МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Інститут математики, економіки та механіки

(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерної алгебри та дискретної математики

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

спеціаліст

(освітньо-кваліфікаційний рівень)

на тему «QS – алгоритм факторизації цілих чисел, його модифікації»

«QS – method of factorization of integers, its modifications»

Виконала: студентка 5 курсу
напряму підготовки (спеціальності)
7.05010201 Комп'ютерні системи та мережі

(шифр і назва напряму підготовки, спеціальності)

Юлія Василівна

Король Ю. В.

(прізвище та ініціали)

Керівник Савастру О. В.

(прізвище та ініціали)

Рецензент Якімова Н. А.

(прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№ 8 від «12» 06. 17 р.

Завідувач кафедри

(підпис) Варбанець П. Д. (прізвище, ініціали)

Захищено на засіданні ДЕК № 4

протокол № 25 від «22» 06. 2017 р.

Оцінка добре (85) В

(за 4-х бальною шкалою, за шкалою ECTS, бал.)

Голова ДЕК

(підпис) Арсірій О.О. (прізвище, ініціали)

Ш/к 593793

Одеса - 2017

АНОТАЦІЯ

В дипломній роботі досліджується проблема факторизації цілих чисел. В даний час широко поширений метод шифрування з відкритим ключем RSA, а також деякі алгоритми цифрового підпису, базуються на практичній трудомісткості рішення задачі факторизації. Існування класичного алгоритму, що вирішує задачу факторизації за поліноміальний час, змусило б повністю відмовитися від RSA в майбутньому, і скомпрометувало б велику кількість вже існуючих систем. Однак, самий швидкий алгоритм факторизації довільних натуральних чисел, відомий на сьогоднішній день, має субекспоненціальну оцінку часу роботи.

Метою даного дипломного проекту є – вивчення існуючих сучасних алгоритмів факторизації великих цілих чисел, а також реалізація розробленого QS-алгоритму у вигляді комп'ютерної програми.

В роботі представлено огляд сучасних алгоритмів факторизації цілих чисел з екпоненціальною та з субекпоненціальною складністю. Реалізовано базовий метод квадратичного решета мовою C#. Розроблено QS – алгоритм для чисел Мерсена.

АННОТАЦИЯ

В дипломной работе исследуется проблема факторизации целых чисел. В настоящее время широко распространен метод шифрования с открытым ключом RSA, а также некоторые алгоритмы цифровой подписи, основанные на практической трудоемкости решения задачи факторизации. Существование классического алгоритма, который решает задачу факторизации за полиномиальное время, заставило бы полностью отказаться от RSA в будущем, и скомпрометировало бы большое количество уже существующих систем. Однако, самый быстрый алгоритм факторизации произвольных натуральных чисел, известный на сегодняшний день, имеет субекспоненциальную оценку времени работы.

Целью данного дипломного проекта является – изучение существующих современных алгоритмов факторизации больших целых чисел, а также реализация разработанного QS-алгоритма в виде компьютерной программы.

В работе представлен обзор современных алгоритмов факторизации целых чисел с екпоненциальной и с субекпоненциальной сложностью. Реализовано базовый метод квадратичного решета на языке C #. Разработан QS - алгоритм для чисел Мерсенна.

ANNOTATION

The given diploma paper investigates the problem of integers' factorization. At present, the widespread public key encryption method RSA, as well as some digital signature algorithms are based on practical work content of factorization task. The existence of classical algorithm, solving the factorization task, would make us renounce the RSA later on, and compromise the great number of pre-existing systems. However, the fastest factorization's algorithm of any natural numbers, which is well- known today, has subexponential estimation of working hours.

The object of given diploma project is the investigation of modern present factorization algorithms of great integers, as well as realization of developed QS-algorithm as computer programme.

This diploma paper describes the review of modern factorization algorithms of integers with exponential and subexponential complication. The basic quadratic sieve method in the language of C# is realized. The QS - algorithm for Mersen's numbers is developed.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ 6
ВВЕДЕННЯ..... 7
1 Допоміжні алгоритми 9
1.1 Розширений алгоритм Евкліда 9
1.2 Символ Лежандра 13
1.3 Алгоритм знаходження квадратичних коренів по модулю p 15
2 Огляд методів з експоненціальною та з субекспоненціальною складністю..... 18
2.1 Факторизація цілих чисел з експоненціальною складністю 18
2.2 Факторизація цілих чисел з субекспоненціальною складністю 24
3 Метод квадратичного решета та його модифікації 33
4 Програмна реалізація та аналіз алгоритмів 60
4.1 Склад і структура 60
4.2 Програмна реалізація..... 61
ВИСНОВОК..... 65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 66
ДОДАТОК..... 68

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ ТА ТЕРМІНІВ

N	множина натуральних чисел
Z	кільце цілих чисел
$\lceil a \rceil$	ціла частина зверху, тобто найменше n є Z, для якого $n \geq a$
$a \equiv b (\mathrm{mod}\ c)$	a - b ділиться на c в розглянутому кільці (цілих чисел або многочленів)
$a \not\equiv b (\mathrm{mod}\ c)$	a - b не ділиться на c
a ∈ A	a – елемент множини A
∅	порожня множина
const	яка-небудь додатня постійна
$O(f(n))$	складність алгоритму, означає, що зі збільшенням параметра n, час роботи алгоритму буде зростати не швидше, ніж деяка константа, помножена на $f(n)$
exp (x)	e^x, де e - число Ейлера

ВИСНОВОК

У даній роботі ми розглянули сучасні алгоритми факторизації цілих чисел. Таким чином, можна зробити наступні висновки: для факторизації натуральних чисел існує досить багато способів, але це завдання далеко не тривіальне, і досить затратне в плані часу, про це говорить оцінка складності алгоритмів факторизації.

В роботі детально розглядався алгоритм факторизації цілих чисел – QS-метод. Спеціально була створена комп'ютерна програма, яка реалізувала цей алгоритм, а саме метод квадратичного решета мовою C#, а також розроблено та реалізовано QS – алгоритм факторизації для чисел Мерсена; виконані обчислювальні експерименти. Наведені стратегії, які дозволяють поліпшити ефективність методу квадаратичного решета.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Miller G. Riemann’s hypothesis and tests for primality // Journal of Computer and System Sciences._ 1976._ Vol. 13, no. 3._ Pp. 300–317.
2. Pollard J.M. Theorems on factorization and primality testing // Proc. Cambridge Phil. Soc. 1974. V. 76. P. 521—528.
3. Cohen H. A course in computational algebraic number theory. Springer-Verlag, 1993.
4. Коблиц Н. Курс теории чисел и криптографии. – М.: Научное изд-во ТВП, 2001,- 254 с.
5. Montgomery P. L., Silverman R.D. A FFT-extension to the p −1 factoring algorithm // Math. Comp. 1990. V. 54 (190). P. 839—854.
6. Pomerance C. Analysis and comparision of some integer factoring algorithms // Computational methods in number theory. V. 1 / H.W. Lenstra and R. Tijdeman, editors. Amsterdam, 1982. P. 89—139.
7. D. Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. Math. Comp., 62(205):333–350, 1994.
8. B. A. LaMacchia and A. M. Odlyzko. Solving large sparse linear systems over finite fields. In Advances in Cryptology—CRYPTO ’90 (Santa Barbara, Calif., 1990), Lecture Notes in Comput. Sci. Springer, Berlin, 1990.
9. M.W. Baldoni, C. Ciliberto, and. G.M. Piacentini Cattaneo. Elementary Number Theory, Cryptography and Codes, 2009.
10. Крэндалл Р., Померанс К. Простые числа. Криптографические и вычислительные аспекты — М.: УРСС, Либроком, 2011. — 664 с
11. Brillhart J., Morrison M.A. A method of factoring and the factorization of F7 // Math. Comp. 1975. V. 29. P. 183—205.
12. Coppersmith D., Winograd S. On the asymptotic complexity of matrix multiplication // SIAM J. Comput. 1982. V. 11. P. 472—492.

13. Kleinjung Th., Aoki K., Franke J., Lenstra A., Thom E., Bos J., Gaudry P., Kruppa A., Montgomery P., Osvik D. A., Timofeev A., Zimmermann P. Factorization of a 768-bit RSA modulus. Online report, Feb 2010.
14. Василенко О. Н. В19. Теоретико-числовые алгоритмы в криптографии. — М. : МЦНМО, 2003. — 328 с. ISBN 5-94057-103-4; An Introduction to Mathematical Cryptography, 2008, Springer-Verlag, New York, 523 p.

ДОДАТОК

Програмна реалізація алгоритмів

QS - алгоритм

Клас HCSRAlgorithm

```
using System;
using System.Numerics;

namespace QuadraticSieve
{
    class HCSRAlgorithm
    {
        private Random random;

        public HCSRAlgorithm(int seed)
        {
            random = new Random(seed);
        }

        public bool Composite(BigInteger n, int t)
        {
            if (n == 2 || n == 3)
                return false;

            BigInteger m = n % 2;

            if (m == 0)
                return true;

            BigInteger n1 = n - 1;
            BigInteger r = n1;
            long s = 0;

            m = r % 2;

            while (m == 0)
            {
                r = r / 2;
                m = r % 2;
                s++;
            }

            BigInteger n2 = n - 2;

            for (int i = 1; i <= t; i++)
            {
                BigInteger a = RandomRange(2, n2);
                BigInteger y = BigInteger.ModPow(a, r, n);
```

```
            if (y != 1 && y != n1)
            {
                int j = 1;

                while (j <= s && y != n1)
                {
                    y = BigInteger.ModPow(y, 2, n);

                    if (y == 1)
                        return true;

                    j++;
                }

                if (y != n1)
                    return true;
            }
        }

        return false;
    }

    public int Jacobi(BigInteger a, BigInteger n)
    {
        if (a == 0)
            return 0;

        if (a == 1)
            return 1;

        int e = 0;
        BigInteger a1 = a;
        BigInteger quotient = a1 / 2;
        BigInteger remainder = a1 % 2;

        while (remainder == 0)
        {
            e++;
            a1 = quotient;
            quotient = a1 / 2;
            remainder = a1 % 2;
        }

        int s = 0;

        if (e % 2 == 0)
            s = 1;

        else
        {
            BigInteger mod8 = n % 8;

            if (mod8 == 1 || mod8 == 7)
```

```
                    s = +1;

                if (mod8 == 3 || mod8 == 5)
                    s = -1;
            }

            BigInteger mod4 = n % 4;
            BigInteger a14 = a1 % 4;

            if (mod4 == 3 && a14 == 3)
                s = -s;

            BigInteger n1 = n % a1;

            return s * Jacobi(n1, a1);
        }

        public BigInteger RandomRange(
           BigInteger lower, BigInteger upper)
        {
            if (lower <= long.MaxValue && upper <= long.MaxValue
&& lower < upper)
            {
                BigInteger r;

                while (true)
                {
                    r = lower + (long)(((long)upper -
(long)lower) * random.NextDouble());

                    if (r >= lower && r <= upper)
                        return r;
                }
            }

            BigInteger delta = upper - lower;
            byte[] deltaBytes = delta.ToByteArray();
            byte[] buffer = new byte[deltaBytes.Length];

            while (true)
            {
                random.NextBytes(buffer);

                BigInteger r = new BigInteger(buffer) + lower;

                if (r >= lower && r <= upper)
                    return r;
            }
        }

        public BigInteger SquareRootModPrime(
           BigInteger a, BigInteger p)
```

```
        // returns square root of a modulo p if it exists 0
otherwise
        {
            long e = 0, r, s;
            BigInteger b = 0, bp = 0, q = p - 1, m = 0, n = 0;
            BigInteger p1 = p - 1, t = 0, x = 0, y = 0, z = 0;

            // p - 1 = 2 ^ e * q with q odd

            m = q % 2;

            while (m == 0)
            {
                q = q / 2;
                m = q % 2;
                e++;
            }

            // find generator

            int jac = 0;

            do
            {
                n = RandomRange(1, p1);
                jac = Jacobi(n, p);
            } while (jac == -1);

            z = BigInteger.ModPow(n, q, p);
            y = z;
            r = e;
            x = BigInteger.ModPow(a, (q - 1) / 2, p);
            b = (((a * x) % p) * x) % p;
            x = (a * x) % p;

            while (true)
            {
                if (b == 1 || b == p1)
                    return x;

                s = 1;

                do
                {
                    bp = BigInteger.ModPow(b, (long)Math.Pow(2,
s), p);
                    s++;
                } while (bp != 1 && bp != p1 && s < r);

                if (s == r)
                    return 0;
```

```
                t = BigInteger.ModPow(y, (long)Math.Pow(2, r - s
- 1), p);
                y = (t * t) % p;
                x = (x * t) % p;
                b = (b * y) % p;
                r = s;
            }
        }
    }
}
```

Клас LinearAlgebra

```
using System.Threading.Tasks;
namespace QuadraticSieve
{
    class LinearAlgebra
    {
        public void KernelOverZ2(int m, int n, int[] d, sbyte[,]
M, ref int r)
        {
            sbyte D;
            int i, j, k, s;
            int[] c = new int[m];

            r = 0;

            for (i = 0; i < m; i++)
                c[i] = -1;

            for (k = 0; k < n; k++)
            {
                for (j = 0; j < m; j++)
                    if (M[j, k] != 0 && c[j] == -1)
                        break;

                if (j < m)
                {
                    D = -1;
                    M[j, k] = -1;

                    for (s = k + 1; s < n; s++)
                        M[j, s] = (sbyte)(D * M[j, s]);

                    for (i = 0; i < m; i++)
                    {
                        if (i != j)
                        {
                            D = M[i, k];
                            M[i, k] = 0;

                            for (s = k + 1; s < n; s++)
```

```
                                    M[i, s] = (sbyte)((M[i, s] + D *
M[j, s]) % 2);
                            }
                        }

                        c[j] = k;
                        d[k] = j;
                    }

                    else
                    {
                        r = r + 1;
                        d[k] = -1;
                    }
                }
            }
        }
}
```

Клас MainForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.Numerics;
using System.Windows.Forms;

namespace QuadraticSieve
{
    public partial class MainForm : Form
    {
        private int result, t;
        private long B0 = 10000000;
        private BackgroundWorker bw;
        private BigInteger N, n;
        private QuadraticSieve qs;
        private Stopwatch sw;
        private List<long> times;
        private List<FactorExpon> lfe;

        public MainForm()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {

                //long BASE = (long)numericUpDown1.Value;
```

```
                // int expon = (int)numericUpDown2.Value;
                 //BigInteger add = Horner(textBox1.Text);
                 t = (int)numericUpDown5.Value;
                 N = n = BigInteger.Parse(textBox3.Text);
                 bw = new BackgroundWorker();
                 bw.WorkerSupportsCancellation = true;
                 bw.DoWork += Bw_DoWork;
                 bw.RunWorkerCompleted += Bw_RunWorkerCompleted;
                 qs = new QuadraticSieve(1, t, B0);
                 bw.RunWorkerAsync();
                 while (!bw.IsBusy) { }
                 button1.Text = "Стоп";
             }
             catch (Exception ex)
             {
                 MessageBox.Show(ex.ToString(), "Warning",
                     MessageBoxButtons.OK,
MessageBoxIcon.Warning);
             }
         }

         private void Bw_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
         {
             TimeSpan ts = sw.Elapsed;
             string text = string.Empty;
             textBox2.Text += "N = ";
             textBox2.Text += N.ToString() + "\r\n\r\n";

             if (result == -2)
             {
                 MessageBox.Show("Число занадто мале",
"Попередження",
                     MessageBoxButtons.OK,
MessageBoxIcon.Warning);
             }

             else if (result == -1)
             {
                 MessageBox.Show("Алгоритму не вдалося знайти
фактор", "Попередження",
                     MessageBoxButtons.OK,
MessageBoxIcon.Warning);
             }

             else if (result == 0)
             {
                 MessageBox.Show("Число просте, введіть інше
число", "Повідомлення",
                    MessageBoxButtons.OK,
MessageBoxIcon.Information);
             }
```

```
            else if (result == 1)
            {
                MessageBox.Show("Число не повністю факторизовано
", "Повідомлення",
                  MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }

            else if (result == 2)
            {
                MessageBox.Show("Число повністю факторизовано",
"Повідомлення",
                  MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }

            if (lfe.Count > 0)
            {
                // selection sort the factors into ascending
order
                textBox2.Text += "Результат:   ";
                textBox2.Text += "\r\n\r\n";
                for (int i = 0; i < lfe.Count - 1; i++)
                {

                    for (int j = i + 1; j < lfe.Count; j++)
                    {
                        if (lfe[i].factor > lfe[j].factor)
                        {
                            FactorExpon temp = lfe[i];

                            lfe[i] = lfe[j];
                            lfe[j] = temp;
                        }
                    }
                }

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

                    textBox2.Text += lfe[i].factor.ToString();

                    if (lfe[i].expon > 1)
                        textBox2.Text += " ^ " +lfe[i].expon ;

                    textBox2.Text += "\r\n";

                }

                textBox2.Text += "\r\n";
            }

            if (result == 1)
```

```
            {
                textBox2.Text += "Composite Residue" + "\r\n";
                textBox2.Text += n.ToString() + "\r\n\r\n";
            }

            text += "Час виконання " + "\r\n\r\n";
            text += ts.Hours.ToString("D2") + ":";
            text += ts.Minutes.ToString("D2") + ":";
            text += ts.Seconds.ToString("D2") + ".";
            text += ts.Milliseconds.ToString("D3") + "\r\n\r\n";

            if (times.Count == 4)
            {
                text += "Prime creation milliseconds = ";
                text += times[0].ToString() + "\r\n";
                text += "Multitasking milliseconds   = ";
                text += times[1].ToString() + "\r\n";
                text += "Linear algebra milliseconds = ";
                text += times[2].ToString() + "\r\n";
                text += "Final phase milliseconds    = ";
                text += times[3].ToString() + "\r\n";
            }

            textBox2.Text += text + "\r\n\r\n";
            button1.Text = "Факторизувати";
        }

        private void Bw_DoWork(object sender, DoWorkEventArgs e)
        {
            times = new List<long>();
            lfe = new List<FactorExpon>();
            sw = new Stopwatch();
            sw.Start();
            result = qs.Sieve(ref n, ref times, ref lfe);
            sw.Stop();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            textBox2.Text = string.Empty;
        }

        private void textBox1_TextChanged(object sender,
EventArgs e)
        {

        }

        private void textBox2_TextChanged(object sender,
EventArgs e)
        {

        }
```

```
    }
}
```

Клас QuadraticSieve

```
using System.Collections.Generic;
using System.Diagnostics;
using System.Numerics;
using System.Threading.Tasks;

namespace QuadraticSieve
{
    public struct FactorExpon
    {
        public BigInteger factor;
        public int expon;

        public FactorExpon(BigInteger factor, int expon)
        {
            this.factor = factor;
            this.expon = expon;
        }
    }

    class QuadraticSieve
    {
        private sbyte[,] e = null;
        private sbyte[,] v = null;
        private int t, t1;
        private long B0;
        private HCSRAlgorithm hcr;
        private LinearAlgebra hcl;
        private SievePrimes sp;
        private Stopwatch sw;
        private List<long> primes;

        public QuadraticSieve(
            int seed, int t, long B0)
        {
            t1 = t + 1;
            e = new sbyte[t1, t];
            v = new sbyte[t1, t];
            this.t = t;
            this.B0 = B0;
            hcr = new HCSRAlgorithm(seed);
            hcl = new LinearAlgebra();
            sp = new SievePrimes();
            sw = new Stopwatch();
            sp.Sieve(B0, ref primes);
        }

        public struct PrimeExpon
```

```
        {
            public int prime;
            public int expon;

            public PrimeExpon(int prime, int expon)
            {
                this.prime = prime;
                this.expon = expon;
            }
        }

        private BigInteger Sqrt(BigInteger n)
        {
            if (n == 0)
                return 0;

            BigInteger r = Sqrt(n >> 2);
            BigInteger r2 = r << 1;
            BigInteger s = r2 + 1;

            if (n < s * s)
                return r2;

            else
                return s;
        }

        private bool CompositeFactor(
             BigInteger factor, ref BigInteger n, ref
List<FactorExpon> lfe)
        {
            FactorExpon fe = new FactorExpon();

            if (!hcr.Composite(factor, 20))
            {
                n /= factor;
                fe.factor = factor;
                fe.expon = 1;
                lfe.Add(fe);

                if (n == 1)
                    return true;

                if (!hcr.Composite(n, 20))
                {
                    fe.factor = n;
                    fe.expon = 1;
                    lfe.Add(fe);
                    n = 1;
                    return true;
                }

                return false;
```

```
        }

        // factor is composite try trial division on factor

        BigInteger root = Sqrt(factor);

        for (int i = 0; i < primes.Count; i++)
        {
            long p = primes[i];

            if (p > root)
                return false;

            if (factor % p == 0)
            {
                int exp = 0;

                do
                {
                    exp++;
                    factor /= p;
                    n /= p;
                } while (n % p == 0);

                fe.factor = p;
                fe.expon = exp;
                lfe.Add(fe);

                if (n == 1)
                    return true;

                if (!hcr.Composite(n, 20))
                {
                    fe.factor = n;
                    fe.expon = 1;
                    lfe.Add(fe);
                    n = 1;
                    return true;
                }

                if (factor == 1)
                    return false;
            }
        }

        return false;
    }

    public struct State
    {
        public int i0, i1, number;
        public BigInteger m, n;
        public List<int> p;
```

```
    public List<BigInteger> la, lb;
    public List<List<PrimeExpon>> lpel;

    public State(
        int i0, int i1, int number,
        BigInteger m, BigInteger n,
        List<int> p)
    {
        this.i0 = i0;
        this.i1 = i1;
        this.number = number;
        this.m = m;
        this.n = n;
        this.p = p;
        la = new List<BigInteger>();
        lb = new List<BigInteger>();
        lpel = new List<List<PrimeExpon>>();
    }
}

public void FactorTask(object o)
{
    int num = 0;
    BigInteger x = 0;
    HCSRAlgorithm hc = new HCSRAlgorithm(1);
    State s = (State)o;

    for (int i = s.i0; i <= s.i1; i++)
    {
        if (i == 1)
        {
            x = 0;
        }

        else if (i % 2 == 0)
        {
            x = +i / 2;
        }

        else
        {
            x = -i / 2;
        }

        BigInteger xm = x + s.m, b = xm * xm - s.n;
        List<PrimeExpon> lpe = new List<PrimeExpon>();
        int result = -1;
        BigInteger tb = b, tsb;
        PrimeExpon pe = new PrimeExpon();

        if (b == 0 || b == 1 || b == -1)
            result = -1;
```

```
                else
                {
                    if (tb < 0)
                    {
                        tb = -tb;
                        pe.prime = -1;
                        pe.expon = +1;
                        lpe.Add(pe);
                    }

                    tsb = Sqrt(tb);

                    for (int j = 1; result == -1 && j <
s.p.Count; j++)
                    {
                        int exp = 0;
                        BigInteger q = s.p[j];

                        if (q > tsb)
                        {
                            result = 0;
                        }

                        else if (tb % q == 0)
                        {
                            do
                            {
                                exp++;
                                tb /= q;
                            } while (tb % q == 0);

                            pe.prime = (int)q;
                            pe.expon = exp;
                            lpe.Add(pe);

                            if (tb == 1 || !hc.Composite(tb,
20))
                            {
                                result = 1;
                            }
                        }
                    }
                }

                if (result == 1)
                {
                    s.la.Add(xm);
                    s.lb.Add(b);
                    s.lpel.Add(lpe);
                    num++;

                    if (num == s.number)
                    {
```

```
                    return;
                }
            }
        }
    }

    public int Sieve(
        ref BigInteger n,
        ref List<long> times,
        ref List<FactorExpon> lfe)
    {
        // resutns -2 number is too small
        // returns -1 algorithm failed to find factor
        // returns  0 number is probably prime
        // returns  1 number was not completely factored
        // returns  2 number was completely factored

        if (n <= 100)
            return -2;

        if (!hcr.Composite(n, 20))
            return 0;

        bool done = false;
        int count = 0, result = -1;
        int number = 1 + t1 / 4;
        object block = new object();
        List<int> p = null;
        List<BigInteger> la = new List<BigInteger>();
        List<BigInteger> lb = new List<BigInteger>();
        List<BigInteger> ff = new List<BigInteger>();
        State[] states = new State[4];
        Task[] tasks = new Task[4];

        sw.Start();

        while (!done)
        {
            p = new List<int>();
            p.Add(-1);

            for (int j = 0; !done && j < primes.Count; j++)
            {
                BigInteger q = primes[j];

                if (hcr.Jacobi(n, q) == 1)
                {
                    p.Add((int)q);
                    done = p.Count == t;
                }
            }

            if (!done)
```

```
                {
                    B0 += 1000000;
                    primes = new List<long>();
                    sp.Sieve(B0, ref primes);
                }
            }

            sw.Stop();
            times.Add(sw.ElapsedMilliseconds);

            BigInteger m = Sqrt(n), x = 0;

            states[0] = new State(       1, 10000000, number, m,
n, p);
            states[1] = new State(10000001, 20000000, number, m,
n, p);
            states[2] = new State(20000001, 30000000, number, m,
n, p);
            states[3] = new State(30000001, 40000000, number, m,
n, p);
            sw.Restart();

            for (int j = 0; j < 4; j++)
            {
                tasks[j] = new Task(FactorTask, states[j]);
                tasks[j].Start();
            }

            Task.WaitAll(tasks);

            // merge data from the four tasks

            for (int j = 0; count < t1 && j < 4; j++)
            {
                State s = states[j];

                if (s.la.Count != number)
                    return -1;

                for (int k = 0; count < t1 && k < s.number; k++)
                {
                    la.Add(s.la[k]);
                    lb.Add(s.lb[k]);

                    for (int l = 0; l < t; l++)
                        e[count, l] = v[count, l] = 0;

                    List<PrimeExpon> lpe = s.lpel[k];

                    for (int l = 0; l < t; l++)
                    {
                        int q = s.p[l];
```

```
                        for (int u = 0; u < lpe.Count; u++)
                        {
                            PrimeExpon pe = lpe[u];

                            if (q == pe.prime)
                            {
                                e[count, l] = (sbyte)pe.expon;
                                v[count, l] = (sbyte)(pe.expon %
2);
                                break;
                            }
                        }
                    }

                    count++;
                }
            }

            sw.Stop();
            times.Add(sw.ElapsedMilliseconds);

            int r = 0;
            int[] d = new int[t];

            sw.Restart();
            hcl.KernelOverZ2(t1, t, d, v, ref r);
            sw.Stop();
            times.Add(sw.ElapsedMilliseconds);
            sw.Restart();

            BigInteger X = 1;

            for (int k = 0; k < t1; k++)
            {
                X = (la[k] * X) % n;
            }

            for (int k = 0; k < t; k++)
            {
                int dk = d[k];

                if (dk >= 0)
                {
                    BigInteger Y = 1;

                    for (int j = 0; j < t; j++)
                    {
                        if (e[dk, j] % 2 != 0)
                        {
                            Y = (p[j] * Y) % n;
                        }
                    }
```

```
                    BigInteger XmY = (X - Y) % n;

                    if (XmY != 0 && XmY != 1)
                    {
                        BigInteger factor =
BigInteger.GreatestCommonDivisor(XmY, n);

                        if (factor > 1 && !ff.Contains(factor))
                        {
                            ff.Add(factor);

                            if (CompositeFactor(factor, ref n,
ref lfe))
                            {
                                result = 2;
                            }

                            else
                            {
                                result = 1;
                            }
                        }
                    }
                }
            }

            sw.Stop();
            times.Add(sw.ElapsedMilliseconds);
            return result;
        }
    }
}
```

Клас SievePrimes

```
using System.Collections.Generic;

namespace QuadraticSieve
{
    class SievePrimes
    {
        public void Sieve(long B0, ref List<long> primes)
        {
            // Sieve of Eratosthenes
            // find all prime numbers
            // less than or equal B0

            bool[] sieve = new bool[B0 + 1];
            long c = 3, i, inc;

            sieve[2] = true;
```

```
            for (i = 3; i <= B0; i++)
                if (i % 2 == 1)
                    sieve[i] = true;

            do
            {
                i = c * c;
                inc = c + c;

                while (i <= B0)
                {
                    sieve[i] = false;

                    i += inc;
                }

                c += 2;

                while (!sieve[c])
                    c++;
            } while (c * c <= B0);

            if (primes == null)
                primes = new List<long>();

            for (i = 2; i <= B0; i++)
                if (sieve[i])
                    primes.Add(i);
        }
    }
}
```

QS – алгоритм з використанням чисел Мерсенна

Клас MainForm

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Diagnostics;
using System.Numerics;
using System.Windows.Forms;

namespace MersenQuadraticSieve
{
    public partial class mersen : Form
    {
        private int result, t;
        private long B0 = 10000000;
        private BackgroundWorker bw;
        private BigInteger N, n;
        private MQuadraticSieve qs;
        private Stopwatch sw;
        private List<long> times;
        private List<FactorExpon> lfe;

        public mersen()
        {
            InitializeComponent();
        }

        private BigInteger Horner(string s)
        {
            bool negative;
            int first;
            BigInteger result;

            if (s[0] == '-')
            {
                negative = true;
                first = 1;
            }
            else if (s[0] == '+')
            {
                negative = false;
                first = 1;
            }

            else
            {
                negative = false;
                first = 0;
            }
```

```
            result = new BigInteger((int)(s[first] - '0'));

            for (int i = first + 1; i < s.Length; i++)
                    result = 10 * result + (int)(s[i] - '0');

            if (negative)
                result = -result;

            return result;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            try
            {

                long BASE = (long)numericUpDown1.Value;
                int expon = (int)numericUpDown2.Value;
                BigInteger add = Horner(textBox1.Text);
                t = (int)numericUpDown5.Value;
                N = n = BigInteger.Pow(BASE, expon) + add;
                bw = new BackgroundWorker();
                bw.WorkerSupportsCancellation = true;
                bw.DoWork += Bw_DoWork;
                bw.RunWorkerCompleted += Bw_RunWorkerCompleted;
                qs = new MQuadraticSieve(1, t, B0);
                bw.RunWorkerAsync();
                while (!bw.IsBusy) { }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString(), "Warning",
                    MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            }
        }

        private void Bw_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {
            TimeSpan ts = sw.Elapsed;
            string text = string.Empty;
            textBox2.Text += "N = " ;
            textBox2.Text += N.ToString() + "\r\n";

            if (result == -2)
            {
                MessageBox.Show("Число дуже мале для розгляду",
"Warning",
                    MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            }
```

```
            else if (result == -1)
            {
                MessageBox.Show("Алгоритму не вдалось знайти
фактор", "Warning",
                    MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            }

            else if (result == 0)
            {
                MessageBox.Show("Число является простим",
"Information",
                   MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }

            else if (result == 1)
            {
                MessageBox.Show("Число не повністю
факторизовано", "Information",
                  MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }

            else if (result == 2)
            {
                MessageBox.Show("Число повністю факторизовано",
"Information",
                  MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }

            if (lfe.Count > 0)
            {
                // selection sort the factors into ascending
order

                for (int i = 0; i < lfe.Count - 1; i++)
                {
                    for (int j = i + 1; j < lfe.Count; j++)
                    {
                        if (lfe[i].factor > lfe[j].factor)
                        {
                            FactorExpon temp = lfe[i];

                            lfe[i] = lfe[j];
                            lfe[j] = temp;
                        }
                    }
                }

                for (int i = 0; i < lfe.Count; i++)
                {
```

```
                textBox2.Text += lfe[i].factor.ToString();

                if (lfe[i].expon > 1)
                    textBox2.Text += " ^ " + lfe[i].expon;

                textBox2.Text += "\r\n";
            }

            textBox2.Text += "\r\n";
        }

        if (result == 1)
        {
            textBox2.Text += "Composite Residue" + "\r\n";
            textBox2.Text += n.ToString() + "\r\n\r\n";
        }

        text += ts.Hours.ToString("D2") + ":";
        text += ts.Minutes.ToString("D2") + ":";
        text += ts.Seconds.ToString("D2") + ".";
        text += ts.Milliseconds.ToString("D3") + "\r\n\r\n";

        if (times.Count == 4)
        {
            text += "Prime creation milliseconds = ";
            text += times[0].ToString() + "\r\n";
            //text += "Multitasking milliseconds   = ";
            //text += times[1].ToString() + "\r\n";
            text += "Linear algebra milliseconds = ";
            text += times[2].ToString() + "\r\n";
            text += "Final phase milliseconds    = ";
            text += times[3].ToString() + "\r\n";
        }

        textBox2.Text += text + "\r\n\r\n";
        button1.Text = "Факторизувати";
    }

    private void Bw_DoWork(object sender, DoWorkEventArgs e)
    {
        times = new List<long>();
        lfe = new List<FactorExpon>();
        sw = new Stopwatch();
        sw.Start();
        result = qs.Sieve(ref n, ref times, ref lfe);
        sw.Stop();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        textBox2.Text = string.Empty;
    }
```

```
        private void textBox1_TextChanged(object sender,
EventArgs e)
        {

        }

        private void numericUpDown5_ValueChanged(object sender,
EventArgs e)
        {

        }

        private void label4_Click(object sender, EventArgs e)
        {

        }

        private void textBox2_TextChanged(object sender,
EventArgs e)
        {

        }
    }
}
```