Алгоритмы для нахождения наибольшего общего делителя и наименьшего общего кратного.

Низамутдинова Артура Салаватовича

2013

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

НОД – наибольший общий делитель;

НОК – наименьшее общее кратное;

ВВЕДЕНИЕ

Алгоритмы для нахождения наибольшего общего делителя (НОД) и наименьшего общего кратного (НОК) в настоящее время являются довольно значимыми. К примеру НОД применяется в криптографическом алгоритме с открытым ключом RSA (аббревиатура от фамилий Rivest, Shamir и Adleman), а также имеет частое применение в математике. В свою очередь с помощью НОК производится приведение дробей к общему знаменателю.

Целью настоящей работы является приведение примеров, а также сравнение эффективности алгоритмов для нахождения НОД и НОК двух натуральных чисел.

Поставленные задачи:

- рассмотреть понятия НОД и НОК;
- рассмотреть алгоритмы для их нахождения;
- подробно описать эти алгоритмы;
- реализовать алгоритмы в Microsoft Visual Studio C++ 2010;
- сравнить эффективность алгоритмов для их нахождения.

1 История

Математики в древней Греции называли алгоритм для нахождения НОД «взаимное вычитание». Этот алгоритм был впервые описан в книге Евклида «Начала» (около 300 г. до н.э.), хотя некоторые ученые предполагают, что данный метод был известен за 200 лет до этого, по крайней мере в той форме, которая использует вычитания. В «Началах» Евклида данный алгоритм описывается два раза — в седьмой книге для вычисления НОД двух натуральных чисел, а также в десятой книге для вычисления НОК двух чисел. В каждом из этих случаев дано геометрическое описание, для поиска «общей меры» двух отрезков. Однако данный алгоритм для нахождения НОД двух натуральных чисел также описан в I книге древнекитайского писания «Математика в девяти томах».

Алгоритм Евклида можно назвать дедушкой всех алгоритмов, поскольку он самый старый из всех нетривиальных алгоритмов, дошедших до наших дней. [1]

2 Наибольший общий делитель и наименьшее общее кратное

Наибольший общий делитель целых a, b — это такой их общий делитель, который делится на любой общий делитель этих чисел.

Наименьшее общее кратное целых a, b — это такое наименьшее натуральное число, которое делится на a и b.

Если числа a и b представить в виде $a = {p_1}^{d_1} \cdot \ldots \cdot {p_n}^{d_n}$ $b = {p_1}^{e_1} \cdot \ldots \cdot {p_n}^{e_n}$, где $p_1, \ldots, p_n$ — простые числа, $e_1, \ldots, e_n$ и $d_1, \ldots, d_n$ — целые неотрицательные числа (некоторые из них могут быть равны нулю, если соответствующего простого числа нет в разложении). Тогда НОД и НОК этих чисел можно представить в следующем виде:

$$\text{НОД}(a,b) = {p_1}^{min(d_1,e_1)} \cdot \ldots \cdot {p_n}^{min(d_n,e_n)}$$

$$\text{НОК}(a,b) = {p_1}^{max(d_1,e_1)} \cdot \ldots \cdot {p_n}^{max(d_n,e_n)}$$

Пусть a и b — два каких-то целых числа, не равных одновременно нулю; рассмотрим совокупность всех натуральных чисел, на которые делятся и a и b. Эта совокупность, несомненно, конечная, так как если, например, $a \neq 0$, то никакое число, большее чем a, не может быть делителем a. Отсюда следует, что число общих делителей a и b конечно; пусть через d обозначен наибольший из них. Число d называется общим наибольшим делителем a и b, и мы условимся обозначать его $d = (a, b)$. Если a и b — достаточно большие числа, например $a = 24157819$, $b = 39088167$, то попытки найти общий наибольший делитель с помощью непосредственных проб довольно утомительны [2]. Короткий и вполне надежный метод вытекает из алгоритмов нахождения НОД.

3 Алгоритмы нахождения наибольшего общего делителя

Для вычисления НОД двух чисел существуют следующие алгоритмы:

1. Античный алгоритм Евклида (через разности);
2. Алгоритм Евклида (через остатки);
3. Нахождение НОД методом перебора;
4. Бинарный алгоритм Стейна.

НОК можно вычислить при помощи следующей формулы

$$\text{НОК} = \frac{ab}{\text{НОД}(a, b)}.$$

3.1 Античный алгоритм Евклида

Суть алгоритма заключается в том, что если даны два числа, и одно из них равно нулю, то в ответ записывается большее из них. Если ни одно из них не равно нулю, тогда вычитаем из большего числа меньшее, а потом снова проверяем условие равенства одного из них нулю.

3.2 Алгоритм Евклида

В алгоритме Евклида, если одно из чисел равно нулю, то в ответ записывается наибольшее из них. Иначе большему числу присваиваем остаток от деления его на меньшее, а потом снова проверяем условие равенства одного из них нулю.

3.3 Алгоритм нахождения НОД методом перебора

В данном случае мы выбираем $max(a, b)$ и уменьшаем это число на единицу до тех пор, пока a и b не станут одновременно кратны ему.

3.4 Бинарный алгоритм Стейна

В 1961 году израильский физик и программист Джозеф Стейн (Josef Stein) представил совершенно иной алгоритм нахождения наибольшего общего делителя [3], использующий, прежде всего, на бинарную арифметику. Этому новому алгоритму совершенно не нужны команды, совершающие операции деления. Основанный исключительно на операциях вычитания, он проверяет, является ли число четным, и делит пополам четные числа (что соответствует в бинарной арифметике сдвигу вправо). Бинарный алгоритм поиска наибольшего общего делителя основан прежде всего на четырех простых фактах относительно положительных целых чисел a и b:

1. Если оба числа четны, то

$$\text{НОД}(2a, 2b) = 2\,\text{НОД}(a, b).$$

2. Если первое число четно, а второе нечетно, то

$$\text{НОД}(2a, 2b + 1) = \text{НОД}(a, 2b + 1).$$

3. Также как и в алгоритме Евклида,

$$\text{НОД}(a, b) = \text{НОД}(a - b, b).$$

4. Если оба числа нечетные, то $a - b$ - четно и

$$|a - b| < max(a, b).$$

4 Реализация алгоритмов

В данном разделе приведены описания следующих алгоритмов:

1. Античный алгоритм Евклида (через разности);
2. Алгоритм Евклида (через остатки);
3. Нахождение НОД методом перебора;
4. Бинарный алгоритм Стейна.

Листинг программ, реализующих эти алгоритмы можно увидеть в приложениях А—Г. Стоит заметить, что алгоритмы представленные в этих приложениях были написаны на языке C++ в среде Visual Studio 2010, и их тестирование на скорость вычисления поставленной задачи проводилось в операционной системе Windows 8, на компьютере HP Envy m6 1154er с процессором Intel Core i5 3210M, тактовая частота которого 2.5 ГГц.

4.1 Подробное описание алгоритмов

Далее приведены краткие рекомендации по работе с представленными программами, блок-схемы и скриншоты работы программ.

4.1.1 Античный алгоритм Евклида

На вход программе должно быть подано два целых положительных числа (те самые числа a и b) и нажата клавиша Enter. Далее во второй, третьей и четвертой строке появятся НОД(a, b), НОК(a, b) и время работы программы в миллисекундах соответственно. Общий вид реализованного алгоритма представлен на рисунке 1. Код программы можно увидеть в приложении А.

На рисунке 2 можно увидеть скриншот работы программы.

4.1.2 Алгоритм Евклида

Здесь также, как и в предыдущем описании, на вход программе должно быть подано два целых положительных числа (те самые числа a и b) и нажата клавиша Enter. Далее во второй, третьей и четвертой строке появятся НОД(a, b), НОК(a, b) и время работы программы в миллисекундах соответственно. Код программы можно увидеть в приложении Б. Общий вид реализованного алгоритма представлен на рисунке 3.

На рисунке 4 можно увидеть скриншот работы программы.

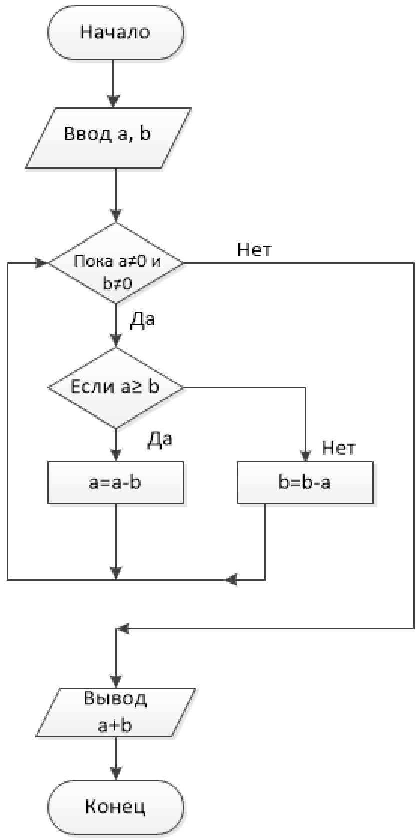

Рисунок 1 – Блок-схема античного алгоритма Евклида

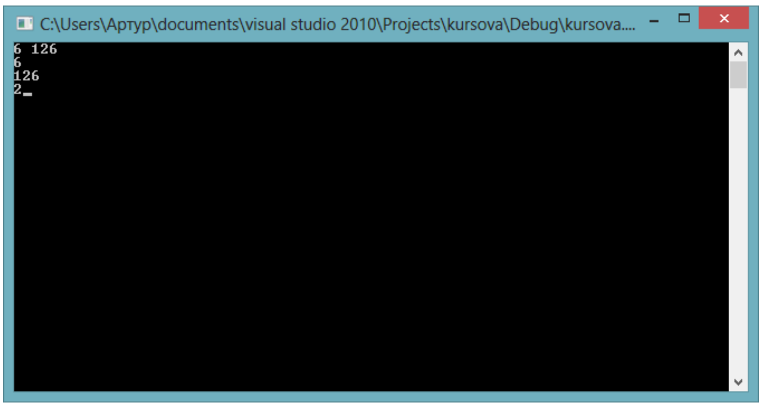

Рисунок 2 – Работа античного алгоритма Евклида

4.1.3 Алгоритм нахождения НОД методом перебора

Здесь также, как и в двух предыдущих описаниях, на вход программе должно быть подано два целых положительных числа (те самые числа a и b) и нажата клавиша Enter. Далее во второй, третьей и четвертой строке появятся НОД(a, b), НОК(a, b) и время работы программы в миллисекундах соответственно. Код программы можно увидеть в приложении В. На рисунке 5 представлена блок-схема реализованного алгоритма.

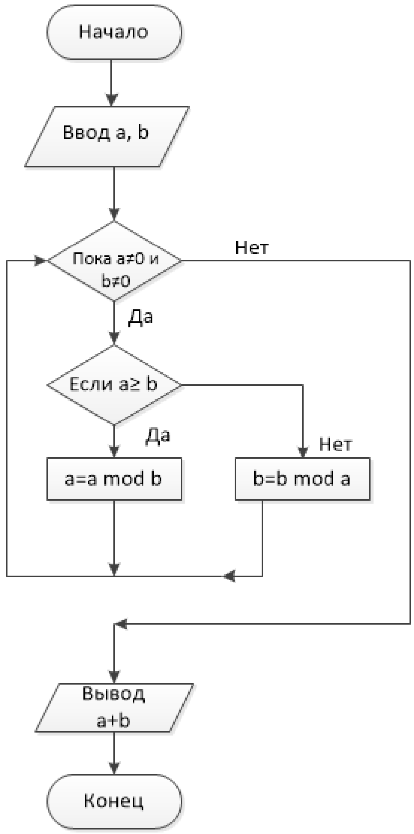

Рисунок 3 – Блок-схема алгоритма Евклида

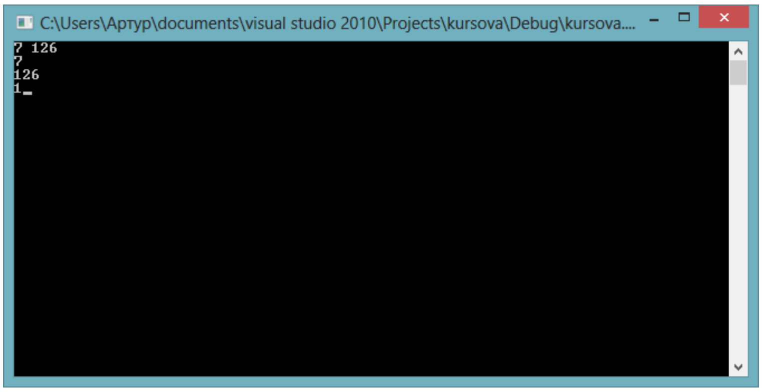

Рисунок 4 – Работа алгоритма Евклида

На рисунке 6 можно увидеть скриншот работы программы.

4.1.4 Бинарный алгоритм Стейна

Здесь также, как и в трех предыдущих описаниях, на вход программе должно быть подано два целых положительных числа a и b и нажата клавиша Enter. Далее во второй, третьей и четвертой строке появятся НОД(a, b), НОК(a, b) и время работы программы в миллисекундах соответственно. Код программы можно увидеть в приложении Г. Общий вид реализованного алго-

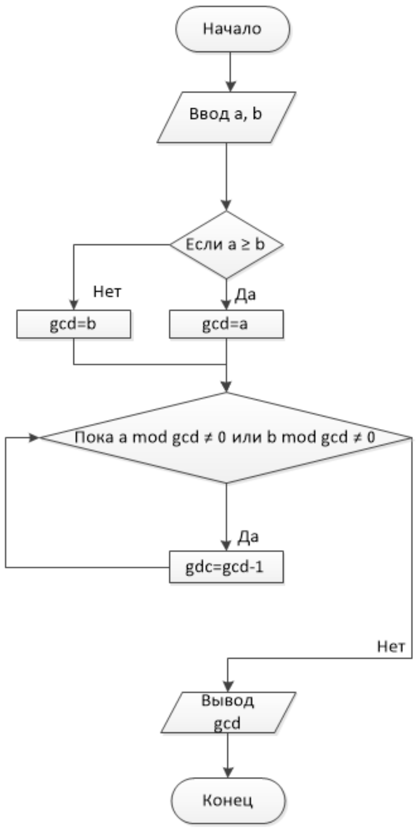

Рисунок 5 – Блок-схема алгоритма для нахождения НОД методом перебора

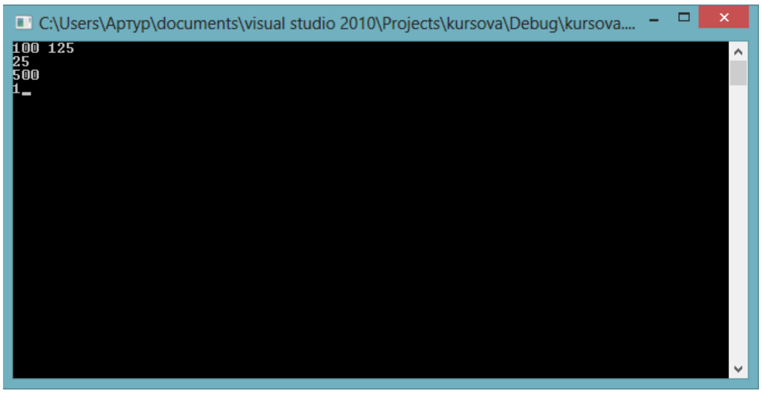

Рисунок 6 – Работа алгоритма для нахождения НОД методом перебора

ритма представлен в виде блок-схемы на рисунке 7.

На рисунке 8 можно увидеть скриншот работы программы.

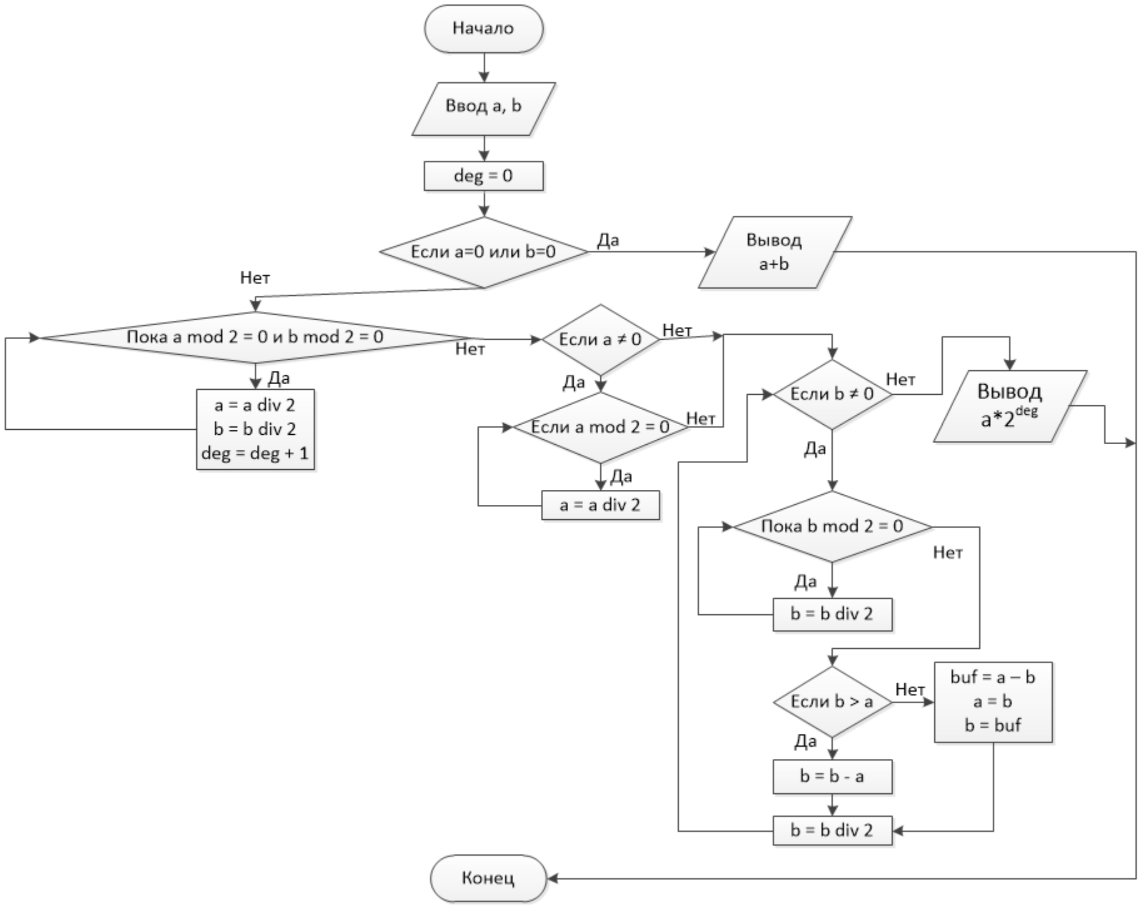

Рисунок 7 – Блок-схема бинарного алгоритма Стейна

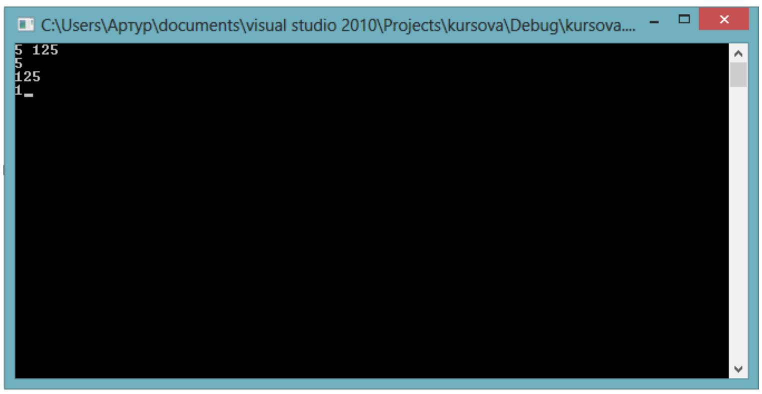

Рисунок 8 – Работа бинарного алгоритма Стейна

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Кнут, Д. Э.* Искусство программирования Том 2 / Д. Э. Кнут. — М.: Вильямс, 2002.

2 *Курант, Р.* Что такое математика? / Р. Курант, Г. Роббинс. — М.: МЦНМО, 2000.

3 *Stein, J.* Computational problems associated with racah algebra / J. Stein // *Journal of Computational Physics*. — 1967. — Vol. 1. — P. 397.

ПРИЛОЖЕНИЕ А

Листинг античного алгоритма Евклида, написанного на Microsoft Visual Studio C++ 2010

Код приложения obalg.cpp.

```
// обычный алгоритм Евклида через вычитания
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <math.h>
#include "windows.h"

using namespace std;

unsigned long long int gcd(unsigned long long int a, unsigned long long int b)
{
    while (a && b) //пока оба числа не равны нулю
        if (a >= b)//если a больше или равно b
           a -= b; //вычитаем из a число b
        else       //иначе
           b -= a; //вычитаем из b число a
    return a | b;  //возвращаем a + b
}

int main()
{
        //объявление переменных, предназначенных
        //для вычисления времени работы программы
        LARGE_INTEGER start, finish, freq;

        unsigned long long int a, b, nod;
        //a,b - исходные числа
        //nod - наибольший общий делитель
        cin >> a >> b;

        // запоминаем частоту операций

        QueryPerformanceFrequency( &freq );

        // засекаем начало работы алгоритма

```

```
        QueryPerformanceCounter( &start );

        nod = gcd(a,b);

        cout << nod << endl;//наибольший общий делитель

        cout << a*b/nod << endl;//наименьшее общее кратное

        // time - время в секундах

        double time = (finish.QuadPart - start.QuadPart) / (double)freq.QuadPart;

        cout << time << endl;

        getch();
           return 0;
}
```

ПРИЛОЖЕНИЕ Б

Листинг алгоритма Евклида, написанного на Microsoft Visual Studio C++ 2010

Код приложения ostalg.cpp.

```
// обычный алгоритм Евклида через остатки
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <math.h>
#include "windows.h"

using namespace std;

unsigned long long int gcd(unsigned long long int a, unsigned long long int b)
{
    while (a && b) //пока оба числа не равны нулю
        if (a >= b)//если a больше или равно b
           a %= b; //a присваиваем остаток от деления его на b
        else            //иначе
           b %= a; //b присваиваем остаток от деления его на a
    return a | b;  //возвращаем a + b
}

int main()
{
        //объявление переменных, предназначенных
        //для вычисления времени работы программы
        LARGE_INTEGER start, finish, freq;

        unsigned long long int a, b, nod;
        //a,b - исходные числа
        //nod - наибольший общий делитель
        cin >> a >> b;

        // запоминаем частоту операций

        QueryPerformanceFrequency( &freq );

        // засекаем начало работы алгоритма

```

```
        QueryPerformanceCounter( &start );

        nod = gcd(a,b);

        cout << nod << endl;//наибольший общий делитель

        cout << a*b/nod << endl;//наименьшее общее кратное

        // time - время в секундах

        double time = (finish.QuadPart - start.QuadPart) / (double)freq.QuadPart;

        cout << time << endl;

        getch();
           return 0;
}
```

ПРИЛОЖЕНИЕ В

Листинг алгоритма для нахождения НОД методом перебора, написанного на Microsoft Visual Studio C++ 2010

Код приложения `pralg.cpp`.

```
// Алгоритм для нахождения НОД методом перебора
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <math.h>
#include "windows.h"

using namespace std;

int main()
{
        //объявление переменных, предназначенных
        //для вычисления времени работы программы
        LARGE_INTEGER start, finish, freq;

        unsigned long long int a, b, gcd;
        //a,b - исходные числа
        //gcd - наибольший общий делитель

        cin >> a >> b;

        // запоминаем частоту операций

        QueryPerformanceFrequency( &freq );

        // засекаем начало работы алгоритма

        QueryPerformanceCounter( &start );

        if (a >= b) gcd = a; //выбираем max(a,b)
        else gcd = b;


        //пока a и b не кратны gcd, уменьшаем значение
        //gcd на единицу
        while (a % gcd != 0 || b % gcd != 0) gcd--;
```

```
    // засекаем окончание работы алгоритма
    QueryPerformanceCounter( &finish );


    cout << gcd << endl;//наибольший общий делитель a и b

    cout << a*b/gcd << endl;//наименьшее общее кратное a и b
    // time - время в секундах

    double time = (finish.QuadPart - start.QuadPart) / (double)freq.QuadPart;

    cout << time << endl;

    getch();
    return 0;
}
```

ПРИЛОЖЕНИЕ Г

Листинг бинарного алгоритма Стейна, написанного на Microsoft Visual Studio C++ 2010.

Код приложения `binalg.cpp`.

```
// бинарный алгоритм Стейна
#include "stdafx.h"
#include <iostream>
#include <conio.h>
#include <math.h>
#include "windows.h"

using namespace std;

unsigned long long int gcd(unsigned long long int a, unsigned long long int b)
{
    unsigned long long int buf, deg = 0;

    if (a == 0 || b == 0)//если a или b равно нулю
        return a | b;    //возвращаем a + b

    while (((a | b) & 1) == 0)//пока a и b нечётны
    {
        deg++;  //увеличиваем степень двойки на единицу
        a >>= 1;//делим a и b нацело на 2
        b >>= 1;
    }

    if (a)                     //если a не равно нулю
        while ((a & 1) == 0)//пока a чётно
            a >>= 1;        //делим его на 2

    while (b)//пока b не равно нулю
    {
        while ((b & 1) == 0)//пока b чётно
            b >>= 1;        //делим его на 2

        if (a < b)          //если a < b
            b -= a;         //вычитаем из b число a
        else                //иначе
        {
```

```
            buf = a - b;//сохраняем a - b
            a = b;        //присваиваем a число b
            b = buf;        //присваиваем b сохранённую разность
        }
        b >>= 1;        //делим b нацело на 2
    }

    //возвращаем a умноженное на 2 в степени deg
    return (a << deg);
}

int main()
{
        //объявление переменных, предназначенных
        //для вычисления времени работы программы
        LARGE_INTEGER start, finish, freq;

        unsigned long long int a, b, nod;
        //a,b - исходные числа
        //nod - наибольший общий делитель

        cin >> a >> b;


        // запоминаем частоту операций

        QueryPerformanceFrequency( &freq );

        // засекаем начало работы алгоритма

        QueryPerformanceCounter( &start );

        nod = gcd(a,b);

        cout << nod << endl;//наибольший общий делитель

        cout << a*b/nod << endl;//наименьшее общее кратное


        // time - время в секундах

```

```
        double time = (finish.QuadPart - start.QuadPart) / (double)freq.QuadPart;

        cout << time << endl;

        getch();
            return 0;
}
```

ПРИЛОЖЕНИЕ Д

Листинг программы для нахождения чисел Фибоначчи, написанного на Delphi 7.0

Код приложения `fib.dpr`.

```
program fib;//алгоритм для нахождения чисел Фибоначчи
            //использующий длинную арифметику
{$APPTYPE CONSOLE}
uses
  SysUtils;

  Const MaxDig = 1000;//максимальная длина числа
        Osn = 10000;  //основание системы счисления
  Type TLong = Array[0..MaxDig] Of LongInt;
  Var i, j, n : LongInt;
      a, b, c : TLong;

  //процедура для сложения a и b
  Procedure SumLongTwo(A, B : TLong; Var C : TLong);
   Var i, k : LongInt;
   Begin
     FillChar(C, SizeOf (C), 0);//заполняем массив C нулями

     //присваиваем k наибольшее из длин чисел a и b
     If A[0] > B[0] Then k := A[0]
     Else k := B[0];

     For i := 1 To k Do // C присваиваем A + B
               Begin
                 C[i+1] := (C[i] + A[i] + B[i]) Div Osn;
                 C[i] := (C[i] + A[i] + B[i]) Mod Osn;
               End;

     //уточняем длину числа C
     If C[k+1] = 0 Then C[0] := k
     Else C[0] := k + 1
   End;

  //процедура для вывода на экран длинного числа
  Procedure WriteLong(Const A : TLong);
   Var ls, s : String;
```

```
       i : LongInt;
   Begin
     //преобразуем число Osn Div 10
     //в строку ls
     Str(Osn Div 10, ls);

     Write(A[A[0]]); {выводим старшие цифры числа}

     For i := A[0] - 1 DownTo 1 Do
      Begin
        Str(A[i], s); {преобразуем чило A[i] в строку s}

        {дополняем незначащими нулями}
        While Length(s) < Length(ls) Do s := '0' + s;

        Write(s)
      End;
   End;

Begin
  ReadLn(n);

  //присваиваем a, b, c значение 1
  a[0] := 1;  a[1] := 1;
  b := a;  c := a;

  //вычисляем n-ое число Фибоначчи
  For i := 3 To n Do
   Begin
     SumLongTwo(a,b,c);
     a := b;
     b := c;
   End;

  //вывод n-го числа Фибоначчи
  WriteLong(c);

  ReadLn;
End.
```