) {        // Check if n is 2 or 3
             is_prime = true          // 2 and 3 are prime numbers. So the flag variable now is True.
        } else {
             i = 2               // Initialize the divider i to 2
             while (i*i <= n) {        // Loop through all numbers up to sqrt(n)
                  if (n % i == 0) {      // If i divides n evenly
                         is_prime = false    // n is not a prime number. Flag variable now is False.
                         break*          // Exit the loop
                  }
                  i = i + 1           // Move to the next potential divisor
             } // while
   }
   return is_prime              // Return the answer of the algorithm.
```

***I used break statement to exit the loop immediately after finding the pair and not to run more times, this optimizes the code.**