Министерство науки и высшего образования РФ
Пензенский государственный университет
Кафедра «Вычислительная техника»

Отчет

по лабораторной работе № 2

по курсу «Арифметические и логические основы вычислительной техники»

на тему «Оценка времени выполнения программ»

Выполнили
студенты группы 24ВВВ2:
Макаров Я.С.
Бегичев Д.Д.
Родионов К.Е.
Приняли
Юрова О.В.
Деев М.В.

Пенза, 2025

Цель работы

Провести исследование эффективности различных алгоритмов и операций (перемножения матриц и сортировки массивов) путем оценки их временной сложности с использованием теоретических моделей (О-символика), измерения реального времени выполнения на различных входных данных (матрицы и массивы разного размера и структуры), а также визуализации результатов для сопоставления теории с практикой

Лабораторное задание

1. Проанализировать алгоритм перемножения матриц и определить его временную сложность в терминах О-символики.
2. Реализовать программу на языке С, которая:
    - Выделяет динамическую память под две входные матрицы и одну матрицу результата.
    - Инициализирует входные матрицы случайными числами.
    - Выполняет умножение матриц классическим алгоритмом.
    - Измеряет время выполнения умножения с помощью функций из библиотеки time.h.
3. Выполнить замеры времени работы программы для матриц размером 100, 200, 400, 1000, 2000, 4000, 10000.
4. Построить график зависимости времени выполнения умножения матриц от их размера.
5. Сравнить экспериментальные результаты с теоретической оценкой сложности $O(N^3)$ и сделать выводы о соответствии времени выполнения теоретической модели.

Описание метода решения

Для оценки временной сложности алгоритма анализируют количество операций в зависимости от размера входных данных N. Подсчитывают шаги во вложенных циклах, выделяют главный член с наибольшим ростом (например, N^3 для тройного цикла) и обозначают сложность как $O(N^3)$, игнорируя константы и менее значимые слагаемые.

Потом измеряют реальное время выполнения программы с помощью функций из time.h для разных размеров данных и сравнивают экспериментальные результаты с теоретической оценкой.

Выделение памяти - операции malloc выполняются константное количество раз (3 раза), временная сложность — O(1).

Первый двойной цикл:

```
for (i = 0; i < N; i++) {
   for (j = 0; j < N; j++) {
      a[i * N + j] = rand() % 10;
      b[i * N + j] = rand() % 10;
      f[i * N + j] = 0;
   }
}
```

Здесь выполняется $N \times N = N^2$ операций - инициализация трех массивов за каждую итерацию. Сложность — $O(N^2)$.

Тройной вложенный цикл перемножения матриц:

```
for (i = 0; i < N; i++) {
   for (j = 0; j < N; j++) {
      c = 0;
```

```
        for (r = 0; r < N; r++) {
            c += a[i * N + r] * b[r * N + j];
        }
        f[i * N + j] = c;
    }
}
```

Этот участок кода - классический алгоритм перемножения матриц. Вложенные циклы i, j, r все идут от 0 до N, итого операций порядка $N \times N \times N = N^3$. Сложность — $O(N^3)$.

Псевдокод

1-ое задание

дано

две матрицы a и b размера N × N

результат умножения в матрицу f того же размера

надо

- вычислить произведение матриц f = a × b

```
нач цел
матрицы a, b, f размером N × N
переменные i, j, r, c, время

инициализировать генератор случайных чисел
нц для i от 0 до N-1
   нц для j от 0 до N-1
      a[i, j] := случайное число от 0 до 9
      b[i, j] := случайное число от 0 до 9
      f[i, j] := 0
   кц
кц

записать время начала вычислений

нц для i от 0 до N-1
   нц для j от 0 до N-1
      c := 0
```

```
        нц для r от 0 до N-1
            c := c + a[i, r] * b[r, j]
        кц
        f[i, j] := c
    кц
кц
```

записать время окончания вычислений
вычислить время выполнения как разницу времени конца и начала

вывести первые 10 строк и 10 столбцов матрицы f
вывести время выполнения

кон

2-ое задание

алг сравнение_сортировок

дано

массивы inc (возрастающий), dec (убывающий), random (случайный), korn (угловатый) размером 100

процедуры сортировок: Shell и Быстрая

надо измерить и вывести время работы каждой сортировки на каждом массиве

```
нач цел
массивы inc, dec, random, korn, temp
время timed
переменные i, size

// Копирование массива
процедура копировать(откуда, куда, размер)
нц для i от 0 до размер-1
    куда[i] := откуда[i]
кц
конец процедуры

// Сортировка Шелла
функция Shell(массива arr, размер n)
измерить старт времени
нц для gap от n/2 до 1 с шагом деления gap на 2
    нц для i от gap до n-1
        temp := arr[i]
        j := i
        пока j ≥ gap и arr[j - gap] > temp
            arr[j] := arr[j - gap]
```

```
            j := j - gap
        конец пока
        arr[j] := temp
    кц
кц
измерить конец времени
вернуть время работы
конец функции

// Быстрая сортировка (QuickSort)
процедура quick_sort(массива arr, low, high)
если low ≥ high тогда вернуть
pivot := arr[high]
i := low - 1
нц для j от low до high-1
    если arr[j] ≤ pivot тогда
        i := i + 1
        поменять arr[i] и arr[j]
    конец если
кц
i := i + 1
поменять arr[i] и arr[high]
вызвать quick_sort(arr, low, i-1)
вызвать quick_sort(arr, i+1, high)
конец процедуры

функция quick(массива arr, low, high)
измерить старт времени
вызвать quick_sort(arr, low, high)
измерить конец времени
вернуть время работы
конец функции

// Основной блок
инициализировать генератор случайных чисел
заполнить массивы inc, dec, random, korn согласно условиям

для каждого массива из {random, inc, dec, korn} делать
    размер := длина массива
    копировать(исходный, temp, размер)
    timed := Shell(массив, размер)
    вывести "Время работы сортировки Шелла на массиве:", timed
    копировать(temp, массив, размер)
    timed := quick(массив, 0, размер-1)
    вывести "Время работы быстрой сортировки на массиве:", timed
конец для
```

кон

Листинг

1-ое задание

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N 100 // Спросите почему одномерный? Потому что malloc читает массив как одномерный двух мерного нет
int main() {
    int i, j, r, c;
    int *a, *b, *f;

   // выделение памяти под массив
    a = (int*)malloc(N * N * sizeof(int));
    b = (int*)malloc(N * N * sizeof(int));
    f = (int*)malloc(N * N * sizeof(int));

    srand(time(NULL));
    clock_t start = clock();

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            a[i * N + j] = rand() % 10; // [i * N + J] - по формеле как константа
            b[i * N + j] = rand() % 10;
            f[i * N + j] = 0;
        }
    }

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
```

```
            c = 0;
            for (r = 0; r < N; r++) {
                c += a[i * N + r] * b[r * N + j];
            }
            f[i * N + j] = c;
        }
    }
    clock_t end = clock();
    double time = (double)(end - start) / CLOCKS_PER_SEC;
    printf("10 el:\n"); // мматрица 10 на 10, в задание есть
10000 на 10000, считать будешь вечность
    for (i = 0; i < 10; i++) {
        for (j = 0; j < 10; j++) {
            printf("%d ", f[i * N + j]);
        }
        printf("\n");
    }
    printf("time: %.10f", time);

    free(a);
    free(b);
    free(f);

    return 0;
}
```

2-ое задание

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>

double Shell(int arr[], int n) {
    LARGE_INTEGER frequency, start, end;
    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&start);
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -=
gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
    QueryPerformanceCounter(&end);
    double time_spent = (double)(end.QuadPart - start.QuadPart)
/ frequency.QuadPart;
    return time_spent;
}

void quick_sort(int arr[], int low, int high) {
    if (low >= high) return;

    int pivot = arr[high];
    int i = low - 1;
```

```
    for (int j = low; j < high; j++) {
        if (arr[j] <= pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    i++;
    int temp = arr[i];
    arr[i] = arr[high];
    arr[high] = temp;

    quick_sort(arr, low, i - 1);
    quick_sort(arr, i + 1, high);
}

double quick(int arr[], int low, int high) {
    LARGE_INTEGER frequency, start, end;
    QueryPerformanceFrequency(&frequency);
    QueryPerformanceCounter(&start);

    quick_sort(arr, low, high);

    QueryPerformanceCounter(&end);
    return (double)(end.QuadPart - start.QuadPart) /
frequency.QuadPart;
}

int copy(int arr[], int temp[]) {
    int n = sizeof(arr) / sizeof(arr[0]);
```

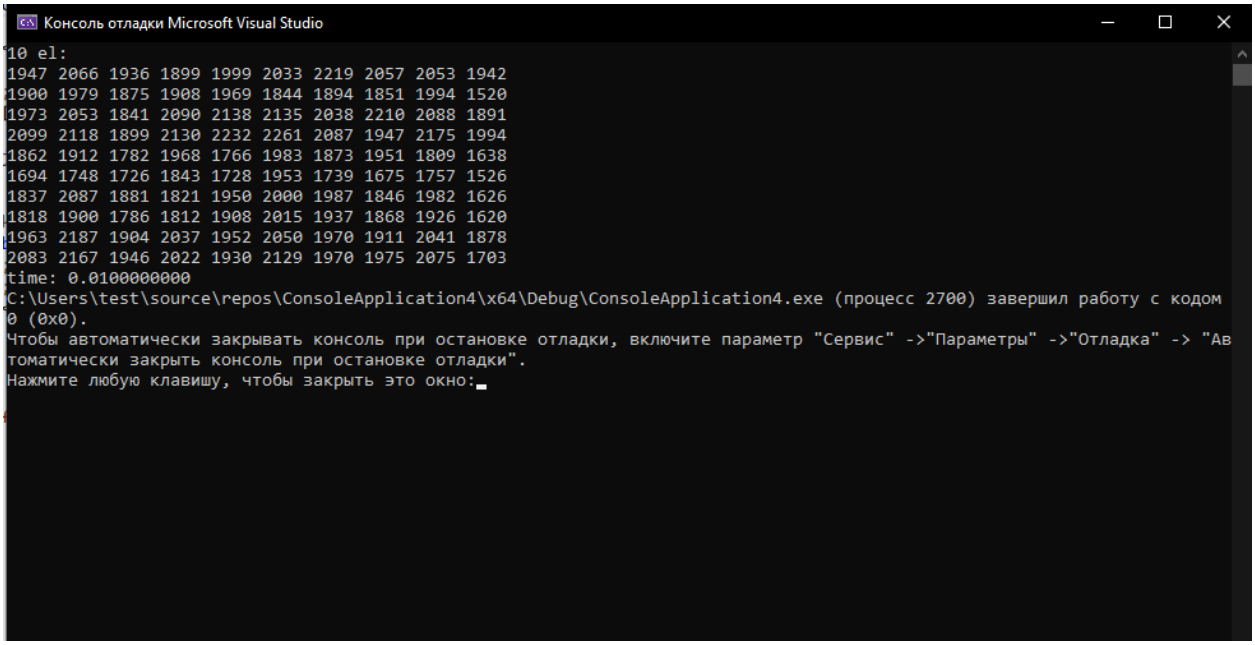
```
    for (int i = 0; i < n; i++) {
        temp[i] = arr[i];
    }
}
int main() {
    system("chcp 1251");
    system("cls");
    srand(time(NULL));
    int inc[100], dec[100], random[100], korn[100], temp[100];
    double timed;
    for (int i = 0; i < 100; i++) {
        inc[i] = i + 1;
        dec[i] = 100 - i;
        random[i] = rand() % 100;
        if (i < 48) {
            korn[i] = i + 1;
        }
        else {
            korn[i] = 100 - i;
        }
    }
    int size1 = sizeof(inc) / sizeof(inc[0]);
    int size2 = sizeof(dec) / sizeof(dec[0]);
    int size3 = sizeof(random) / sizeof(random[0]);
    int size4 = sizeof(korn) / sizeof(korn[0]);
    int size5 = sizeof(temp) / sizeof(temp[0]);
    copy(random, temp);
    timed = Shell(random, size3);
    printf("Время работы сортировки Шелла на случайном массиве:
%lf\n", timed);
    copy(temp, random);
    timed = quick(random, 0, size3 - 1);
```

```
    printf("Время работы быстрой сортировки на случайном
массиве: %lf\n", timed);

    copy(inc, random);

    timed = Shell(inc, size1);

    printf("Время работы сортировки Шелла на возрастающем
массиве: %lf\n", timed);

    copy(temp, inc);

    timed = quick(inc, 0, size1 - 1);

    printf("Время работы быстрой сортировки на возрастающем
массиве: %lf\n", timed);

    copy(dec, temp);

    timed = Shell(dec, size2);

    printf("Время работы сортировки Шелла на убывающем массиве:
%lf\n", timed);

    copy(temp, dec);

    timed = quick(dec, 0, size2 - 1);

    printf("Время работы быстрой сортировки на убывающем
массиве: %lf\n", timed);

    copy(korn, temp);

    timed = Shell(korn, size4);

    printf("Время работы сортировки Шелла на угловатом массиве:
%lf\n", timed);

    copy(temp, korn);

    timed = quick(korn, 0, size4 - 1);

    printf("Время работы быстрой сортировки на угловатом
массиве: %lf\n", timed);

}
```

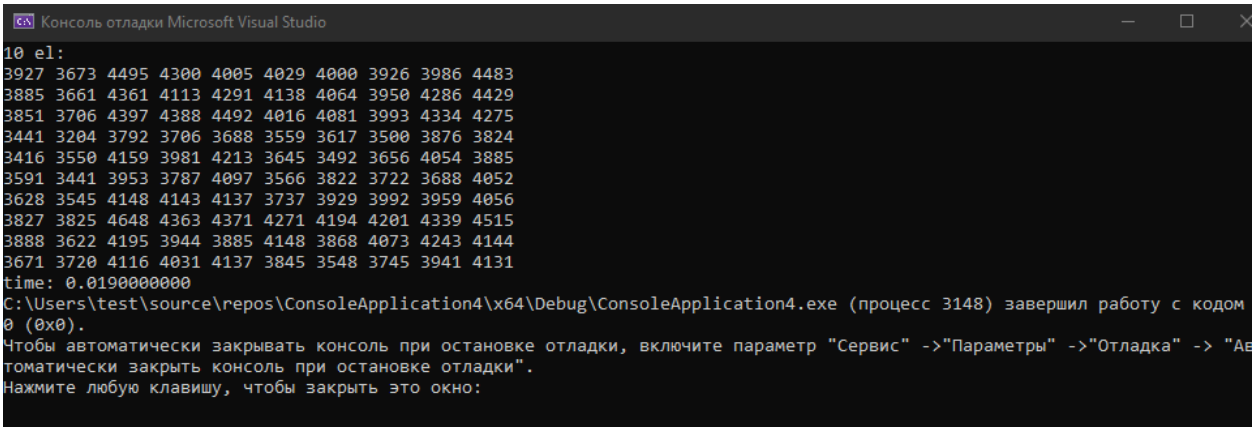
Результат работы программы

1-ое задание

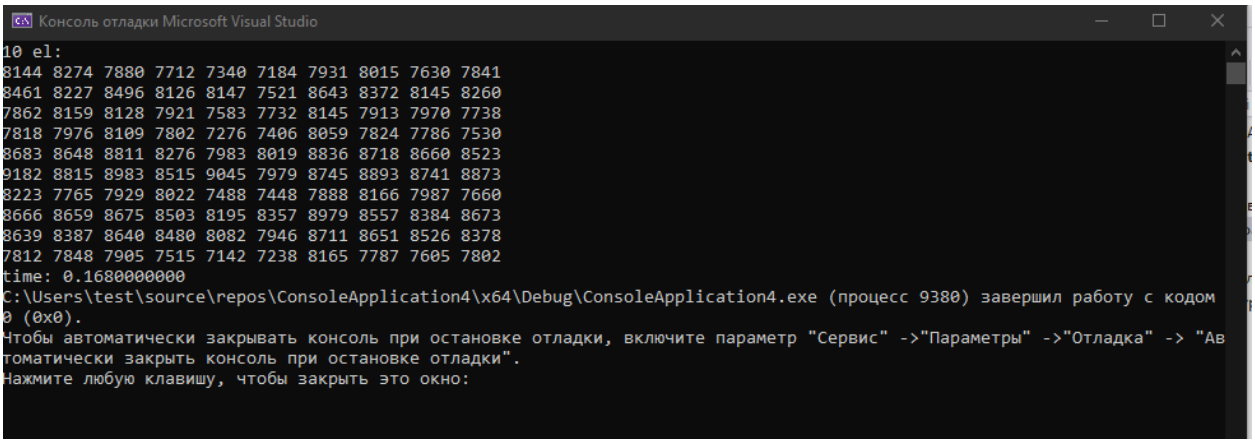
100



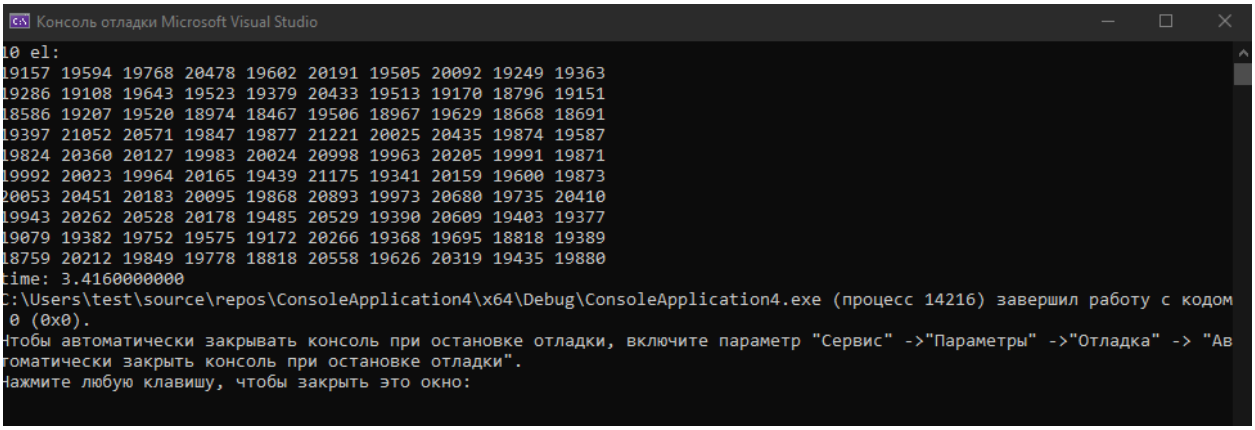
200



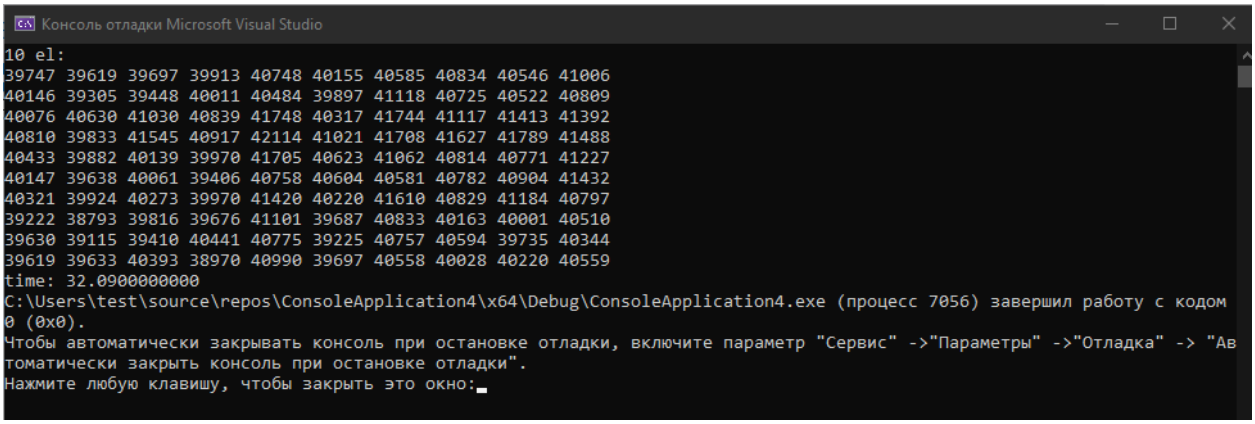
400



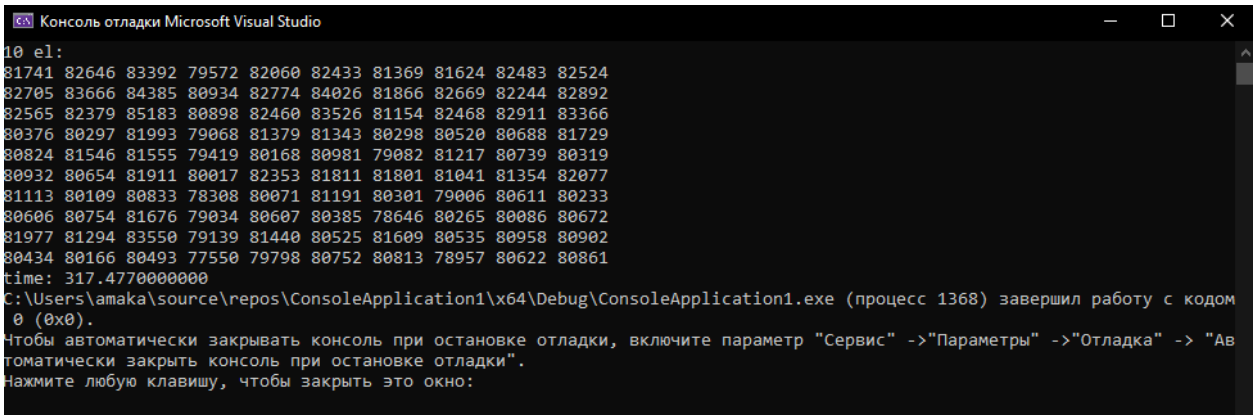
1000



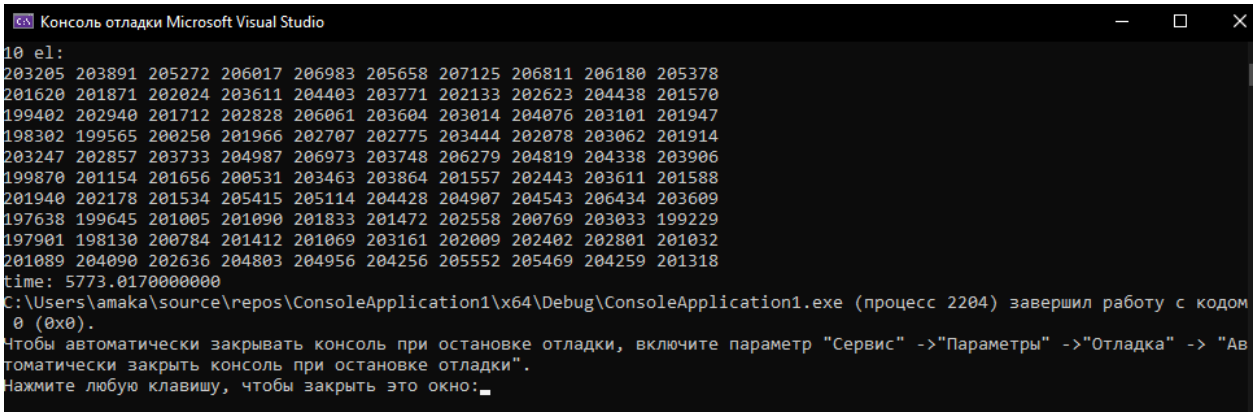
2000



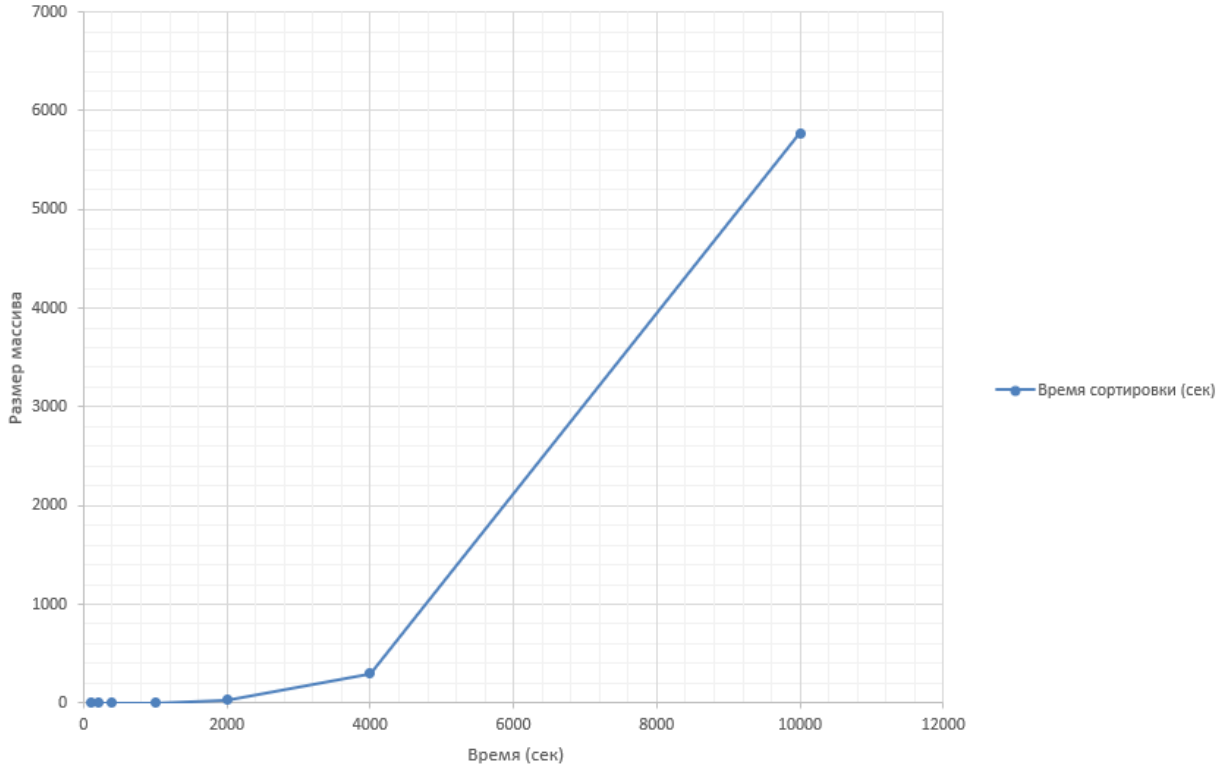
4000

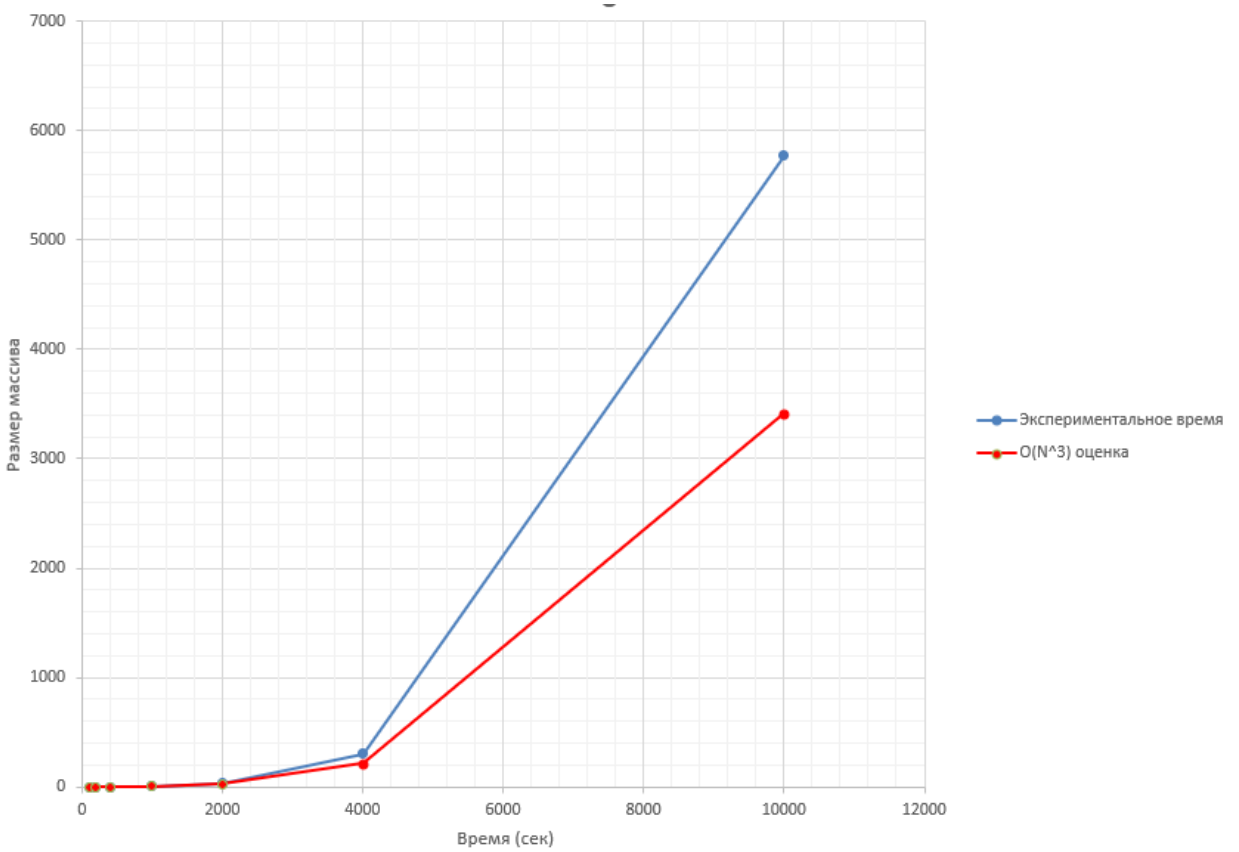


10000

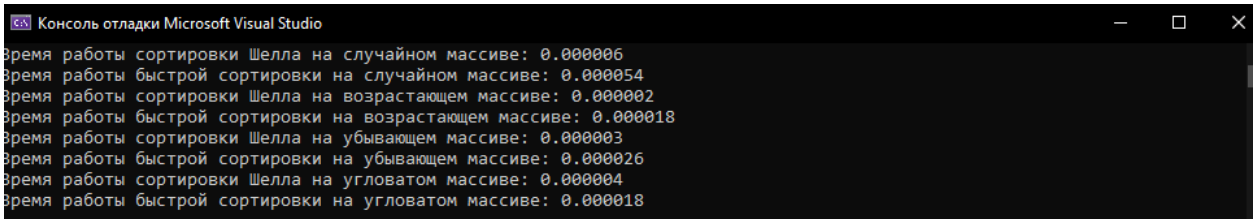


Размер массива (N)	Время сортировки (сек)
100	0.01
200	0.019
400	0.168
1000	3.416
2000	32.09
4000	300.672
10000	5773.017




7000
6000
5000
4000
3000
2000
1000
0
Размер массива
0
2000
4000
6000
8000
10000
12000
Время (сек)
Экспериментальное время
O(N^3) оценка

2-ое задание


Консоль отладки Microsoft Visual Studio
Время работы сортировки Шелла на случайном массиве: 0.000006
Время работы быстрой сортировки на случайном массиве: 0.000054
Время работы сортировки Шелла на возрастающем массиве: 0.000002
Время работы быстрой сортировки на возрастающем массиве: 0.000018
Время работы сортировки Шелла на убывающем массиве: 0.000003
Время работы быстрой сортировки на убывающем массиве: 0.000026
Время работы сортировки Шелла на угловатом массиве: 0.000004
Время работы быстрой сортировки на угловатом массиве: 0.000018

Вывод

1-ое задание

График показывает, что время сортировки растёт очень быстро с увеличением размера массива, примерно, как кубический рост. Это значит, что алгоритм работает медленно на больших объёмах данных, и время его работы растёт очень сильно. Для маленьких массивов разница между теорией и практикой может быть заметна из-за разных технических деталей, но в целом поведение совпадает с ожиданиями. Такой алгоритм не подходит для работы с очень большими массивами, лучше использовать более быстрые методы.

В сравнении с теоретической оценкой времени по сложности $O(N^3)$ экспериментальные данные показывают примерно такой же быстрый рост времени с увеличением размера массива, что подтверждает высокую вычислительную сложность алгоритма.

2-ое задание

Разница времени работы сортировки Шелла и быстроты сортировки (quicksort) на разных типах массивов (случайный, возрастающий, убывающий, угловой) очень мала. Это говорит о высокой эффективности обеих сортировок на небольших тестовых массивах: быстрая сортировка и сортировка Шелла выполняются фактически за доли миллисекунды независимо от начального состояния данных.