



Государственное образовательное учреждение  
высшего профессионального образования  
«Московский Государственный Технический Университет  
имени Н.Э. Баумана»

### **Отчет**

По лабораторной работе №2

По курсу «Анализ Алгоритмов»

На тему «Исследование сложности алгоритмов умножения матриц»

Кизилов Дмитрий, ИУ7-54

Москва, 2017

# Оглавление

Постановка задачи . . . . .	2
Модель вычислений . . . . .	2
Блок-схемы . . . . .	3
Листинг . . . . .	6
Временные эксперименты . . . . .	12
Теоретическая оценка . . . . .	14
Выводы . . . . .	15
Заключение . . . . .	16

## Постановка задачи

Реализовать алгоритмы умножения матриц:

1. Стандартный
2. Винограда
3. Улучшенного Винограда

Рассчитать сложность алгоритмов и провести временные эксперименты

## Модель вычислений

Операнды  $+$ ,  $-$ ,  $*$ ,  $/$ ,  $<$ ,  $>$ ,  $<=$ ,  $>=$ ,  $==$ ,  $!=$ ,  $[ ]$  имеют значение  $f = 1$ ;  
Значение для цикла  $f = 2 + N(\dots)$ , где  $N$  - количество итераций.

## Блок-схемы

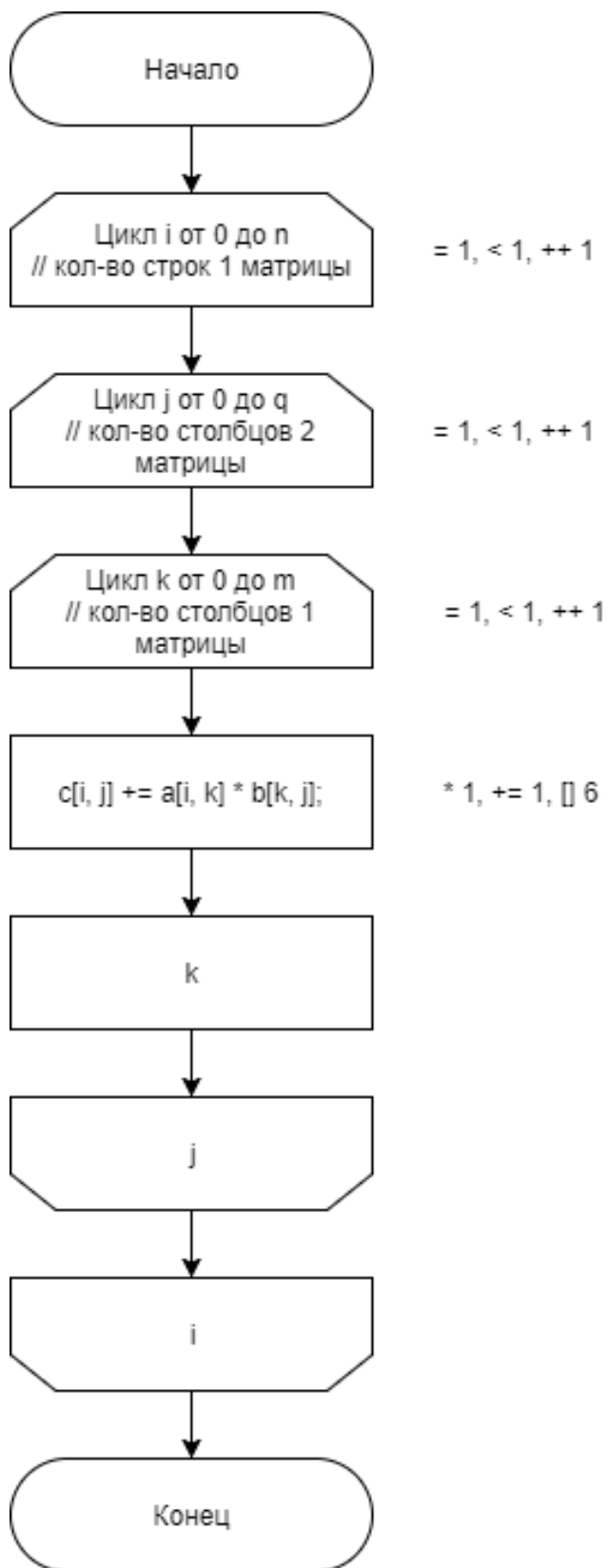


Рис. 1: Блок-схема базового алгоритма

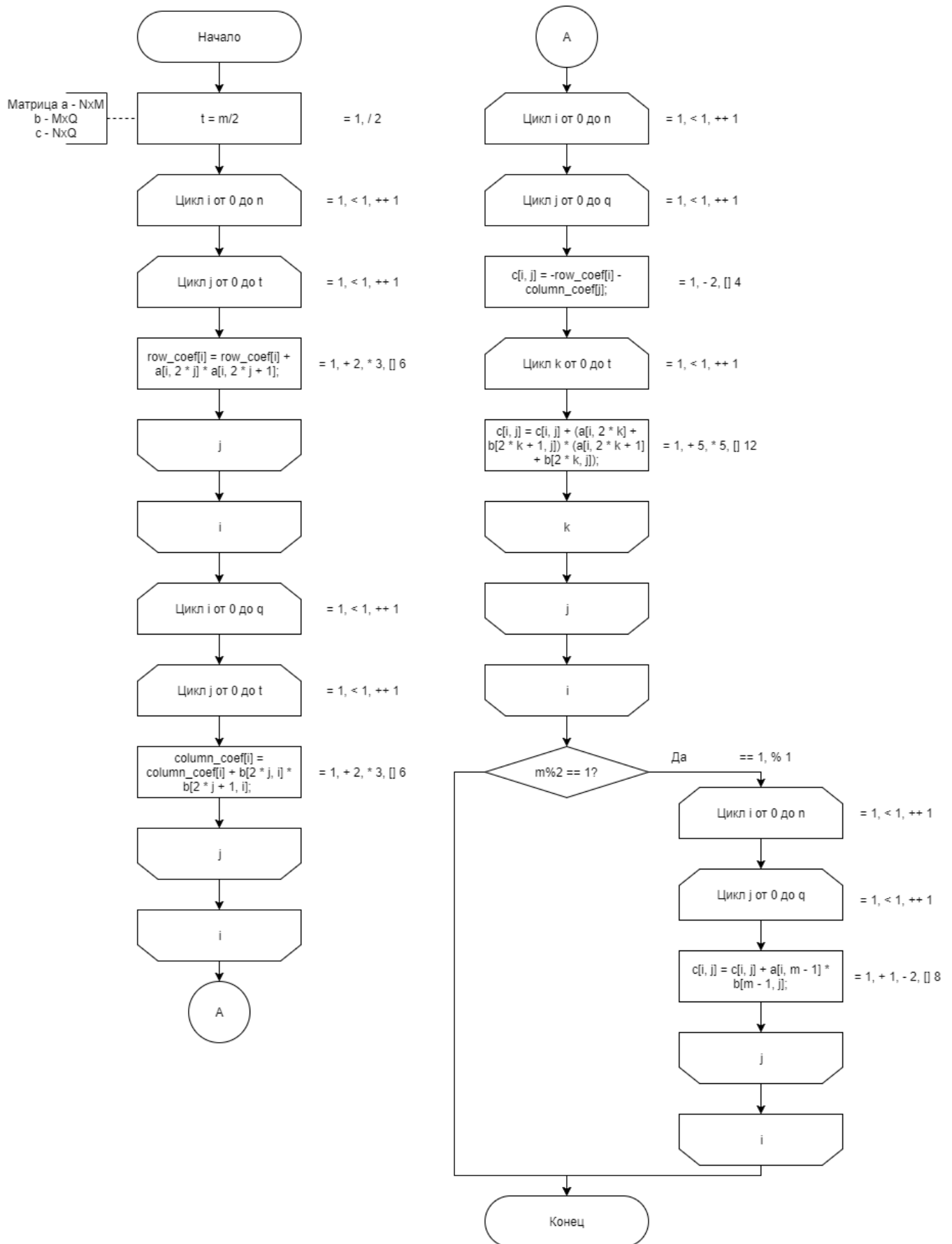


Рис. 2: Блок-схема алгоритма Винограда

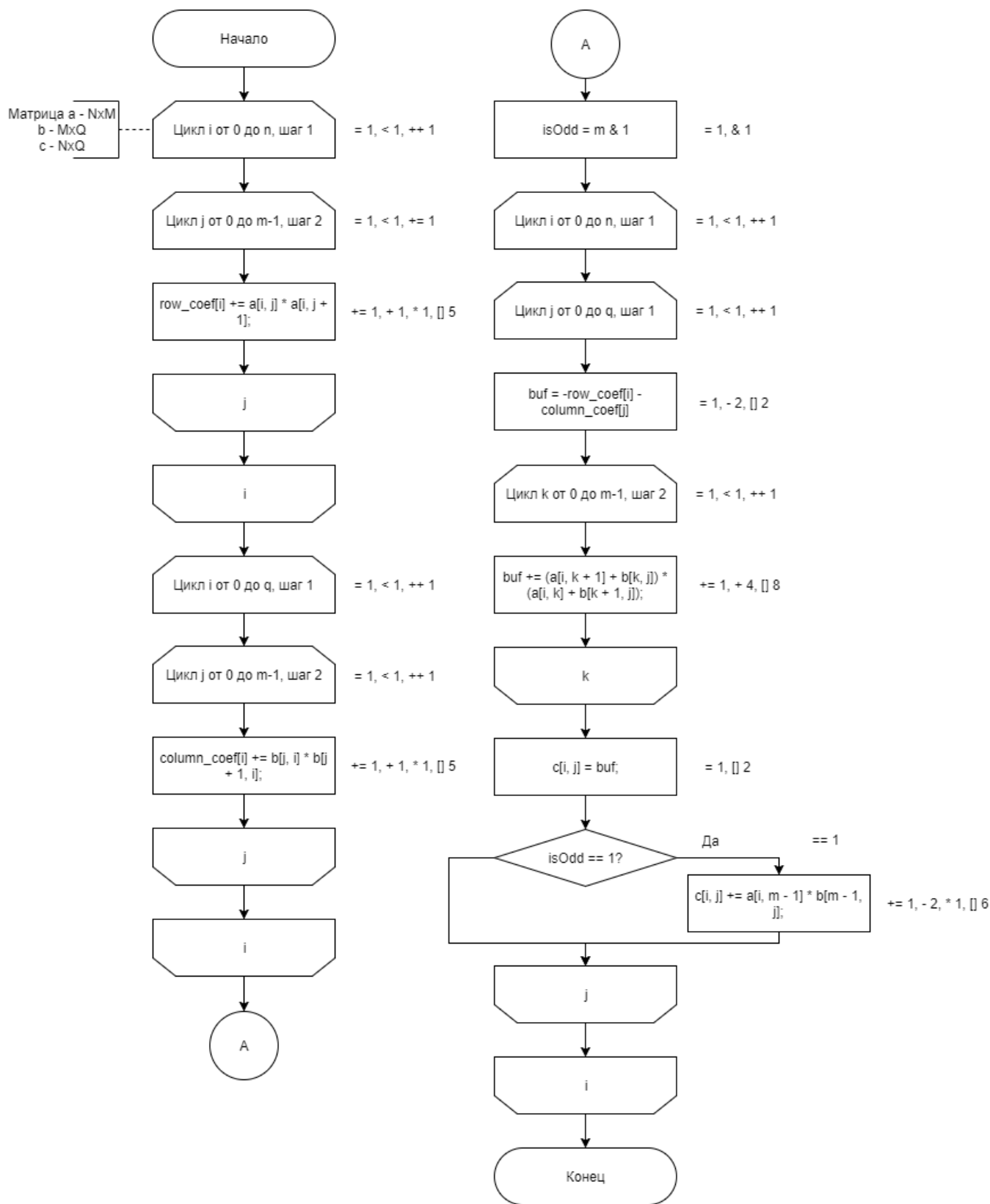


Рис. 3: Блок-схема улучшенного алгоритма Винограда

## Листинг

MAIN.CPP:

```
1  #include <iostream>
2  #include <fstream>
3  #include "mymat.cpp"
4
5  #define REP_NUM 7          // std == 7
6  #define REP_SIZE_NUM 10   // std == 10
7
8  using namespace std;
9
10 template <typename T>
11 void timeCounter(int st_size, int d_size, int rep_size_num, ofstream
    & file, int rep_num,
12                 long double (*f)(const vector<vector<T> >&, const
    vector<vector<T> >&))
13 {
14     vector<vector<T> > mat1;
15     vector<vector<T> > mat2;
16     long double res;
17
18     for (int i = 0; i < rep_size_num; i++)
19     {
20         randFill(mat1, st_size + d_size * i, st_size + d_size * i);
21         randFill(mat2, st_size + d_size * i, st_size + d_size * i);
22
23         res = 0;
24         for (int j = 0; j < rep_num; j++)
25         {
26             res += f(mat1, mat2);
27         }
28         res /= rep_num;
29
30         cout.setf(ios::fixed);
31         cout.precision(4);
32         cout << st_size+i*d_size << " : " << res << endl;
33
34         file.setf(ios::fixed);
35         file.precision(4);
36         file << res << "\n";
37     }
38     file << "\n";
39 }
40
41 int main(void)
42 {
43     srand(time(0));
44     ofstream file_ev("results_even.txt");
45     ofstream file_odd("results_odd.txt");
```

```

46
47     timeCounter<double>(100, 100, REP_SIZE_NUM, file_ev , REP_NUM,
        classicMatMult<double>);
48     timeCounter<double>(101, 100, REP_SIZE_NUM, file_odd , REP_NUM,
        classicMatMult<double>);
49
50     timeCounter<double>(100, 100, REP_SIZE_NUM, file_ev , REP_NUM,
        VinogradMatMult<double>);
51     timeCounter<double>(101, 100, REP_SIZE_NUM, file_odd , REP_NUM,
        VinogradMatMult<double>);
52
53     timeCounter<double>(100, 100, REP_SIZE_NUM, file_ev , REP_NUM,
        betterVinogradMatMult<double>);
54     timeCounter<double>(101, 100, REP_SIZE_NUM, file_odd , REP_NUM,
        betterVinogradMatMult<double>);
55
56     file_ev.close();
57     file_odd.close();
58
59     return 0;
60 }

```

#### MYMAT.CPP:

```

1  #include "mymat.h"
2
3  template <typename T>
4  long double classicMatMult(const vector<vector<T> >& mat1, const
        vector<vector<T> >& mat2)
5  {
6      vector<vector<T> > ans;
7      vector<T> tmp;
8      int n = mat1.size();
9      int m = mat2.size();
10     int q = mat2[0].size();
11
12     tmp.resize(q, 0);
13     ans.resize(n, tmp);
14
15     unsigned long int start = clock();
16
17     for (int i = 0; i < n; i++)
18     {
19         for (int j = 0; j < q; j++)
20         {
21             for (int k = 0; k < m; k++)
22             {
23                 ans[i][j] += mat1[i][k] * mat2[k][j];
24             }
25         }
26     }

```



```

27
28     return double(clock() - start)/double(CLOCKS_PER_SEC)*1000;
29 }
30
31 template <typename T>
32 long double VinogradMatMult(const vector<vector<T> >& mat1, const
    vector<vector<T> >& mat2)
33 {
34     vector<vector<T> > ans;
35     vector<T> tmp;
36
37     int n = mat1.size();
38     int m = mat2.size();
39     int q = mat2[0].size();
40
41     tmp.resize(q, 0);
42     ans.resize(n, tmp);
43
44     vector<T> row_factor;
45     vector<T> col_factor;
46
47     row_factor.resize(n, 0);
48     col_factor.resize(q, 0);
49
50     int d = m/2;
51
52     unsigned long int start = clock();
53
54     // Row Factor calc
55     for (int i = 0; i < n; i++)
56     {
57         for (int j = 0; j < d; j++)
58         {
59             row_factor[i] += mat1[i][2*j] * mat1[i][2*j+1];
60         }
61     }
62
63     // Col Factor calc
64     for (int i = 0; i < q; i++)
65     {
66         for (int j = 0; j < d; j++)
67         {
68             col_factor[i] += mat2[2*j][i] * mat2[2*j+1][i];
69         }
70     }
71
72     // Ans calc
73     for (int i = 0; i < n; i++)
74     {
75         for (int j = 0; j < q; j++)

```

```

76         {
77             ans[i][j] = - row_factor[i] - col_factor[j];
78             for (int k = 0; k < d; k++)
79             {
80                 ans[i][j] += (mat1[i][2*k] + mat2[2*k+1][j]) * (mat1
                        [i][2*k+1] + mat2[2*k][j]);
81             }
82         }
83     }
84
85     // For odd matrix
86     if (m % 2)
87     {
88         for (int i = 0; i < n; i++)
89         {
90             for (int j = 0; j < q; j++)
91             {
92                 ans[i][j] += mat1[i][m-1] * mat2[m-1][j];
93             }
94         }
95     }
96
97     return double(clock() - start)/double(CLOCKS_PER_SEC)*1000;
98 }
99
100 template <typename T>
101 long double betterVinogradMatMult(const vector<vector<T> >& mat1,
    const vector<vector<T> >& mat2)
102 {
103     vector<vector<T> > ans;
104     vector<T> tmp;
105
106     int n = mat1.size();
107     int m = mat2.size();
108     int q = mat2[0].size();
109
110     tmp.resize(q, 0);
111     ans.resize(n, tmp);
112
113     vector<T> row_factor;
114     vector<T> col_factor;
115
116     row_factor.resize(n, 0);
117     col_factor.resize(q, 0);
118
119     int d = m-1;
120
121     unsigned long int start = clock();
122
123     // Row Factor calc

```

```

124     for (int i = 0; i < n; i++)
125     {
126         for (int j = 0; j < d; j += 2)
127         {
128             row_factor[i] += mat1[i][j] * mat1[i][j+1];
129         }
130     }
131
132     // Col Factor calc
133     for (int i = 0; i < q; i++)
134     {
135         for (int j = 0; j < d; j += 2)
136         {
137             col_factor[i] += mat2[j][i] * mat2[j+1][i];
138         }
139     }
140
141     // Ans calc
142     for (int i = 0; i < n; i++)
143     {
144         for (int j = 0; j < q; j++)
145         {
146             T buf = -row_factor[i] - col_factor[j];
147             for (int k = 0; k < d; k += 2)
148             {
149                 buf += (mat1[i][k] + mat2[k+1][j]) * (mat1[i][k+1] +
150                     mat2[k][j]);
151             }
152             ans[i][j] = buf;
153         }
154     }
155
156     // For odd matrix
157     if (m % 2)
158     {
159         for (int i = 0; i < n; i++)
160         {
161             for (int j = 0; j < q; j++)
162             {
163                 ans[i][j] += mat1[i][m-1] * mat2[m-1][j];
164             }
165         }
166     }
167
168     return double(clock() - start)/double(CLOCKS_PER_SEC)*1000;
169 }
170
171 template <typename T>
172 void randFill(vector<vector<T> >& mat, const int row, const int col)

```

```

173 {
174     matClr(mat);
175     vector<T> tmp;
176
177     for (int i = 0; i < row; i++)
178     {
179         tmp.clear();
180
181         for (int j = 0; j < col; j++)
182         {
183             tmp.push_back(rand());
184         }
185
186         mat.push_back(tmp);
187     }
188 }
189
190 template <typename T>
191 void matClr(vector<vector<T> >& mat)
192 {
193     for (auto &x : mat)
194     {
195         x.clear();
196     }
197     mat.clear();
198 }

```

## Временные эксперименты

Измерения проводились для квадратных вещественных матриц

Размеры матриц	Обычное	Виноград	Улучшенный Виноград
100x100	24.6549	14.0676	11.3303
200x200	170.8249	137.3591	119.8764
300x300	488.6610	429.6541	355.9341
400x400	1358.2967	1082.4930	935.0479
500x500	3132.0469	2562.7789	2248.1820
600x600	5765.4314	4467.8990	3802.2509
700x700	8566.0461	7090.2327	6045.3397
800x800	12530.9540	10513.3806	9130.0609
900x900	18801.5250	15415.6954	13241.0050
1000x1000	26042.9380	24677.6349	26579.2120

ЗАМЕРЫ ВРЕМЕНИ В МИЛИСЕКУНДАХ (СРЕДНЕЕ ИЗ 7 ЗАМЕРОВ)



Рис. 4: Сравнение времени при четном кол-ве элементов

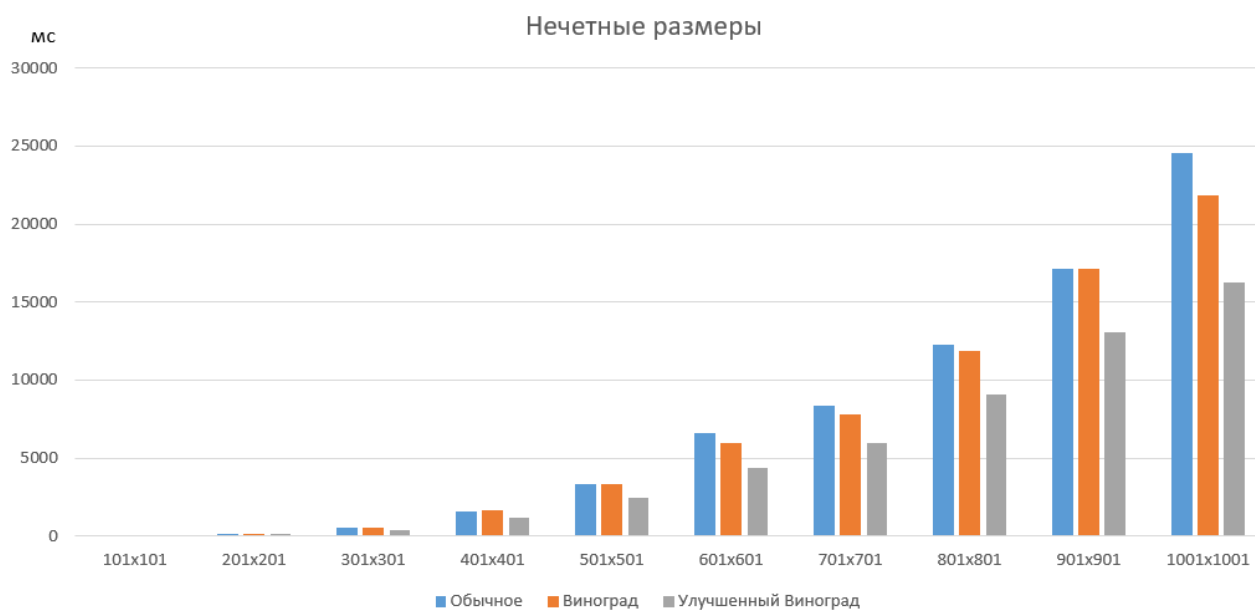


Рис. 5: Сравнение времени при нечетном кол-ве элементов

## Теоретическая оценка

а) Обычное умножение:

$$f = 2 + N * (2 + 2 + Q(2 + 2 + M(2 + 8))) = 2 + 4N + 4NQ + 10MNQ \sim O(n^3)$$

б) Умножение алгоритмом Винограда:

- Переменная t:  $f = 2$
- row coef:  $f = 2 + N(2 + 2 + 2 + M/2(2 + 12)) = 2 + 6N + 7MN$
- column coef:  $f = 2 + Q(2 + 2 + 2 + M/2(2 + 12)) = 2 + 6Q + 7MQ$
- Вычисление результирующей матрицы:  
 $f = 2 + N(2 + 2 + Q(2 + 7 + 2 + M/2(2 + 23))) = 2 + 4N + 11QN + 25MNQ/2$
- Вычисления последней строки:  $f = 2 + 2 + N(2 + 2 + Q(2 + 13)) = 4 + 4N + 15NQ$

Лучший случай (размер матрицы четный):

$$f = 2 + 2 + 6N + 7MN + 2 + 6Q + 7MQ + 2 + 4N + 11QN + 25MNQ/2 = 8 + 10N + 7MN + 6Q + 7QM + 11QN + 25QMN/2 \sim O(n^3)$$

Худший случай (размер матрицы нечетный):

$$f = 2 + 2 + 6N + 7MN + 2 + 6Q + 7MQ + 2 + 4N + 11QN + 25MNQ/2 + 4 + 4N + 15NQ = 12 + 14N + 7MN + 6Q + 7QM + 26QN + 25QMN/2 \sim O(n^3)$$

в) Умножение улучшенным алгоритмом Винограда:

- Переменная t:  $f = 2$
- row coef:  $f = 2 + N(2 + 2 + 2 + (M - 1)/2(2 + 8)) = 5MN + N + 2$
- column coef:  $f = 2 + Q(2 + 2 + 2 + (M - 1)/2(2 + 8)) = 5MQ + Q + 2$
- Переменная isOdd:  $f = 2$
- Вычисление результирующей матрицы:  
 $f = 2 + N(2 + 2 + Q(2 + 5 + 2 + (M - 1)/2(2 + 14) + 3[+1 + 10])) = 2 + 4N + 8MNQ + [5NQ|14NQ]$

Лучший случай (размер матрицы четный):

$$f = 2 + 5MN + N + 2 + 5MQ + Q + 2 + 2 + 2 + 4N + 8MNQ + 5NQ = 10 + Q + 5N + 5MN + 5MQ + 5NQ + 8MNQ \sim O(n^3)$$

Худший случай (размер матрицы нечетный):

$$f = 2 + 2 - 4N + 10MN + 2 - 4Q + 10MQ + 2 + 2 + 4N + 8MNQ + 13NQ = 10 + Q + 5N + 5MN + 5MQ + 13NQ + 8MNQ \sim O(n^3)$$

## Выводы

Сравнение алгоритмов по времени подтверждает теоретические замеры времени.

Алгоритмы работают примерно одинаково, что подтверждает их оценка трудоемкости порядка  $O(n^3)$ .

После умножения матриц размерами примерно 600x600 алгоритм Винограда работает быстрее обычного алгоритма умножения матриц постоянно.



## Заключение

В ходе лабораторной работы были реализованы 3 алгоритма умножения матриц: классический, алгоритм Винограда и улучшенным алгоритмом Винограда. Были получены навыки работы с матрицами и контейнером `std::vector` в C++, а так же с L<sup>A</sup>T<sub>E</sub>X. Изучен подход к вычислению сложности алгоритмов.