МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

Тема: Поиск с возвратом

Вариант 2р

Студент гр. 8303 _____ Колосова М.П.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

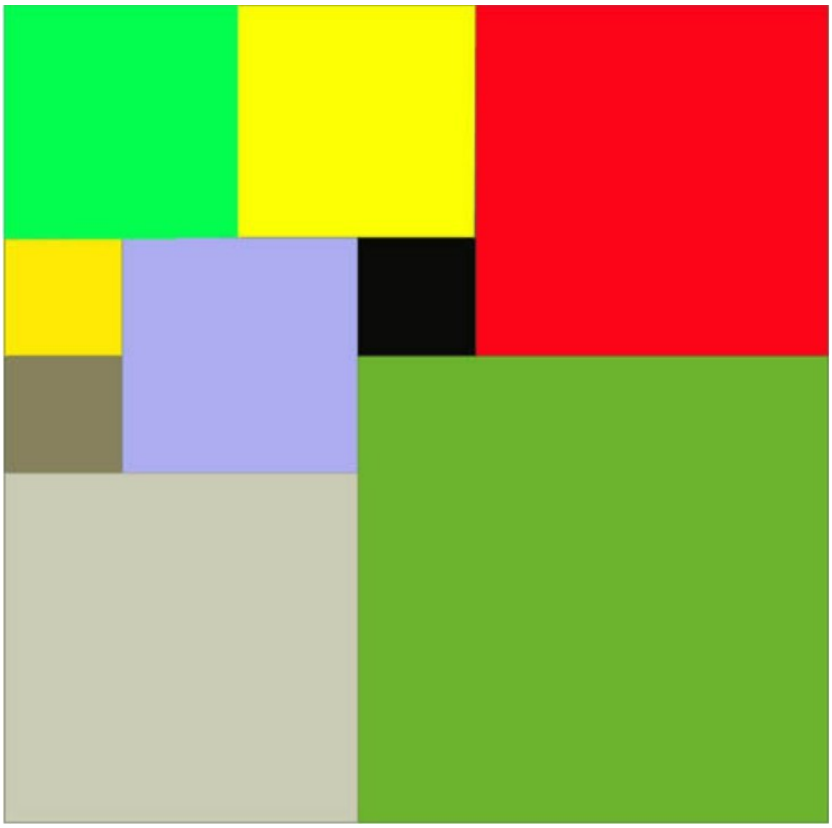
Цель работы.

Изучение алгоритма поиска с возвратом. Исследование времени выполнения программы в зависимости от входного параметра.

Задание.

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 1 до N−1, и у него есть неограниченное число обрезков любого размера. Но ему очень хочется получить большую столешницу – квадрат размера N. Он может получить ее, собрав из уже имеющихся обрезков (квадратов).

Например, столешница размера 7×7 может быть построена из 9 обрезков.



Внутри столешницы не должно быть пустот, обрезки не должны выходить за пределы столешницы и не должны перекрываться. Кроме того, Вова хочет использовать минимально возможное число обрезков.

Входные данные

Размер столешницы – одно целое число N (2≤N≤20).

Выходные данные

Одно число K, задающее минимальное количество обрезков(квадратов), из которых можно построить столешницу (квадрат) заданного размера N. Далее должны идти K строк, каждая из которых должна содержать три целых числа x,

y и w, задающие координаты левого верхнего угла (1≤x, y≤N) и длину стороны соответствующего обрезка (квадрата).

Пример входных данных

7

Соответствующие выходные данные

9
1 1 2
1 3 2
3 1 1
4 1 1
3 2 2
5 1 3
4 4 4
1 5 3
3 4 1

Индивидуализация

Рекурсивный бэктрекинг. Исследование времени выполнения от размера квадрата.

Описание алгоритма

Основная идея программы заключается в поиске с возвратом, а именно — в полном рекурсивном переборе всех заполнений квадрата.

В начале выполняются проверки для частных случаев кратности. Затем вызывается рекурсивная функция полного заполнения исходного квадрата обрезками со сторонами от максимально возможной в данной раскладке до единичной, причем максимальный размер вставляемого обрезка определяется входным параметром — размером квадрата. После каждой вставки рекурсивно вызывается функция вставки следующего максимально возможного обрезка следующего цвета в ближайшие координаты. После выхода из уровня рекурсии происходит очищение текущей вставки и выбор других координат для вставки максимально возможного обрезка. Выход из рекурсии обеспечивается проверкой передаваемых координат, в этом же месте выполняется сравнение текущего количества обрезков с предыдущим минимальным.

Координаты текущей минимальной раскладки записываются в строку и перезаписываются в случае изменения искомого значения.

Вставку квадрата мы начинаем с обрезка размера 4*(N/2 + 1)/5 и заканчиваем единичным. Случай, когда N — простое число, является для данного алгоритма худшим, потому как является минимально оптимизируемым. Для N >= 3 выполняется два поиска. Величина свободной площади в первом - (N/2 + 1)*(N/2 + 1) - 5 , во втором - (N/2 + 1)*(N/2 + 1) - 11. Тогда получаем сложность по количеству операций :

o($(s)^{(N/2+1)^2-5}$), где s — максимально возможный размер обрезка

В программе используется двумерный массив поля и строка для записи координат. Отсюда следует, что сложность по памяти составляет o(N^2).

Использованные оптимизации

Для избежания постоянного поиска подходящих координат для текущего размера обрезка в свободные координаты вставляется максимально возможный обрезок. Это минимизирует дублирования вариантов разложения.

Функция поиска прекращает работу, если размер текущего разложения начинает превышать текущий минимальный результат.

При поиске минимального разложения для непростых чисел размер поля уменьшается в масштабе минимального делителя введенного параметра.

Частный случай кратности двум не вызывает рекурсивный поиск, а использует готовое решение, реализованное в одной из функций.

Для N < 3 поиск не выполняется два раза, потому что размер не позволяет рассматривать два варианта поведения.

Перед запуском перебора вставляются очевидные квадраты, используемые в любом разложении, что уменьшает свободную для заполнения площадь.

Максимально возможный размер обрезка ограничивается теоретически вычисленным значением для избежания бессмысленных попыток вставки.

Описание структур данных

static int table[MAX_N][MAX_N] — двумерный массив для хранения состояния поля

static string coord — строка для хранения координат обрезков минимального разложения

Описание функций

void **initializeField**()
Инициализирует поле нулями

void **fill_2Square**()
Заполняет квадрат с четной стороной минимальным разложением, реализация частного случая

void **fillOddSquare**()
Заполняет квадрат с нечетной стороной четырьмя очевидными обрезками

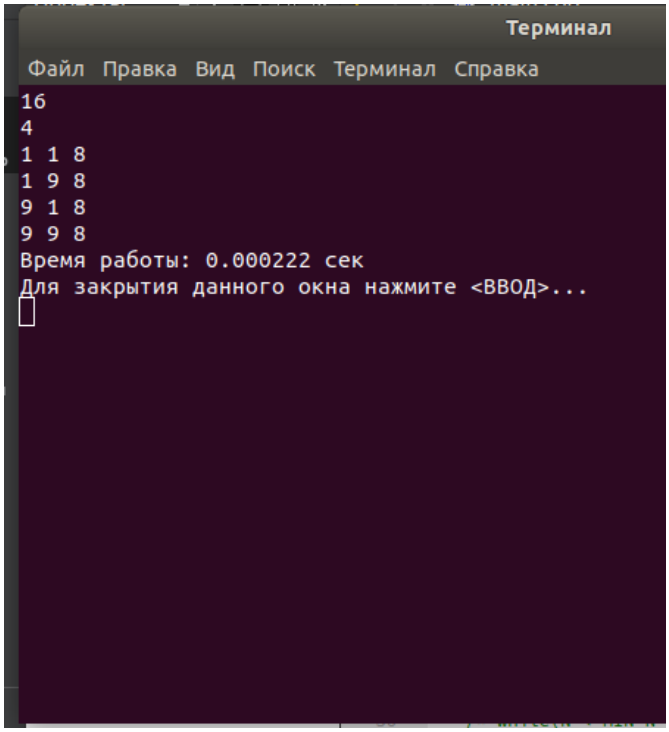
bool **paintSq**(int N, int start_x, int start_y, int color, int tableSize)
Закрашивает область определенного размера, начиная с переданных координат
N — размер обрезка, tableSize — введенный параметр, x и y — координаты начала области для вставки обрезка
Возвращает false или true, если вставка невозможна или возможна и состоялась соответственно

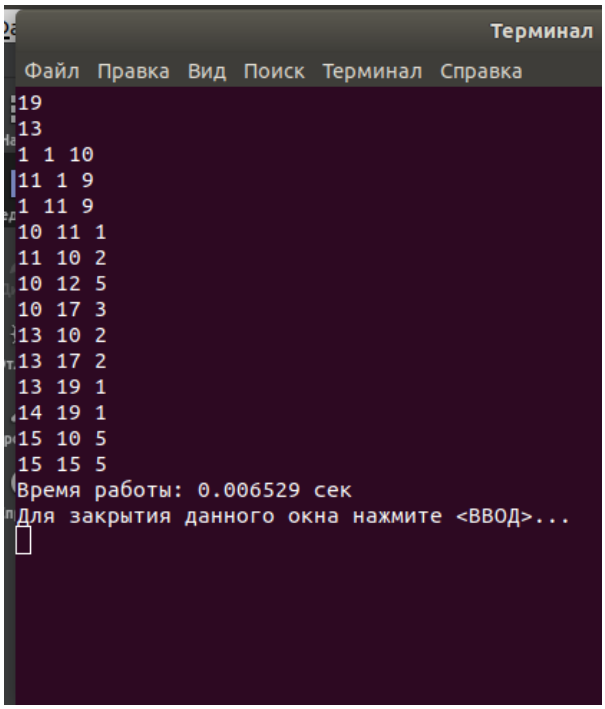
void **printTable**()
Выводит в стандартный поток вывода текущее состояние поля

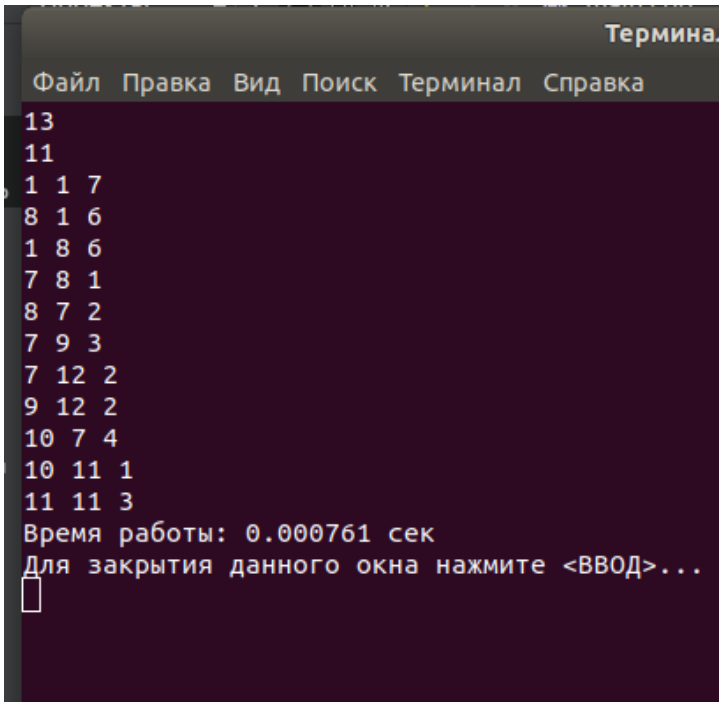
void **printCoord**(int color)
Записывает в строку координаты верхнего левого угла обрезка определенного цвета

void **backTracking**(int x, int y, int size, int color, int& K)
Выполняет рекурсивный поиск с возвратом
x, y — координаты для вставки, size — размер обрезка, color — цвет, &K — ссылка на переменную, хранящую размер текущего минимального разложения

Тестирование



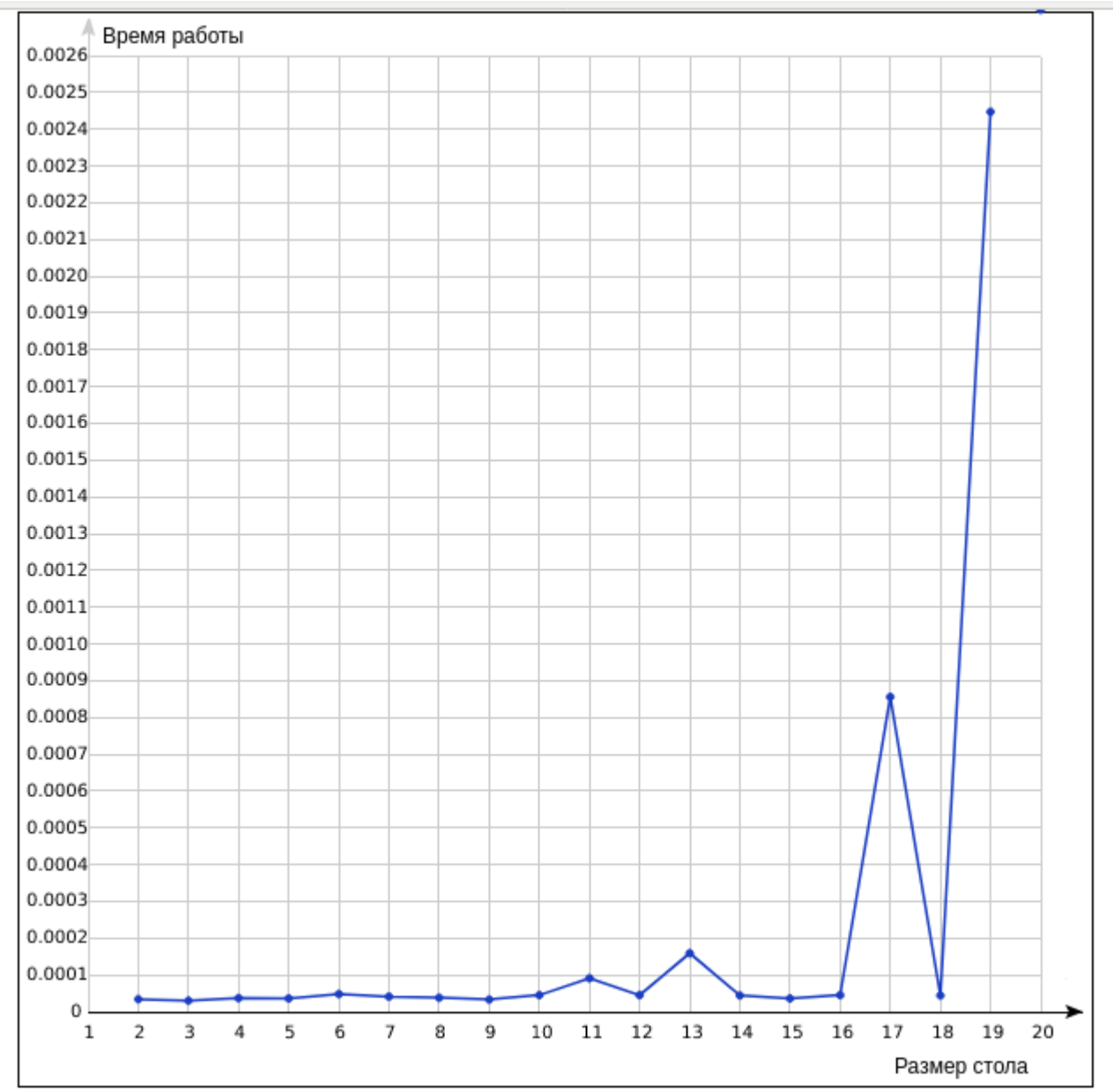




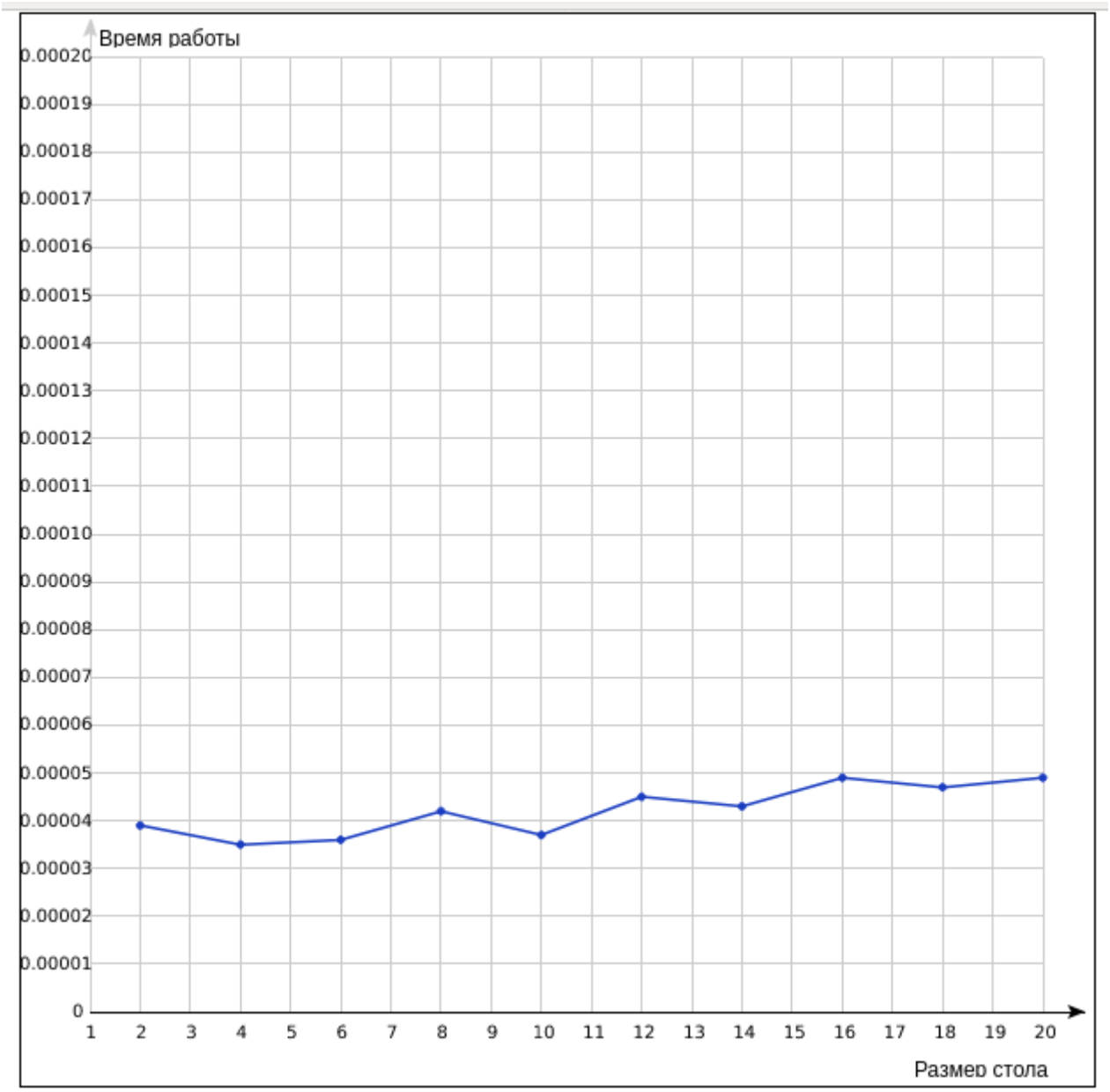
Исследование времени выполнения

Был написан скрипт командой строки, тестирующий программу на числах от 2 до 20 включительно и сохраняющий результаты работы в файл. На основе получившихся данных построены графики зависимости времени от входного параметра.

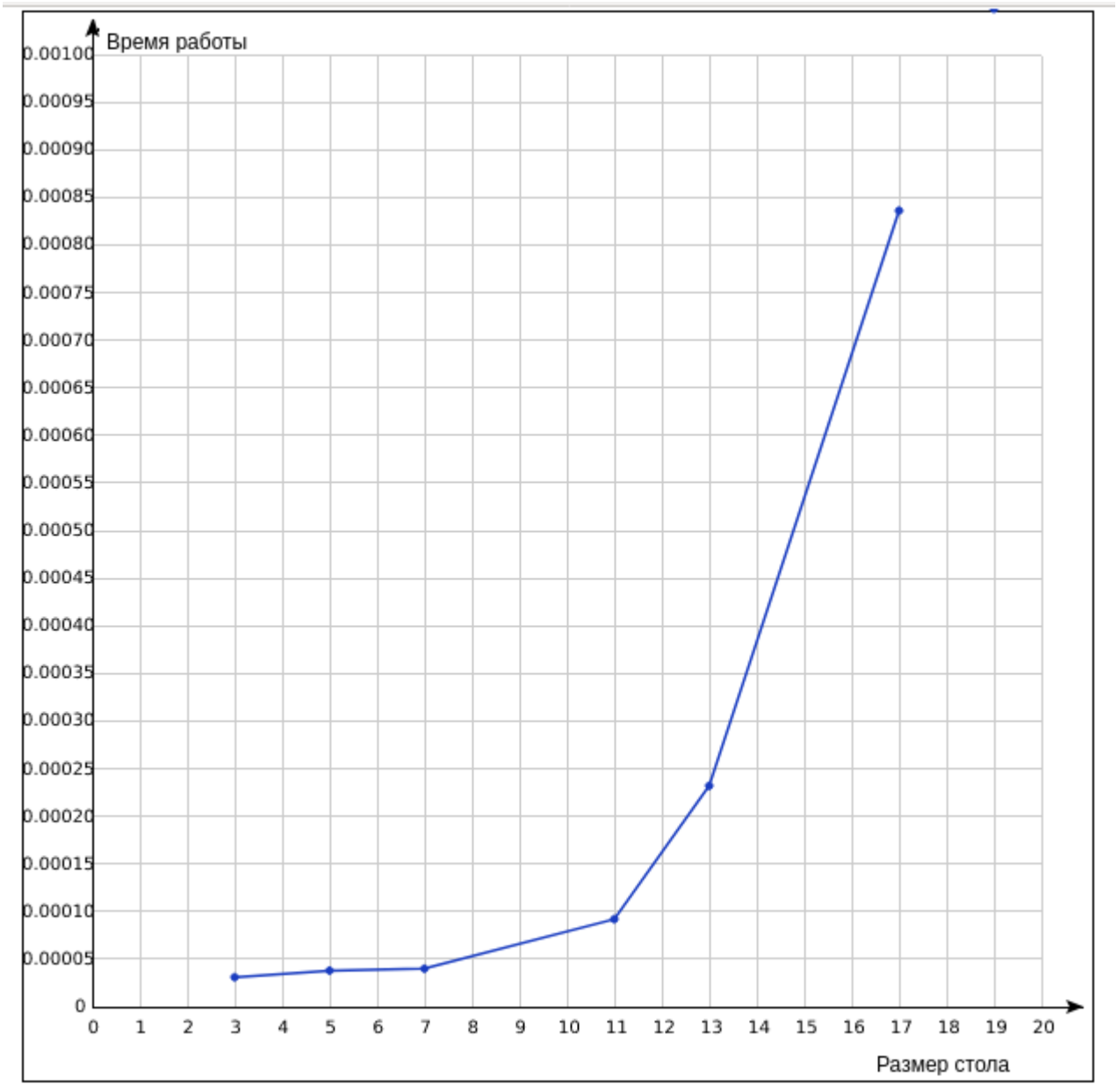
Здесь изображена зависимость времени от всех тестовых данных. Скорость возрастания функции различается на участках, потому что для четных, кратных и простых чисел программа работает неодинаково. Данный набор данных не дает достаточно сведений для анализа.



Была составлена тестовая выборка состоящая только из чисел, кратных двум. Следующий график иллюстрирует изменение времени для четных сторон квадрата. Все значения функции лежат в диапазоне от 0.000034 до 0.00005, что говорит о том, что программа была выполнена примерно за одно время. Решение для таких случаев вызывало другую функцию и не требовало вызова рекурсивного перебора. Возрастание обусловлено увеличением площади для закраски и площади поиска координат цвета.



Далее был проанализирован набор непростых чисел, минимальный делитель которых больше 2. Для заданных по условию граничных значений входного параметра — это 9 и 15. Их минимальные делители совпадают, поэтому результаты очень близки — 4.9e-05 и 4.6e-05. Для смыслового анализа введенной оптимизации данная выборка была расширена путем увеличения правой границы диапазона входного параметра. Для чисел 9, 15, 21, 25, 27 результаты такие: 3.5e-05, 3.3e-05, 3.4e-05, 5.2e-05, 3.1e-05. График для чисел с одинаковым минимальным делителем близок к прямой. Значит, масштабируемая замена стола дает существенный выигрыш во времен. Самый сложный обрабатываемый случай — простые числа. Аналогично предыдущим шагам был получен график для усеченной тестовой выборки. Используемые оптимизации не охватывают эту зону, поэтому возрастание схоже с экспоненциальным ростом.



Приложение А. Исходный код

```
#include <iostream>
#include <cmath>
#include <ctime>

#define MAX_N 40
#define MIN_N 2
using namespace std;


static int N;
static int table[MAX_N][MAX_N];
static int proportion;
static string coord;


void initializeField();//инициализируем поле
void fill_2Square();//частный случай кратности двум
void fillOddSquare();//закрасить четыре очевидных квадрата
bool paintSq(int N, int start_x, int start_y, int color, int
tableSize);//попытка вставить обрезок в опред место
void backTracking(int x, int y, int size, int color, int& K);
void printTable();
void printCoord(int color);

int main()
{
    //0 - некорректный размер квадрата (для захода в след цикл)
    N = 0;
    int K;
    proportion = 1;
   /* while(N < MIN_N || N > MAX_N){
        cout<<"Введите корректный размер столешницы."<<endl;Введите корректны
        cin>>N;
    }*/
    cin>>N;
    initializeField();//заполнили нулями
    long startTime = clock();

    if (N % 2 == 0){//в четном случае лучший вариант очевиден, координаты
зависят от размера стола
        K = 4;
        cout<<K<<endl;
        fill_2Square();
        coord.clear();
        for(int i = 1; i <= K; i++)//печать координат для каждого
цвета=квадрата
            printCoord(i);

        cout<<coord;
        cout<<"Введите корректный размер столешницы."<<endl;Время работы: "Вв
startTime)/CLOCKS_PER_SEC<<"Введите корректный размер столешницы."<<endl; сек

        return 0;

    }
    else{
```

```
        //ищем делитель
        for(int i = 2; i <= N; i++){
            if(N % i == 0){
                proportion = N/i;
                N = i;
                break;
            }
        }

        K = 1600;//потенциально недостижимое число (число свободных
клеток при максимальной величине квадрата)

        //1 var
        fillOddSquare();
        if(N > 3)
            paintSq(2, N/2 + 1,N/2, 5, N);
        else paintSq(1,N/2+1, N/2, 5, N);
        backTracking(N/2, N/2 + 2, 4*(N/2 + 1)/5, 6, K);

        //2 var
        if(N > 3){

            initializeField();
            fillOddSquare();
            paintSq(1, N/2, N/2 + 2, 5, N);
            paintSq(3, N/2 + 1, N/2, 6, N);
            backTracking(N/2, N/2 + 3, 4*(N/2 + 1)/5, 7, K);
        }


        cout<<K<<endl;
        cout<<coord;
        cout<<"Введите корректный размер столешницы."<<endl;Время работы: "Вв
startTime)/CLOCKS_PER_SEC<<"Введите корректный размер столешницы."<<endl; сек"

    }
    return 0;
}

void initializeField(){

    for (int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            table[i][j] = 0;
        }
    }
    return;

}

void fill_2Square(){

    int color = 1;
    for (int i = 0; i < N; i = i + N/2){
        for(int j = 0; j < N; j = j +N/2){
            paintSq(N/2, i, j, color, N);
            color++;
```

```
        }
    }
    return;

}

void fillOddSquare(){

    paintSq(N/2 + 1,  0, 0, 1, N);
    paintSq(N/2, N/2 + 1, 0, 2, N);
    paintSq(N/2, 0, N/2 +1, 3, N);
    paintSq(1, N/2, N/2 + 1, 4, N);

}

bool paintSq(int N, int start_x, int start_y, int color, int tableSize){

    if (start_x + N > tableSize || start_y + N > tableSize){//передача
неверных координат
        return false;
    }

    for(int i = start_x; i < start_x + N; i++){
        for(int j = start_y; j < start_y + N; j++){
            if(table[i][j] != 0){//не хватает свободной площади
                return false;
            }
        }
    }

    //функция закрашивает область только в том случае, если свободной
площади хватает
    for(int i = start_x; i < start_x + N; i++){
        for(int j = start_y; j < start_y + N; j++){
            table[i][j] = color;
        }
    }

    return true;

}

void printTable(){

    cout<<"Введите корректный размер столешницы."<<endl;Текущий вид поля:\n";
        for (int i = 0; i < N; i++){
            for(int l = 1; l <= proportion; l++){
                for (int j = 0; j < N; j++){
                    for(int l = 1; l <= proportion; l++){
                    if(table[i][j] < 10) cout<<"Введите корректный размер сто
                    cout << table[i][j]<<"Введите корректный размер столешниц
                    }
                }
                cout<<endl;
            }
        }

}
```

```
void printCoord(int clr){

    int sqSize = 0;
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            if(table[i][j] == clr){
                sqSize++;
            }
        }
    }
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            if(table[i][j] == clr){
                string s = to_string(proportion*i + 1);
                s.append("Введите корректный размер столешницы."<<endl; "Введ
                s.append(to_string(proportion*j + 1));
                s.append("Введите корректный размер столешницы."<<endl; "Введ
                s.append(to_string(int(proportion*sqrt(sqSize))));
                s.append("Введите корректный размер столешницы."<<endl;\n";n"
                coord.append(s);
                return;
            }
        }
    }

void backTracking(int x, int y, int size, int color, int& K){

    if(size == 0)
        return;
    if(color > K)
        return;

    if (y >= N){//x - номер строки, y - номер столбца
        x++;
        y = N/2;
    }

    if (x >= N){//некуда больше идти и все квадраты вставлены

        if(color - 1 < K){
            K = color - 1;
            coord.clear();

            for(int i = 1; i <= K; i++){
                printCoord(i);
            }
        }

        return;
    }

    while(size){

        if(paintSq(size, x, y, color, N)){
            backTracking(x, y + size, 4*(N/2 + 1)/5, color + 1,
K);//попытка вставить максимальный обрезок
            //удалить только что вставленный квадрат
            for(int i = x; i < x + size; i++){
```

```
                for(int j = y; j < y + size; j++){
                    table[i][j] = 0;
                }
            }
        }
        size--;//не удалось вставить, потому что слишком большой размер

    }

    if(table[x][y]){//не удалось вставить?
        backTracking(x, y+1, 4*(N/2 + 1)/5, color, K);
    }

}
```

Вывод

В ходе лабораторной работы была решена задача заполнения квадратного поля минимальным количеством квадратов меньшего размера. Таким образом, был изучен и реализован алгоритм поиска с возвратом и была исследована зависимость времени выполнения программы от входного параметра. На анализ результатов влияли введенные оптимизации.