

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе №2

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Оценка времени выполнения программ"

Выполнили:

Студенты группы 24ВВВЗ

Плотников И.А.

Виноградов Б.С.

Приняли:

Деев М.В.

Юрова О. В.

Пенза 2025

Цель

Оценка времени выполнения и сложности программ.

Лабораторное задание

Задание 1:

1. Вычислить порядок сложности программы (O-символику).
2. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки `time.h` для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000
3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

Задание 2:

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.
2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.
3. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.
4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.
5. Оценить время работы стандартной функции `qsort`, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

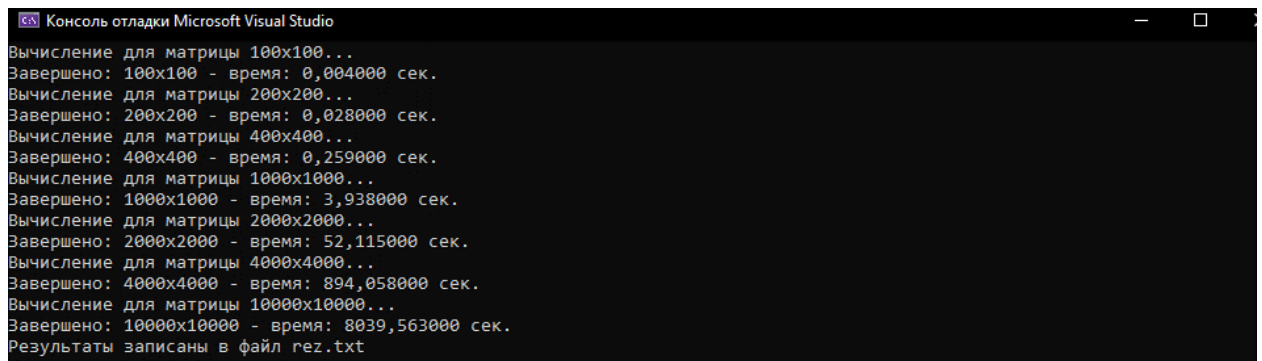
Пояснительный текст к программам

Первая программа предназначена для измерения времени выполнения умножения двух квадратных матриц различных размеров. Результаты замеров сохраняются в текстовый файл `rez.txt` и выводятся на экран.

Вторая программа предназначена для измерения времени работы алгоритмов сортировки данных на разных наборах входных данных. Результаты замеров сохраняются в текстовый файл `rez2.txt` и выводятся на экран.

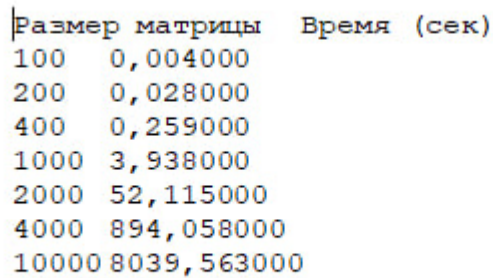
Результаты программ

1 Рис. - Результат работы ex1_lab2.c



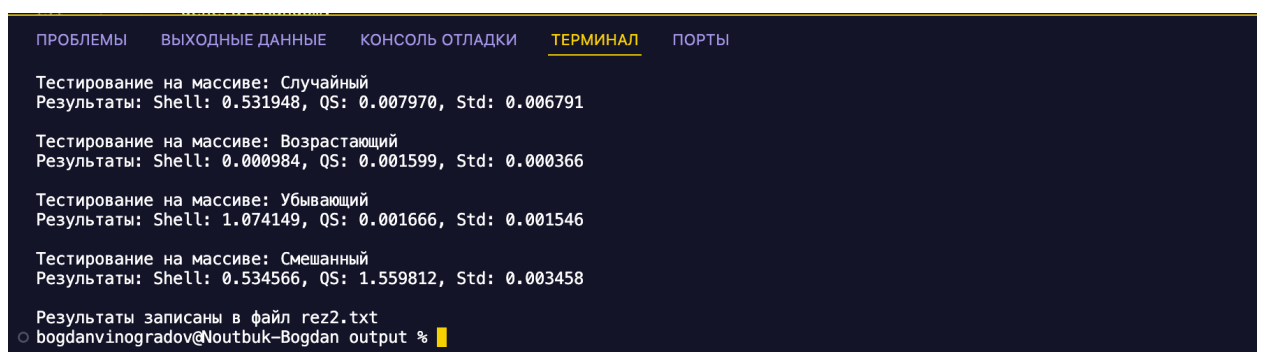
```
Консоль отладки Microsoft Visual Studio
Вычисление для матрицы 100x100...
Завершено: 100x100 - время: 0,004000 сек.
Вычисление для матрицы 200x200...
Завершено: 200x200 - время: 0,028000 сек.
Вычисление для матрицы 400x400...
Завершено: 400x400 - время: 0,259000 сек.
Вычисление для матрицы 1000x1000...
Завершено: 1000x1000 - время: 3,938000 сек.
Вычисление для матрицы 2000x2000...
Завершено: 2000x2000 - время: 52,115000 сек.
Вычисление для матрицы 4000x4000...
Завершено: 4000x4000 - время: 894,058000 сек.
Вычисление для матрицы 10000x10000...
Завершено: 10000x10000 - время: 8039,563000 сек.
Результаты записаны в файл rez.txt
```

2 Рис. – Файл rez.txt



Размер матрицы	Время (сек)
100	0,004000
200	0,028000
400	0,259000
1000	3,938000
2000	52,115000
4000	894,058000
10000	8039,563000

3 Рис. - Результат работы ex2_lab2.c



```
ПРОБЛЕМЫ  ВЫХОДНЫЕ ДАННЫЕ  КОНСОЛЬ ОТЛАДКИ  ТЕРМИНАЛ  ПОРТЫ
Тестирование на массиве: Случайный
Результаты: Shell: 0.531948, QS: 0.007970, Std: 0.006791

Тестирование на массиве: Возрастающий
Результаты: Shell: 0.000984, QS: 0.001599, Std: 0.000366

Тестирование на массиве: Убывающий
Результаты: Shell: 1.074149, QS: 0.001666, Std: 0.001546

Тестирование на массиве: Смешанный
Результаты: Shell: 0.534566, QS: 1.559812, Std: 0.003458

Результаты записаны в файл rez2.txt
bogdanvinogradov@Noutbuk-Bogdan output %
```

4 Рис. – Файл rez2.txt



Сортировка Шелла	Быстрая сортировка	Стандартная qsort
0.545326	0.008142	0.006744
0.000999	0.001595	0.000349
1.074463	0.001659	0.001565
0.534205	1.561755	0.003466

1. Вычисление порядка сложности программы (О-символика)

Главный "тяжёлый" участок программы — это умножение двух матриц размера $n \times n$ в функции multiplyArr.

Умножение матриц в коде реализовано тройным вложенным циклом:

```
for (int i = 0; i < n; i++) {  
    for (int j = 0; j < n; j++) {  
        c[i][j] = 0;  
        for (int k = 0; k < n; k++) {  
            c[i][j] += a[i][k] * b[k][j];  
        }  
    }  
}
```

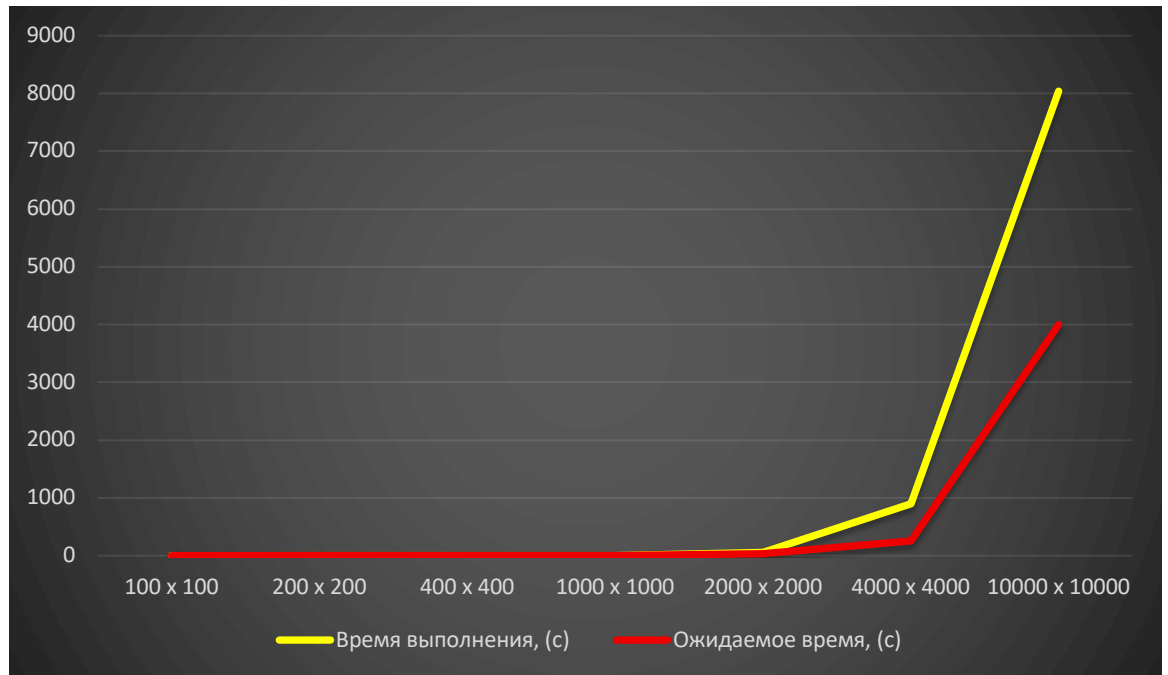
- Внешний цикл i выполняется n раз.
- Второй цикл j выполняется n раз.
- Внутренний цикл k выполняется n раз.

Это значит, что временная сложность алгоритма — $O(n^3)$.

2. Оценка времени выполнения программы и кода, выполняющего перемножение

Размер матрицы	Время выполнения, (с)	Ожидаемое время, (с)
100 x 100	0,004	0,004
200 x 200	0,028	0,032
400 x 400	0,259	0,256
1000 x 1000	3,938	4
2000 x 2000	52,115	32
4000 x 4000	894,058	256
10000 x 10000	8039,563	4000

3. График зависимости времени выполнения операции



4. Оценка времени выполнения алгоритмов сортировки

Алгоритм \ Набор данных	shell	qs	std
Случайный	0.531948	0.007970	0.006791
Возрастающий	0.000984	0.001599	0.000366
Убывающий	1.074149	0.001666	0.001546
Смешанный	0.534566	1.559812	0.003458

Листинг

Файл ex1_lab2.c

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

void clearScreen();
int** createArr(int n);
void freeArr(int** arr, int n);
void fillArr(int** arr, int n);
```

```

double multiplyArr(int n);
void runCalculations();

int main() {
    clearScreen();
    runCalculations();
    return 0;
}

void clearScreen() {
    #ifdef _WIN32
        system("cls");
    #else
        system("clear");
    #endif
}

int** createArr(int n) {
    int** arr = (int**)malloc(n * sizeof(int*));
    for (int i = 0; i < n; i++) {
        arr[i] = (int*)malloc(n * sizeof(int));
    }
    return arr;
}

void freeArr(int** arr, int n) {
    for (int i = 0; i < n; i++) {
        free(arr[i]);
    }
    free(arr);
}

void fillArr(int** arr, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            arr[i][j] = rand() % 100 + 1;
        }
    }
}

```

```

double multiplyArr(int n) {
    int** a = createArr(n);
    int** b = createArr(n);
    int** c = createArr(n);

    fillArr(a, n);
    fillArr(b, n);

    clock_t start = clock();

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            c[i][j] = 0;
            for (int k = 0; k < n; k++) {
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    clock_t end = clock();
    double timeTaken = ((double)(end - start)) / CLOCKS_PER_SEC;

    freeArr(a, n);
    freeArr(b, n);
    freeArr(c, n);

    return timeTaken;
}

void runCalculations() {
    int sizes[] = { 100, 200, 400, 1000, 2000, 4000, 10000 };
    int numSize = sizeof(sizes) / sizeof(sizes[0]);

    FILE* file = fopen("rez.txt", "w");
    if (file == NULL) {
        printf("Ошибка открытия файла!\n");
        return;
    }
}

```



```

fprintf(file, "Размер матрицы\tВремя (сек)\n");

srand(time(NULL));

for (int i = 0; i < numSize; i++) {
    int n = sizes[i];
    printf("Вычисление для матрицы %dx%d...\n", n, n);

    double timeTaken = multiplyArr(n);

    fprintf(file, "%d\t%.6f\n", n, timeTaken);

    printf("Завершено: %dx%d - время: %.6f сек.\n", n, n,
timeTaken);
}

fclose(file);
printf("Результаты записаны в файл rez.txt\n");
}

```

Файл ex2_lab2.c

```

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

void shell(int* items, int count) {
    int i, j, gap, k;
    int x, a[5];
    a[0] = 9; a[1] = 5; a[2] = 3; a[3] = 2; a[4] = 1;

    for (k = 0; k < 5; k++) {
        gap = a[k];
        for (i = gap; i < count; ++i) {
            x = items[i];
            for (j = i - gap; (x < items[j]) && (j >= 0); j = j
- gap)

```

```

        items[j + gap] = items[j];
        items[j + gap] = x;
    }
}

void qs(int* items, int left, int right) {

    if (left >= right) {
        return;
    }

    while (left < right) {
        int i = left;
        int j = right;
        int pivot = items[(left + right) / 2];

        while (i <= j) {
            while (items[i] < pivot) i++;
            while (items[j] > pivot) j--;
            if (i <= j) {
                int temp = items[i];
                items[i] = items[j];
                items[j] = temp;
                i++;
                j--;
            }
        }

        if (j - left < right - i) {
            qs(items, left, j);
            left = i;
        }
        else {
            qs(items, i, right);
            right = j;
        }
    }
}

```

```

        }
    }
}

int compare(const void* a, const void* b) {
    return (*(int*)a - *(int*)b);
}

void generateRandom(int* arr, int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 10000;
    }
}

void generateAscending(int* arr, int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = i;
    }
}

void generateDescending(int* arr, int n) {
    for (int i = 0; i < n; i++) {
        arr[i] = n - i;
    }
}

void generateMixed(int* arr, int n) {
    for (int i = 0; i < n / 2; i++) {
        arr[i] = i;
    }
    for (int i = n / 2; i < n; i++) {
        arr[i] = n - i;
    }
}

int* copyArray(const int* source, int n) {
    int* copy = malloc(n * sizeof(int));
    if (copy == NULL) {
        printf("Ошибка выделения памяти!\n");
    }
}

```

```

        exit(1);
    }
    memcpy(copy, source, n * sizeof(int));
    return copy;
}

double measureShellTime(int* arr, int n) {
    int* temp = copyArray(arr, n);
    clock_t start = clock();
    shell(temp, n);
    clock_t end = clock();
    free(temp);
    return ((double)(end - start)) / CLOCKS_PER_SEC;
}

double measureQsTime(int* arr, int n) {
    int* temp = copyArray(arr, n);
    clock_t start = clock();
    qs(temp, 0, n - 1);
    clock_t end = clock();
    free(temp);
    return ((double)(end - start)) / CLOCKS_PER_SEC;
}

double measureStdQsortTime(int* arr, int n) {
    int* temp = copyArray(arr, n);
    clock_t start = clock();
    qsort(temp, n, sizeof(int), compare);
    clock_t end = clock();
    free(temp);
    return ((double)(end - start)) / CLOCKS_PER_SEC;
}

int main() {
    const int n = 100000;
    int* originalArray = malloc(n * sizeof(int));

    if (originalArray == NULL) {
        printf("Ошибка выделения памяти!\n");
    }
}

```

```

        return 1;
    }

    const char* arrayTypes[] = {
        "Случайный",
        "Возрастающий",
        "Убывающий",
        "Смешанный"
    };

    void (*generators[])(int*, int) = {
        generateRandom,
        generateAscending,
        generateDescending,
        generateMixed
    };

    FILE* file = fopen("rez2.txt", "w");
    if (file == NULL) {
        printf("Ошибка открытия файла!\n");
        free(originalArray);
        return 1;
    }

    fprintf(file, "Сортировка Шелла      Быстрая сортировка\n");
    fprintf(file, "Стандартная qsort\n");

    srand(time(NULL));

    for (int i = 0; i < 4; i++) {
        printf("Тестирование на массиве: %s\n", arrayTypes[i]);

        generators[i](originalArray, n);

        double timeShell = measureShellTime(originalArray, n);

        double timeQs = measureQsTime(originalArray, n);

```

```

        double timeStdQsort = measureStdQsortTime(originalArray,
n);

        fprintf(file, "    %.6f\t\t%.6f\t\t%.6f\n",
            timeShell, timeQs, timeStdQsort);

        printf("Результаты: Shell: %.6f, QS: %.6f, Std:
%.6f\n\n",
            timeShell, timeQs, timeStdQsort);
    }

    fclose(file);
    free(originalArray);

    printf("Результаты записаны в файл rez2.txt\n");

    return 0;
}

```

Вывод:

В ходе выполнения лабораторной работы были разработаны программы для выполнения заданий Лабораторной работы №2. В процессе выполнения работы была произведена оценка сложности программы и выявлена зависимость скорости выполнения от исходного набора данных.