

**МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»**

**(МИИТ)**

|  |
| --- |
| **ИНСТИТУТ ТРАНСПОРТНОЙ ТЕХНИКИ И СИСТЕМ УПРАВЛЕНИЯ**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Кафедра «Управление и защита информации»

**Лабораторная работа №6**

**«Перемножение матриц»**

**по дисциплине**

**«Методы программирования»**

**Выполнил:** студент группы ТКИ-311

Куминов В. П.

**Проверил:** к.т.н., доц. Логинова Л. Н.,

к.т.н., доц. Сафронов А. И.

**Москва – 2022 г.**

**1. Цель работы**

Изучить реализацию алгоритма перемножения матриц.

**2. Формулировка задачи**

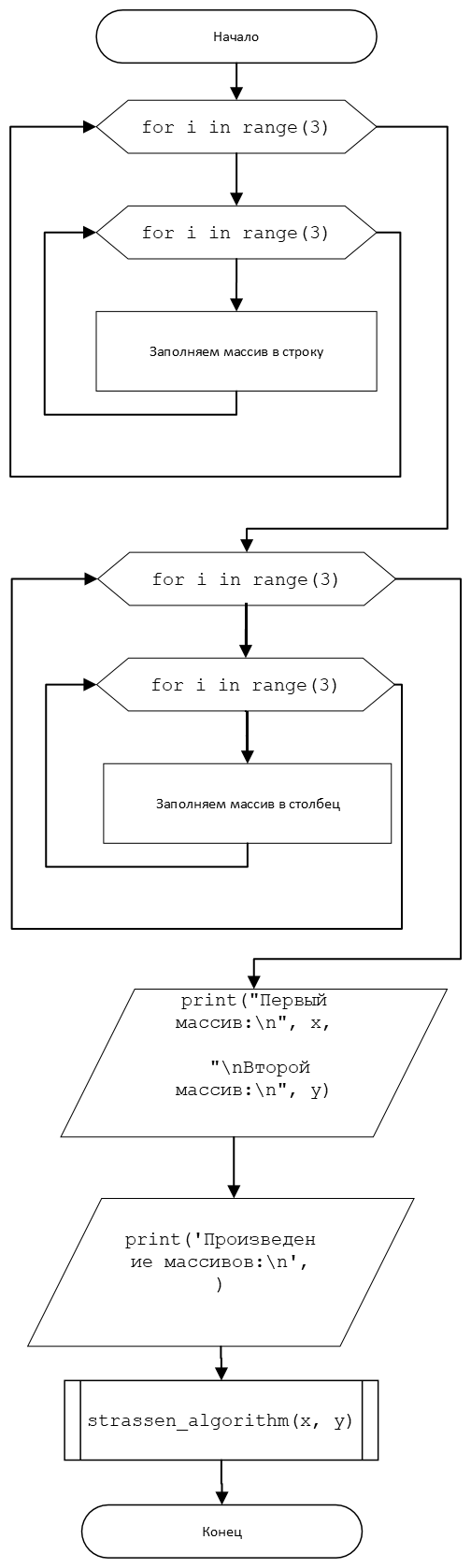
Реализуйте 3 программы, выполняющие перемножения матриц следующими методами:

1. Метод «Грубой силы»
2. Метод Штрассена
3. Метод «разделяй и властвуй»

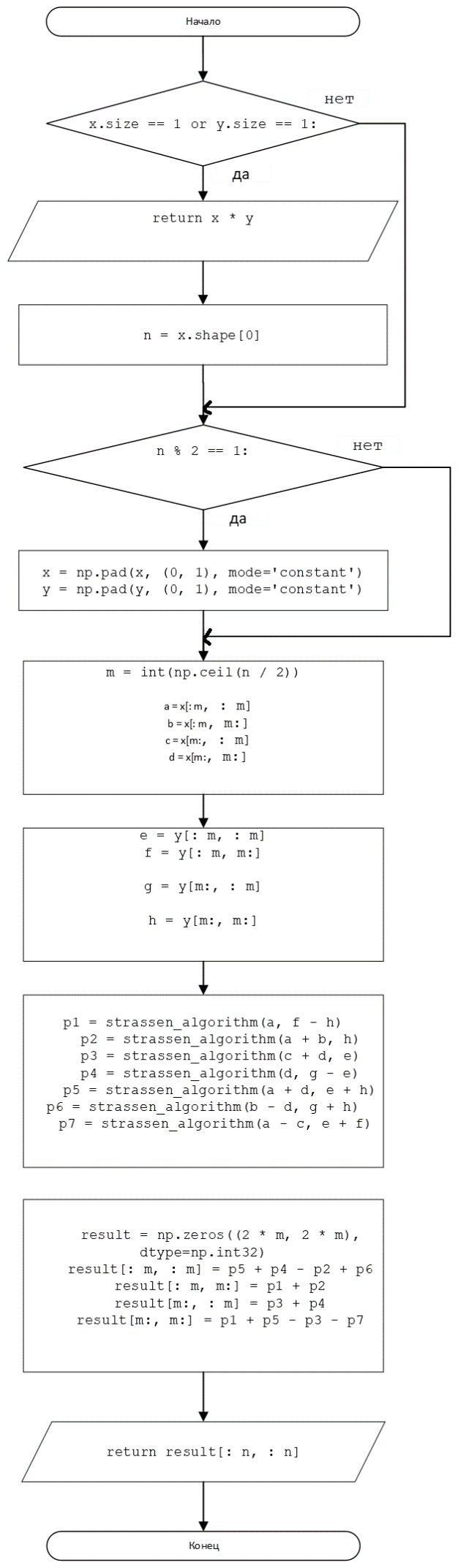
**3. Блок схема**

3.1. Метод Штрассена

Основная программа:

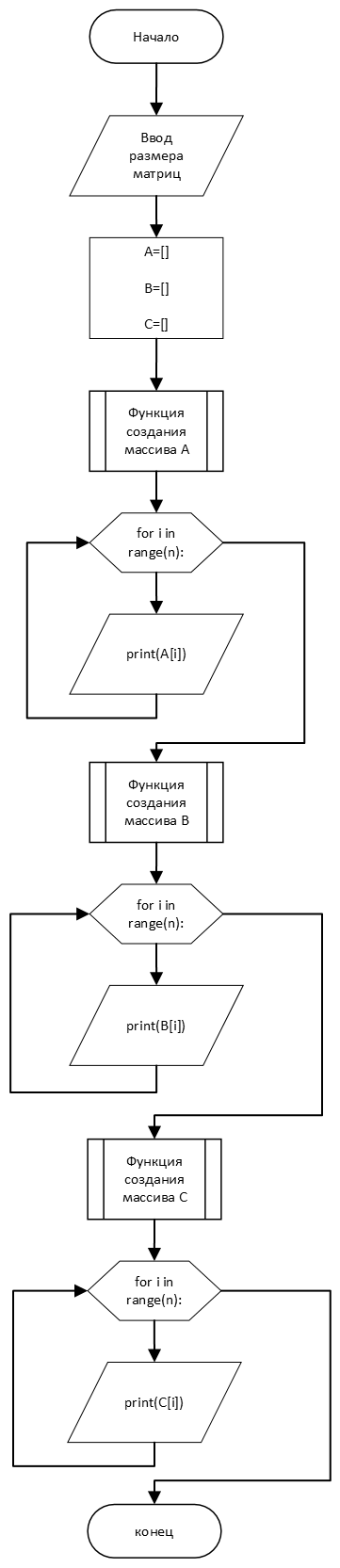


Алгоритм Штрассена

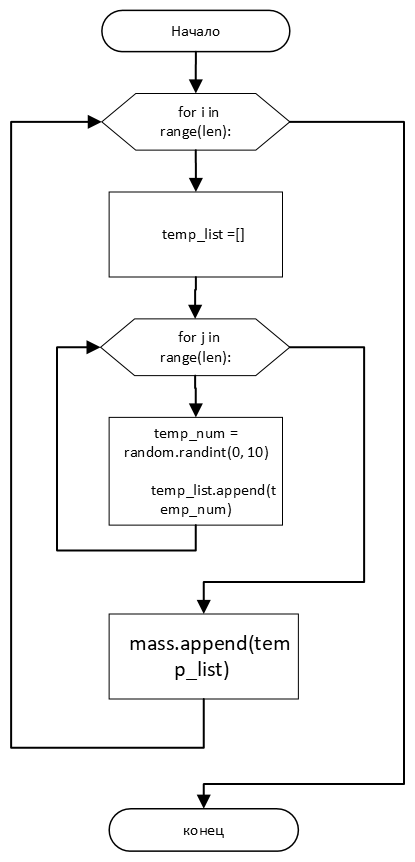


3.2. Метод «грубой силы»

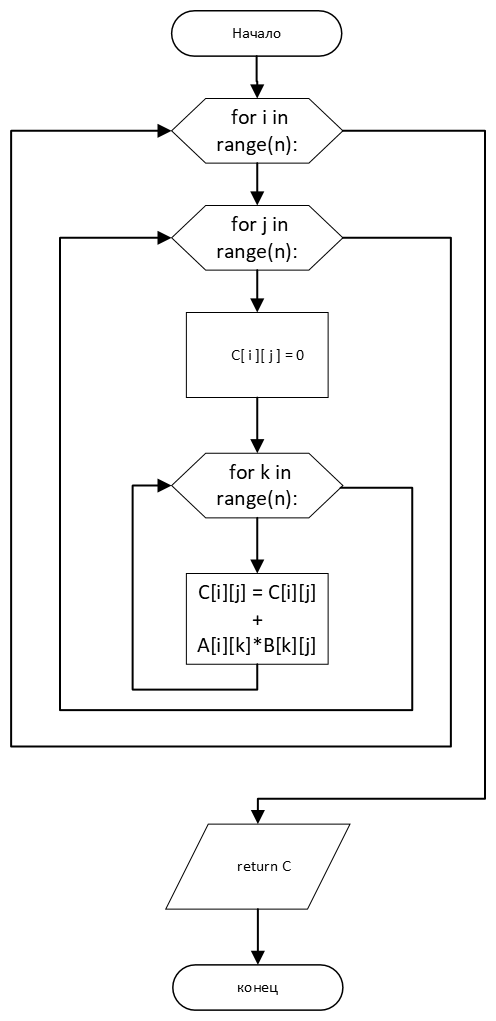
Основная программа:



Функция создания массивов:

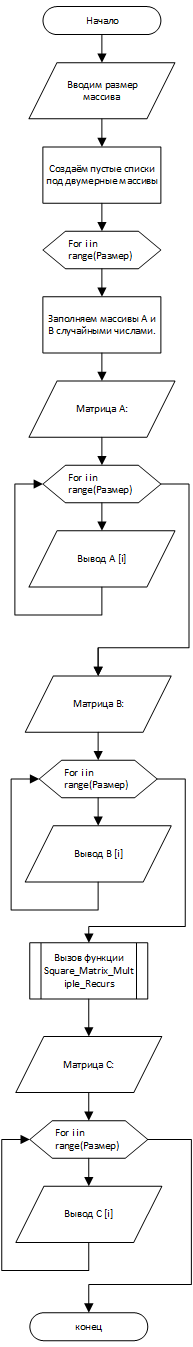


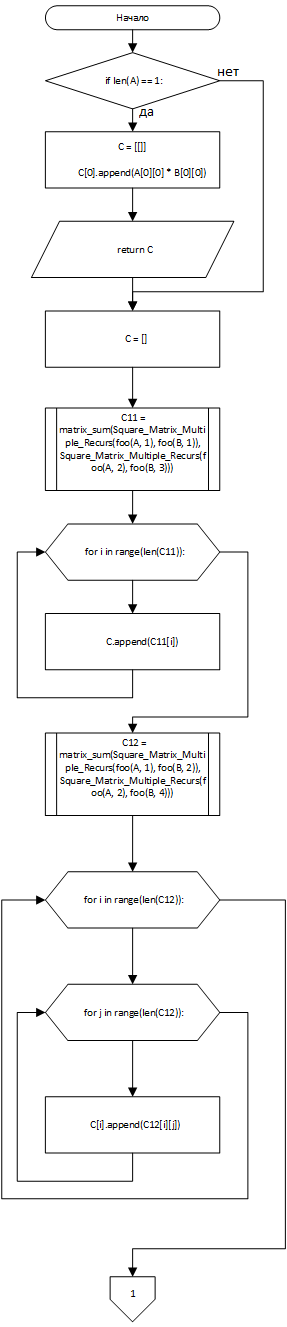
Функция перемножения массивов:

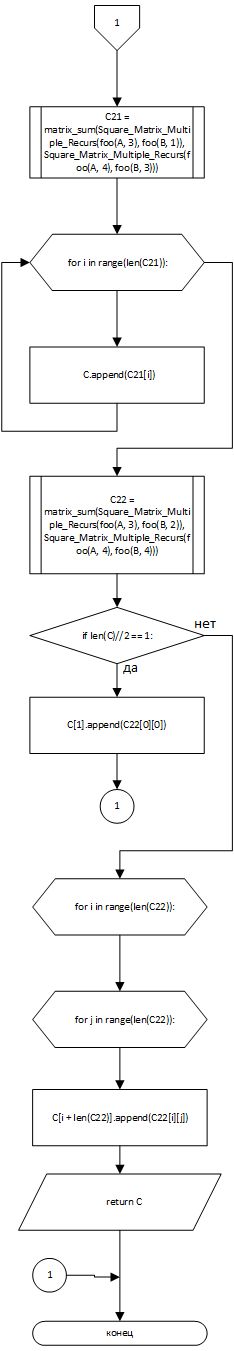


3.2. Метод «Разделяй и властвуй»

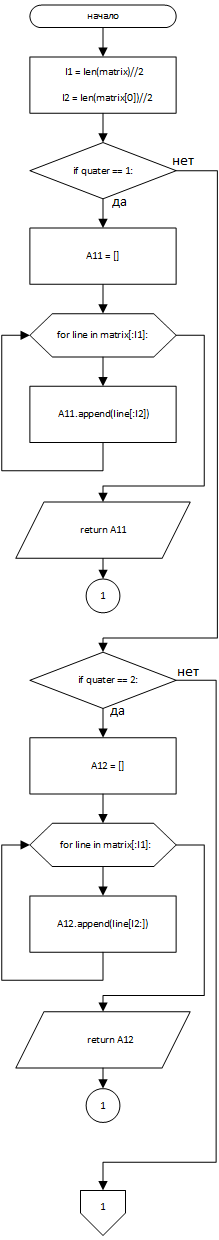
Основная программа

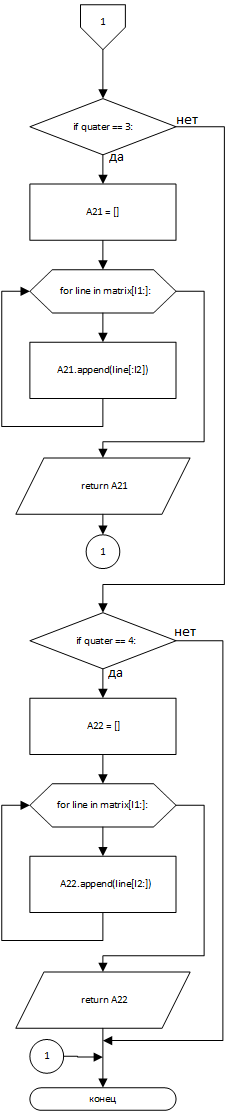


Функция Square\_Matrix\_Multiple\_Recurs  


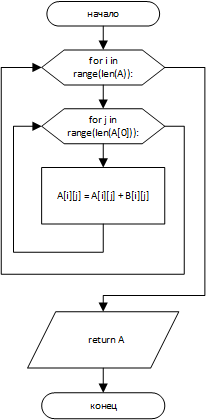


Функция деления матриц на 4 части





Функция matrix\_sum



**4. Подбор тестовых примеров**

4.1. Метод Штрассена

1) Получаем на вход 2 двумерных массива. и

3) Найдём х по следующим формулам:

|  |  |
| --- | --- |
| x1 = (G1,1 + G2,2)(H1,1 + H2,2); | x5 = (G1,1 + G2,2)H2,2; |
| x2 = (G2,1 + G2,2)H1,1; | x6 = (G2,1 - G1,1)(H1,1 + H1,2); |
| x3 = G1,1(H1,2 - H2,2); | x7 = (G2,1 - G2,2)(H2,1 + H2,2); |
| x4 = G2,2(H2,1 - H1,1); |  |

x1 = (1 + 7)\*(3 + 0) = 24 x3 = 1\*(5 - 0) = 5;

x2 = (6 + 7)\*3 = 39; x4 = 7\*(0 - 3) = -21;

x5 = (1+ 7)\*0 = 0 x6 = (6 - 1)\*(3 + 5) = 40

x7 = (6 - 7)\*(0 + 0) = 0

4) Теперь элементы матрицы R могут вычисляться по формулам:

|  |  |
| --- | --- |
| R1,1 = x1 + x4 - x5 + x7; | R1,2 = x3 + x5; |
| R2,1 = x2 + x4; | R2,2 = x1 + x3 - x2 + x6. |

R1,1 = 24 -21 - 0 + 0 = 3 R1,2 = 5 + 0 = 5

R2,1 = 39 -21 = 18 R2,2 = 24 + 5 - 39 + 40 = 30

5) Получаем умноженную матрицу С

4.2. Метод «Грубой силы»

1) Получаем на вход 2 двумерных массива.

и

2) Матрица A Вычисляем элемент новой матрицы (1,1): работаем с 1-ой строкой и с 1-м столбцом.

Получаем: 1\*9+2\*6+3\*3 = 30

3) Вычисляем элемент новой матрицы (1,2): работаем с 1-ой строкой и с 2-м столбцом.

Получаем: 1\*8+2\*5+3\*2 = 24

4) Вычисляем элемент новой матрицы (2,1): работаем с 2-ой строкой и с 1-м столбцом.

Получаем: 4\*9+5\*6+6\*3 = 84

5) Вычисляем элемент новой матрицы (2,2): работаем с 2-ой строкой и с 2-м столбцом.



Получаем: 4\*8+5\*5+6\*2 = 69



6) В итоге получаем матрицу C



4.3. Метод «Разделяй и властвуй»



Получаем на вход 2 двумерных массива.

и

С = =

**5. Код программы**

**5.1. Метод Штрассена**

import numpy as np

import random

import time

def strassen\_algorithm(x, y):

if x.size == 1 or y.size == 1:

return x \* y

n = x.shape[0]

if n % 2 == 1:

x = np.pad(x, (0, 1), mode='constant')

y = np.pad(y, (0, 1), mode='constant')

m = int(np.ceil(n / 2))

a = x[: m, : m]

b = x[: m, m:]

c = x[m:, : m]

d = x[m:, m:]

e = y[: m, : m]

f = y[: m, m:]

g = y[m:, : m]

h = y[m:, m:]

p1 = strassen\_algorithm(a, f - h)

p2 = strassen\_algorithm(a + b, h)

p3 = strassen\_algorithm(c + d, e)

p4 = strassen\_algorithm(d, g - e)

p5 = strassen\_algorithm(a + d, e + h)

p6 = strassen\_algorithm(b - d, g + h)

p7 = strassen\_algorithm(a - c, e + f)

result = np.zeros((2 \* m, 2 \* m), dtype=np.int32)

result[: m, : m] = p5 + p4 - p2 + p6

result[: m, m:] = p1 + p2

result[m:, : m] = p3 + p4

result[m:, m:] = p1 + p5 - p3 - p7

return result[: n, : n]

x = np.array([[random.randint(-10, 10) for i in range(3)] for i in range(3)])

y = np.array([[random.randint(-10, 10) for i in range(3)] for i in range(3)])

print("Первый массив:\n", x, "\nВторой массив:\n", y)

print('Произведение массивов:\n', strassen\_algorithm(x, y))

print("Время выполнения программы:: ", time.process\_time())

**5.2. Метод «Грубой силы»**

import random

import time

def cr\_arr(mass, len):

for i in range(len):

temp\_list =[]

for j in range(len):

temp\_num = random.randint(0, 10)

temp\_list.append(temp\_num)

mass.append(temp\_list)

def square\_Matrix\_Mult(A,B):

for i in range(n):

for j in range (n):

C[i][j] = 0

for k in range(n):

C[i][j] = C[i][j] + A[i][k]\*B[k][j]

return C

n = int(input("Введите размер матриц: "))

A=[]

B=[]

C=[]

cr\_arr(A, n)

print("Массив A:")

for i in range(n): print(A[i])

cr\_arr(B, n)

print("Массив B:")

for i in range(n): print(B[i])

cr\_arr(C, n)

square\_Matrix\_Mult(A,B)

print("Массив C:")

for i in range(n): print(C[i])

print("Время выполнения программы:: ", time.process\_time())

**5.3. Метод «Разделяй и властвуй»**

import random

import time

def foo(matrix, quater):

l1 = len(matrix)//2

l2 = len(matrix[0])//2

if quater == 1:

A11 = []

for line in matrix[:l1]:

A11.append(line[:l2])

return A11

if quater == 2:

A12 = []

for line in matrix[:l1]:

A12.append(line[l2:])

return A12

if quater == 3:

A21 = []

for line in matrix[l1:]:

A21.append(line[:l2])

return A21

if quater == 4:

A22 = []

for line in matrix[l1:]:

A22.append(line[l2:])

return A22

def matrix\_sum(A, B):

for i in range(len(A)):

for j in range(len(A[0])):

A[i][j] = A[i][j] + B[i][j]

return A

def Square\_Matrix\_Multiple\_Recurs(A, B):

if len(A) == 1:

C = [[]]

C[0].append(A[0][0] \* B[0][0])

return C

C = []

C11 = matrix\_sum(Square\_Matrix\_Multiple\_Recurs(foo(A, 1), foo(B, 1)), Square\_Matrix\_Multiple\_Recurs(foo(A, 2), foo(B, 3)))

for i in range(len(C11)):

C.append(C11[i])

C12 = matrix\_sum(Square\_Matrix\_Multiple\_Recurs(foo(A, 1), foo(B, 2)), Square\_Matrix\_Multiple\_Recurs(foo(A, 2), foo(B, 4)))

for i in range(len(C12)):

for j in range(len(C12)):

C[i].append(C12[i][j])

C21 = matrix\_sum(Square\_Matrix\_Multiple\_Recurs(foo(A, 3), foo(B, 1)), Square\_Matrix\_Multiple\_Recurs(foo(A, 4), foo(B, 3)))

for i in range(len(C21)):

C.append(C21[i])

C22 = matrix\_sum(Square\_Matrix\_Multiple\_Recurs(foo(A, 3), foo(B, 2)), Square\_Matrix\_Multiple\_Recurs(foo(A, 4), foo(B, 4)))

if len(C)//2 == 1:

C[1].append(C22[0][0])

else:

for i in range(len(C22)):

for j in range(len(C22)):

C[i + len(C22)].append(C22[i][j])

return C

N = int(input("Введите размер матрицы: "))

A = []

B = []

for i in range(N):

A\_row = []

B\_row = []

for j in range(N):

A\_row.append(random.randint(-10, 10))

B\_row.append(random.randint(-10, 10))

A.append(A\_row)

B.append(B\_row)

print("Матрица A:")

for i in range(N):

print(A[i])

print()

print("Матрица B:")

for i in range(N):

print(B[i])

C = Square\_Matrix\_Multiple\_Recurs(A, B)

print()

print("Matrix C: ")

for i in range(N):

print(C[i])

print("Время выполнения программы:: ", time.process\_time())

**6. Расчёт тестовых примеров на ПК**

6.1. Метод Штрассена

Изображение выглядит как текст

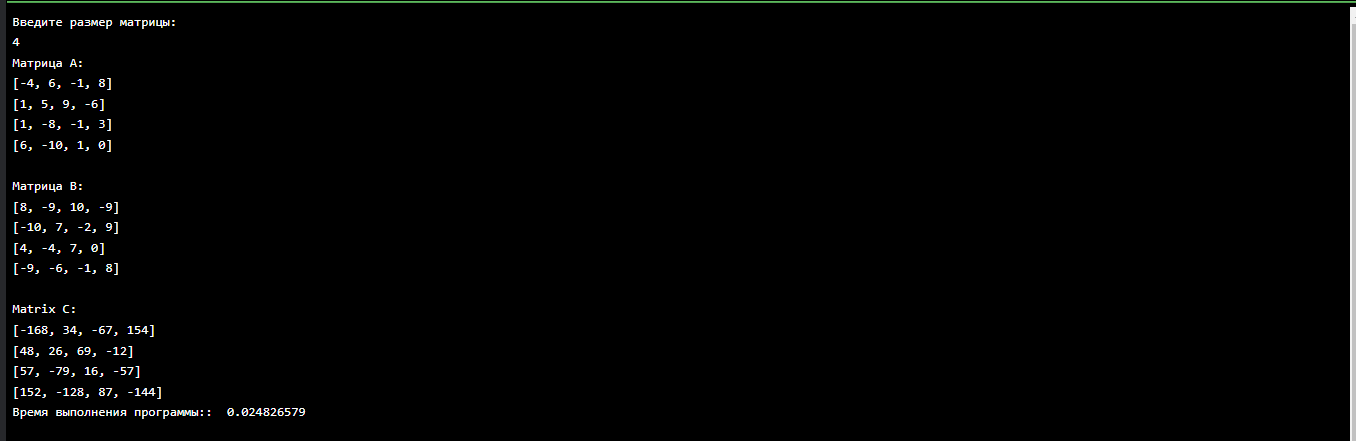
Автоматически созданное описание

6.2. Метод «Грубой силы»

Изображение выглядит как текст

Автоматически созданное описание

6.2. Метод «Разделяй и властвуй»



**7. Вывод**

В данной практической работе был реализован псевдокод алгоритма перемножения матриц. Мной был реализован данный алгоритм тремя способами: методом Штрассена, методом «грубой силы», методом «Разделяй и властвуй». При выполнении задания, самым сложно реализуемым оказался алгоритм методом «Разделяй и властвуй», поскольку было очень легко запутаться в рекурсиях. Исследовать, какой алгоритм работает быстрее я не стал, поскольку мне доподлинно известно, что время работы алгоритма «Разделяй и властвуй» - n3; Метода Штрассена - n2,81; «Грубой силы» - n3. Исходя из этих сведений, могу сделать вывод, что самый быстрый алгоритм – это алгоритм методом Штрассена.