МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра «Вычислительная техника»

ОТЧЁТ

По лабораторной работе №2

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Оценка времени выполнения программ»

Выполнили
студенты
группы
23ВВВ4:
Святов И.Ю.
Епинин Д.В.

Приняли:
Юрова О.В.
Деев М.В.

Пенза 2024

Цель работы: научиться определять порядок сложности программы и оценивать время выполнения программы.

Задание 1:

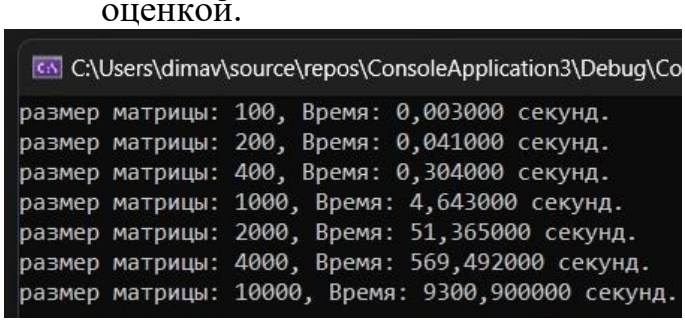
1. Вычислить порядок сложности программы (*O*-символику).

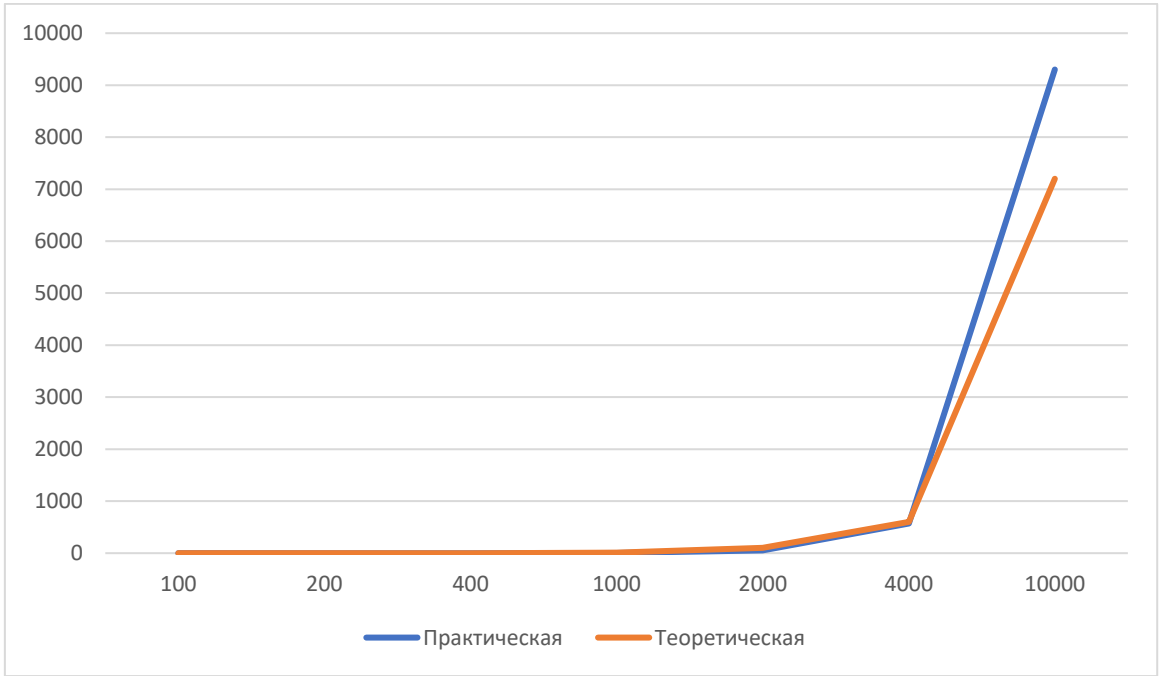
Присутствуют 3 вложенных цикла:
Внешний цикл по i (от 0 до 199) - O(n). Средний цикл по j (от 0 до 199) - O(n). Внутренний цикл по r (от 0 до 199) - O(n). Следовательно, сложность умножения двух матриц составит O(n^3).

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.

Время выполнения программы должно увеличиться в связи увеличения матрицы от размера 100x100 к 10000x10000.

3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.





Задание 2:

```
count == 100
Shell sort (random): 0.000000 seconds
Quick sort (random): 0.000000 seconds
qsort (random): 0.000000 seconds
Shell sort (ascending): 0.000000 seconds
Quick sort (ascending): 0.000000 seconds
qsort (ascending): 0.000000 seconds
Shell sort (descending): 0.000000 seconds
Quick sort (descending): 0.000000 seconds
qsort (descending): 0.000000 seconds
Shell sort (half ascending, half descending): 0.000000 seconds
Quick sort (half ascending, half descending): 0.000000 seconds
qsort (half ascending, half descending): 0.000000 seconds

count == 1000
Shell sort (random): 0.005000 seconds
Quick sort (random): 0.000000 seconds
qsort (random): 0.001000 seconds
Shell sort (ascending): 0.000000 seconds
Quick sort (ascending): 0.000000 seconds
qsort (ascending): 0.000000 seconds
Shell sort (descending): 0.000000 seconds
Quick sort (descending): 0.000000 seconds
qsort (descending): 0.001000 seconds
Shell sort (half ascending, half descending): 0.000000 seconds
Quick sort (half ascending, half descending): 0.001000 seconds
qsort (half ascending, half descending): 0.000000 seconds

count == 7000
Shell sort (random): 0.006000 seconds
Quick sort (random): 0.008000 seconds
qsort (random): 0.002000 seconds
Shell sort (ascending): 0.000000 seconds
Quick sort (ascending): 0.000000 seconds
qsort (ascending): 0.001000 seconds
Shell sort (descending): 0.017000 seconds
Quick sort (descending): 0.000000 seconds
qsort (descending): 0.002000 seconds
Shell sort (half ascending, half descending): 0.006000 seconds
Quick sort (half ascending, half descending): 0.043000 seconds
qsort (half ascending, half descending): 0.005000 seconds
```

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.

Сортировка Шелла(shell sort):

- Случайные данные: Сортировка Шелла демонстрирует отличные результаты на небольших массивах (100 элементов), однако её эффективность существенно падает на более крупных массивах (1000 и 10000 элементов).

Быстрая сортировка(quick sort):

- Случайные данные: Быстрая сортировка показывает высокую эффективность на массивах любого размера, особенно на больших.

Функция qsort:

- Случайные данные: Функция qsort показывает неплохую производительность на массивах любого размера, сравнимую с быстрой сортировкой.

2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.

Сортировка Шелла(shell sort):

- Возрастающие данные: На упорядоченных по возрастанию массивах сортировка Шелла работает очень быстро, независимо от размера массива.

Быстрая сортировка(quick sort):

- Возрастающие данные: Быстрая сортировка также демонстрирует хорошие результаты, хотя немного медленнее, чем сортировка Шелла при работе с возрастающими данными.

Функция qsort:

- Возрастающие данные: qsort также работает эффективно, хотя немного медленнее, чем сортировка Шелла на упорядоченных массивах.

3. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

Сортировка Шелла(shell sort):

- Убывающие данные: Сортировка Шелла эффективно работает на малых массивах, но на больших она начинает замедляться.

Быстрая сортировка(quick sort):

- Убывающие данные: По производительности быстрая сортировка близка к результатам, полученным на возрастающих данных.

Функция qsort:

- Убывающие данные: По производительности qsort сопоставима с быстрой сортировкой на возрастающих данных.

4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

Сортировка Шелла(shell sort):

- Полувозрастающие/полуубывающие данные: При этих данных сортировка Шелла демонстрирует среднюю производительность, которая лучше, чем на случайных данных, но хуже, чем на полностью возрастающих.

Быстрая сортировка(quick sort):

-Полувозрастающие/полуубывающие данные: На таких массивах быстрая сортировка значительно замедляется, особенно с увеличением размера массива.

Функция qsort:

- Полувозрастающие/полуубывающие данные: qsort показывает хорошие результаты, значительно превышающие производительность быстрой сортировки на таких массивах.

5. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

1. Сортировка Шелла эффективно работает с возрастающими данными и небольшими массивами, но проявляет слабую производительность на крупных массивах случайного порядка.

2. Быстрая сортировка демонстрирует стабильные и хорошие результаты на случайных, возрастающих и убывающих данных, однако значительно замедляется, когда обрабатываются полувозрастающие/полуубывающие массивы.

3. Функция qsort обеспечивает надежную и высокую производительность на всех типах данных и размерах массивов, что делает её универсальным инструментом для сортировки.

Вывод: в ходе лабораторной работы научились определять порядок сложности программы и оценивать время выполнения программы.