МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной 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. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при 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 $O(n^2)$, что хуже, чем худшее гарантированное время для сортировки Шелла.

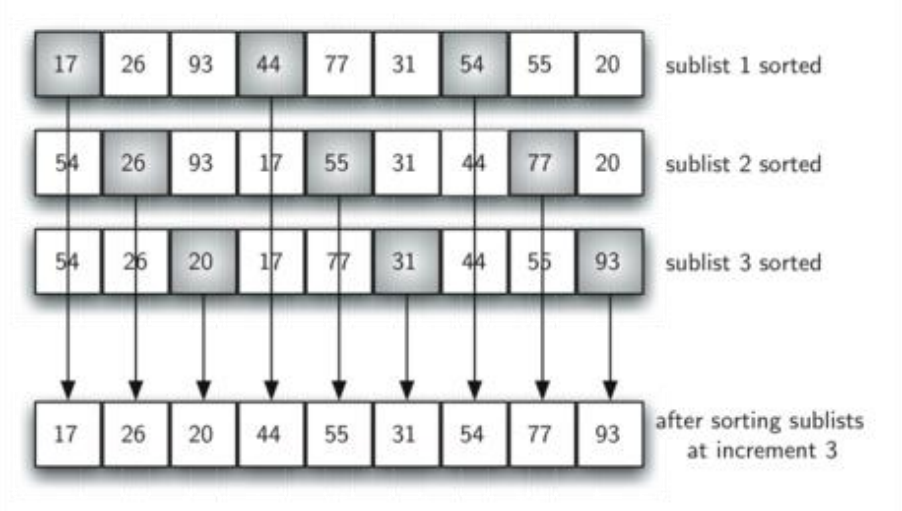

Рис.1.Графическая схема первого шага сортировки Шелла.

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 $b1 \leq b2 \leq b3 \leq \ldots \leq bn$.

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 $O(n^2)$;
- 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: $O(n^{7/6})$, а в худшем случае порядка $O(n^{4/3})$. При использовании формулы Седжвика следует остановиться на значении inc[s-1], если 3*inc[s] > size;
- предложенная Хиббардом последовательность: все значения 2^{i} - $1 \leq N$, $i \in N$ Такая последовательность шагов приводит к алгоритму сложностью $O(n^{4/3})$;
- предложенная Праттом последовательность: все значения 2^{i} * 3^{j} $\leq N/2$, $i, j \in N$; В таком случае сложность алгоритма составляет $O(N(\log N)^2)$;

Таблица 1 - Функции для алгоритма сортировки Шелла

№	Имя	Аргументы	Назначение
1	**shellSort**	int a[], int size, int sequence	Сортировка вставками с выбранным промежутком
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 – $O(n^2)$. Последовательность Хиббарда – $2^n — 1$, асимптотика в худшем случае – $O(n^{1,5})$, последовательность Седжвика (формула нетривиальна, была приведена выше) — $O(n^{4/3})$, Пратта (все произведения степеней двойки и тройки) — $O(n\log^2 n)$.

Анализируя алгоритм Шелла можно прийти к выводу, что время выполнения сортировки не обязательно квадратично, количество сравнений в алгоритме в худшем случае пропорционально $N^{3/2}$. К этому выводу приходим из многочисленных экспериментов, которые дают основание полагать, что среднее количество сравнений на один шаг может быть равно $N^{1/5}$, и данное свойство не зависит от входных данных. Таким образом, время исполнения алгоритма равно $O(n^{1,5})$.

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 C++, 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
```