Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах.»

на тему: «Оценка времени выполнения программ.»

Выполнили**:**

студенты группы 21ВВ4

Федоренко Вероника

Роганов Данила

Проверили:

Юрова О.В,

Акифьев И.В.

Пенза 2022

**Название**

Оценка времени выполнения программ.

**Цель работы**

Изучение функций, работающих со временем, оценки сложности и времени выполнения программ.

**Лабораторное задание**

**Задание 1**: Вычислить порядок сложности программы (О-символику).

Листинг

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

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

const int maxCountElements = 200;

int i = 0, j = 0, r;

short a[maxCountElements][maxCountElements], b[maxCountElements][maxCountElements], c[maxCountElements][maxCountElements], elem\_c;

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

while (i < maxCountElements)

{

while (j < maxCountElements)

{

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 < maxCountElements)

{

while (j < maxCountElements)

{

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

j++;

}

i++;

}

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

{

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

{

elem\_c = 0;

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

{

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

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

}

}

}

return 0;

}

Результат

Данная программа имеет оценку сложности O(n3).

**Задание 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

Листинг

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main()

{

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

const int maxCountElements = 2000;

int i = 0, j = 0, r;

short a[maxCountElements][maxCountElements], b[maxCountElements][maxCountElements], c[maxCountElements][maxCountElements], elem\_c;

start = clock();

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

while (i < maxCountElements)

{

while (j < maxCountElements)

{

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 < maxCountElements)

{

while (j < maxCountElements)

{

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

j++;

}

i++;

}

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

{

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

{

elem\_c = 0;

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

{

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

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

}

}

}

end = clock();

double diff = (end - start) / CLOCKS\_PER\_SEC;

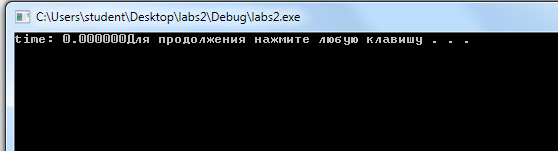
printf("time: %lf", diff);

return 0;

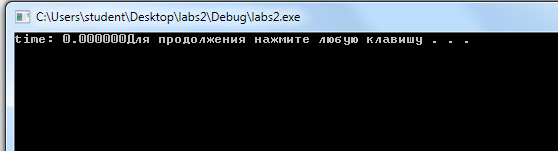
}

Результат

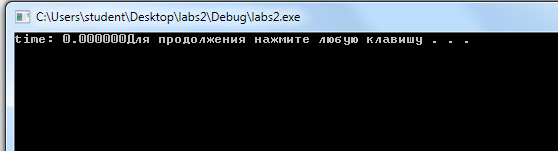
Матрицы от 100:



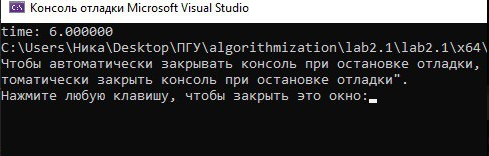
Матрицы от 200:



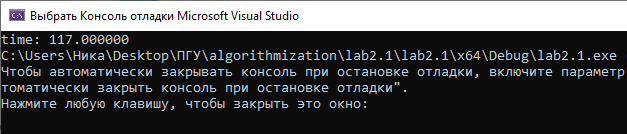
Матрицы от 400:



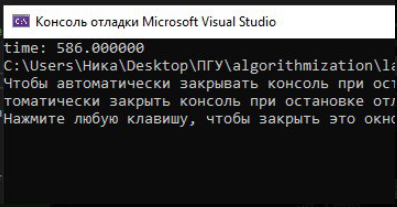
Матрицы от 1000:



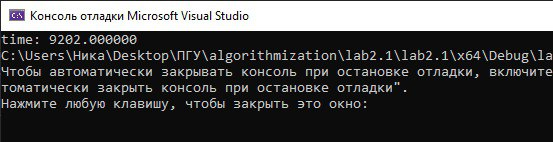
Матрицы от 2000:



Матрицы от 4000:



Матрицы от 10000:



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



**Задание 4:** Измененный код к первому заданию.

Листинг

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void allocateMatrix(int\*\*\* sourceMatrix, int countElements) {

\*sourceMatrix = (int\*\*)malloc(countElements \* sizeof(int\*));

for (int i = 0; i < countElements; i++)

{

(\*sourceMatrix)[i] = (int\*)malloc(countElements \* sizeof(int));

}

}

void fillMatrixRandomElements(int\*\* sourseMatrix, int countElements) {

for (int i = 0; i < countElements; i++)

{

for (int j = 0; j < countElements; j++)

{

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

}

}

}

int main()

{

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

const int maxCountElements = 1000;

int\*\* firstMatrix = NULL;

int\*\* secondMatrix = NULL;

int\*\* resultMatrix = NULL;

start = clock();

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

allocateMatrix(&firstMatrix, maxCountElements);

fillMatrixRandomElements(firstMatrix, maxCountElements);

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

allocateMatrix(&secondMatrix, maxCountElements);

fillMatrixRandomElements(secondMatrix, maxCountElements);

allocateMatrix(&resultMatrix, maxCountElements);

for (int i = 0; i < maxCountElements; i++)

{

for (int j = 0; j < maxCountElements; j++)

{

int sumElem = 0;

for (int r = 0; r < maxCountElements; r++)

{

sumElem = sumElem + firstMatrix[i][r] \* secondMatrix[r][j];

resultMatrix[i][j] = sumElem;

}

}

}

end = clock();

double diff = (end - start) / CLOCKS\_PER\_SEC;

printf("time: %lf", diff);

free(firstMatrix);

free(secondMatrix);

free(resultMatrix);

return 0;

}

**Задание 5:** оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.

**Сортировка Шелла.**

Листинг

int main()

{

clock\_t start, end;

const int maxCountElements = 20000;

int \*sourseArray, index = 0;

srand(time(NULL));

sourseArray = (int\*)calloc(maxCountElements, sizeof(int));

for (index = 0; index < maxCountElements; index++)

{

sourseArray[index] = rand() % 100;

printf("%-3d", sourseArray[index]);

}

start = clock();

shellSort(sourseArray, maxCountElements);

end = clock();

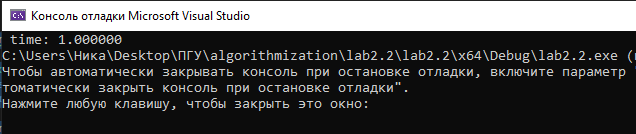
double diff = (end - start);

printf(" time: %lf", diff);

free(sourseArray);

}

Результат



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

Листинг

int main()

{

clock\_t start, end;

const int maxCountElements = 20000;

int \*sourseArray, index = 0;

sourseArray = (int\*)calloc(maxCountElements, sizeof(int));

for (index = 0; index < maxCountElements; index++)

{

sourseArray[index] = index;

//printf("%-3d", sourseArray[index]);

}

start = clock();

shellSort(sourseArray, maxCountElements);

end = clock();

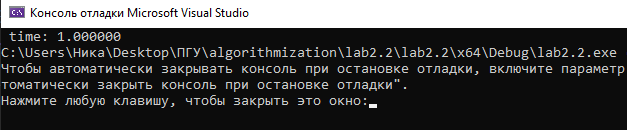
double diff = (end - start);

printf(" time: %lf", diff);

free(sourseArray);

}

Результат



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

Листинг

int main()

{

clock\_t start, end;

const int maxCountElements = 20000;

int \*sourseArray, index = 0;

sourseArray = (int\*)calloc(maxCountElements, sizeof(int));

for (index = 0; index < maxCountElements; index++)

{

sourseArray[index] = -index;

//printf("%-3d", sourseArray[index]);

}

start = clock();

shellSort(sourseArray, maxCountElements);

end = clock();

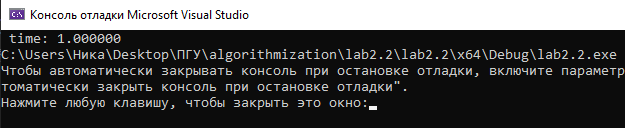
double diff = (end - start);

printf(" time: %lf", diff);

free(sourseArray);

}

Результат



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

Листинг

int main()

{

clock\_t start, end;

const int maxCountElements = 20000;

;

int\* sourseArray, index = 0;

//srand(time(NULL));

sourseArray = (int\*)calloc(maxCountElements, sizeof(int));

for (index = 0; index < maxCountElements; index++)

{

if ( index < (maxCountElements / 2)) {

sourseArray[index] = index;

}

else

{

sourseArray[index] = -index;

}

}

start = clock();

shell(sourseArray, maxCountElements);

end = clock();

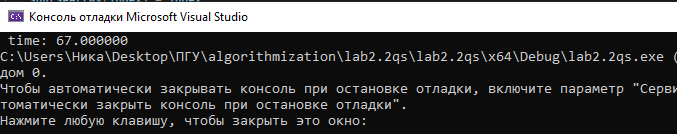
double diff = (end - start);

printf(" time: %lf", diff);

free(sourseArray);

}

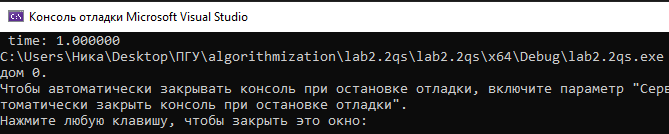
Результат



**Быстрая сортировка.**

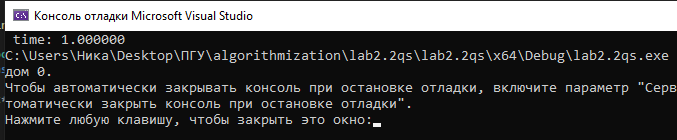
Случайное заполнение массива.

Результат



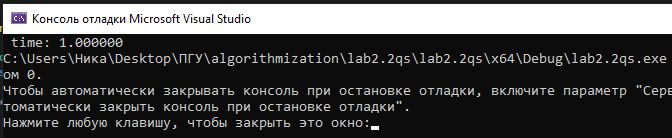
Монотонно возрастающая последовательность.

Результат



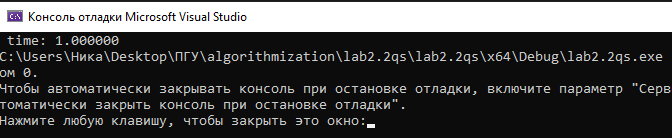
Монотонно убывающая последовательность.

Результат



Треугольник.

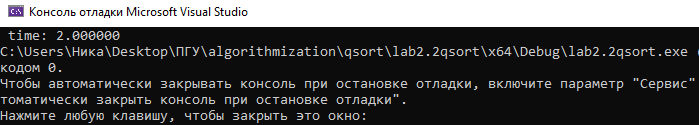
Результат



**Qsort().**

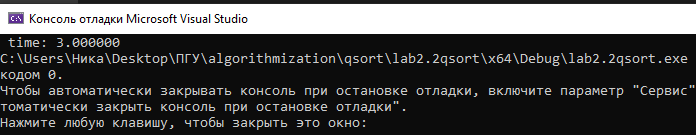
Случайное заполнение массива.

Результат



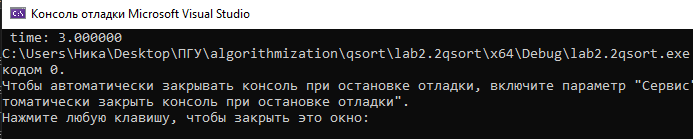
Монотонно возрастающая последовательность.

Результат



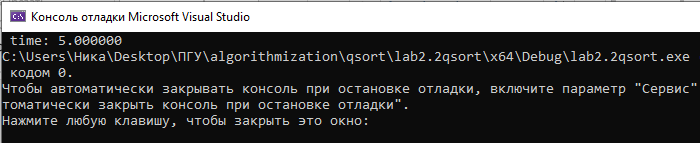
Монотонно убывающая последовательность.

Результат

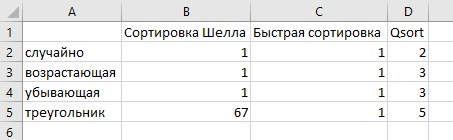


Треугольник.

Результат



**Сравнение производительности сортировок.**



Вывод: в ходе выполнения лабораторной работы были разработаны программы, реализующие работу с разными видами сортировок.