Пензенский государственный университет

Кафедра «Вычислительная техника»

Пенза 2022

**ОТЧЕТ**

по лабораторной работе №2

на тему «Оценка времени выполнения программ»

**Выполнил**

**студент группы 21ВВ3:**

Столяров А.

**Приняли:**

Митрохин М. А.

Юрова О. В.

### Цель работы:

### Оценка времени выполнения программ языка Си или их частей.

**Лабораторное задание:**

Дана программа, вычисляющая произведение двух матриц:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(void)

{

setvbuf(stdin, NULL, \_IONBF, 0);

setvbuf(stdout, NULL, \_IONBF, 0);

clock\_t start, end; // объявляем переменные для определения времени выполнения

int i=0, j=0, r;

int a[200][200], b[200][200], c[200][200], elem\_c;

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

while(i<200)

{

while(j<200)

{

a[i][j]=rand()% 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

i=0; j=0;

while(i<200)

{

while(j<200)

{

b[i][j]=rand()% 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

for(i=0;i<200;i++)

{

for(j=0;j<200;j++)

{

elem\_c=0;

for(r=0;r<200;r++)

{

elem\_c=elem\_c+a[i][r]\*b[r][j];

c[i][j]=elem\_c;

}

}

}

return(0);

}

**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.

Даны реализации алгоритмов сортировки Шелла и быстрой сортировки:

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) //вызов функции: qs(items, 0, count-1);

{

int i, j;

int x, y;

i = left; j = right;

/\* выбор компаранда \*/

x = items[(left+right)/2];

do {

while((items[i] < x) && (i < right)) i++;

while((x < items[j]) && (j > left)) j--;

if(i <= j) {

y = items[i];

items[i] = items[j];

items[j] = y;

i++; j--;

}

} while(i <= j);

if(left < j) qs(items, left, j);

if(i < right) qs(items, i, right);

}

**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.

**Результаты выполнения заданий:**

**Задание 1:**

1. Рассмотрев программу, определили сложность алгоритма программы О(n3).
2. Мы оценили время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000. Рис 1-7.

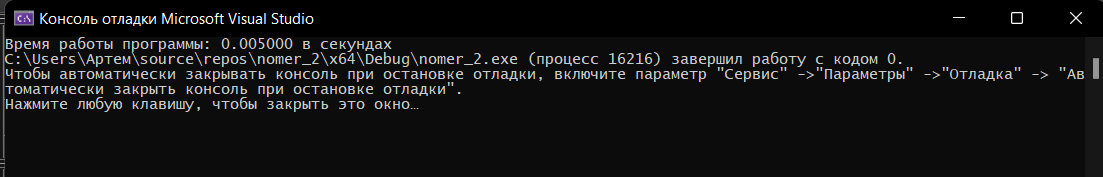


Рисунок 1 — Матрица размером 100

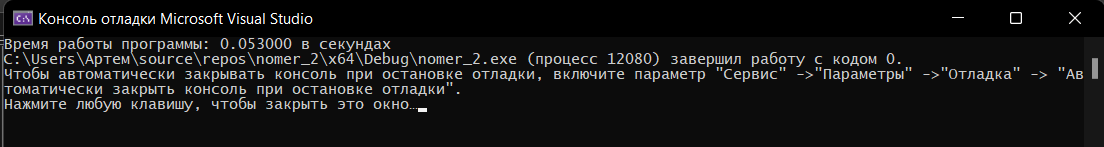


Рисунок 2 — Матрица размером 200

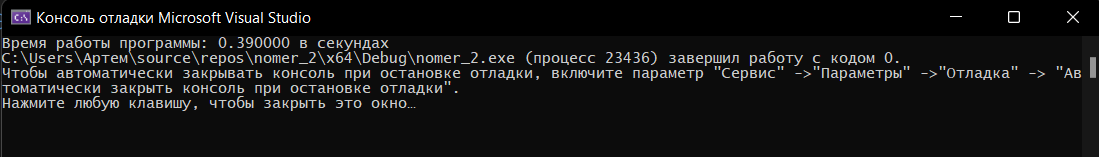


Рисунок 3 — Матрица размером 400

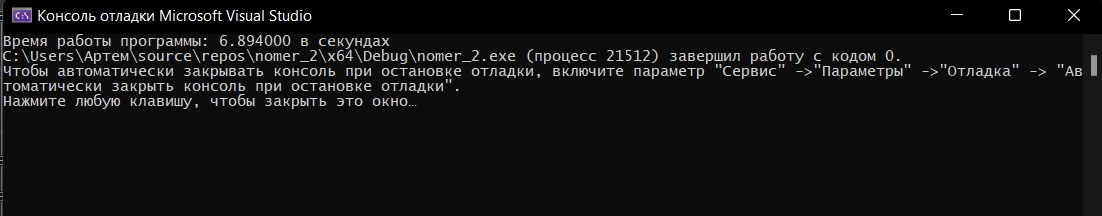


Рисунок 4 — Матрица размером 1000

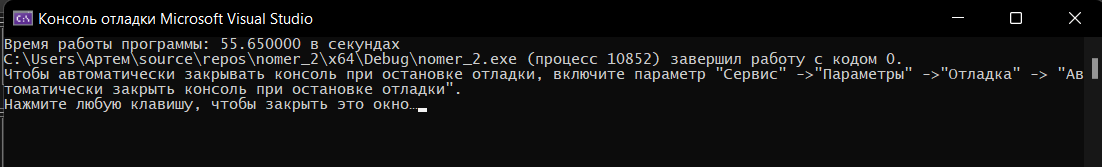


Рисунок 5 — Матрица размером 2000

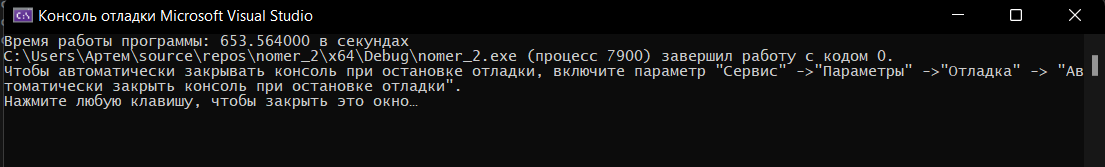


Рисунок 6 — Матрица размером 4000

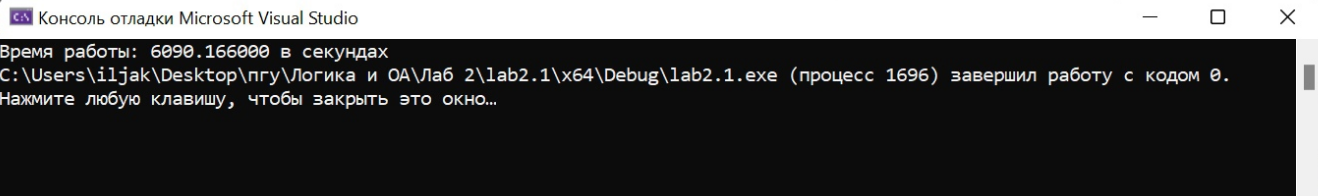


Рисунок 7 — Матрица размером 10000

Рисунок 8 — График зависимостей

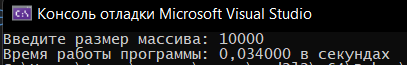
**Задание 2:**

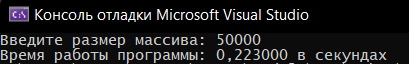
1. Оценили время работы каждого из реализованных алгоритмов на случайном наборе значений массива.

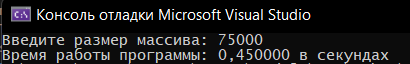
Shell

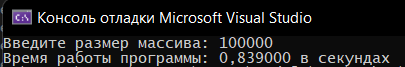


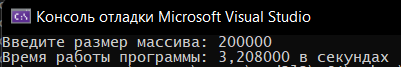












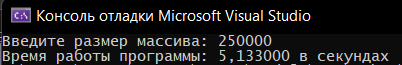
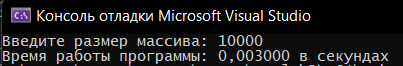


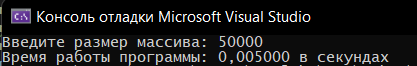
Рисунок 9 — коллаж измерений shell

Qs















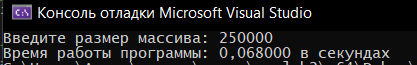


Рисунок 10 — коллаж измерений qs

**Shell:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Время | 0,001 | 0,006 | 0,034 | 0,223 | 0,450 | 0,839 | 3,208 | 5,133 |
| Кол-во данных | 3000 | 4000 | 10000 | 50000 | 75000 | 100000 | 200000 | 250000 |

**Qs:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Время | 0,001 | 0,001 | 0,003 | 0,005 | 0,026 | 0,025 | 0,037 | 0,068 |
| Кол-во данных | 3000 | 4000 | 10000 | 50000 | 75000 | 100000 | 200000 | 250000 |

Таблица 1 — сравнение shell и qs

Время выполнения программы увеличивается прямо пропорционально увеличению размера массива. Qs лучше справляется с массивами больших размеров нежели Shell.

1. Оценили время работы каждого из реализованных алгоритмов на массиве, представляющих собой возрастающую последовательность чисел.

Shell

90000



150000



200000



250000



Рисунок 11 — коллаж измерений shell

Qs

90000



150000



200000



250000



Рисунок 12 — коллаж измерений qs

**Shell:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во данных | 90000 | 150000 | 200000 | 250000 |
| Время | 0,002 | 0,003 | 0,004 | 0,019 |

**Qs:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во данных | 90000 | 150000 | 200000 | 250000 |
| Время | 0,003 | 0,007 | 0,010 | 0,009 |

Таблица 2 — сравнение shell и qs

Время выполнения программы увеличивается прямо пропорционально увеличению размера массива. Shell лучше справляется с массивами, представляющими собой возрастающую последовательность чисел.

1. Оценили время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

Shell

90000



150000



200000



250000



**Рисунок 13 — коллаж измерений shell**

Qs









**Рисунок 14 — коллаж измерений qs**

**Shell:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во данных | 90000 | 150000 | 200000 | 250000 |
| Время | 2,079 | 4,814 | 7,528 | 11,126 |

**Qs:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во данных | 90000 | 150000 | 200000 | 250000 |
| Время | 0,004 | 0,007 | 0,008 | 0,010 |

**Таблица 3 — сравнение shell и qs**

Время выполнения программы увеличивается прямо пропорционально увеличению размера массива. QS лучше справляется с массивами, представляющими собой убывающую последовательность чисел.

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

Shell

10000



40000



70000



100000



**Рисунок 15 — коллаж измерений shell**

Qs

10000



40000



70000



100000



**Рисунок 16 — коллаж измерений qs**

**Shell:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во данных | 10000 | 40000 | 70000 | 100000 |
| Время | 0,005 | 0,069 | 0,397 | 0,589 |

**Qs:**

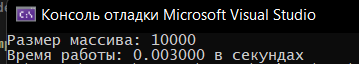
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Кол-во данных | 10000 | 40000 | 70000 | 100000 |
| Время | 0,001 | 0,002 | 0,003 | 0,006 |

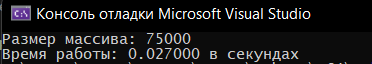
**Таблица 4 — сравнение shell и qs**

Время выполнения программы увеличивается прямо пропорционально увеличению размера массива. Qs лучше справляется с массивами, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

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

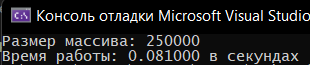
Случайные числа:









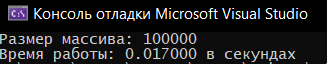


**Рисунок 17 — коллаж измерений qsort**

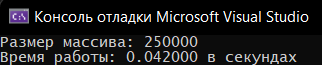
Возрастающая последовательность:







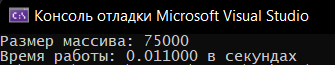




**Рисунок 18 — коллаж измерений qsort**

Убывающая последовательность:





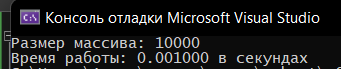


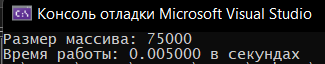


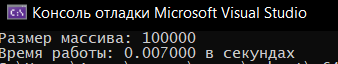


**Рисунок 19 — коллаж измерений qsort**

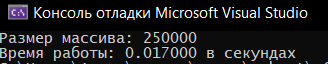
Первая половина, убывающая вторая возрастающая:











**Рисунок 20 — коллаж измерений qsort**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер | Случайные числа | Возрастающая последовательность | Убывающая последовательность | Первая половина, возрастающая, вторая убывающая |
| 10000 | 0,003 | 0,003 | 0,004 | 0,001 |
| 75000 | 0,027 | 0,009 | 0,011 | 0,005 |
| 100000 | 0,032 | 0,017 | 0,019 | 0,007 |
| 200000 | 0,086 | 0,046 | 0,031 | 0,023 |
| 250000 | 0,081 | 0,042 | 0,034 | 0,017 |

**Таблица 5 — сравнение результатов выполнения qsort**

Время выполнения программы увеличивается прямо пропорционально увеличению размера массива. Qsort одинаково хорошо справляется со всеми указанными выше наборами данных.

### Вывод:

Рассмотрев программу, определили сложность алгоритма программы О(n3). При выполнении первого задания, оценили время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки Time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000, и сделали вывод, что по мере увеличения матрицы, время работы программы увеличивается. Так же построили график зависимостей, который отражает данные изменения.

Анализ и оценка работы алгоритмов Shell, QS, показали, что с работой с данными типа случайных наборов значений массива, лучше справляется Qs.

Анализ и оценка работы алгоритмов Shell, QS, показали, что с работой с данными типа возрастающей последовательности чисел, лучше справляется Shell.

Анализ и оценка работы алгоритмов Shell, QS, показали, что с работой с данными типа убывающей последовательности чисел, лучше справляется Qs.

Анализ и оценка работы алгоритмов Shell, QS, показали, что с работой с данными типа, где одна половина — возрастающие числа, а вторая — убывающие, лучше справляется Qs.

Анализ работы со стандартной функцией qsort показал, что при работе со случайными числами, он быстрее, чем Shell, но медленнее, чем Qs.

Анализ работы со стандартной функцией qsort показал, что при работе с возрастающей последовательностью чисел, он медленнее чем Shell и Qs.

Анализ работы со стандартной функцией qsort показал, что при работе с убывающей последовательностью чисел, он быстрее, чем Shell, но медленнее, чем Qs.

Анализ работы со стандартной функцией qsort показал, что при работе с массивом данных, в котором первая половина — возрастающая, а вторая — убывающая, он быстрее чем Shell, а с Qs – время примерно одинаковое.