Пензенский государственный университет

Кафедра "Вычислительной техники"

**Отчёт**

По лабораторной №2

по дисциплине "Л и ОА в ИЗ"

на тему "Оценка времени выполнения программ"

***Выполнили студенты группы 24ввв1:***

*Цыбузин Д.В*

*Тусков А.А*

***Приняли:***

*Юрова О. В.*

Пенза 2025

# Цель работы: Оценить время работы программ.

**Лабораторное задание:**

**Задание 1:**

1. Вычислить порядок сложности программы (*О*-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.

**Теоретическая часть:**

Для оценки времени выполнения программ языка Python или его частей могут использоваться средства, предоставляемые библиотекой time. Данная библиотека содержит функции для работой с датой и временем.

1. time.time() - в**озвращает текущее время в секундах с момента эпохи**

2. time.perf\_counter() - **возвращает значение счётчика производительности** в виде числа с плавающей точкой

Для оценки времени выполнения программ языка Си или их частей могут использоваться средства, предоставляемые библиотекой **time.h**. Данная библиотека содержит описания типов и прототипы функций для работы с датой и временем.

**Практическая часть:**

**Листинг:**

import time

import random

from matplotlib import pyplot as plt

def fill\_mas(mas, var):

    if var == 1:  *# Возрастающий*

        return list(range(len(mas)))

    if var == 2:  *# Убывающий*

        return list(range(len(mas), 0, -1))

    if var == 3:  *# Возрастающе-убывающий*

        half = len(mas) // 2

        return list(range(half)) + list(range(len(mas) - half, 0, -1))

    if var == 4:  *# Случайный*

        return [random.**randint**(-100, 100) for \_ in range(len(mas))]

def shell(nums):

    arr = nums.copy()  *# Работаем с копией*

    inc = len(arr) // 2

    while inc:

        for i in range(inc, len(arr)):

            temp = arr[i]

            j = i

            while j >= inc and arr[j - inc] > temp:

                arr[j] = arr[j - inc]

                j -= inc

            arr[j] = temp

        inc = inc // 2

    return arr

def built\_in\_sort(nums):

    return sorted(nums.copy())  *# Работаем с копией*

def quicksort(nums):

    if len(nums) <= 1:

        return nums.copy()  *# Возвращаем копию*

    q = random.**choice**(nums)

    s\_nums = [n for n in nums if n < q]

    e\_nums = [n for n in nums if n == q]

    m\_nums = [n for n in nums if n > q]

    return quicksort(s\_nums) + e\_nums + quicksort(m\_nums)

def fill\_matr(matr):

    for i in range(len(matr)):

        for j in range(len(matr[0])):

            matr[i][j] = random.**randint**(-100, 100)

def matrix\_multiply(A, B):

    rows\_A, cols\_A = len(A), len(A[0])

    rows\_B, cols\_B = len(B), len(B[0])

    if cols\_A != rows\_B:

        raise ValueError("Невозможно умножить матрицы")

    result = [[0 for \_ in range(cols\_B)] for \_ in range(rows\_A)]

    for i in range(rows\_A):

        for j in range(cols\_B):

            for k in range(cols\_A):

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

    return result

def test(razm):

*# ПРАВИЛЬНОЕ создание матриц*

    matrix\_1 = [[0] \* razm for \_ in range(razm)]

    matrix\_2 = [[0] \* razm for \_ in range(razm)]

    fill\_matr(matrix\_1)

    fill\_matr(matrix\_2)

    start = time.perf\_counter()

    multiply = matrix\_multiply(matrix\_1, matrix\_2)

    end = time.perf\_counter()

    time\_of\_work = end - start

    print(f"Время умножения матриц {razm}X{razm}: {time\_of\_work:.6f} секунд")

    return time\_of\_work

def sortir(mas, var):

    arr = mas.copy()  *# Всегда работаем с копией*

    if var == 1:

        return shell(arr)

    if var == 2:

        return built\_in\_sort(arr)

    if var == 3:

        return quicksort(arr)

def process\_array(arr, fill\_idx, result\_list, time\_list):

    original\_arr = fill\_mas(arr, fill\_idx)

    for sort\_type in [1, 2, 3]:

*# Создаем копию для каждой сортировки*

        arr\_copy = original\_arr.copy()

        start = time.perf\_counter()

        sorted\_arr = sortir(arr\_copy, sort\_type)

        end = time.perf\_counter()

        result\_list.append(sorted\_arr)

        time\_list.append(end - start)

def sortirovki():

*# Уменьшим размеры для тестирования*

    c = [[0] \* 10000 for \_ in range(4)]  *# было 100000*

    d = [[0] \* 30000 for \_ in range(4)]  *# было 300000*

    result\_1, time\_1 = [], []

    result\_2, time\_2 = [], []

    for i in range(4):

        process\_array(c[i], i + 1, result\_1, time\_1)

    for i in range(4):

        process\_array(d[i], i + 1, result\_2, time\_2)

    return result\_1, result\_2, time\_1, time\_2

def draw(X, Y):

    plt.figure(*figsize*=(10, 6))

    plt.plot(X, Y, 'bo-')

    plt.xlabel("Размерность матриц (N x N)")

    plt.ylabel("Время работы (секунды)")

    plt.title("Зависимость времени умножения матриц от их размера")

    plt.plot([i for i in range(10)], [i\*\*3 for i in range(10)])

    plt.show()

def main():

    choose = meet()

    if choose == 1:

        print("Запуск тестирования сортировок...")

        res\_1, res\_2, tim\_1, tim\_2 = sortirovki()

        print("\n" + "="\*80)

        print("10000 чисел")

        print("Вид\\Набор данных       Возрастающий   Убывающий   Возраст-убыв   Случайный")

        print("-"\*80)

        print(f"Сортировка Шелла      {tim\_1[0]:.6f}    {tim\_1[1]:.6f}    {tim\_1[2]:.6f}    {tim\_1[3]:.6f}")

        print(f"Встроенная сортировка {tim\_1[4]:.6f}    {tim\_1[5]:.6f}    {tim\_1[6]:.6f}    {tim\_1[7]:.6f}")

        print(f"Быстрая сортировка    {tim\_1[8]:.6f}    {tim\_1[9]:.6f}    {tim\_1[10]:.6f}    {tim\_1[11]:.6f}")

        print("\n" + "="\*80)

        print("30000 чисел")

        print("Вид\\Набор данных       Возрастающий   Убывающий   Возраст-убыв   Случайный")

        print("-"\*80)

        print(f"Сортировка Шелла      {tim\_2[0]:.6f}    {tim\_2[1]:.6f}    {tim\_2[2]:.6f}    {tim\_2[3]:.6f}")

        print(f"Встроенная сортировка {tim\_2[4]:.6f}    {tim\_2[5]:.6f}    {tim\_2[6]:.6f}    {tim\_2[7]:.6f}")

        print(f"Быстрая сортировка    {tim\_2[8]:.6f}    {tim\_2[9]:.6f}    {tim\_2[10]:.6f}    {tim\_2[11]:.6f}")

    elif choose == 2:

        print("Запуск тестирования умножения матриц...")

*# Создаем размерности матриц*

        rows\_cols = []

        current = 100

        while len(rows\_cols) < 10:  *# Ограничим до 10 размеров для теста*

            rows\_cols.append(current)

            if current < 1000:

                current += 100

            else:break

*#current += 500  # Для больших размеров увеличиваем шаг*

        times = []

        for size in rows\_cols:

            try:

                time\_taken = test(size)

                times.append(time\_taken)

            except MemoryError:

                print(f"Пропускаем размер {size} - недостаточно памяти")

                break

            except Exception as e:

                print(f"Ошибка при размере {size}: {e}")

                break

        print(f"\nРазмеры матриц: {rows\_cols}")

        print(f"Время выполнения: {times}")

        if len(rows\_cols) == len(times):

            draw(rows\_cols, times)

*#draw()*

        else:

*# Если какие-то размеры были пропущены, обрезаем lists*

            min\_len = min(len(rows\_cols), len(times))

            draw(rows\_cols[:min\_len], times[:min\_len])

def meet():

    while True:

        try:

            print("\nВыберите операцию:")

            print("1 - Сравнение сортировок")

            print("2 - Перемножение матриц")

            choose = int(input("Введите 1 или 2: "))

            if choose in [1, 2]:

                return choose

            else:

                print("Пожалуйста, введите 1 или 2")

        except ValueError:

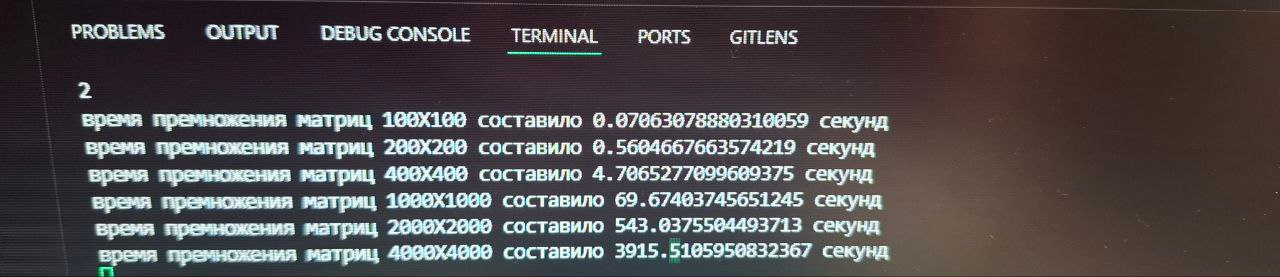
            print("Пожалуйста, введите число")

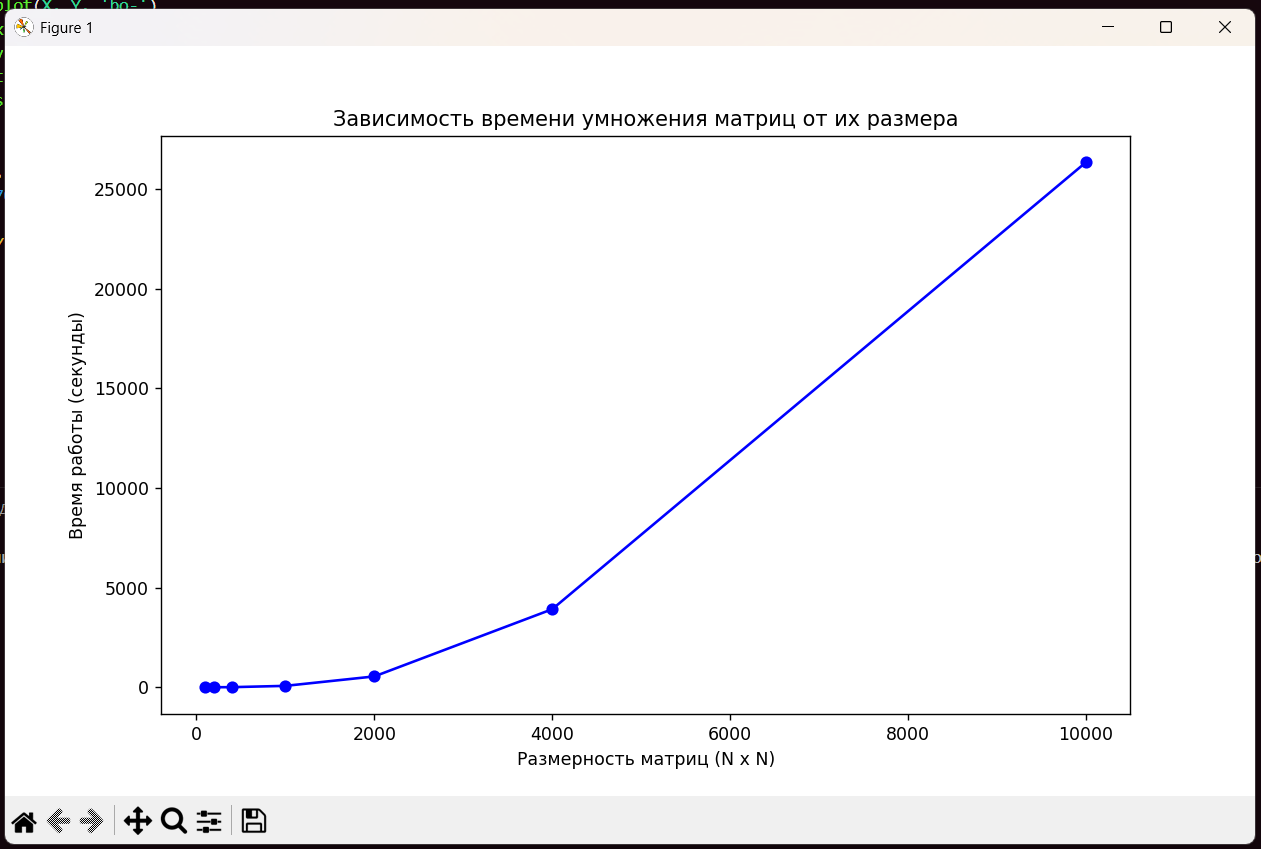
if \_\_name\_\_ == "\_\_main\_\_":

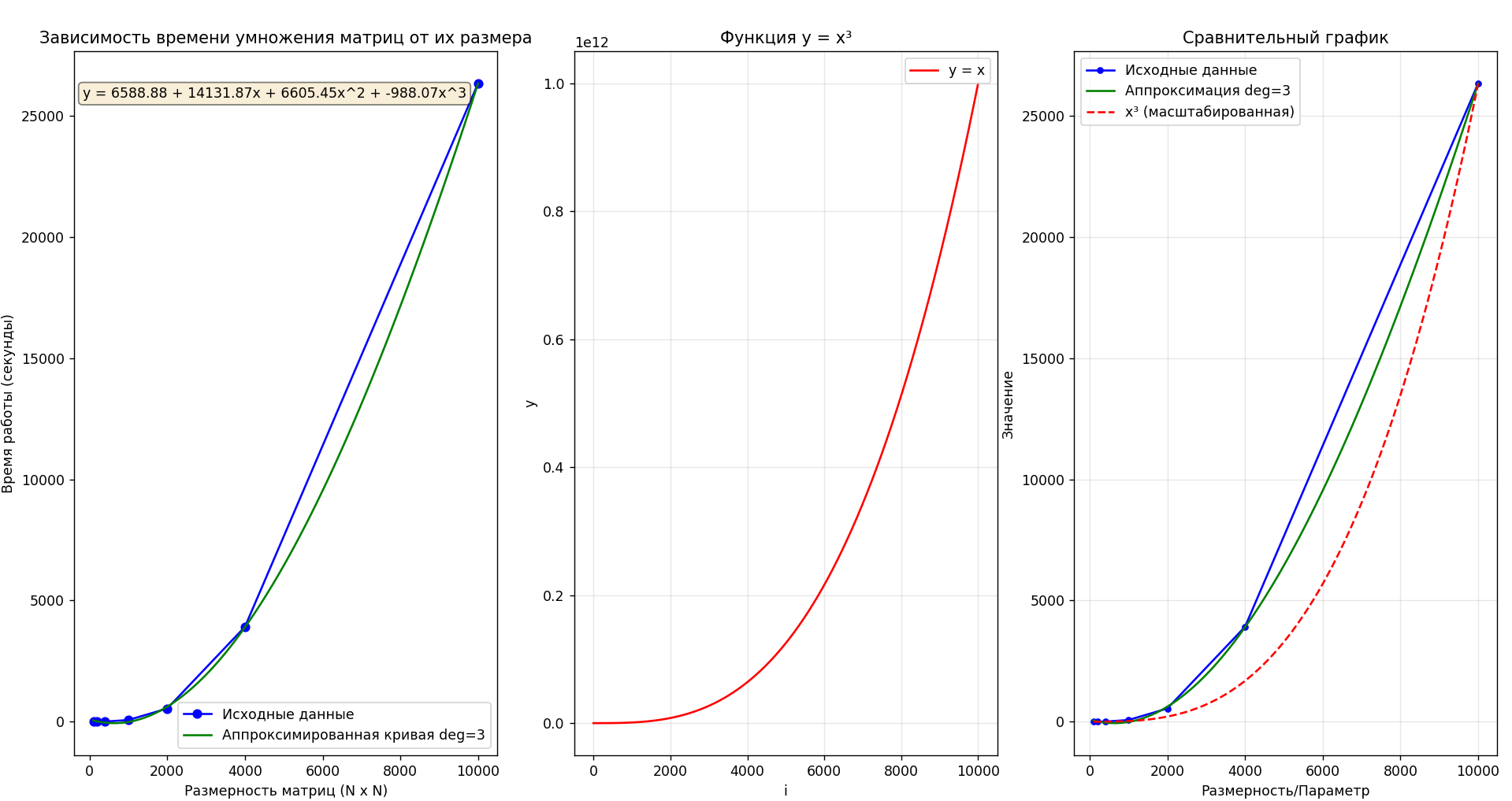
    main()

**Задание 1.**

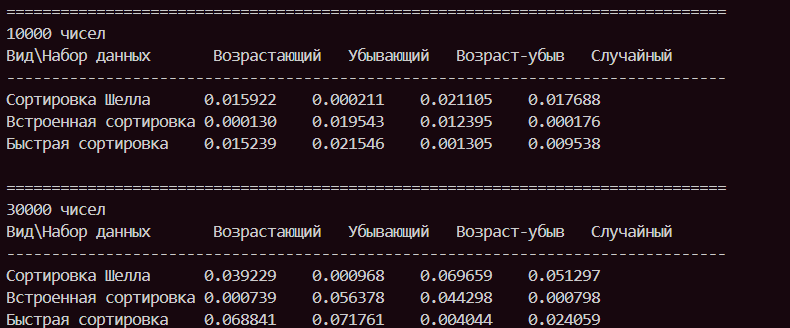
**Результат работы программы:**

****

Дополнительное задание: Нарисовать на графике кривую, по которой идут точки и кривую предположительных результатов



**Задание 2:**

****

**Вывод:**

1. Выяснили, что алгоритм быстрой сортировки, реализованный вручную, быстрее всех работает на возрастающе-убывающем набором данных в массиве
2. С увеличением количества данных в массиве, увеличивается, время, затрачиваемое на сортировку
3. Сортировка Шелла требует наибольше количество времени на работу, так как он сравнивает не только элементы, стоящие рядом, но и на определённом расстоянии
4. Функция быстрой сортировки из библиотеки требует большее количество времени, чем реализованный вручную