Dane jest poniższa implementacja algorytmu badania czy zadana liczba jest pierwsza:

```
bool IsPrime(BigInteger Num)
{
  if (Num < 2) return false;
  else if (Num < 4) return true;
  else if (Num % 2 == 0) return false;
  else for (BigInteger u = 3; u < Num / 2; u += 2)
            if (Num % u == 0) return false;
  return true;
}
```

Celem projektu jest zaproponowanie bardziej efektywnego algorytmu przy zachowaniu niezmienionego interfejsu podprogramu. Przeprowadzić analizę za pomocą instrumentacji i pomiarów czasu. Przyjąć, że operacją dominującą jest dzielenie modulo (%).

W sprawozdaniu przedstawić dla obu algorytmów:

- kod źródłowy przed instrumentacją
- kod źródłowy po instrumentacji
- zebrane wyniki w postaci tekstu i wykresów
- wnioski z analizy zebranych danych (ocena złożoności)

Badanie przeprowadzić dla następującego zbioru punktów pomiarowych (liczb pierwszych):
{ 100913, 1009139, 10091401, 100914061, 1009140611, 10091406133, 100914061337, 1009140613399 }

1. *Przykładowy algorytm*

*W tym algorytmie obliczanie dwóch ostatnich liczb pierwszych trwało zbyt długo i wartość „Counter” i „Time” zostały obliczone metodą proporcji

- Implementacja metody obliczającej, czy liczba jest pierwsza bez instrumentacji:

```
private static bool IsPrime(BigInteger Num)
{
    if (Num < 2) return false;
    else if (Num < 4) return true;
    else if (Num % 2 == 0) return false;
    else
    {
        for (BigInteger u = 3; u < Num / 2; u += 2)
        {
            if (Num % u == 0) return false;
        }
    }
    return true;
}
```

- Implementacja metody obliczającej, czy liczba jest pierwsza z instrumentacją:

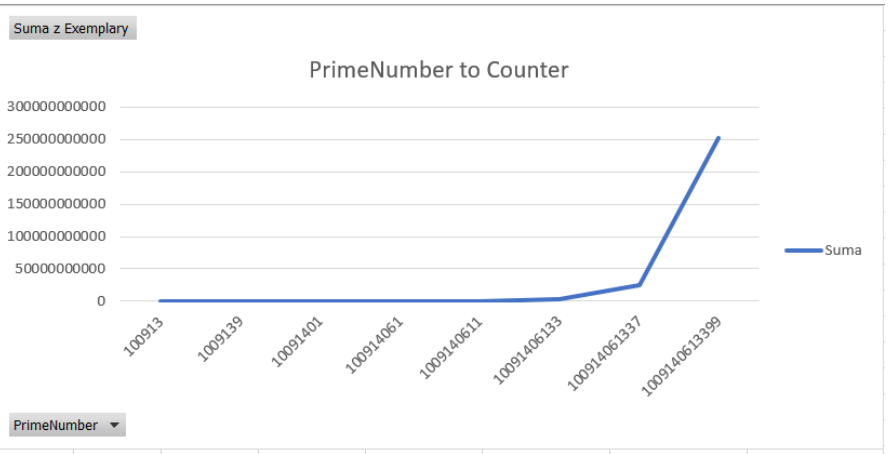
```
private static bool IsPrimeInstr(BigInteger Num)
{
    Counter = 0;
    if (Num < 2) return false;
    else if (Num < 4) return true;
    else if (Num % 2 == 0) { Counter++; return false; }
    else
    {
        for (BigInteger u = 3; u < Num / 2; u += 2)
        {
            Counter++;
            if (Num % u == 0) return false;
        }
    }
    return true;
}
```

- Metoda main wykorzystana w algorytmie przykładowym:

```
private static readonly int Iter = 10;
private static ulong Counter;
static void Main(string[] args)
{
    long min = long.MaxValue;
    long max = long.MinValue;
    long timeElapsed = 0;
    long iterTimeElapsed;
    long start;
    long stop;
    double elapsedSeconds;
    bool isPrimeNumber;
    string isPrimeNumberString;
    BigInteger n;
    BigInteger[] primeNumbers = new BigInteger[] { 100913, 1009139, 10091401, 100914061, 1009140611, 10091406133, 100914061337, 1009140613399 };
    Console.WriteLine("PrimeNumber;Exemplary;Time;IsPrime");
    for(int i = 0; i < primeNumbers.Length; i++)
    {
        n = primeNumbers[i];
        for (int j = 0; j < Iter + 2; ++j)
        {
            start = Stopwatch.GetTimestamp(); // start time
            IsPrime(n); // calling function IsPrime without counter
            stop = Stopwatch.GetTimestamp();   // stop time
            iterTimeElapsed = stop - start;
            timeElapsed += iterTimeElapsed;
            if (iterTimeElapsed < min) min = iterTimeElapsed;
            if (iterTimeElapsed > max) max = iterTimeElapsed;
        }
        timeElapsed = (max - min);
        isPrimeNumber = IsPrimeInstr(n); ; // calling function IsPrime with counter
        elapsedSeconds = timeElapsed * (1.0 / (Iter * Stopwatch.Frequency));
        if (isPrimeNumber == true) isPrimeNumberString = "Prime";
        else isPrimeNumberString = "Not Prime";
        Console.WriteLine(n + ";" + Counter + ";" + (elapsedSeconds.ToString("F6")) + ";" + isPrimeNumberString); // Prime Number; Counter; Time; Is Prime
    }
}
```

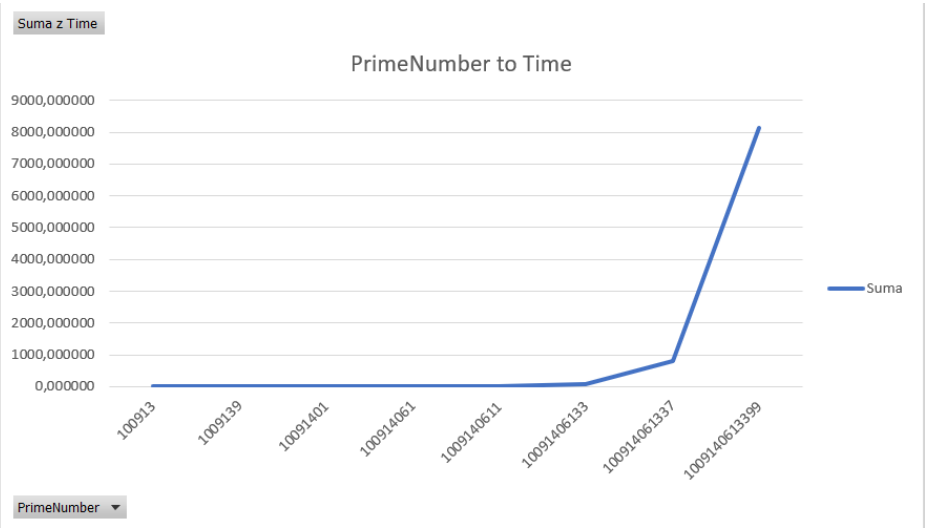
a) Zebrane dane podczas instrumentacji:

Etykiety wierszy	Suma z Exemplary
100913	25227
1009139	252283
10091401	2522849
100914061	25228514
1009140611	252285151
10091406133	2522851532
100914061337	25228515322
1009140613399	252285153225
Suma końcowa	**280316834103**



b) Zebrane dane podczas pomiaru czasu:

Etykiety wierszy	Suma z Time
100913	0,000039
1009139	0,003886
10091401	0,040416
100914061	0,422036
1009140611	4,289907
10091406133	81,463449
100914061337	814,634490
1009140613399	8146,344901
Suma końcowa	**9047,199124**



2. ***Przyzwoity algorytm***

- Implementacja metody obliczającej, czy liczba jest pierwsza bez instrumentacji:

```
private static bool IsPrime(BigInteger Num)
{
    if (Num < 2) return false;
    else if (Num < 4) return true;
    else if (Num % 2 == 0) return false;
    else
    {
        for (BigInteger u = 3; (u * u) <= Num; u += 2)
        {
            if (Num % u == 0) return false;
        }
    }
    return true;
}
```

- Implementacja metody obliczającej, czy liczba jest pierwsza z instrumentacją:

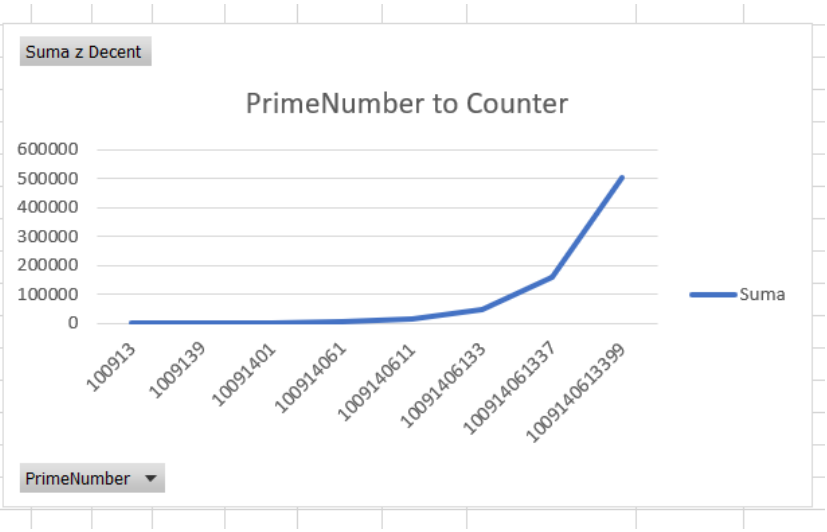
```
private static bool IsPrimeInstr(BigInteger Num)
{
    Counter = 0;
    if (Num < 2) return false;
    else if (Num < 4) return true;
    else if (Num % 2 == 0) { Counter++; return false; }
    else
    {
        for (BigInteger u = 3; (u * u) <= Num; u += 2)
        {
            Counter++;
            if (Num % u == 0) return false;
        }
    }
    return true;
}
```

- Metoda main wykorzystana w algorytmie przyzwoitym:

```
private static readonly int Iter = 10;
private static ulong Counter;
static void Main(string[] args)
{
    long min = long.MaxValue;
    long max = long.MinValue;
    long timeElapsed = 0;
    long iterTimeElapsed;
    long start;
    long stop;
    double elapsedSeconds;
    bool isPrimeNumber;
    string isPrimeNumberString;
    BigInteger n;
    BigInteger[] primeNumbers = new BigInteger[] { 100913, 1009139, 10091401, 100914061, 1009140611, 10091406133, 100914061337, 1009140613399 };
    Console.WriteLine("PrimeNumber;Decent;Time;IsPrime");
    for (int i = 0; i < primeNumbers.Length; i++)
    {
        n = primeNumbers[i];
        for (int j = 0; j < Iter + 2; ++j)
        {
            start = 0;
            stop = 0;
            start = Stopwatch.GetTimestamp(); // start time
            IsPrime(n); // calling function IsPrime without counter
            stop = Stopwatch.GetTimestamp();   // stop time
            iterTimeElapsed = stop - start;
            timeElapsed += iterTimeElapsed;
            if (iterTimeElapsed < min) min = iterTimeElapsed;
            if (iterTimeElapsed > max) max = iterTimeElapsed;
        }
        timeElapsed = (max - min);
        elapsedSeconds = timeElapsed * (1.0 / (Iter * Stopwatch.Frequency));
        isPrimeNumber = IsPrimeInstr(n); ; // calling function IsPrime with counter
        if (isPrimeNumber == true) isPrimeNumberString = "Prime";
        else isPrimeNumberString = "Not Prime";
        Console.WriteLine(n + ";" + Counter + ";" + (elapsedSeconds.ToString("F6")) + ";" + isPrimeNumberString); // Prime Number; Counter; Time; Is Prime
    }
}
```

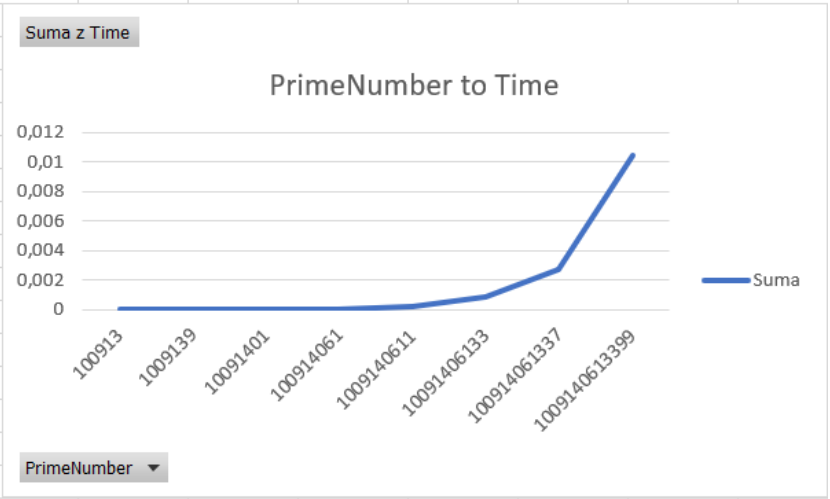
c) Zebrane dane podczas instrumentacji:

Etykiety wierszy	Suma z Decent
100913	158
1009139	501
10091401	1587
100914061	5022
1009140611	15882
10091406133	50227
100914061337	158834
1009140613399	502279
Suma końcowa	**734490**



d) Zebrane dane podczas pomiaru czasu:

Etykiety wierszy	Suma z Time
100913	0,000027
1009139	0,000027
10091401	0,000027
100914061	0,000077
1009140611	0,000215
10091406133	0,000899
100914061337	0,002694
1009140613399	0,010461
Suma końcowa	**0,014427**



3. ***Optymalny algorytm***

- Implementacja metody obliczającej, czy liczba jest pierwsza bez instrumentacji:

```
private static bool IsPrime(BigInteger Num)
{
    if (Num < 2) return false;
    else if (Num == 2) return true;
    else if (Num == 3) return true;
    else if (Num == 5) return true;
    else if (Num == 7) return true;
    else if (Num % 2 == 0 || Num % 3 == 0 || Num % 5 == 0) return false;
    else if (Num <= 30) return IsPrime2(Num);
    else
    {
        for (BigInteger u = 7; (u * u) <= Num; u += 30)
        {
            if (Num % u == 0 || Num % (u + 4) == 0 || Num % (u + 6) == 0 || Num % (u + 10) == 0 || Num % (u + 12) == 0
                || Num % (u + 16) == 0 || Num % (u + 22) == 0 || Num % (u + 24) == 0) return false;
        }
    }
    return true;
}
private static bool IsPrime2(BigInteger Num)
{
    if (Num < 2) return false;
    else if (Num < 4) return true;
    else if (Num % 2 == 0) return false;
    else
    {
        for (BigInteger u = 3; (u * u) <= Num; u += 2)
        {
            if (Num % u == 0) return false;
        }
    }
    return true;
}
```

- Implementacja metody obliczającej, czy liczba jest pierwsza z instrumentacją:

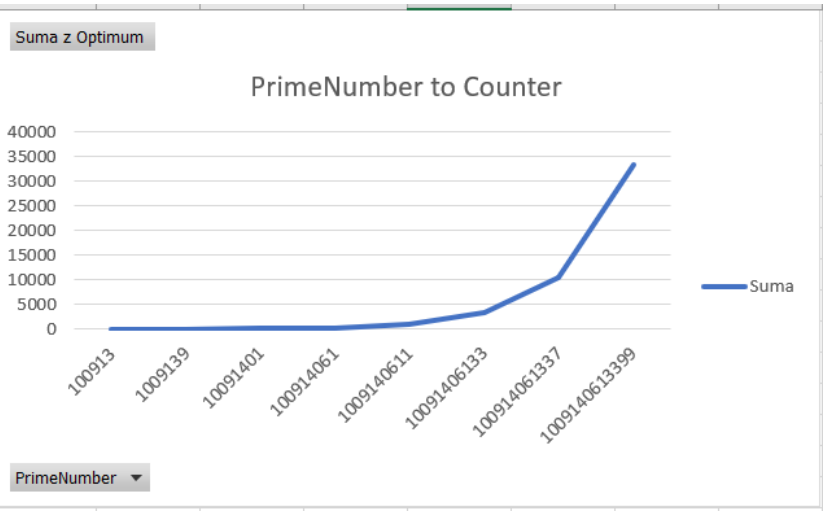
```
private static bool IsPrimeInstr2(BigInteger Num)
{
    if (Num < 2) return false;
    else if (Num < 4) return true;
    else if (Num % 2 == 0) { Counter++; return false; }
    else
    {
        for (BigInteger u = 3; (u * u) <= Num; u += 2)
        {
            Counter++;
            if (Num % u == 0) return false;
        }
    }
    return true;
}
private static bool IsPrimeInstr(BigInteger Num)
{
    Counter = 0;
    if (Num < 2) return false;
    else if (Num == 2) return true;
    else if (Num == 3) return true;
    else if (Num == 5) return true;
    else if (Num == 7) return true;
    else if (Num % 2 == 0 || Num % 3 == 0 || Num % 5 == 0) { Counter++; return false; }
    else if (Num <= 30) return IsPrimeInstr2(Num);
    else
    {
        for (BigInteger u = 7; (u * u) <= Num; u += 30)
        {
            Counter++;
            if (Num % u == 0 || Num % (u + 4) == 0 || Num % (u + 6) == 0 || Num % (u + 10) == 0 || Num % (u + 12) == 0
                || Num % (u + 16) == 0 || Num % (u + 22) == 0 || Num % (u + 24) == 0) return false;
        }
    }
    return true;
}
```

- Metoda main wykorzystana w algorytmie Optymalnym:

```
private static readonly int Iter = 10;
private static ulong Counter;
static void Main(string[] args)
{
    long min = long.MaxValue;
    long max = long.MinValue;
    long timeElapsed = 0;
    long iterTimeElapsed;
    long start;
    long stop;
    double elapsedSeconds;
    bool isPrimeNumber;
    string isPrimeNumberString;
    BigInteger n;
    BigInteger[] primeNumbers = new BigInteger[] { 100913, 1009139, 10091401, 100914061, 1009140611, 10091406133, 100914061337, 1009140613399 };
    Console.WriteLine("PrimeNumber;Optimum;Time;IsPrime");
    for (int i = 0; i < primeNumbers.Length; i++)
    {
        n = primeNumbers[i];
        for (int j = 0; j < Iter + 2; ++j)
        {
            start = Stopwatch.GetTimestamp(); // start time
            IsPrime(n); // calling function IsPrime without counter
            stop = Stopwatch.GetTimestamp();   // stop time
            iterTimeElapsed = stop - start;
            timeElapsed += iterTimeElapsed;
            if (iterTimeElapsed < min) min = iterTimeElapsed;
            if (iterTimeElapsed > max) max = iterTimeElapsed;
        }
        timeElapsed = (max - min);
        isPrimeNumber = IsPrimeInstr(n); ; // calling function IsPrime with counter
        elapsedSeconds = timeElapsed * (1.0 / (Iter * Stopwatch.Frequency));
        if (isPrimeNumber == true) isPrimeNumberString = "Prime";
        else isPrimeNumberString = "Not Prime";
        Console.WriteLine(n + ";" + Counter + ";" + (elapsedSeconds.ToString("F6")) + ";" + isPrimeNumberString); // Prime Number; Counter; Time; Is Prime
        //Sito(primeNumbers[i]);
    }
}
```

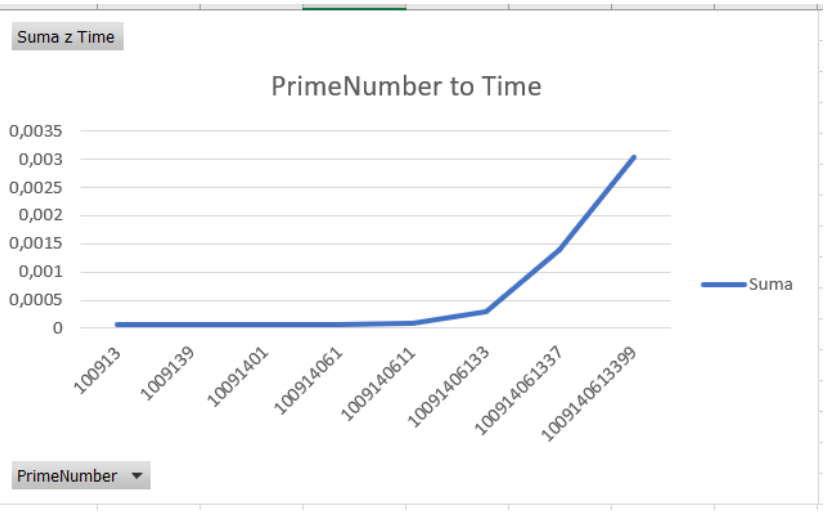
e) Zebrane dane podczas instrumentacji:

Etykiety wierszy	Suma z Optimum
100913	11
1009139	34
10091401	106
100914061	335
1009140611	1059
10091406133	3349
100914061337	10589
1009140613399	33486
Suma końcowa	**48969**



f) Zebrane dane podczas pomiaru czasu:

Etykiety wierszy	Suma z Time
100913	0,000066
1009139	0,000066
10091401	0,000066
100914061	0,000066
1009140611	0,000079
10091406133	0,000304
100914061337	0,001391
1009140613399	0,003046
Suma końcowa	**0,005084**



Eksperymenty zostały przeprowadzone na komputerze wyposażonym w procesor Intel Core i7-5600U, w Visual Studio 2017.

Podsumowanie badań:

- Przykładowy algorytm jest bardzo wolny i sprawdzi on się dla małych liczb, jego wydajność nie jest zadawalająca

- Przyzwoity algorytm jest znacznie wydajniejszy niż przykładowy, ale przy dużych liczbach znacznie maleje jego wydajność

- Optymalny algorytm jest znacznie wydajniejszy niż przykładowy oraz przyzwoity, na pewno lepiej się nadaje do sprawdzania dużych liczb pierwszych, bo uzyskujemy coraz mniejszą złożoność tego algorytmu, a co za tym idzie mniejsza liczba modulo oraz czasy są bardziej zadawalające.

Prace wykonał Denis Grabiszewski Grupa K35.2