Kacper Gendosz k_28

Projekt 2

Optymalizacja przykładowego algorytmu sprawdzania czy dana liczba jest liczbą pierwszą .

Metody badania wykorzystane do zbadania złożoności algorytmu :

- pomiar czasu potrzebnego na wykonanie sprawdzenia.
- zliczanie operacji dominujących. W tym przypadku operacji "%" .
- przyjęte liczby pomiarowe : { 100913, 1009139 ,10091401, 100914061, 1009140611, 10091406133, 100914061337, 1009140613399 }

I. Algorytm przykładowy :

Kod algorytmu bez instrumentacji :

```
            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;
    }
```

Kod z instrumentacją i pomiarem czasu:

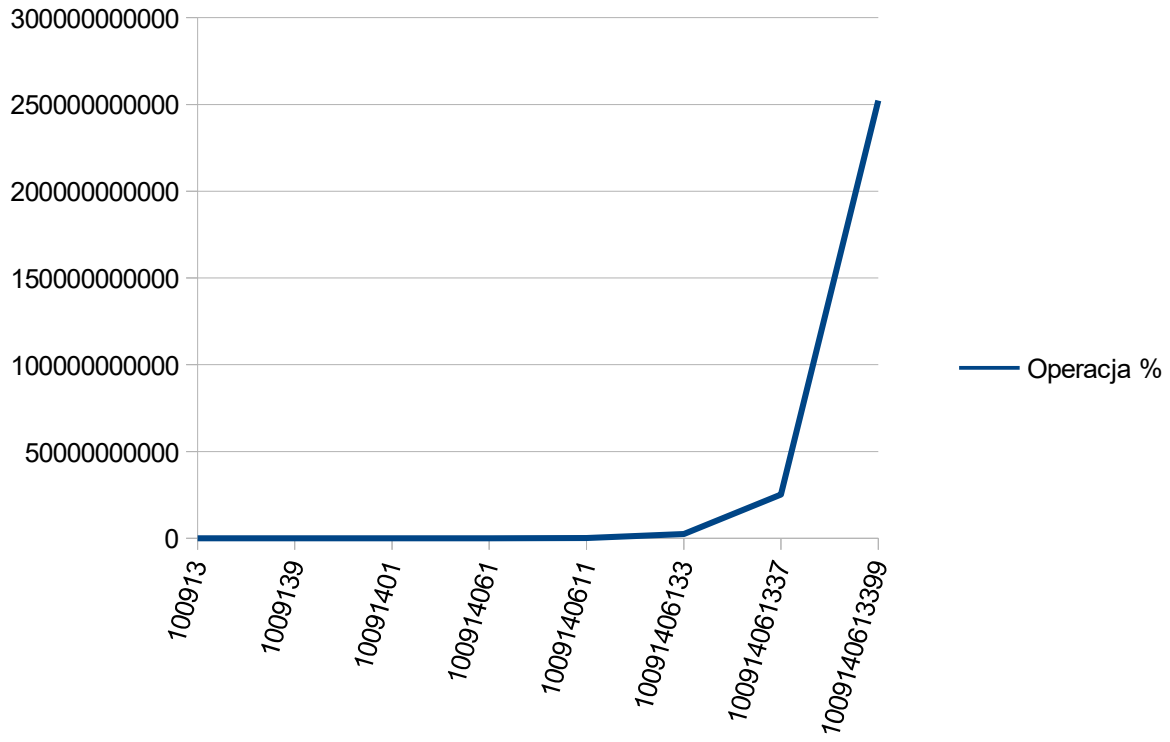
```
        class Program
{
    static void Main(string[] args)
    {
      Alg Algorytmy = new Alg();
      BigInteger[] tab = { 100913, 1009139, 10091401, 100914061, 1009140611,
                           10091406133, 100914061337, 1009140613399 };
        Algorytmy.IsPrime(0,out BigInteger a); // pierwsze wywołanie w celu uniknięcia
        przekłamania przy obliczania czasu.
      for (int i = 0; i < tab.Length; i++)
      {
         long StartTime = Stopwatch.GetTimestamp();
         Algorytmy.IsPrime(tab[i],out BigInteger counter);
         long StopTime = Stopwatch.GetTimestamp();
         double ElapsedSeconds = (StopTime - StartTime) * (1.0 / Stopwatch.Frequency);
      Console.WriteLine($"{tab[i]} Is Prime Hops = {counter} Time ={ElapsedSeconds}");
      }
    }
}
```

```
class Alg
 {
     public bool IsPrime(BigInteger Num,out BigInteger counter)
     {
         counter = 1;
         if (Num < 2) return false;
         else if (Num < 4) return true;
         else if (Num % 2 == 0) return false;// counter ++
         else for (BigInteger u = 3; u < Num / 2; u += 2)
             {
                 counter++;
                 if (Num % u == 0) return false;
             }
         return true;
     }
```

Wyniki :

ID	Liczba badana	Ilość operacji dominującej %	Czas
1	100913	25228	0,0135943743
2	1009139	252284	0,1362938433
3	10091401	2522850	1,3634466219
4	100914061	25228515	15,2250316978
5	1009140611	252285152	135,9231975453
6	10091406133	2522851533	2876,1208847204
7	100914061337	25228515334	38902,1901522711
8	1009140613399	252285153349	- (brak danych)*

* trwa obliczanie.



II. Algorytm przyzwoity.

Optymalizacja algorytmu polegała na zmianie warunku wykonywania pętli z " u < **Num / 2** " na " **u * u <= Num".** Ponieważ każda liczba złożona ma dzielnik mniejszy od jej pierwiastka to sprawdzanie podzielności liczby za nim jest bezcelowe . Takie rozwiązanie znacznie zredukowało ilość operacji dominujących.

Kod algorytmu bez instrumentacji :

```
class Alg
    {
        public 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;
        }
    }
```

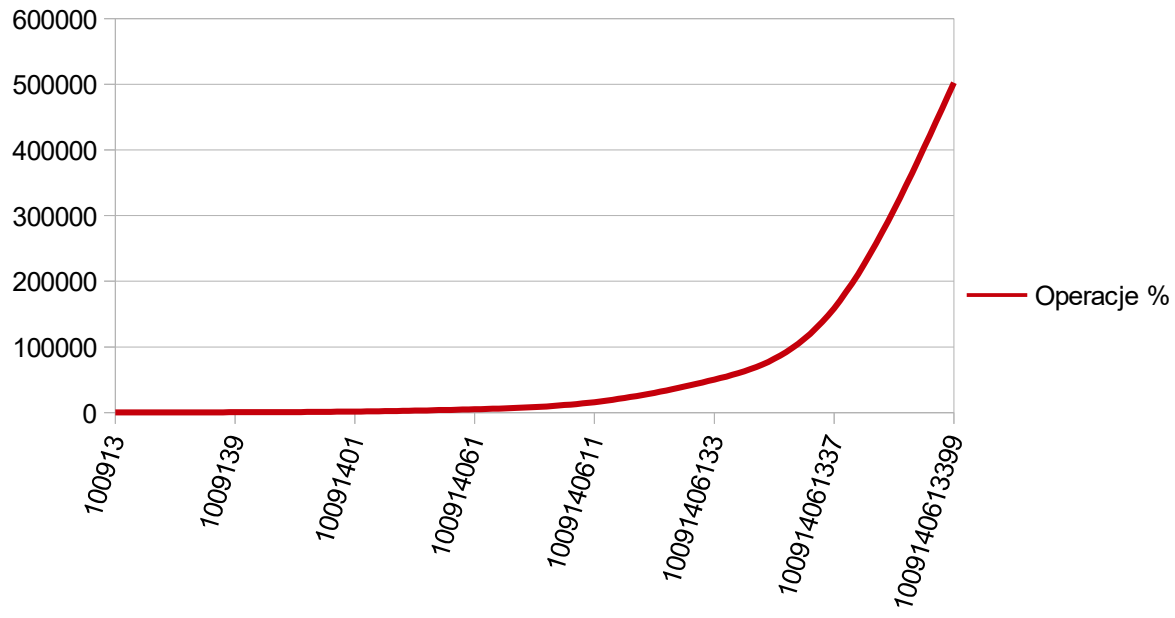
Kod z instrumentacją i pomiarem czasu:

```
class Program
    {
        static void Main(string[] args)
        {
            Alg Algorytmy = new Alg();
            BigInteger[] tab = {  100913, 1009139, 10091401, 100914061, 1009140611,
                                  10091406133, 100914061337, 1009140613399 };
              Algorytmy.IsPrime(0,out BigInteger a); // pierwsze wywołanie w celu uniknięcia
              przekłamania przy obliczania czasu.
            for (int i = 0; i < tab.Length-1; i++)
            {
                long StartTime = Stopwatch.GetTimestamp();
                Algorytmy.IsPrime(tab[i], out BigInteger counter);
                long StopTime = Stopwatch.GetTimestamp();
                double ElapsedSeconds = (StopTime - StartTime) * (1.0 /
Stopwatch.Frequency);

                Console.WriteLine($"{tab[i]} Is Prime Hops = {counter} Time =
{ElapsedSeconds}");

            }
        }
    }
```

```
class Alg
{
    public bool IsPrime(BigInteger Num, out BigInteger counter)
    {
        counter = 1;
        if (Num < 2) return false;
        else if (Num < 4) return true;
        else if (Num % 2 == 0) return false; // counter ++
        else for (BigInteger u = 3;  u*u<= Num ; u += 2)
            {
                counter++;
                if (Num % u == 0) return false;
            }
        return true;
    }
}
```

Wyniki :

ID	Liczba badana	Ilość operacji dominującej %	Czas
1	100913	159	0,0001062461
2	1009139	502	0,0003014572
3	10091401	1588	0,0007814845
4	100914061	5023	0,0025063828
5	1009140611	15883	0,0085156851
6	10091406133	50228	0,0491932020
7	100914061337	158835	0,3113591750
8	1009140613399	502280	0,6793110136



III. Algorytm optymalny :

Kolejnym sposobem optymalizacji jest sprawdzanie podzielności danej liczby przez liczby pierwsze mniejsze bądź równe jej pierwiastkowi,pozwala to uniknąć sprawdzenia wielokrotności przebadanych dzielników i zmniejszy to liczbę operacji dominujących. Jednak problemem jest tu wyznaczenie liczb pierwszych z danego zakresu co jest operacją czasochłonną jednak jest ona jednorazowa i jeśli wyznaczymy takie liczby dla maksymalnego zakresu możemy użyć takiego zbioru dla innych nie większych liczb.
Do wyznaczenia liczb pierwszych użyto metody SitoErastotenesa();

Kod algorytmu bez instrumentacji :

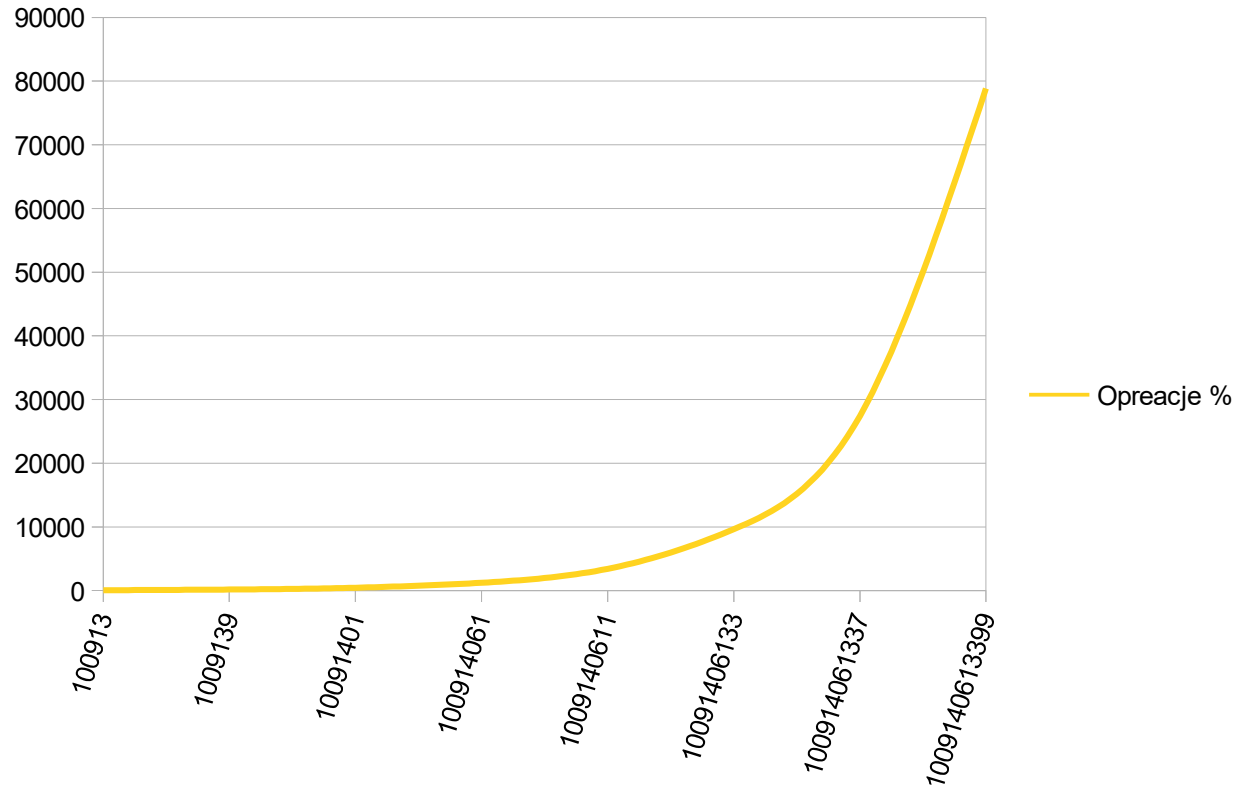
```
static void Main(string[] args)
        {
            Alg Algorytmy = new Alg();
            List<int> b ;
            BigInteger[] tab = { 100913, 1009139, 10091401, 100914061, 1009140611,
                                 10091406133, 100914061337, 1009140613399 };
            Algorytmy.IsPrime(0,b=new List<int>(), out BigInteger a); // pierwsze wywołanie
w celu uniknięcia przekłamania przy obliczania czasu.
            for (int i = 0; i < tab.Length; i++)
            {
                List<int> sito = Algorytmy.SitoEratostenesa(tab[i]);
                Algorytmy.IsPrime(tab[i],sito, out BigInteger counter);
            }
        }
    }


        public bool IsPrime(BigInteger Num, List<int> sito, out BigInteger counter)
        {

            counter = 1;
            if (Num < 2) return false;
            else if (Num < 4) return true;
            else if (Num % 2 == 0) return false;

            else for (int i = 0;i<sito.Count;i++)
                {
                    if (Num % sito[i] == 0) return false;
                }
            return true;
        }
```

Kod z instrumentacją i pomiarem czasu:

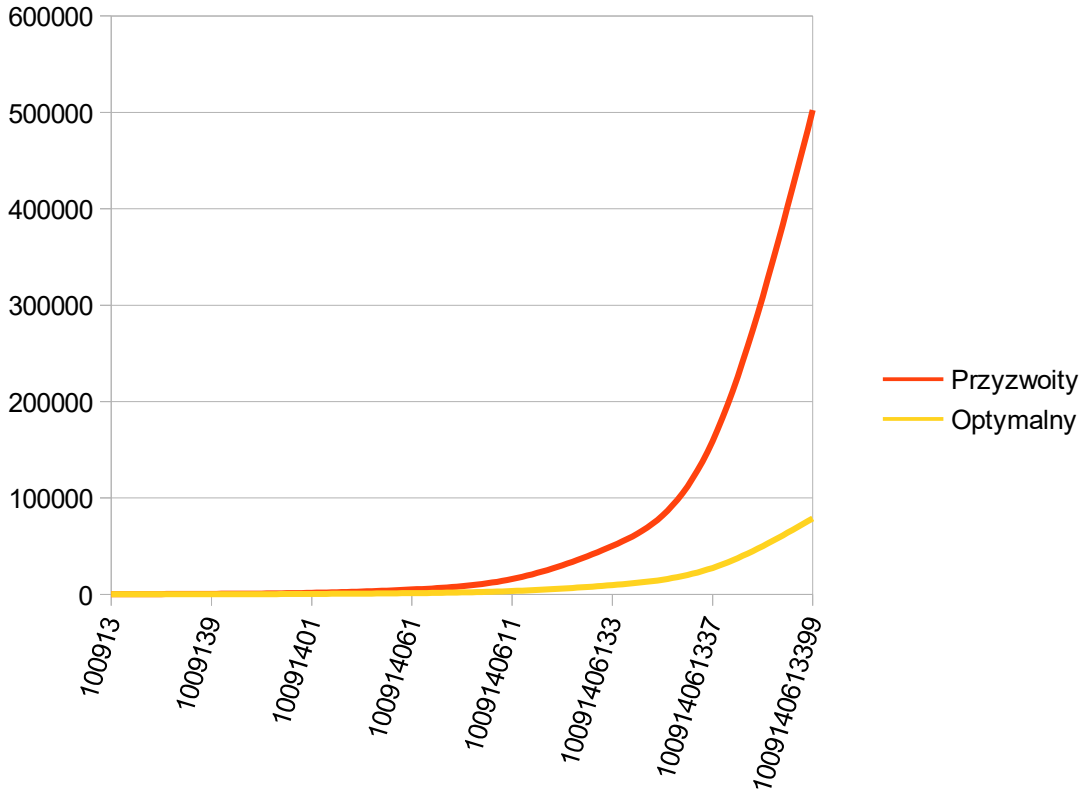
```
static void Main(string[] args)
        {
            Alg Algorytmy = new Alg();
            List<int> b ;
            BigInteger[] tab = { 100913, 1009139, 10091401, 100914061, 1009140611,
                                 10091406133, 100914061337, 1009140613399 };
            Algorytmy.IsPrime(0,b=new List<int>(), out BigInteger a); // pierwsze wywołanie
w celu uniknięcia przekłamania przy obliczania czasu.
            for (int i = 0; i < tab.Length; i++)
            {
                List<int> sito = Algorytmy.SitoEratostenesa(tab[i]);
                long StartTime = Stopwatch.GetTimestamp();
                Algorytmy.IsPrime(tab[i],sito, out BigInteger counter);
                long StopTime = Stopwatch.GetTimestamp();
               double ElapsedSeconds = (StopTime - StartTime) * (1.0 / Stopwatch.Frequency);

          Console.WriteLine($"{tab[i]} Is Prime Hops = {counter} Time = {ElapsedSeconds}");

            }
        }
    }


        public bool IsPrime(BigInteger Num, List<int> sito, out BigInteger counter)
        {

            counter = 1;
            if (Num < 2) return false;
            else if (Num < 4) return true;
            else if (Num % 2 == 0) return false; // counter ++

            else for (int i = 0;i<sito.Count;i++)
                {

                    counter++;
                    if (Num % sito[i] == 0) return false;
                }
            return true;
        }
```

Wyniki :

ID	Liczba badana	Ilość operacji dominującej %	Czas
1	100913	66	0,0000524830
2	1009139	168	0,0000512029
3	10091401	449	0,0001267272
4	100914061	1233	0,0003436996
5	1009140611	3415	0,0009184523
6	10091406133	9628	0,0032168232
7	100914061337	27411	0,0090046729
8	1009140613399	78840	0,0270677818



Zestawienie ilości operacji dominujących



IV. Wnioski i ocena złożonosci.

- Algorytm przykładowy.
  Koszt minimalny : 1 (kiedy liczba jest mniejsza od 2 lub podzielna przez 3)
  Koszt średni : [1+ ((Num-1) /4)] / 2.
  Koszt maksymalny : (Num-1)/4 W przypadku gdy liczba jest pierwsza.
  Złożoność : n!

- Algorytm przyzwoity.
  Koszt minimalny : 1 (kiedy liczba jest mniejsza od 2 lub podzielna przez 3)
  Koszt średni : (1+[Sqrt(Num-1)]/2)/2
  Koszt maksymalny : [Sqrt(Num-1)]/2 W przypadku gdy liczba jest pierwsza.
  Złożoność : n^3

- Algorytm optymalny.
  Koszt minimalny : 1 (kiedy liczba jest mniejsza od 2 lub podzielna przez 3)
  Koszt średni : --
  Koszt maksymalny : Zależny od ilości liczb pierwszych mniejszych niż Sqrt(Num)
  Złożoność : n^2

Wnioski z przeprowadzonej optymalizacji:

Sprawdzanie czy liczba jest liczbą pierwszą czy złożoną powinno być wykonywane do jej pierwiastka włącznie, jeżeli liczba nie ma dzielnika do tego miejsca nie będzie go miała i za nim. Waże jest również wyeliminowanie z potencjalnych dzielników liczb złożonych poniważ są one wielokrotnościami wystarczy sprwadzić liczby pierwsze co zmniejszy ilość operacji, jednak ustalenie takich dzielników może być czasochłonne przy czym raz obliczona tablica liczb pierwszych może być użyta do sprawdzenia następnych .