**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по практической работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Алгоритмы сортировки

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Киреев К.А. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2019

## Цель работы.

Ознакомиться с основными понятиями алгоритмов сортировки на языке программирования C++. Разработать программу, реализующую сортировку Шелла с разными способами задания длин промежутков.

## Задание.

Вариант №15

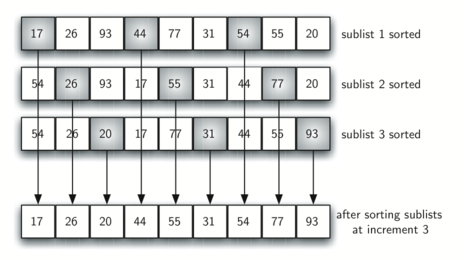
Реализовать сортировку Шелла с графическим интерфейсом.

## Основные теоретические положения.

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии d{\displaystyle d}. После этого процедура повторяется для некоторых меньших значений d{\displaystyle d}, а завершается сортировка Шелла упорядочиванием элементов при {\displaystyle d=1} d = 1(то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

* отсутствие потребности в памяти под стек;
* отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до O(n²), что хуже, чем худшее гарантированное время для сортировки Шелла.

Рисунок 1 - Графическая схема первого шага сортировки Шелла.

## Выполнение работы.

Для решения задачи был разработан графический интерфейс с помощью QtCreator. В интерфейсе реализовано поле для генерации случайного массива для заданного количества элементов, кнопка сортировки массива и кнопка пошаговой сортировки массива, а также список для выбора последовательности.

Для решения задачи используются следующие методы:

* on\_array\_clicked() – генерация массива случайных чисел
* on\_incrementShell\_clicked() – сортировка массива и вывод отсортированного массива в поле after\_sort
* on\_step1\_clicked() - сортировка массива по шагам во всплывающем окне и вывод отсортированного массива в поле after\_sort

Также были реализованы информационные кнопки on\_info1\_clicked() и on\_info2\_clicked() для помощи пользователю.

**Реализация алгоритма сортировки:**

В зависимости от определенной последовательности промежутков производится сортировка вставками с определенным расстоянием. На вход алгоритма подаётся последовательность n чисел: a1, a2,…,an. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности b1, b2,…, bn, чтобы выполнялось следующее соотношение b1 ≤ b2 ≤ b3 ≤ … ≤ bn.

**Выбор длины промежутка:**

Среднее время работы алгоритма зависит от длин промежутков — d, на которых будут находиться сортируемые элементы исходного массива ёмкостью N на каждом шаге алгоритма. Существует несколько подходов к выбору этих значений:

* первоначально используемая Шеллом последовательность длин промежутков: d[1] = N/2, d[i] = d[i-1]/2, d[k] = 1 в худшем случае, сложность алгоритма составит O(n2);
* предложенная Седжвиком последовательность: d[i] = 9\*2^{i} - 9\*2^{i/2} + 1, если i четное и d[i] = 8\*2^{i} - 6\*2^{(i+1)/2} + 1, если i нечетное. При использовании таких приращений средняя сложность алгоритма составляет: O(n7/6), а в худшем случае порядка O(n4/3). При использовании формулы Седжвика следует остановиться на значении inc[s-1], если 3\*inc[s] > size;
* предложенная Хиббардом последовательность: все значения 2^{i} - 1 ≤ N, i ∈ N Такая последовательность шагов приводит к алгоритму сложностью O(n4/3);
* предложенная Праттом последовательность: все значения 2^{i} \* 3^{j} ≤ N/2, i, j ∈ N; В таком случае сложность алгоритма составляет O(N(logN)2) ;

Таблица 1 - Функции для алгоритма сортировки Шелла

|  |  |  |  |
| --- | --- | --- | --- |
| № | Имя | Аргументы | Назначение |
| 1 | **shellSort** | int a[], int size, int sequence | Сортировка вставками с выбранным промежутком |

Окончание таблицы 1

|  |  |  |  |
| --- | --- | --- | --- |
| 2 | **incrementShell** | int inc[], int size | Последовательность Шелла |
| 3 | **incrementSedgewick** | int inc[], int size | Последовательность Седжвика |
| 4 | **incrementHibbard** | int inc[], int size | Последовательность Хиббарда |
| 5 | **incrementPratt** | int inc[], int size | Последовательность Пратта |
| 6 | **reverseArray** | int a[], int b[] | Разворот массива |

**Алгоритм работы:**

*Шаг 1.* Создание массива длин промежутков в зависимости от поданного значения sequence (способы создания массива были описаны ранее). Если сортировка производится по шагам – переход в шагу 2б, иначе к шагу 2а.

*Шаг 2а.* В цикле while производится сортировка вставками для каждой длины промежутка от большего к меньшему.

*Шаг 2б.* В цикле while производится сортировка вставками для каждой длины промежутка от большего к меньшему, но с остановками с помощью QMessageBox, который позволяет остановить функцию до нажатия соответствующей кнопки или закрытия окна.

*Шаг 3.*Отсортированный массив записывается в строку и выводится на экран.

## Оценка эффективности алгоритма.

Существует довольно много последовательностей с разными оценками. Последовательность Шелла – первый элемент равен длине массива, каждый следующий вдвое меньше предыдущего. Асимптотика в худшем случае – O(n2). Последовательность Хиббарда – 2n — 1, асимптотика в худшем случае – O(n1,5), последовательность Седжвика (формула нетривиальна, была приведена выше) — O(n4/3), Пратта (все произведения степеней двойки и тройки) — O(nlog2n).

Анализируя алгоритм Шелла можно прийти к выводу, что время выполнения сортировки не обязательно квадратично, количество сравнений в алгоритме в худшем случае пропорционально N3/2. К этому выводу приходим из многочисленных экспериментов, которые дают основание полагать, что среднее количество сравнений на один шаг может быть равно N1/5, и данное свойство не зависит от входных данных. Таким образом, время исполнения алгоритма равно O(n1,5).

Диаграмма 1 – зависимость op от N при последовательности Шелла

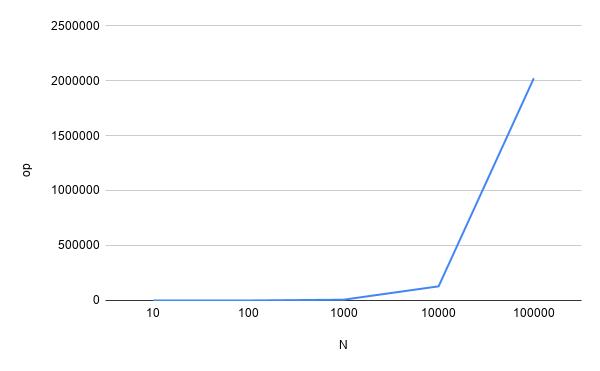
****

Диаграмма 2 – зависимость op от N при последовательности Сэджвика

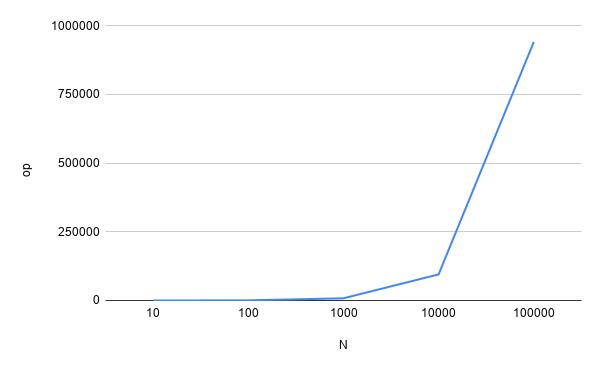
****

Диаграмма 3 – зависимость op от N при последовательности Хиббарда

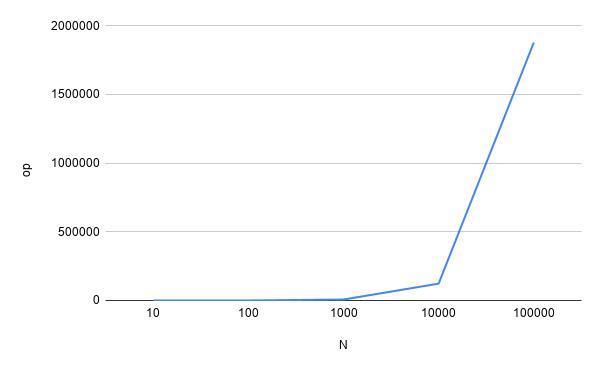
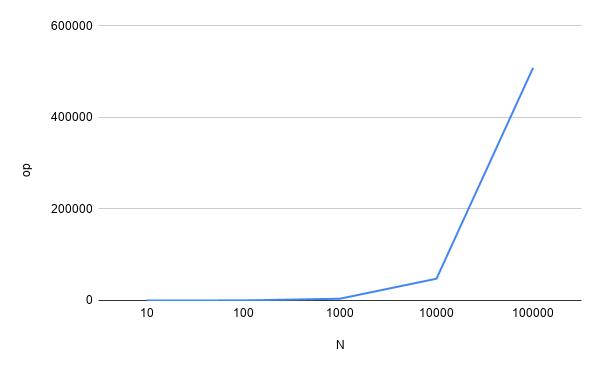
****

Диаграмма 4 – зависимость op от N при последовательности Пратта

****

## Тестирование программы.

|  |  |  |  |
| --- | --- | --- | --- |
| Набор тестовых данных | | Предполагаемые результаты, высчитанные вручную | Результаты выполнения программы |
| № | Данные |
| 1 | 971 331 447 835 324 | 324 331 447 835 971 | 324 331 447 835 971 |
| 2 | 2 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 2 | 1 1 1 1 1 1 1 2 |
| 3 | 6 5 1 9 4 | 1 4 5 6 9 | 1 4 5 6 9 |
| 4 | 0 0 0 | 0 0 0 | 0 0 0 |
| 5 | 444 986 80 608 610 862 902 393 219 351 | 80 219 351 393 444 608 610 862 902 986 | 80 219 351 393 444 608 610 862 902 986 |

## Выводы.

В ходе выполнения лабораторной работы был изучен такой алгоритм сортировки как сортировка Шелла. Была реализована программа на С++, которая анализирует массив и сортирует его в зависимости от определённой последовательности промежутков, а также реализован графический интерфейс.

# Приложение А Исходный код программы

Название файла cinput.cpp

#include <cinput.h>

#include <shellsort.h>

int **cinput**()

{

srand(time(0));

int n, sequence;

cout << "Num of elements: ";

cin >> n;

int arr[n];

if(!n)

{ cout << "!Zero elements" << endl; exit(1); }

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

{ arr[i] = rand()%1000; cout << arr[i] << " "; }

cout << endl << "Gap length sequence: ";

cout << endl << "0 for Shell gap";

cout << endl << "1 for Sedgewick gap";

cout << endl << "2 for Hibbard gap";

cout << endl << "3 for Pratt gap";

cout << endl << "Enter: ";

cin >> sequence;

shellSort(arr, n, sequence);

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

cout << arr[i] << " ";

cout << endl;

return 0;

}

void **shellSort**(int a[], int size, int sequence)

{

int inc, i, j, s, inc\_array[100];

switch(sequence)

{

case 0:

{ s = incrementShell(inc\_array, size); break; }

case 1:

{ s = incrementSedgewick(inc\_array, size); break; }

case 2:

{ s = incrementHibbard(inc\_array, size); break; }

case 3:

{ s = incrementPratt(inc\_array, size); break; }

default:

{ cout << "!Wrong sequence" << endl; exit(1); }

}

while (s >= 0)

{

inc = inc\_array[s--];

for (i = inc; i < size; i++)

{

int temp = a[i];

for (j = i-inc; (j >= 0) && (a[j] > temp); j -= inc)

a[j+inc] = a[j];

a[j+inc] = temp;

}

}

}

int **incrementShell**(int inc[], int size)

{

if(size == 1) inc[0] = 1;

int s = -1;

size /= 2;

while(size > 0)

{ inc[++s] = size; size /= 2; }

reverseArray(inc, inc+s);

return s >= 0 ? s : 0;

}

int **incrementSedgewick**(int inc[], int size)

{

int pf = 1;//pow(2, s)

int ps = 1;//pow(2, s/2)

int pt = 1;//pow(2, (s+1)/2)

int s = -1;

do

{

if (++s % 2)

inc[s] = 8\*pf - 6\*pt + 1;

else

{

inc[s] = 9\*pf - 9\*ps + 1;

ps \*= 2;

pt \*= 2;

}

pf \*= 2;

} while(3\*inc[s] < size);

return s > 0 ? --s : 0;

}

int **incrementHibbard**(int inc[], int size)

{

int s = -1, i = 1;

while(pow(2, i) - 1 <= size)

inc[++s] = pow(2, i++) - 1;

return s--;

}

int **incrementPratt**(int inc[], int size)

{

int pow2, pow3 = 1;

int s = -1;

size /= 2;

while (pow3 <= size)

{

pow2 = pow3;

while (pow2 <= size)

{

inc[++s] = pow2;

pow2 = pow2 \* 2;

}

pow3 = pow3 \* 3;

}

sort(inc, inc+s+1);

return s;

}

void **reverseArray**(int a[], int b[])

{

if (a < b)

{ \*a += \*b; \*b = \*a - \*b; \*a -= \*b;

reverseArray(a + 1, b - 1); }

}

Название файла main.cpp

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

bool flag = false;

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

{

if(!strcmp("-console", argv[i]))

flag = true;

}

if(flag)

cinput();

else

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

return 0;

}

Название файла mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

QPixmap pixmap\_sort(":/resources/img/sort.jpg");

QIcon ButtonIcon(pixmap\_sort);

ui->incrementShell->setIcon(ButtonIcon);

ui->incrementShell->setIconSize(pixmap\_sort.rect().size());

QPixmap pixmap\_array(":/resources/img/array.png");

QIcon ButtonIconArray(pixmap\_array);

ui->array->setIcon(ButtonIconArray);

ui->array->setIconSize(pixmap\_array.rect().size());

QPixmap pixmap\_info(":/resources/img/info.png");

QIcon ButtonIconInfo(pixmap\_info);

ui->info1->setIcon(ButtonIconInfo);

ui->info1->setIconSize(pixmap\_info.rect().size());

ui->info2->setIcon(ButtonIconInfo);

ui->info2->setIconSize(pixmap\_info.rect().size());

QPixmap pixmap\_steps(":/resources/img/steps.png");

QIcon ButtonIconSteps(pixmap\_steps);

ui->step1->setIcon(ButtonIconSteps);

ui->step1->setIconSize(pixmap\_steps.rect().size());

ui->comboBox->addItem("Шелла");

ui->comboBox->addItem("Седжвика");

ui->comboBox->addItem("Хиббарда");

ui->comboBox->addItem("Пратта");

}

MainWindow::~***MainWindow***()

{

delete ui;

}

void MainWindow::**on\_array\_clicked**()

{

srand(time(0));

QString str\_n = ui->lineEdit->text();

int n = str\_n.toInt();

int array[n];

QString arrayStr;

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

{

array[i] = rand()%1000;

arrayStr += QString::number(array[i]) + " ";

}

ui->before\_sort->setText(arrayStr);

}

void MainWindow::**on\_info1\_clicked**()

{

QMessageBox::information(this, "Массив", "Генерация массива");

}

void MainWindow::**on\_incrementShell\_clicked**()

{

QString before\_sort\_str = ui->before\_sort->toPlainText();

QString after\_sort\_str;

ShellSort sort;

int value = ui->comboBox->currentIndex();

switch(value)

{

case(0):

{ after\_sort\_str = sort.Shell(before\_sort\_str, 0, 0); break; }

case(1):

{ after\_sort\_str = sort.Shell(before\_sort\_str, 1, 0); break; }

case(2):

{ after\_sort\_str = sort.Shell(before\_sort\_str, 2, 0); break; }

case(3):

{ after\_sort\_str = sort.Shell(before\_sort\_str, 3, 0); break; }

}

ui->after\_sort->setText(after\_sort\_str);

}

void MainWindow::**on\_info2\_clicked**()

{

int value = ui->comboBox->currentIndex();

switch(value)

{

case(0):

{ QMessageBox::information(this, "Последовательность Шелла", "Первоначально используемая Шеллом последовательность длин промежутков: d[1] = N/2, d[i] = d[i-1]/2, d[k] = 1\nВ худшем случае, сложность алгоритма составит O(N^{2})"); break; }

case(1):

{ QMessageBox::information(this, "Последовательность Седжвика", "Предложенная Седжвиком последовательность: d[i] = 9\*2^{i} - 9\*2^{i/2} + 1, если i четное и d[i] = 8\*2^{i} - 6\*2^{(i+1)/2} + 1, если i нечетное\nПри использовании таких приращений средняя сложность алгоритма составляет: O(N^{7/6}), а в худшем случае порядка O(N^{4/3})"); break; }

case(2):

{ QMessageBox::information(this, "Последовательность Хиббарда", "Предложенная Хиббардом последовательность:\nвсе значения 2^{i} - 1 ≤ N, i ∈ N\nТакая последовательность шагов приводит к алгоритму сложностью O(N^{3/2})"); break; }

case(3):

{ QMessageBox::information(this, "Последовательность Пратта", "Предложенная Праттом последовательность: все значения 2^{i} \* 3^{j} ≤ N/2, i,j ∈ N; В таком случае сложность алгоритма составляет O(N(logN)^{2})"); break; }

default: exit(1);

}

}

void MainWindow::**on\_step1\_clicked**()

{

QString before\_sort\_str = ui->before\_sort->toPlainText();

QString after\_sort\_str;

ShellSort sort;

int value = ui->comboBox->currentIndex();

switch(value)

{

case(0):

{ after\_sort\_str = sort.Shell(before\_sort\_str, 0, 1); break; }

case(1):

{ after\_sort\_str = sort.Shell(before\_sort\_str, 1, 1); break; }

case(2):

{ after\_sort\_str = sort.Shell(before\_sort\_str, 2, 1); break; }

case(3):

{ after\_sort\_str = sort.Shell(before\_sort\_str, 3, 1); break; }

}

ui->after\_sort->setText(after\_sort\_str);

}

Название файла shellsort.cpp

#include "shellsort.h"

ShellSort::**ShellSort**()

{

}

QString ShellSort::**Shell**(QString str\_n, int sequence, int flag){

QStringList array = str\_n.split(" ");

int\* a = new int[array.length()];

for (int i = 0; i < array.length(); ++i)

a[i] = array[i].toInt();

int inc, i, j, s, inc\_array[100];

int size = array.length()-1;

switch(sequence)

{

case 0:

{ s = incrementShell(inc\_array, size); break; }

case 1:

{ s = incrementSedgewick(inc\_array, size); break; }

case 2:

{ s = incrementHibbard(inc\_array, size); break; }

case 3:

{ s = incrementPratt(inc\_array, size); break; }

default:

{ exit(1); }

}

while (s >= 0)

{

inc = inc\_array[s--];

for (i = inc; i < size; i++)

{

int temp = a[i];

for (j = i-inc; (j >= 0) && (a[j] > temp); j -= inc)

a[j+inc] = a[j];

a[j+inc] = temp;

if(flag == 1)

{

QString arrayStr = QString::number(inc) + " : ";

for(int k = 0; k < size; k++)

{

if(a[k] == a[j+inc] && a[j+inc] != a[i])

arrayStr += "[" + QString::number(a[k]) + "]" + " ";

else

arrayStr += QString::number(a[k]) + " ";

}

QMessageBox out;

out.setInformativeText("Ок - шаг алгоритма\nCancel - остановка");

out.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

out.setIcon(QMessageBox::Information);

out.setDefaultButton(QMessageBox::Ok);

out.setText(arrayStr);

out.setStyleSheet("QLabel{ width:250 px; font-size: 14px; } QPushButton{ width:125px; font-size: 18px; }");

int res = out.*exec*();

if (res == QMessageBox::Cancel)

return arrayStr;

}

}

}

QString arrayStr;

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

arrayStr += QString::number(a[i]) + " ";

return arrayStr;

}

int ShellSort::**incrementShell**(int inc[], int size)

{

if(size == 1) inc[0] = 1;

int s = -1;

size /= 2;

while(size > 0)

{ inc[++s] = size; size /= 2; }

reverseArray(inc, inc+s);

return s >= 0 ? s : 0;

}

int ShellSort::**incrementSedgewick**(int inc[], int size)

{

int pf = 1;//pow(2, s)

int ps = 1;//pow(2, s/2)

int pt = 1;//pow(2, (s+1)/2)

int s = -1;

do

{

if (++s % 2)

inc[s] = 8\*pf - 6\*pt + 1;

else

{

inc[s] = 9\*pf - 9\*ps + 1;

ps \*= 2;

pt \*= 2;

}

pf \*= 2;

} while(3\*inc[s] < size);

return s > 0 ? --s : 0;

}

int ShellSort::**incrementHibbard**(int inc[], int size)

{

int s = -1, i = 1;

while(pow(2, i) - 1 <= size)

inc[++s] = int(pow(2, i++) - 1);

return s--;

}

int ShellSort::**incrementPratt**(int inc[], int size)

{

int pow2, pow3 = 1;

int s = -1;

size /= 2;

while (pow3 <= size)

{

pow2 = pow3;

while (pow2 <= size)

{

inc[++s] = pow2;

pow2 = pow2 \* 2;

}

pow3 = pow3 \* 3;

}

std::sort(inc, inc+s+1);

return s;

}

void ShellSort::**reverseArray**(int a[], int b[])

{

if (a < b)

{ \*a += \*b; \*b = \*a - \*b; \*a -= \*b;

reverseArray(a + 1, b - 1); }

}

Название файла cinput.h

#include <shellsort.h>

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <algorithm>

using namespace std;

int **cinput**();

void **shellSort**(int a[], int size, int sequence);

int **incrementShell**(int inc[], int size);

int **incrementSedgewick**(int inc[], int size);

int **incrementHibbard**(int inc[], int size);

int **incrementPratt**(int inc[], int size);

void **reverseArray**(int \*a, int \*b);

Название файла mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QGraphicsEffect>

#include <QGraphicsView>

#include <QFileDialog>

#include <QStandardPaths>

#include <QtGui>

#include <QLabel>

#include <QColorDialog>

#include <QInputDialog>

#include <QMainWindow>

#include <QPushButton>

#include <QMessageBox>

#include <QPixmap>

#include <iostream>

#include <fstream>

#include <shellsort.h>

#include <cinput.h>

using namespace std;

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

private slots:

void on\_array\_clicked();

void on\_incrementShell\_clicked();

void on\_info1\_clicked();

void on\_info2\_clicked();

void on\_step1\_clicked();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

Название файла shellsort.h

#ifndef SHELLSORT\_H

#define SHELLSORT\_H

#include <iostream>

#include <cstdlib>

#include <cmath>

#include <algorithm>

#include <QMessageBox>

class **ShellSort**

{

public:

**ShellSort**();

QString **Shell**(QString str\_n, int sequence, int flag);

private:

int **incrementShell**(int inc[], int size);

int **incrementSedgewick**(int inc[], int size);

int **incrementHibbard**(int inc[], int size);

int **incrementPratt**(int inc[], int size);

void **reverseArray**(int a[], int b[]);

};

#endif // SHELLSORT\_H

Название файла ShellSort.pro

QT += core gui

greaterThan(QT\_MAJOR\_VERSION, 4): QT += widgets

CONFIG += c++11

# The following define makes your compiler emit warnings if you use

# any Qt feature that has been marked deprecated (the exact warnings

# depend on your compiler). Please consult the documentation of the

# deprecated API in order to know how to port your code away from it.

DEFINES += QT\_DEPRECATED\_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.

# In order to do so, uncomment the following line.

# You can also select to disable deprecated APIs only up to a certain version of Qt.

#DEFINES += QT\_DISABLE\_DEPRECATED\_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \

cinput.cpp \

main.cpp \

mainwindow.cpp \

shellsort.cpp

HEADERS += \

cinput.h \

mainwindow.h \

shellsort.h

FORMS += \

mainwindow.ui

# Default rules for deployment.

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /opt/$${TARGET}/bin

!isEmpty(target.path): INSTALLS += target

RESOURCES += \

resourse.qrc