**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

(национальный исследовательский университет)»

Институт №3 «Системы управления, информатика и электроэнергетика»

Кафедра № 304 «Вычислительные машины, системы и сети»

Структуры и алгоритмы обработки данных

Отчет по лабораторной работе № 3

«Алгоритмы поиска»

Выполнили студенты группы М3О-210Б-21  
Пысларь А. И.

Осминнов Н. М.

Проверили Чечиков Ю.Б.  
Секретарём В. Е.

**Оглавление.**

[**Цель работы.** 3](#_heading=h.gjdgxs)

[**Блок-схемы функций алгоритма поиска** 4](#_heading=h.30j0zll)

[**Листинг программы.** 7](#_heading=h.1fob9te)

[**Работа программы.** 10](#_heading=h.3znysh7)

[**1 этап работы** 11](#_heading=h.2et92p0)

[**2 этап работы** 12](#_heading=h.tyjcwt)

[**Вывод.** 13](#_heading=h.3dy6vkm)

# **Цель работы.**

Исследование и сравнение алгоритмов сортировки:

* шейкерная сортировка,
* сортировка выбором,
* сортировка вставкой,
* сортировка слиянием,
* быстрая сортировка,
* сортировка Шелла.

**ЗАДАНИЕ**

Для массива из ***n*** элементов выполнить сортировку с помощью двух указанных методов для:

1. заданной произвольным образом последовательности чисел,
2. уже отсортированных последовательностей в возрастающем и убывающем порядке (лучший и худший случаи для выполнения сортировки).

Этапы выполнения ЛР:

1. Промежуточные результаты сортировки представить по каждой итерации для массива размерностью ***n*** = 15 (необходимо выводить на печать весь массив на каждом промежуточном этапе сортировки). Сравнить число **необходимых** **сравнений и число пересылок**.
2. Выполнить сортировку массивов размерности ***n*** = 1000 ( 10000, 100000, 500000 ). Сравнить **время выполнения алгоритмов, число необходимых сравнений и число пересылок**:

**Варианты заданий**

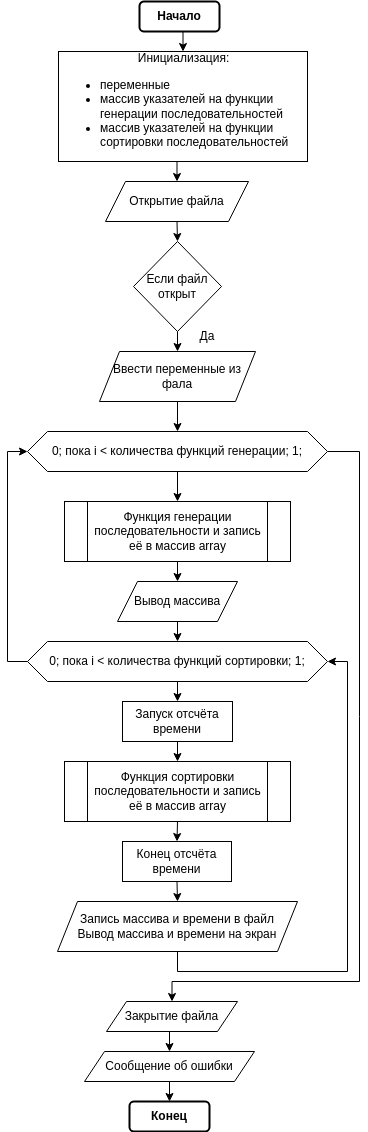
|  |  |
| --- | --- |
| № вар. | метод |
| 6 | вставкой, Шелла |

**Отчет по лабораторной работе должен содержать:**

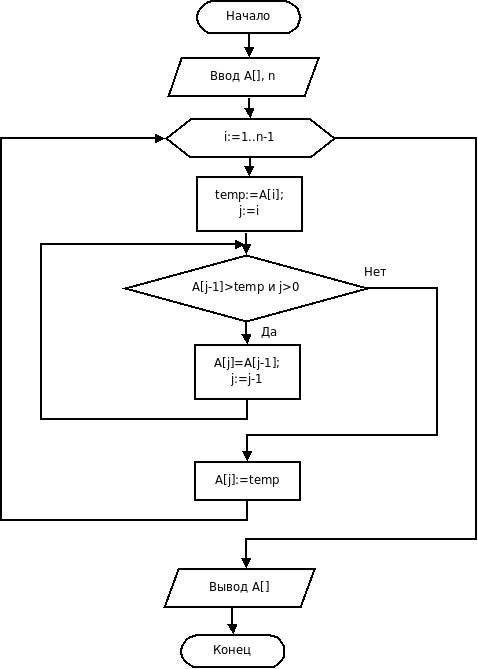
1. задание;
2. структурные схемы главной функции, алгоритмов сортировки варианта работы;
3. тексты программ;
4. результаты работы программ по этапам и результаты сопоставительного анализа – в виде таблиц;
5. выводы по работе, выводы о целесообразности применения методов для различных наборов данных.

# **Блок-схемы функций алгоритма сортировки**

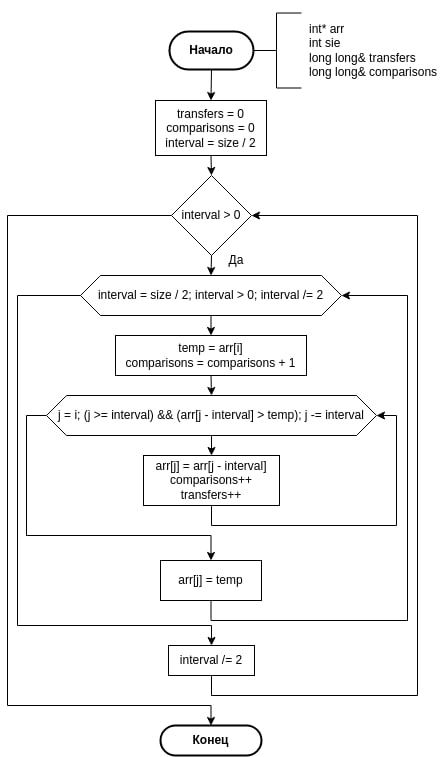
Блок-схема main ()



Блок-схема insertSort ()



Блок-схема shellSort ()



# **Листинг программы.**

**1. main.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Курс ПРОГРАММИРОВАНИЕ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Project type : CMAKE\_CXX\_STANDARD 11 \*

\*Project name : lab-3-sorting \*

\*File name : function.cpp \*

\*Language : CPP, GCC 11.3.0 \*

\*Programmers : Пысларь Александр Игоревич, Осминнов Никита Михайлович М3О-210Б-21 \*

\*Modified By : \*

\*Created : 03.12.2022 \*

\*Last revision: 13.12.2022 \*

\*Comment : Алгоритмы сортировки \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <chrono>

#include <fstream>

#include <iomanip>

#include <string>

#include "functions.h"

using namespace std;

const int FUNCTION\_GENERATION\_COUNT = 3; // Количество генераторов последовательностей

const int FUNCTION\_SORT\_COUNT = 2; // Количество методов сортировки

const char OUT\_PATH[] = "output//"; // Путь для сохранения выходных данных

const char input[] = "Input.txt"; // Название входного файла

const char VerticalLine = (char)186; //"║" - Вертикальная линия

const char HorizontalLine = (char)205; //"═" - Горизонтальная линия

const char TopLeftCorner = (char)201; //"╔" - Левый верхний угол

const char TopRightCorner = (char)187; //"╗" - Правый верхний угол

const char BottomLeftCorner = (char)200; //"╚" - Левый нижний угол

const char BottomRightCorner = (char)188; //"╝" - Правый нижний угол

const char LeftJunction = (char)185; //"╣" - вертикальная "труба влево"

const char RightJunction = (char)204; //"╠" - вертикальная "труба вправо"

const char BottomJunction = (char)203; //"╦" - Горизонтальная вниз "труба"

const char TopJunction = (char)202; //"╩" - Горизонтальная вниз "труба"

const char CrossJunction = (char)206; //"╬" - пересечение линий

const char Space = (char)32;

const int space = 20;

const char\* GENERATORS[] = {

{"allRandom"},

{"upRegularize"},

{"downRegularize"}

};

const char\* SORT\_FUNCTIONS[] = {

{"insertSort"},

{"shellSort"}

};

int main()

{

setlocale(LC\_ALL, "ru");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ПЕРЕМЕННЫЕ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool devKey; // Отладочная печать

int size; // Размер массива

int minValue; // Нижняя граница генерации

int maxValue; // Верхняя граница генерации

long long transfers = 0;

long long comparisons = 0;

int\* array; // Массив с исходными данными

int\* arr; // Сортируемый массив в который мы переносим данные из исходного массива

void\* (\*menuFunctionGeneration[FUNCTION\_GENERATION\_COUNT]) (

int\* arr, // Указатель на наш массив

int size, // Размер массива

int minValue, // Нижняя граница

int maxValue // Верхняя граница

) = {

&allRandom, // Генерация псевдослучайной последовательности чисел

&upRegularize, // Генерация упорядоченной по возрастанию последовательности

&downRegularize // Генерация упорядоченной по убыванию последовательности

};

void\* (\*menuFunctionSort[FUNCTION\_SORT\_COUNT]) (

int\* arr, // Указатель на сортируемый массив

int size, // Размер массива

bool devKey,

long long & transfers,

long long & comparisons

) = {

&insertSort, // Сортировка вставками

&shellSort // Сортировка Шелла

};

ifstream ifs(input);

if (ifs.is\_open()) { //Считываем данные из файла

ifs >> devKey;

ifs >> size;

ifs >> minValue;

ifs >> maxValue;

array = new int[size];

arr = new int[size];

// Проходим по всем вариантам набора данных всеми алгоритмами сортировки

for (int i = 0; i < FUNCTION\_GENERATION\_COUNT; i++) {

cout << " -------------------------------------------" << endl;

menuFunctionGeneration[i](array, size, minValue, maxValue);

if (devKey) {

cout << " " << setw(5) << "Исходный массив:" << endl;

printArray(array, size);

cout << endl;

}

else {

setlocale(LC\_ALL, "C");

cout << TopLeftCorner << setfill(HorizontalLine) << setw(space \* 4)

<< TopRightCorner << endl;

cout << VerticalLine << setfill(Space)

<< setw(space \* 4 - 1) << right << GENERATORS[i] << VerticalLine << endl;

cout << RightJunction << setfill(HorizontalLine)

<< setw(space) << BottomJunction << setw(space)

<< BottomJunction << setw(space)

<< BottomJunction << setw(space)

<< LeftJunction << endl;

cout << VerticalLine << setfill(Space)

<< setw(space - 1) << right << "Function" << VerticalLine

<< setw(space - 1) << right << "time" << VerticalLine

<< setw(space - 1) << right << "counts" << VerticalLine

<< setw(space - 1) << right << "transfers" << VerticalLine << endl;

}

for (int j = 0; j < FUNCTION\_SORT\_COUNT; j++) {

for (int i = 0; i < size; i++) {

arr[i] = array[i];

}

auto begin = chrono::steady\_clock::now(); // Запуск отсчёта

menuFunctionSort[j](arr, size, devKey, transfers, comparisons); //Сам вызов функции

auto end = std::chrono::steady\_clock::now(); // получаем время по окончанию

auto timeOfWork = std::chrono::duration\_cast<std::chrono::microseconds>(end - begin);

ofstream ofs(string(GENERATORS[i]) + '\_' + string(SORT\_FUNCTIONS[j]) + string(".txt"));

ofs << "Исходный массив:" << endl; // Печать в файл

for (int i = 0; i < size; i++) {

ofs << setw(5) << right << array[i];

}

ofs << endl << endl;

ofs << "Отсортированный массив:" << endl;

for (int i = 0; i < size; i++) {

ofs << setw(5) << right << arr[i];

}

ofs << endl;

if (!devKey) {

// ╠══╬══╬══╬══╣

cout << RightJunction << setfill(HorizontalLine)

<< setw(space) << CrossJunction

<< setw(space)

<< CrossJunction << setw(space)

<< CrossJunction << setw(space)

<< LeftJunction << endl;

cout << VerticalLine << setfill(Space)

<< setw(space - 1) << right << SORT\_FUNCTIONS[j] << VerticalLine

<< setw(space - 1) << right << timeOfWork.count() << VerticalLine

<< setw(space - 1) << right << comparisons << VerticalLine

<< setw(space - 1) << right << transfers << VerticalLine << endl;

}

ofs << "Время выполнения сортировки: " << timeOfWork.count() << endl

<< "Сравнений: " << setw(8) << right << comparisons << endl

<< "Перемещений: " << setw(8) << right << transfers << endl;

if (devKey) {

cout << " " << setw(5) << "Отсортированный массив:" << endl;

printArray(arr, size);

}

ofs.close();

}

if (!devKey) {

// ╚══╩══╩══╩══╝

cout << BottomLeftCorner << setfill(HorizontalLine)

<< setw(space) << TopJunction << setw(space)

<< TopJunction << setw(space) << TopJunction

<< setw(space) << BottomRightCorner << endl;

}

}

}

else {

cout << "File not founded." << endl;

}

}

**2. function.h**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Курс ПРОГРАММИРОВАНИЕ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Project type : CMAKE\_CXX\_STANDARD 11 \*

\*Project name : lab-3-sorting \*

\*File name : function.cpp \*

\*Language : CPP, GCC 11.3.0 \*

\*Programmers : Пысларь Александр Игоревич, Осминнов Никита Михайлович М3О-210Б-21 \*

\*Modified By : \*

\*Created : 03.12.2022 \*

\*Last revision: 13.12.2022 \*

\*Comment : Алгоритмы сортировки \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef FUNCTION\_H

#define FUNCTION\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ФУНКЦИИ ВЫВОДА \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printArray( // Вывод в консоль

int\* arr, // Массив

int size // Размер массива

);

void filePrintArray(

int\* arr, // Массив

int size, // Размер массива

char\* f\_name // Название создаваемого файла

);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ГЕНЕРАТОРЫ ПОСЛЕДОВАТЕЛЬНОСТЕЙ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void\* allRandom( // Функция формирующая случайную последовательность

int\* arr, // Массив

int size, // Количество элементов

int minValue, // Ограничение на минимальное число

int maxValue // Ограничение на максимальное число

);

void\* upRegularize( // Функция формирующая упорядоченную последовательность по возрастанию

int\* arr, // Массив

int size, // Количество элементов

int minValue, // Ограничение на минимальное число

int maxValue // Ограничение на максимальное число

);

void\* downRegularize( // Функция формирующая упорядоченную последовательность по убыванию

int\* arr, // Массив

int size, // Количество элементов

int minValue, // Ограничение на минимальное рандомное число

int maxValue // Ограничение на максимальное рандомное число

);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* АЛГОРИТМЫ СОРТИРОВКИ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void\* insertSort( // Сортировка вставками

int\* arr, // Массив элементов

int size, // Размер массива

bool devKey, // Нужен ли промежуточный вывод

long long & transfers, // Количество перемещений

long long & comparisons // Количесвто сравнений

);

void\* shellSort( // Сортировка Шелла

int\* arr, // Массив элементов

int size, // Размер массива

bool devKey, // Нужен ли промежуточный вывод

long long & transfers, // Количество перемещений

long long & comparisons // Количесвто сравнений

);

#endif

**3. function.cpp**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Курс ПРОГРАММИРОВАНИЕ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Project type : CMAKE\_CXX\_STANDARD 11 \*

\*Project name : lab-3-sorting \*

\*File name : function.cpp \*

\*Language : CPP, GCC 11.3.0 \*

\*Programmers : Пысларь Александр Игоревич, Осминнов Никита Михайлович М3О-210Б-21 \*

\*Modified By : \*

\*Created : 03.12.2022 \*

\*Last revision: 13.12.2022 \*

\*Comment : Алгоритмы сортировки \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include "functions.h"

#include <iomanip>

#include <fstream>

#include <math.h>

using namespace std;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ВЫВОД МАССИВА \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void printArray(int\* arr, int size) {

for (int i = 0; i < size; i++) {

cout << setw(5) << right << arr[i];

}

cout << endl;

}

void filePrintArray(int\* arr, int size, char\* f\_name) {

ofstream ofs(f\_name);

for (int i = 0; i < size; i++) {

ofs << setw(5) << right << arr[i];

}

ofs << endl;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ГЕНЕРАТОРЫ ПОСЛЕДОВАТЕЛЬНОСТЕЙ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ЦЕЛОЧИСЛЕННЫЕ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void\* allRandom( // Функция формирующая псевдослучайную последовательность

int\* arr, // Массив

int size, // Количество элементов

int minValue, // Ограничение на минимальное число

int maxValue // Ограничение на максимальное число

) {

srand(time(0));

for (int i = 0; i < size; i++) {

arr[i] = minValue + rand() % (maxValue - minValue);

}

return 0;

}

void\* upRegularize( // Функция формирующая упорядоченную последовательность по возрастанию

int\* arr, // Массив

int size, // Количество элементов

int minValue, // Ограничение на минимальное число

int maxValue // Ограничение на максимальное число

) {

for (int i = 0; i < size; i++) {

arr[i] = i \* (maxValue - minValue) / (size - 1) + minValue;

}

return 0;

}

void\* downRegularize( //функция формирующая упорядоченную последовательность по убыванию

int\* arr, //массив

int size, //количество элементов

int minValue, //ограничение на минимальное рандомное число

int maxValue //ограничение на максимальное рандомное число

)

{

for (int i = 0; i < size; i++) {

arr[i] = i \* (minValue - maxValue) / (size - 1) + maxValue;

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* АЛГОРИТМЫ СОРТИРОВКИ \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void\* insertSort( // Сортировка вставками

int\* arr, // Массив элементов

int size, // Размер массива

bool devKey, // Выводить ли промежуточный результат

long long & transfers, // Число перемещений

long long & comparisons // Число сравнений

) {

transfers = 0;

comparisons = 0;

int temp; // Временная переменная для хранения значения элемента сортируемого массива

int ind; // Индекс предыдущего элемента

for (int i = 1; i < size; i++)

{

temp = arr[i]; // Инициализируем временную переменную текущим значением элемента массива

ind = i - 1; // Запоминаем индекс предыдущего элемента массива

comparisons++; // Считаем сранения

while (ind >= 0 && arr[ind] > temp) // пока индекс не равен 0 и предыдущий элемент массива больше текущего

{

comparisons++; // Считаем сранения

transfers++; // Считаем переносы

arr[ind + 1] = arr[ind]; // Перестановка элементов массива

arr[ind] = temp;

ind--;

}

if (devKey) {

printArray(arr, size);

}

}

return 0;

}

void\* shellSort( // Сортировка Шелла

int\* arr, // Массив элементов

int size, // Размер массива

bool devKey, // Выводить ли промежуточный результат

long long & transfers, // Число перемещений

long long & comparisons // Число сравнений

)

{

transfers = 0;

comparisons = 0;

int interval = (size / 2); // Длина промежутков между элементами

while (interval > 0) // Пока интервал не равен нулю, сортировка продолжается

{

for (int interval = size / 2; interval > 0; interval /= 2)

{

for (int i = interval; i < size; i += 1)

{

int temp = arr[i];

int j;

comparisons++;

for (j = i; (j >= interval) && (arr[j - interval] > temp); j -= interval) {

arr[j] = arr[j - interval];

comparisons++;

transfers++;

}

arr[j] = temp;

}

}

if (devKey) { // Отладочная печать

printArray(arr, size);

}

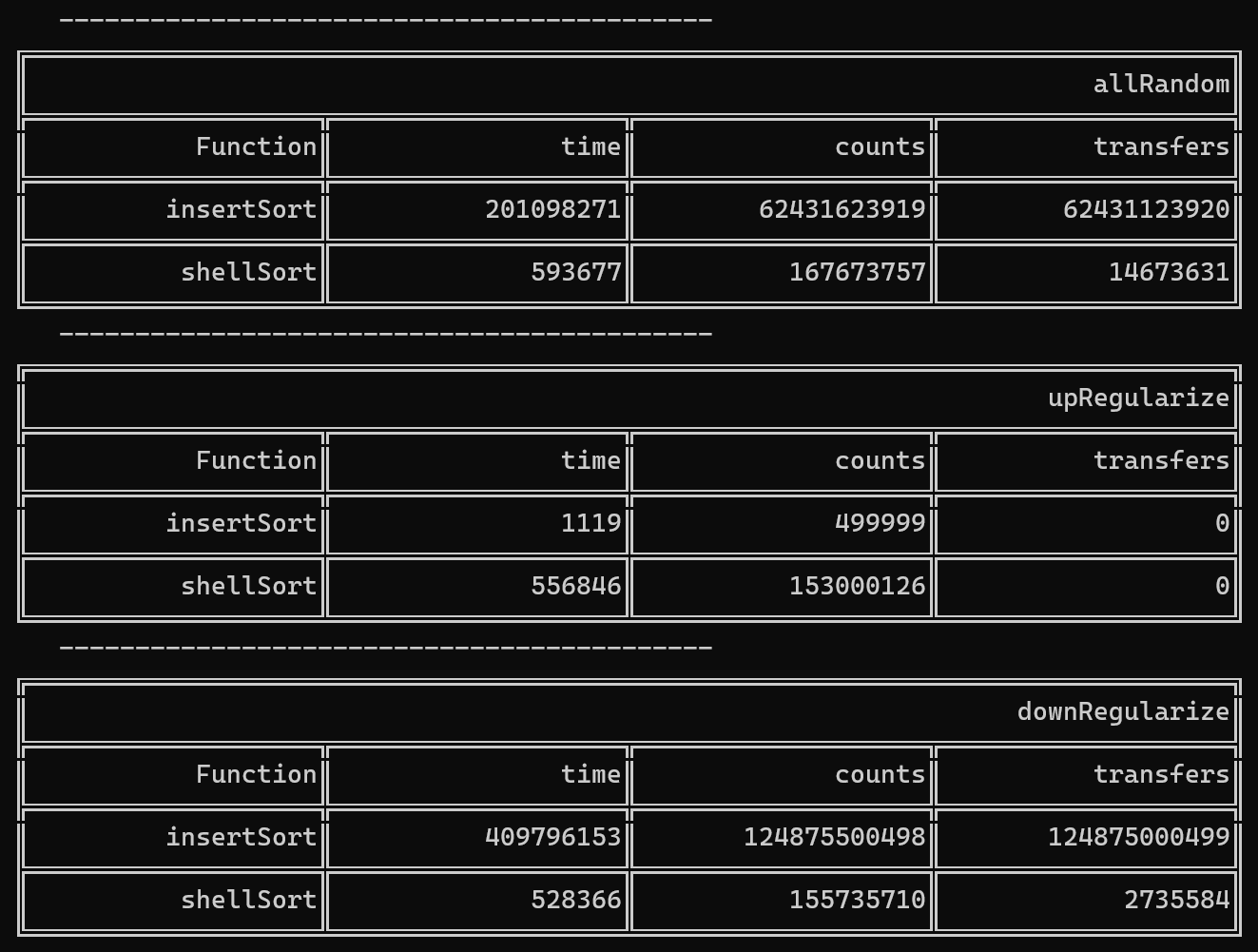
interval /= 2; // Уменьшаем интервал

}

return 0;

}

# **Работа программы.**



# 

# **1 этап работы**

**Количество элементов - 15**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Массив со случайными числами | | Отсортированный массив по возрастанию | | Отсортированный массив по  убыванию | |
|  | Сравнения | Пересылки | Сравнения | Пересылки | Сравнения | Пересылки |
| Сортировка вставкой | 66 | 52 | 14 | 0 | 119 | 105 |
| Сортировка Шелла | 130 | 28 | 102 | 0 | 129 | 27 |

# **2 этап работы**

**Количество элементов** - 1000  
**Сортировка по возрастанию**  
**Время сортировки** (в мкС)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Массив со случайными числами | | | Отсортированный массив по возрастанию | | | Отсортированный массив по  убыванию | | |
|  | Время | Сравнения | Пересылки | Время | Сравнения | Пересылки | Время | Сравнения | Пересылки |
| Сортировка вставками | 859 | 251754 | 250755 | 2 | 999 | 0 | 1727 | 500499 | 499500 |
| Сортировка Шелла | 346 | 78896 | 6842 | 205 | 72054 | 0 | 248 | 76754 | 4700 |

**Количество элементов** - 10000  
**Сортировка по возрастанию**  
**Время сортировки** (в мкС)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Массив со случайными числами | | | Отсортированный массив по возрастанию | | | Отсортированный массив по  убыванию | | |
|  | Время | Сравнения | Пересылки | Время | Сравнения | Пересылки | Время | Сравнения | Пересылки |
| Сортировка вставками | 73000 | 249011197 | 24891198 | 23 | 9999 | 0 | 160397 | 49960008 | 49950009 |
| Сортировка Шелла | 6092 | 1690153 | 130088 | 4623 | 1560065 | 0 | 6188 | 1612657 | 52592 |

**Количество элементов** - 100000  
**Сортировка по возрастанию**  
**Время сортировки** (в мкС)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Массив со случайными числами | | | Отсортированный массив по возрастанию | | | Отсортированный массив по  убыванию | | |
|  | Время | Сравнения | Пересылки | Время | Сравнения | Пересылки | Время | Сравнения | Пересылки |
| Сортировка вставками | 7522828 | 2498357738 | 2498257739 | 218 | 99999 | 0 | 15327894 | 4995100098 | 4995000099 |
| Сортировка Шелла | 89192 | 25934968 | 1934872 | 76982 | 24546096 | 0 | 78417 | 24546096 | 546000 |

**Количество элементов** - 500000  
**Сортировка по возрастанию**  
**Время сортировки** (в мкС)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Массив со случайными числами | | | Отсортированный массив по возрастанию | | | Отсортированный массив по  убыванию | | |
|  | Время | Сравнения | Пересылки | Время | Сравнения | Пересылки | Время | Сравнения | Пересылки |
| Сортировка вставками | 201095271 | 62431623919 | 62431123920 | 1119 | 499999 | 0 | 409796153 | 124875500498 | 124875000499 |
| Сортировка Шелла | 593677 | 167673757 | 14673631 | 556846 | 153000126 | 0 | 528366 | 155735310 | 2735584 |

**Вывод.**

Разработка программы успешно завершена, т.к.:

1. Полученные результаты совпадают с ожидаемыми
2. Изучены основные принципы работы алгоритмов сортировок
3. В 1 этапе работы на малой размерности массива, заданным случайным образом, были протестированы алгоритмы сортировок.   
   Сортировка Шелла имеет худший случай O(n\* n), если неудачны выбраны промежутки разбиения массива.   
   Примерная сложность сортировки вставками O().
4. В 2 этапе работы на большом массиве данных были протестированы алгоритмы сортировок.
5. Итак, в конечном счете можно сказать следующее: при небольшой размерности массива эффективно использовать сортировку вставками, так как её сложность в худшем случае намного меньше. Но на больших размерностях лучше использовать сортировку Шелла (за исключением случая, где последовательность почти упорядочена. В таких сценариях лучше воспользоваться сортировкой вставками)