**Министерство науки и высшего образования   
Пензенский государственный университет  
Кафедра “Вычислительная техника”  
  
  
  
  
  
Отчёт**по лабораторной работе №2  
по курсу “Логика и основы алгоритмизации в ИЗ”   
тема:  Оценка времени выполнения программ

Выполнили студенты группы 22ВВС1:  
 Разин Д.С  
 Беккаревич К.В  
 Приняли: Акифьев И.В.  
 Юрова О.В.

Пенза 2023

**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.

**Ход работы:**

1. Сложность программы – О(A + B + C^3) O(N^3)

2. Время выполнения программы матриц размерами 100



Время выполнения программы матриц размерами 200



Время выполнения программы матриц размерами 400



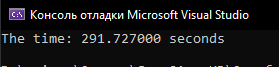
Время выполнения программы матриц размерами 1000



Время выполнения программы матриц размерами 2000



Время выполнения программы матриц размерами 4000



Время выполнения программы матриц размерами 10000



3. График зависимости времени от размера матрицы



**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();

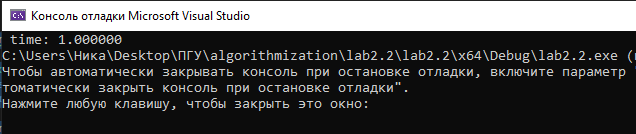
double diff = (end - start);

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();

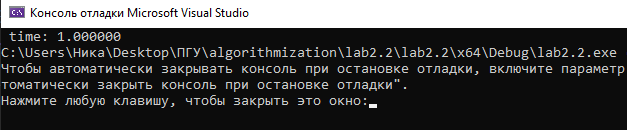
double diff = (end - start);

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();

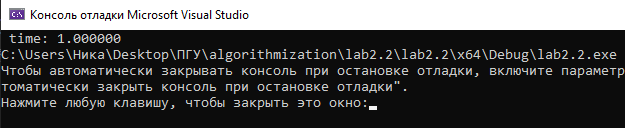
double diff = (end - start);

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();

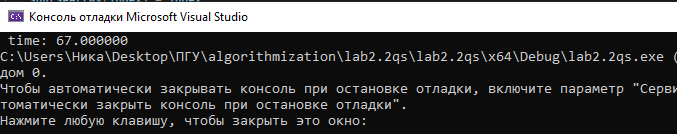
double diff = (end - start);

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

free(sourseArray);

}

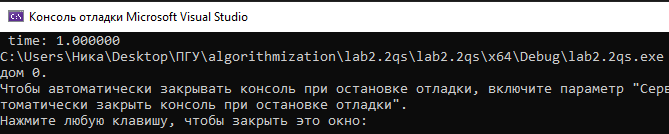
Результат



**Быстрая сортировка.**

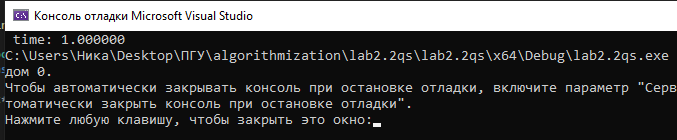
Случайное заполнение массива.

Результат



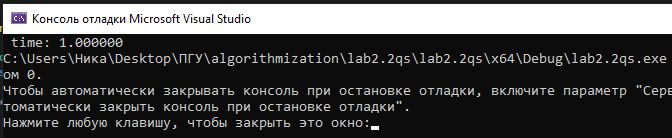
Монотонно возрастающая последовательность.

Результат



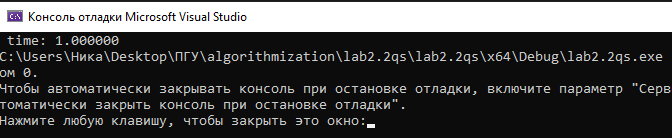
Монотонно убывающая последовательность.

Результат



Треугольник.

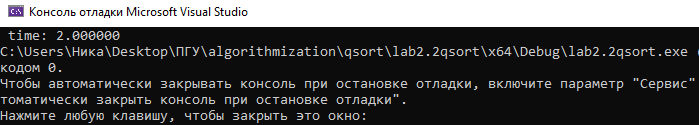
Результат



**Qsort().**

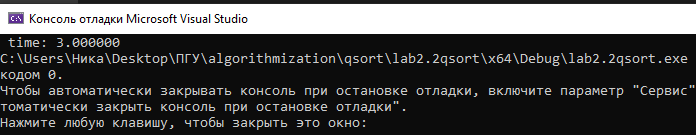
Случайное заполнение массива.

Результат



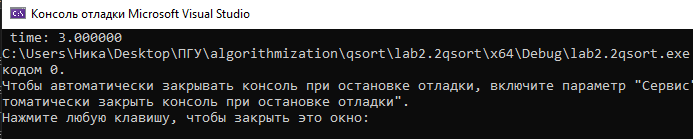
Монотонно возрастающая последовательность.

Результат



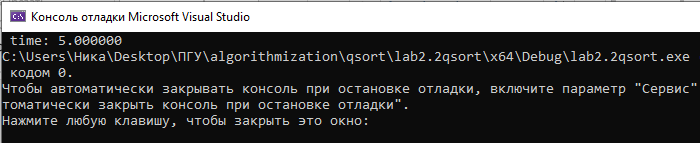
Монотонно убывающая последовательность.

Результат

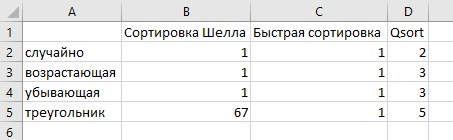


Треугольник.

Результат



**Сравнение производительности сортировок.**



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