

Projekt 2

"Pierwszość Liczby"

Sebastian Krzyżaniak

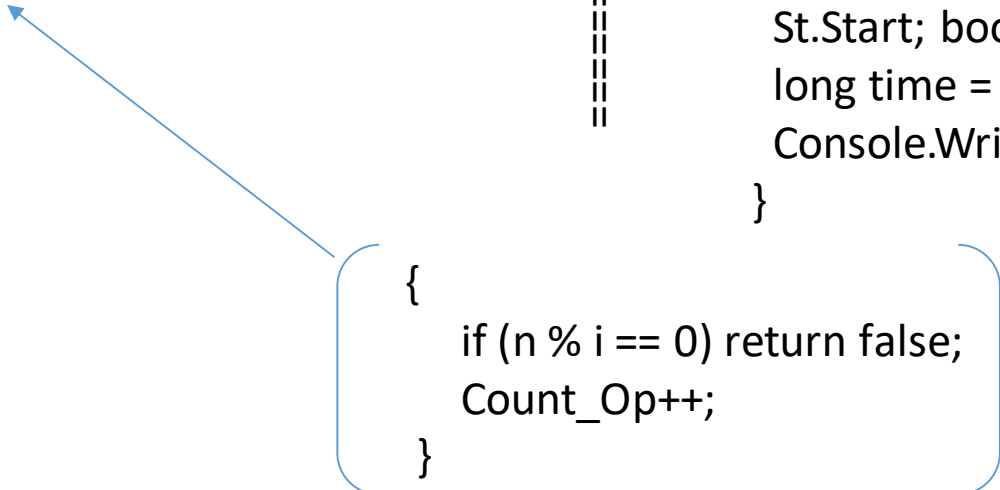
Przykładowy kod

```
static bool exPrime(ulong n)    //exemplary
{
    if (n < 2) return false;
    else if (n < 4) return true;
    else if (n % 2 == 0) return false;
    else
    {
        ulong m = (n >> 1);    // n/2;
        for (ulong i = 3; i < m; i += 2)
        {
            if (n % i == 0) return false;
        }
    }
    return true;
}
```

```
static void Main()
{
    ulong n = 1009140613399;
    Console.WriteLine(exPrime(n));
}

===== INSTRUMENTACJA

static int Count_OP = 0;    //counts operations
static void Main()
{
    ulong n = 1009140613399;
    Stopwatch st = new Stopwatch();
    St.Start; bool t = OptPrime(n); St.Stop;
    long time = st.ElapsedMilliseconds;
    Console.WriteLine(t \n time \n Count_OP);
}
```



```
{
    if (n % i == 0) return false;
    Count_Op++;
}
```

Przyzwoity kod

```
static bool OptPrime(ulong n)    //optimum
{
    if (n < 2) return false;
    else if (n < 4) return true;
    else if (n % 2 == 0) return false;
    else if ((n + 1) % 6 != 0 && (n - 1) % 6 != 0) return false;
    else
    {
        ulong s = (ulong)Math.Sqrt(n);    //square n
        for (ulong i = 3; i <= s; i += 2)
        {
            if (n % i == 0) return false;
        }
    }
    return true;
}
```

```
static void Main()
```

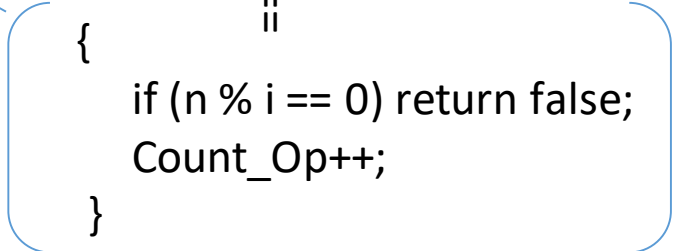
```
{
    ulong n = 1009140613399;
    Console.WriteLine(exPrime(n));
}
```

===== INSTRUMENTACJA

```
static int Count_OP = 0;    //counts operations
```

```
static void Main()
```

```
{ ulong n = 1009140613399;
  Stopwatch st = new Stopwatch();
  St.Start; bool t = OptPrime(n); St.Stop;
  long time = st.ElapsedMilliseconds;
  Console.WriteLine(t \n time \n Count_OP);
}
```



```
{
    if (n % i == 0) return false;
    Count_Op++;
}
```

The diagram shows a blue arrow pointing from the `if (n % i == 0) return false;` line in the `OptPrime` function to a callout box. The callout box contains the code block for the loop body, including the `if` statement and the `Count_Op++` increment, enclosed in curly braces.

Optymalny kod

```
static void Main()
```

```
{
```

```
    ulong d, num;    //multiplier of power 2 in divider num - 1    //d-mnożnik potęgi 2 w dzielniku num - 1
```

```
    int s, quality;
```

```
    bool t;
```

```
    num = 100913;    //number
```

```
    quality = 20;    //quality //chance 1 to 1099511627776 (for 20)
```

```
    s = 0;    //exponent of power 2 in the divider num - 1    //wykładnik potęgi 2 w dzielniku num - 1
```

```
    for (d = num - 1; d % 2 == 0; s++) d /= 2;
```

```
    t = true;
```

```
    t = CheckPrime(d, num, s, quality, t);
```

```
    Console.WriteLine(t);
```

```
}
```

=====INSTRUMENTACJA

//quality == number of operations

```
Stopwatch st = new Stopwatch();  
st.Start;  
t = CheckPrime(d, num, s, quality, t);  
st.Stop;  
long time = st.ElapsedMilliseconds;  
Console.WriteLine(t \n time \n quality)
```

```
private static bool CheckPrime(ulong d, ulong num, int s, int quality, bool t)
{
    ulong rnd_num, x;    //x-number of string    //x-wyraz ciągu Millera-Rabina
    if (MaybePrime(num) == false) return t = false;    //ADDTL SCRTY SPEED
    for (int i = 1; i <= quality; i++)
    {
        rnd_num = Random(num - 2);    //number selected randomly
        x = ExptnMod(rnd_num, d, num);    //modulo exponentiation    //potęgowanie modulo
        if ((x == 1) || (x == num - 1)) continue;    //num has passed the exam --> num is prime
        for (int j = 1; (j < s) && (x != num - 1); j++)
        {
            x = MultiplctnMod(x, x, num);    //multiplication of modulo    //mnożenie modulo
            if (x == 1)
            {
                t = false; break;
            }
        }
        if (!t) break;    //additional security
        if (x != num - 1)
        {
            t = false; break;
        }
    }
    return t;
}
```

```
static bool MaybePrime(ulong num)    //additional method (security and speed)
```

```
{
    if ((num + 1) % 6 == 0 || (num - 1) % 6 == 0) return true;
    else return false;
}
```

```
static ulong _w = 0;
```

```
static ulong Random(ulong b)
```

```
{
    Random rnd = new Random();
    for (int i = 1; i <= 8; i++)    //8x for better randomness
    {
        _w = (ulong)rnd.Next(2, 2147483647);
    }
    return _w;
}
```

```
static ulong MultiplctnMod(ulong a, ulong b, ulong n)
```

```
{
    ulong w = 0;
    for (ulong m = 1; m != 0; m <<= 1)    //64x loops
    {
        if ((b & m) != 0) w = (w + a) % n;
        a = (a << 1) % n;
    }
    return w;
}
```

```
static ulong ExptnMod(ulong a, ulong e, ulong n)
```

```
{
    ulong p, w;

    p = a; w = 1;
    for (ulong m = 1; m != 0; m <<= 1)    //64x loops
    {
        if ((e & m) != 0) w = MultiplctnMod(w, p, n);
        p = MultiplctnMod(p, p, n);
    }
    return w;
}
```


Number of operat.

n	Przykładowy	Przyzwoity	Optymalny
100913	25227	158	20
1009139	252283	501	20
10091401	2522849	1587	20
100914061	25228514	5022	20
1009140611	252285151	15882	20
10091406133	2522851522	50227	20
100914061337	25228515166	158834	20
1009140613399	252285151643	502279	20
n	Przykładowy	Przyzwoity	Optymalny
100913	0	1	1074
1009139	5	5	1222
10091401	49	16	1343
100914061	514	55	1390
1009140611	5476	177	1493
10091406133	73453	761	2252
100914061337	732608	2432	2360
1009140613399	7995539	7907	2657

przy quality == 20

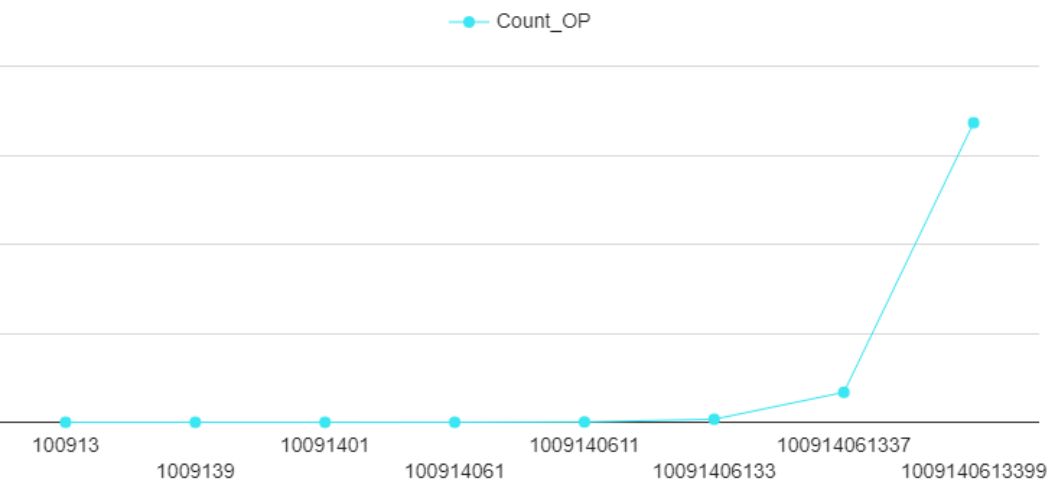


Przyzwoity x500

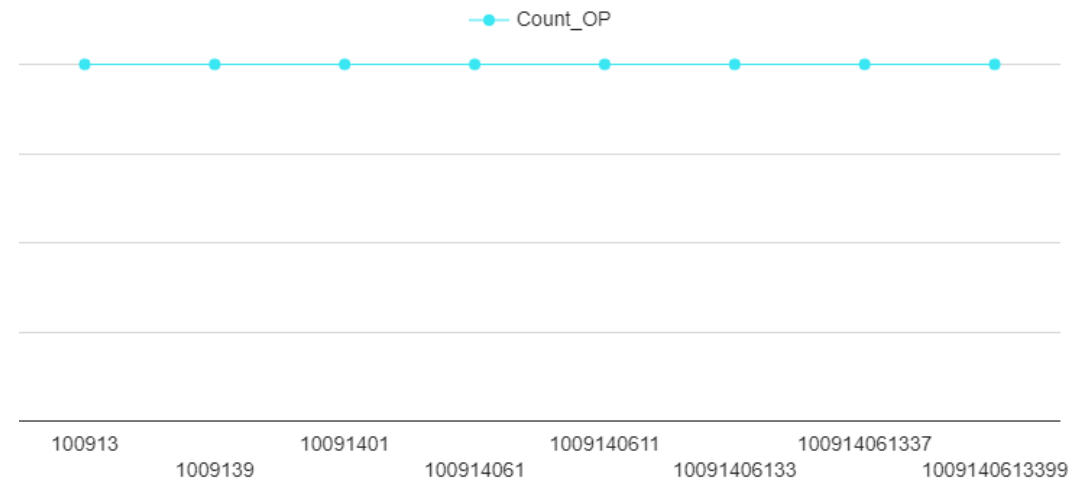
Optymalny x500
quality == 20

Time[ms]

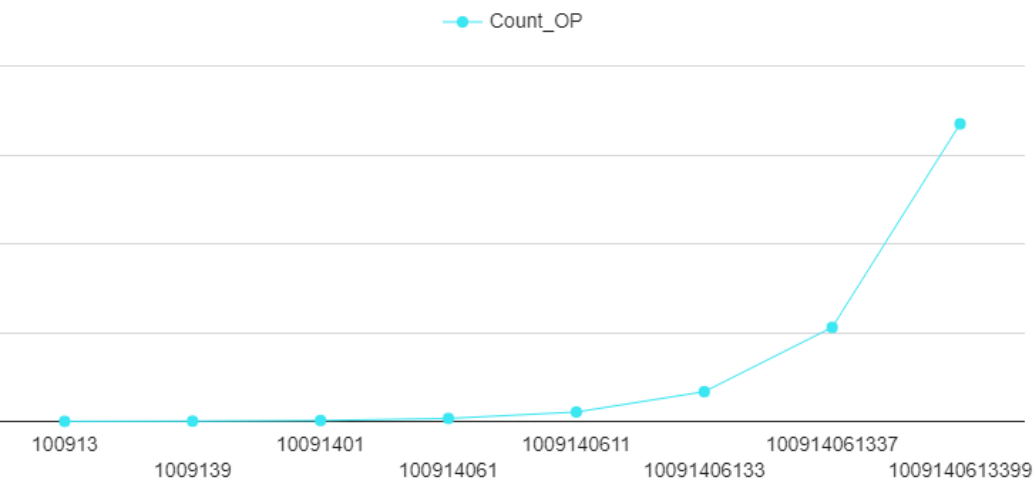
for quality == 1
== 116
== 124
== 137



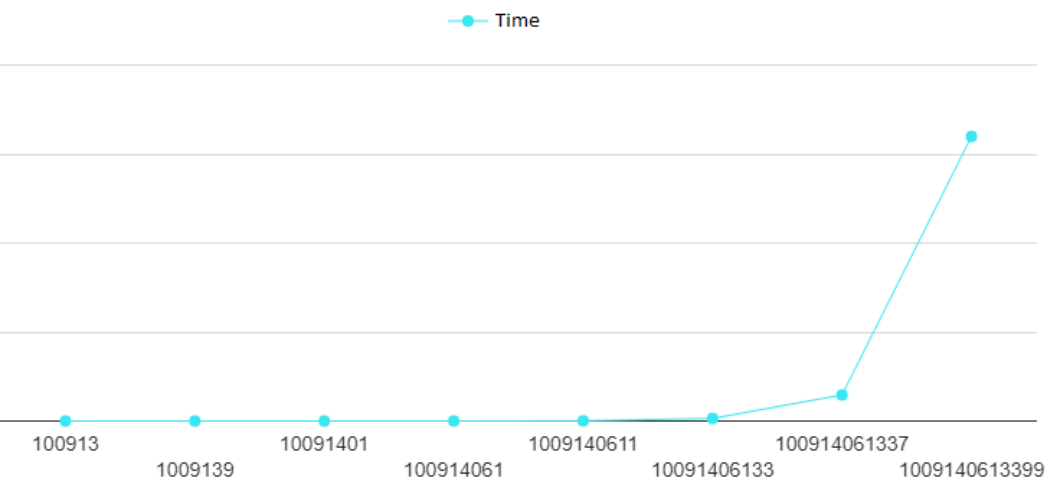
Przykładowy



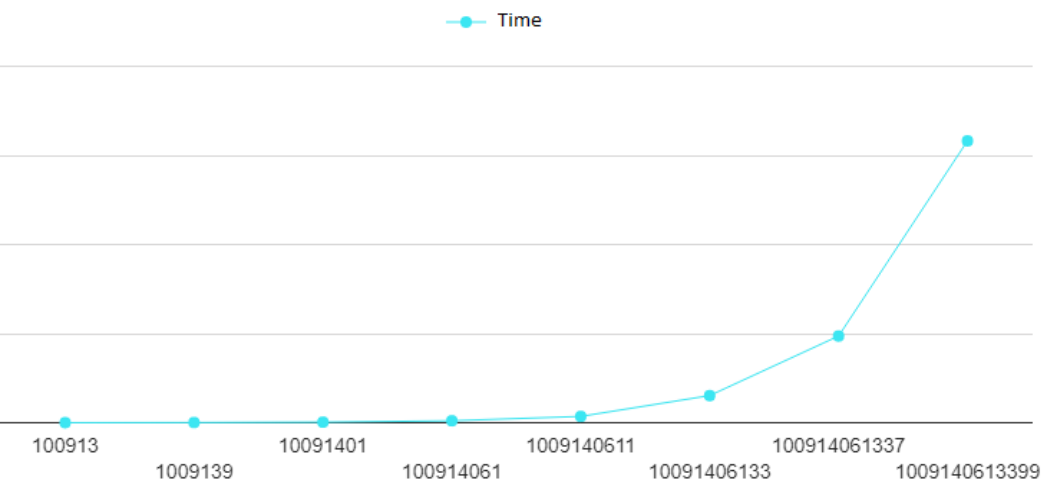
Optymalny



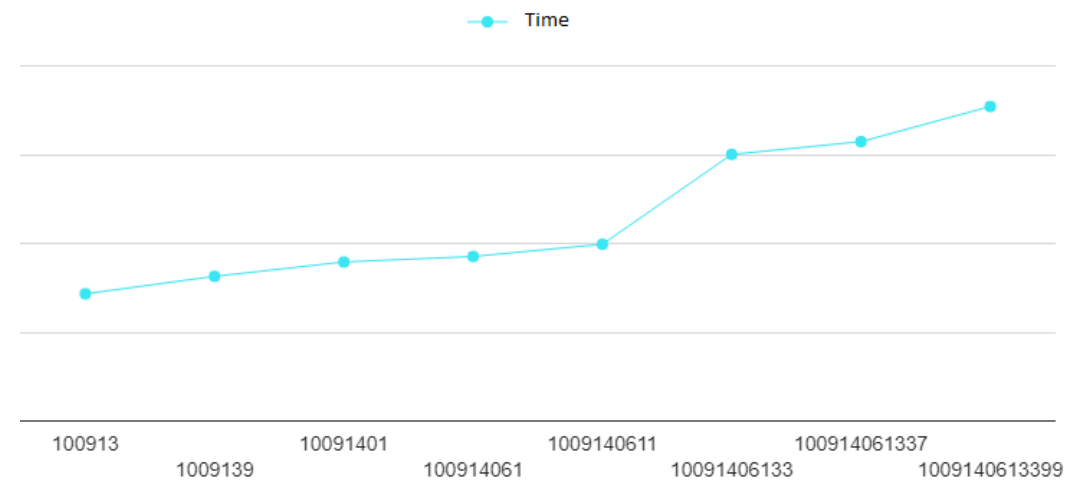
Przyzwoity



Przykładowy



Optymalny



Przyzwoity

Wnioski

Algorytm **przykładowy** początkowo ma niewielkie koszty, lecz już gdy pojawiają się "pierwsze duże liczby", rosną one diametralnie szybko.

Dla małych liczb nie ma znaczenia czy użyjemy **przykładowego**, czy **przyswoitego**, różnice są niewielkie. Wraz ze wzrostem liczby, przewaga **przyswoitego** nad **przykładowym** rośnie. Dla dużych liczb najlepszy okazuje się **optymalny**, gdyż jego złożoność rośnie bardzo powoli a już przy niedużych liczbach zaczyna mieć przewagę; przy małych jest mniej efektywny.

Algorytm optymalny [złożoność logarytmiczna] – napisany przy użyciu testu Millera Rabina, działa podobnie jak: "Chiński Warunek Pierwszości", "Twierdzenie Fermata" czy "Liczby Carmichaela".

Algorytm bada daną liczbę z szansą $(\frac{1}{4})^n$, że liczba złożona zostanie uznana za pierwszą, a później robi jej 'testy', które zmniejszając tę szansę z każdym obiegiem pętli (ilość testów == n == quality). Za to największą zaletą programu jest jego ogromna szybkość, nieporównywalna dla dużych liczb.

Zestawienie złożoności - time[ms]

n	Przykładowy	Przyzwoity	Optymalny
100913	0	0	2
1009139	5	0	2
10091401	49	0	2
10094061	514	0	2
100940611	5476	0	2
1009406133	73453	1	4
10094061337	732608	2	4
100950613399	7995539	7	6

Zestawienie złożoności - time[ms] - Wykres

