Piotr Waszak – Projekt 2

Cel projektu: Zaproponować bardziej efektywny algorytm sprawdzający czy liczba jest pierwsza przy zachowaniu niezmienionego interfejsu podprogramu. Przeprowadzić analizę za pomocą instrumentacji i pomiarów czasu. Przyjąć, że operacją dominującą jest dzielenie modulo (%).

Liczba pierwsza jest to liczba naturalna większa od jeden, która ma tylko dwa dzielniki jedynkę, oraz siebie samą.

Algorytm przykładowy:

```
public static bool isPrime_exampleTime(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;
        }
```

Opis algorytmu podanego w zadaniu, na którym został wykonany pomiar czasu:
Po wywołaniu funkcja isPrime, ma za zadanie zwrócić True lub False.
1. Pierwsza linijka w klamrach sprawdza liczbę w nawiasie Czy jest mniejsza od 2 (ustalono, że liczba pierwszą nie jest ani 0, ani 1).
2. następnie czy liczba jest mniejsza od 4 (ponieważ liczba 3 jest liczbą pierwszą zwracane jest True).
3. W trzeciej linijce sprawdzamy wynik dzielenia modulo, jeśli liczba po dzieleniu przez 2, ma reszty 0 to znaczy, że jest liczbą parzystą.

4. Pętla for zaczynając od liczby nieparzystej 3, zaczyna sprawdzać pokolei każdą następną liczbę nieparzystą, do momentu najwyższej liczby, mniejszej od ilorazy liczby przez 2. Jeśli w którymś momencie reszta z dzielenia, dała wynik 0, instrukcja zwracała false, jeśli nie otrzymywaliśmy liczbę pierwszą.

Algorytm zaproponowany przeze mnie:

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

Opis:
Początek jest taki sam, ponieważ interfejs podprogramu miał zostać zachowany.
Przechodząc do pętli for, zaczynamy tak jak poprzednio liczenie od liczby nieparzystej, do momenty najwyższej potęgi „i" mniejszej lub równej liczbie sprawdzanej. Wynika to z stwierdzenia przedstawionego w książce prof. Macieja Sysło „Piramidy, szyszki i inne konstrukcje algorytmiczne":
„Każda liczba złożona n ma czynnik p spełniający nierówność $p \leq \sqrt{n}$".

Podane algorytmy zostały poddane pomiarowi czasu przez osobne funkcje.
Pomiar czasu dla algorytmu przykładowego:

```
static void exampleTime(BigInteger Num)
        {
            double ElapsedSeconds = 0;
            long StartingTime = Stopwatch.GetTimestamp();
            bool isPrimeisTrue = isPrime_exampleTime(Num);
            long EndingTime = Stopwatch.GetTimestamp();

            long ElapsedTime = EndingTime - StartingTime;
            ElapsedSeconds = ElapsedTime * (1.0 / Stopwatch.Frequency);

            Console.Write(Num+"\t"+isPrimeisTrue+"\t"+ElapsedSeconds.ToString("F4"));
        }
```

1. W pierwszej linijce, wewnatrz funkcji, zostaje wyzerowany czas.
2. Tworzymy zmienną StartingTime, do której przypisujemy pobrany czas z funkcji Stopwatch.GetTimestamp(), która zatrzymuje zegar czasu bijącego od 1 stycznia 1970 r.

3. Następnie wywołujemy funkcje, której wynik boolowski przypisujemy do utworzonej zmiennej.
4. W trzeciej linijce do zmiennej EndingTime przypisujemy nowy wynik czasu, który zatrzymał się po zakończonym działaniu funkcji.
5. W ostatnich krokach przypisujemy różnice miedzy pobranymi wynikami, i mnożymy ją z częstotliwościa, aby otrzymać czytelny wynik.

Pomiar czasu dla algorytmu przyzwoitego wygląda tak samo, z taką różnica że wywołuje funkcje algorytmu lepszego:

```
static void betterTime(BigInteger Num)
        {
            double ElapsedSeconds = 0;
            long StartingTime = Stopwatch.GetTimestamp();
            bool isPrimeisTrue = isPrime_betterTime(Num);
            long EndingTime = Stopwatch.GetTimestamp();

            long ElapsedTime = EndingTime - StartingTime;
            ElapsedSeconds = ElapsedTime * (1.0 / Stopwatch.Frequency);

            Console.Write("\t" + isPrimeisTrue + "\t" +ElapsedSeconds.ToString("F4"));
        }
```

Algorytm przykładowy z instumentacją:

```
public static bool isPrime_exampleInstr(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)
                {
                    ++OpComparisonEQ;
                    if (Num % u == 0) return false;
                }


            return true;
        }
```

Do poprzedniego algorytmu, został zaimplementowany licznik w postaci preinkrementacji zmiennej globalnej OpComparisonEQ. Algorytm wywoływany jest poprzez funkcje exampleInstr:

```
static void exampleInstr(BigInteger Num)
        {
            OpComparisonEQ = 1;
            bool isPrimeisTrue = isPrime_exampleInstr(Num);

            Console.Write("\t" + OpComparisonEQ+"\t");
        }
```

Zmienna globalna OpComparsion zostaje zresetowana, następnie zostaje wywołana funkcja, w której aktywuje się instrumentacja.

Algorytm ulepszony z instrumentacją.

```
public static bool isPrime_betterInstr(BigInteger Num)
        {
            if (Num < 2) return false;
            else if (Num < 4) return true;
            else if (Num % 2 == 0) return false;
            for (BigInteger i = 3; i * i  <= Num ; i+=2)
            {
                OpComparisonEQ++;
                if (Num % i == 0) return false;
            }
            return true;
        }
```

Tak samo jak poprzednio został wstawiony licznik OpComparisonEQ, a funkcja zostaje wywolana z funkcji betterInstr.

```
static void betterInstr(BigInteger Num)
        {
            OpComparisonEQ = 1;
            bool isPrimeisTrue = isPrime_betterInstr(Num);

            Console.WriteLine("\t" + OpComparisonEQ);
        }
```

Cały kod:

```
using System;
using System.Numerics;
using System.Diagnostics;

namespace LiczbyPierwsze
{
    class Program
    {
        public static BigInteger[] primeNumber = { 1619, 100913, 1009139, 10091401,
100914061, 1009140611, 10091406133, 100914061337, 1009140613399 };
        static BigInteger OpComparisonEQ;

        static void Main(string[] args)
        {
            Console.WriteLine("\tnumber \tisPrime? \texampleTime \texampleInstr
\tBetterisPrime? \tbetterTime \tbetterInstr");
            foreach (BigInteger item in primeNumber)
            {
                exampleTime(item);
                exampleInstr(item);
                betterTime(item);
                betterInstr(item);

            }
            Console.Write("\n");

        }
```

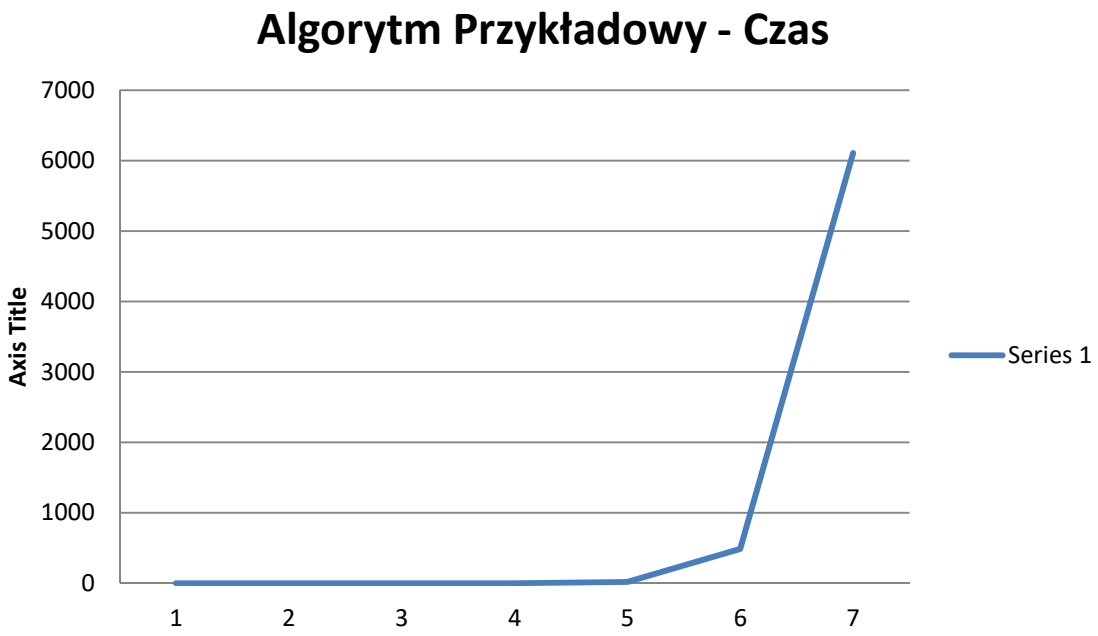
```
        // Algorytm sprawdzający czy liczba jest pierwsza - Algorytm przykładowy
        public static bool isPrime_exampleTime(BigInteger Num)
        {
            if (Num < 2) return false;
            else if (Num < 4) return true;
            else if (Num % 2 == 0) return false;
            else for (BigInteger u = 3; u < Num / 2; u += 2)
                    if (Num % u == 0) return false;

            return true;
        }
        // Algorytm sprawdzajacy czy liczba jest pierwsza - Instrumentacja algorytm
przykładowy
        public static bool isPrime_exampleInstr(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)
                {
                    ++OpComparisonEQ;
                    if (Num % u == 0) return false;
                }


            return true;
        }
        // funkcja mierząca czas potrzebny do sprawdzenia czy liczba jest pierwsza -
algorytmu przykładowego
        static void exampleTime(BigInteger Num)
        {
            double ElapsedSeconds = 0;
            long StartingTime = Stopwatch.GetTimestamp();
            bool isPrimeisTrue = isPrime_exampleTime(Num);
            long EndingTime = Stopwatch.GetTimestamp();

            long ElapsedTime = EndingTime - StartingTime;
            ElapsedSeconds = ElapsedTime * (1.0 / Stopwatch.Frequency);

            Console.Write(Num+"\t"+isPrimeisTrue+"\t"+ElapsedSeconds.ToString("F4"));
        }
        // funkcja mierząca ilosc powtórzeń potrzebnych do sprawdzenia czy liczba jest
pierwsza - algorytmu przykładowego
        static void exampleInstr(BigInteger Num)
        {
            OpComparisonEQ = 1;
            bool isPrimeisTrue = isPrime_exampleInstr(Num);

            Console.Write("\t" + OpComparisonEQ+"\t");
        }

        // Algorytm sprawdzający czy liczba jest pierwsza - Algorytm ulepszony
        public static bool isPrime_betterTime(BigInteger Num)
        {
            if (Num < 2) return false;
            else if (Num < 4) return true;
            else if (Num % 2 == 0) return false;
            for (BigInteger i = 3; i * i <= Num; i+=2)
            {
                if (Num % i == 0) return false;
            }
```

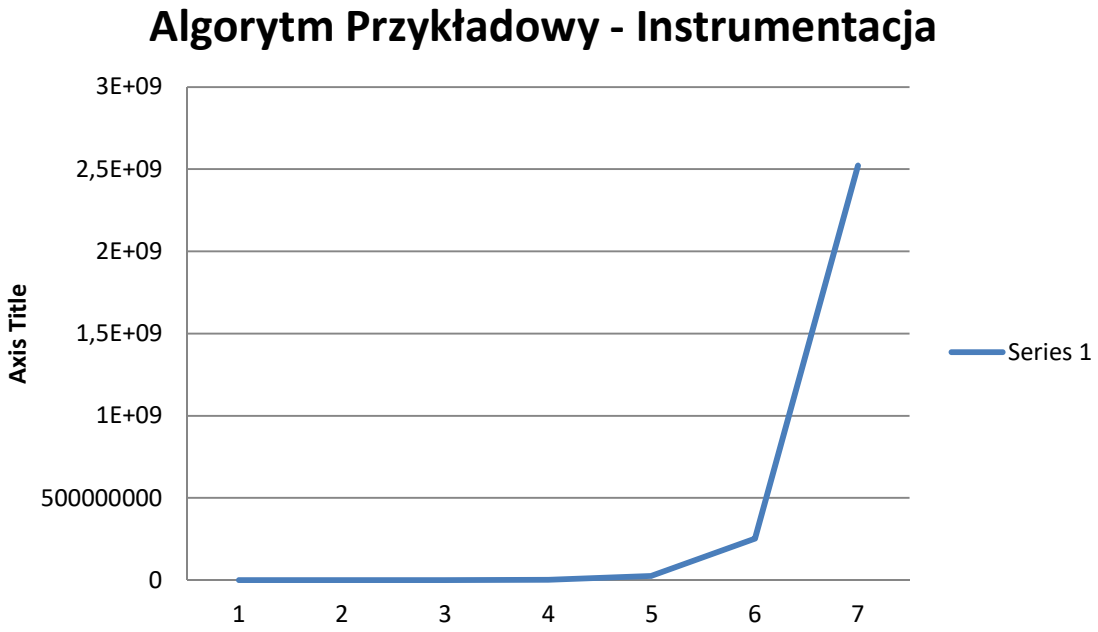
```
            return true;
        }
        // Algorytm sprawdzający czy liczba jest pierwsza - instrumentacja algorytmu
ulepszonego
        public static bool isPrime_betterInstr(BigInteger Num)
        {
            if (Num < 2) return false;
            else if (Num < 4) return true;
            else if (Num % 2 == 0) return false;
            for (BigInteger i = 3; i * i  <= Num ; i+=2)
            {
                OpComparisonEQ++;
                if (Num % i == 0) return false;
            }
            return true;
        }
        static void betterTime(BigInteger Num)
        {
            double ElapsedSeconds = 0;
            long StartingTime = Stopwatch.GetTimestamp();
            bool isPrimeisTrue = isPrime_betterTime(Num);
            long EndingTime = Stopwatch.GetTimestamp();

            long ElapsedTime = EndingTime - StartingTime;
            ElapsedSeconds = ElapsedTime * (1.0 / Stopwatch.Frequency);

            Console.Write("\t" + isPrimeisTrue + "\t" +
ElapsedSeconds.ToString("F4"));
        }
        // Algorytm sprawdzający ilość powtórzeń potrzebnych do sprawdzenia czy liczba
jest pierwsza - Algorytm ulepszony
        static void betterInstr(BigInteger Num)
        {
            OpComparisonEQ = 1;
            bool isPrimeisTrue = isPrime_betterInstr(Num);

            Console.WriteLine("\t" + OpComparisonEQ);
        }
    }
}
```

Liczba „1619” nie jest brana pod uwagę w eksperymencie – jest liczbą testową, która zapewnie prawidłowe wyniki pozostałych liczb.

Wyniki dla algorytmu przykładowego:

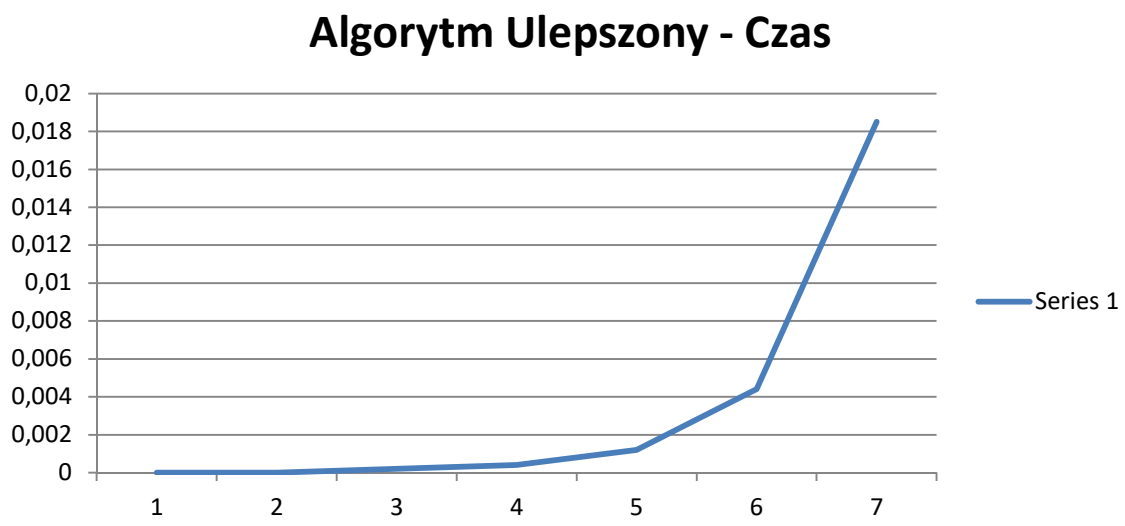
Liczba	Czy jest pierwsza?	Czas	Instrumentacja
100913	True	0,0020	25228
1009139	True	0,0236	252284
10091401	True	0,2338	2522850
100914061	True	2,1547	25228515
1009140611	True	20,0838	252285152
10091406133	True	487,8335	2522851533
100914061337	True	6109,7355	25228515334

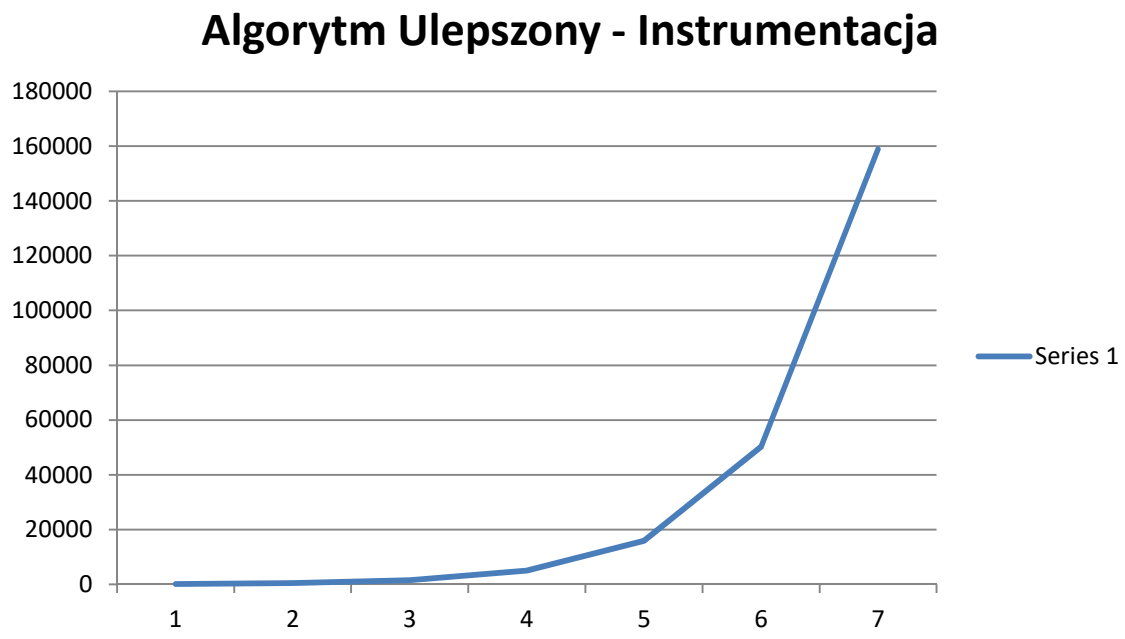




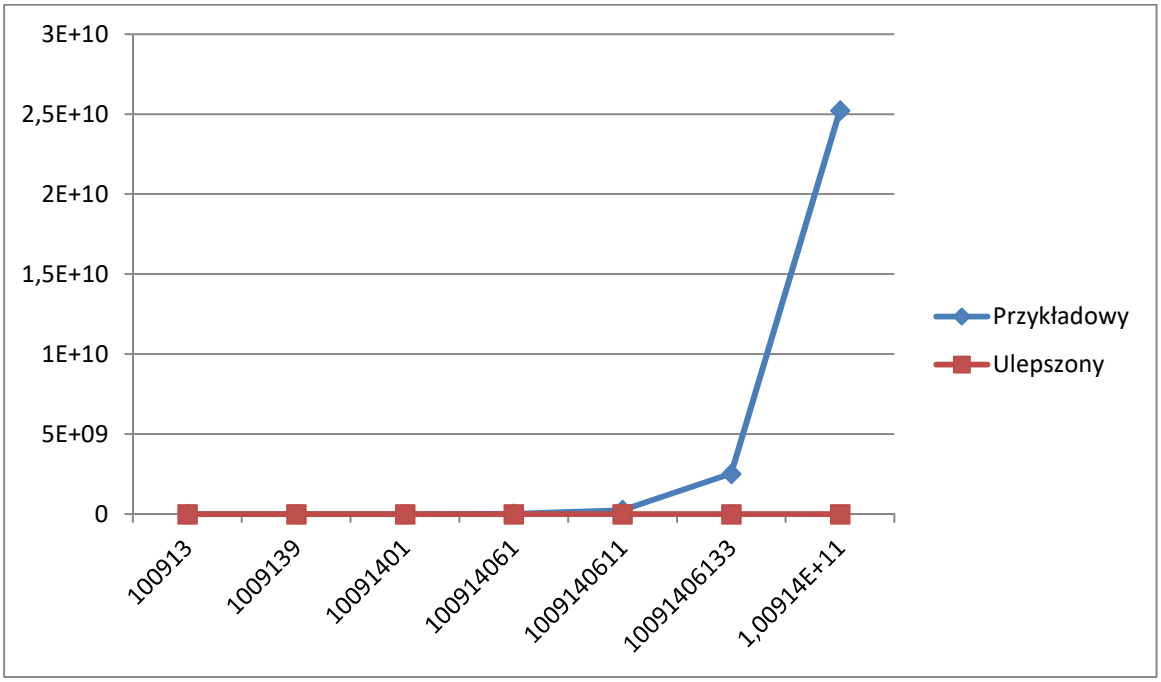
Wyniki dla algorytmu ulepszonego:

Liczba	Czy jest pierwsza?	Czas	Instrumentacja
100913	True	0,0000	159
1009139	True	0,0000	502
10091401	True	0,0002	1588
100914061	True	0,0004	5023
1009140611	True	0,0012	15883
10091406133	True	0,0044	50228
100914061337	True	0,0185	158835

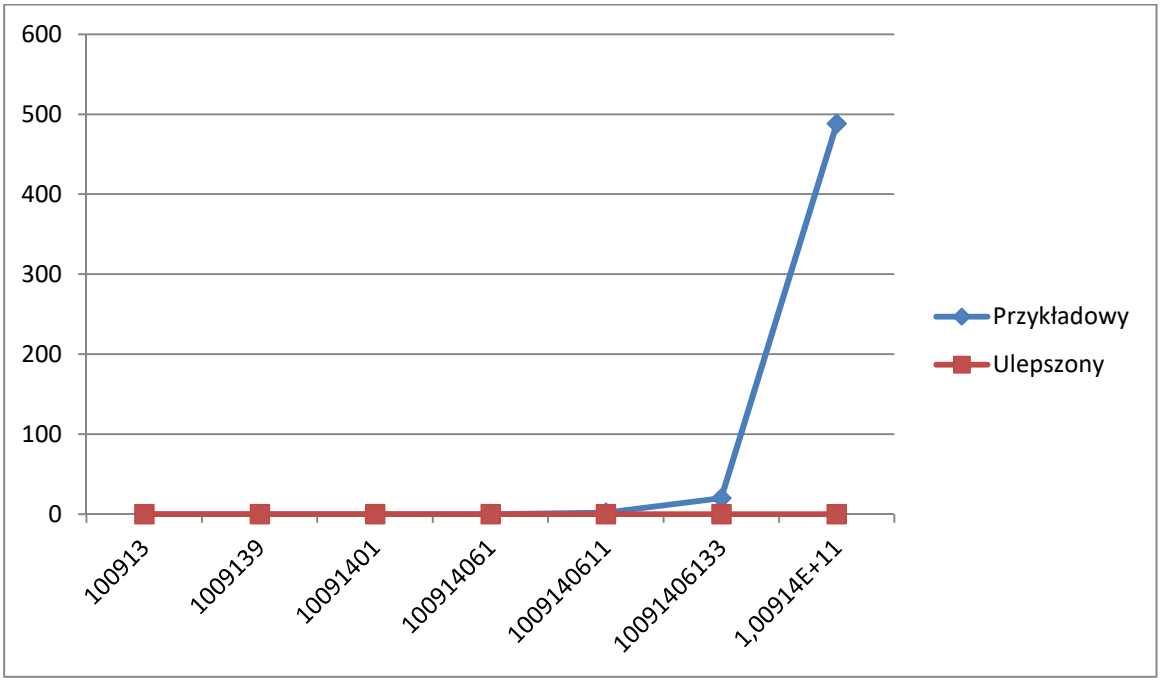

Algorytm Ulepszony - Czas
0,02
0,018
0,016
0,014
0,012
0,01
0,008
0,006
0,004
0,002
0
1
2
3
4
5
6
7
Series 1


Algorytm Ulepszony - Instrumentacja
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
1
2
3
4
5
6
7
Series 1

Wykres zestawiający instrumentacje algorytmu przykładowego z algorytmem ulepszonym:



Wykres zestawiający czas algorytmu przykładowego z algorytmem ulepszonym:



Niestety, w wynikach brakuje liczby „1009140613399”, której bezowocne sprawdzanie trwało łącznie 2 doby. Pozostawiając tylko tą liczbę w tablicy, algorytm wykonywał się na laptopie z procesorem i5-8265u 12 godzin.

Następnego dnia, próba na komputerze z Ryzenem 2600, po 13h również nie przyniosła efektu.

Wnioski:

Złożoność algorytmu jest pierwiastkowa.