НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



**ЗВІТ**

**Про виконання лабораторної роботи № 3**

**з дисципліни «Алгоритми і структури даних»**

**Лектор:**

Доцент кафедри ПЗ

Коротєєва Т.О.

**Виконав:**

студ. групи ПЗ-23

Михалевич П.-І.В.

**Прийняла:**

Доцент кафедри ПЗ

Мельник Н.Б.

«\_\_\_» \_\_\_\_\_\_ 2021 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2021

**Тема роботи**: Метод сортування Шелла.

**Мета роботи**: Вивчити алгоритм сортування Шелла. Здійснити програмну реалізацію алгоритму сортування Шелла. Дослідити швидкодію алгоритму.

**Теоретичні відомості**

Сортування Шелла (англійською «Shell Sort») — це алгоритм сортування, що є узагальненням сортування включенням.

Алгоритм базується на двох тезах:

Сортування включенням ефективне для майже впорядкованих масивів.

Сортування включенням неефективне, тому що переміщує елемент тільки на одну позицію за раз.

Тому сортування Шелла виконує декілька впорядкувань включенням, кожен раз порівнюючи і переставляючи елементи, що знаходяться на різній відстані один від одного.

Сортування Шелла не є стабільним.

Сортування Шелла названо начесть автора — Дональда Шелла, який опублікував цей алгоритм у 1959 році. В деяких пізніших друкованих виданнях алгоритм називають сортуванням Шелла-Мацнера, за ім’ям Нортона Мацнера. Але сам Мацнер стверджував: «Мені не довелось нічого робити з цим алгоритмом, і моє ім’я не має пов’язуватись з ним».

На початку обираються m елементів: d1, d2, …, dm, причому d1 > d2 > … > dm = =1.

Потім виконується m впорядкувань методом включення, спочатку для елементів, що стоять через d1, потім для елементів через d2 і так далі до dm = 1.

Ефективність досягається тим, що кожне наступне впорядкування вимагає меншої кількості перестановок, оскільки деякі елементи вже стали на свої місця.

Оскільки dm = 1, то на останньому кроці виконується звичайне впорядкування включенням всього масиву, а отже кінцевий масивгарантовано буде впорядкованим.

Час роботи залежить від вибору значень елементів масиву d. Існує декілька підходів вибору цих значень:

* При виборі , , , …, *dm* = 1 час роботи алгоритму, в найгіршому випадку, становить *O*(*N*2).
* Якщо *d* — впорядкованний за спаданням набір чисел виду ½(3*j* – 1), *j* ∈ ℕ, *di* < *N*, то час роботи є *O*(*N*1.5).
* Якщо d — впорядкованний за спаданням набір чисел виду 2*i*3*j*, *i*,*j* ∈ ℕ, *dk* < *N*, то час роботи є *O*(*N*∙log2*N*).

**Покроковий опис роботи алгоритму**

***Алгоритм SH.***

Задано масив елементів *R1 ,R2 ,…,Rn.* Даний алгоритм реорганізує масив у висхідному порядку, тобто для його елементів буде мати місце співвідношення *Ri < Rj -* для всіх *i,j=1..n. ; d –* інтервал між елементами.

SH1. Цикл за індексом проходження. Повторювати крок S2 при

.

SH2. Виконуємо сортування елементів, що стоять з інтервалом , методом простої вибірки.

SH3. Кінець. Вихід.

**Постановка завдання**

12. Задано одномірний масив дійсних чисел. Впорядкувати елементи, розташовані після максимального елемента в порядку спадання.

**Код програми**

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QMessageBox>

#include <ctime>

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

: QMainWindow(*parent*)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

connect(ui->pushButton\_clear,SIGNAL(clicked()),*this*,SLOT(MySlot()));

connect(ui->pushButton\_init,SIGNAL(clicked()),*this*,SLOT(MySlot()));

connect(ui->pushButton\_start,SIGNAL(clicked()),*this*,SLOT(MySlot()));

}

void MainWindow::**MySlot**(){

QPushButton\* btn = (QPushButton\*)sender();

*if*(btn->text() == "Clear"){

ui->lineEdit\_enter->clear();

ui->textEdit\_steps->clear();

ui->lineEdit\_result->clear();

}

*else* *if*(btn->text() == "Random init"){

ui->lineEdit\_enter->clear();

ui->textEdit\_steps->clear();

ui->lineEdit\_result->clear();

srand(time(NULL));

QString str\_enter = "";

int count = (rand() % 7) + 5;

*for* (int i = 0; i < count; i++) {

double rand\_val = (rand() % 100) / 2.3;

str\_enter += QString::number(rand\_val,'f',1);

*if*(i != count-1){

str\_enter += ", ";

}

}

ui->lineEdit\_enter->setText(str\_enter);

}

*else* *if*(btn->text() == "Start"){

QString text = ui->lineEdit\_enter->text();

int count\_1 = 0;

*if*(!text.isEmpty()){

QStringList valuestr = text.split(',');

count\_1 = valuestr.size();

}

double\* array = *new* double[count\_1];

*if*(!text.isEmpty()){

QStringList valuestr = text.split(',');

*for*(int j = 0; j < count\_1; j++){

*if*(!valuestr[j].isEmpty()){

bool ok;

double value = valuestr[j].toDouble(*&ok*);

*if*(ok){

array[j] = value;

}*else* QMessageBox::warning(NULL,"Warning!","Enter the numbers and comas!");

}

}

}*else* QMessageBox::warning(NULL,"Warning!","Enter the array");

*//* *task*

int max\_Index = 0;

*for* (int i = 0;i< count\_1 ;i++ ) {

*if*(array[max\_Index] < array[i]){

max\_Index = i;

}

}

*//Shell* *sort*

int start\_Index = max\_Index + 1;

int d = (count\_1 - start\_Index) / 2;

QString result = "";

*if*(array){

*for* (int i = 0; i < count\_1; i++ ) {

result += QString::number(array[i],'f',1);

*if*(i != count\_1-1){

result += ", ";

}

}

}

ui->textEdit\_steps->append(result);

*while*(d >= 1){

*for* (int i = start\_Index + d; i < count\_1; i++) {

*for* (int j = i; j >= start\_Index; j -= d ) {

int min\_Index = j;

*for* (int k = j-d; k >= start\_Index; k -= d ) {

*if*(array[k]<array[min\_Index]){

min\_Index = k;

}

}

*if*(min\_Index != j){

double tmp = array[min\_Index];

array[min\_Index] = array[j];

array[j] = tmp;

}

}

}

ui->textEdit\_steps->append("");

ui->textEdit\_steps->append("Step d = "+QString::number(d)+" (Max = "+QString::number(array[max\_Index],'f',1)+") :");

QString result = "";

*if*(array){

*for* (int i = 0; i < count\_1; i++ ) {

result += QString::number(array[i],'f',1);

*if*(i != count\_1-1){

result += ", ";

}

}

}

ui->textEdit\_steps->append(result);

d /= 2;

}

result = "";

*if*(array){

*for* (int i = 0; i < count\_1; i++ ) {

result += QString::number(array[i],'f',1);

*if*(i != count\_1-1){

result += ", ";

}

}

}

ui->lineEdit\_result->setText(result);

*delete* [] array;

}

}

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

{

*delete* ui;

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTextEdit>

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 MySlot();

*private*:

Ui::MainWindow \*ui;

};

#endif *//* *MAINWINDOW\_H*

**main.cpp**

#include "mainwindow.h"

#include <QApplication>

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

{

QApplication a(*argc*, *argv*);

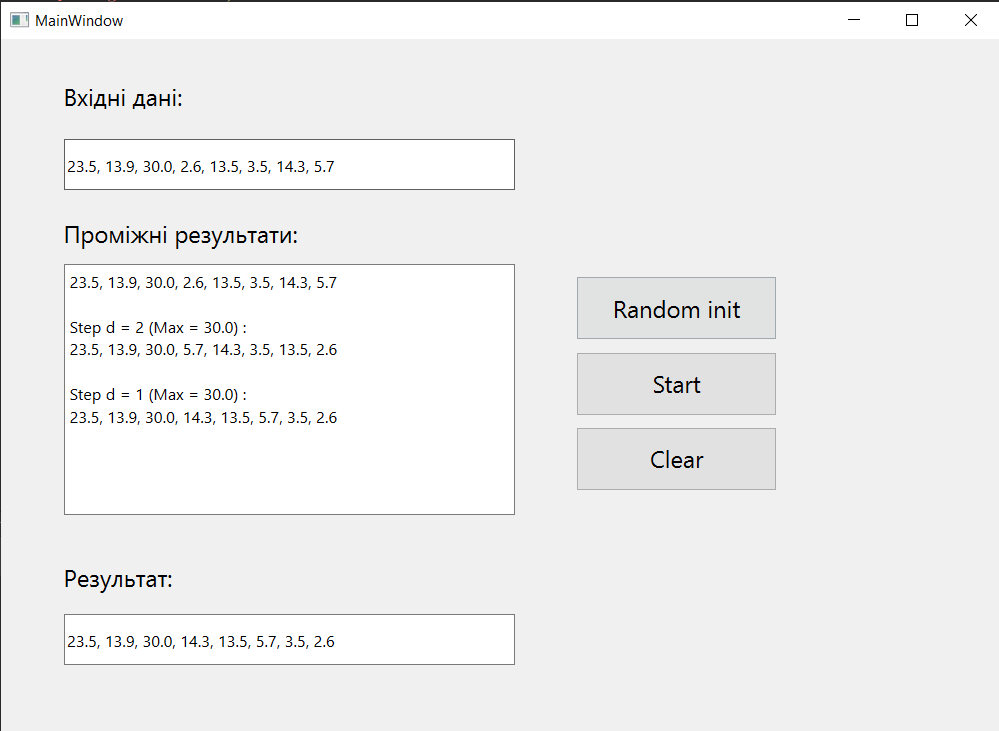
MainWindow w;

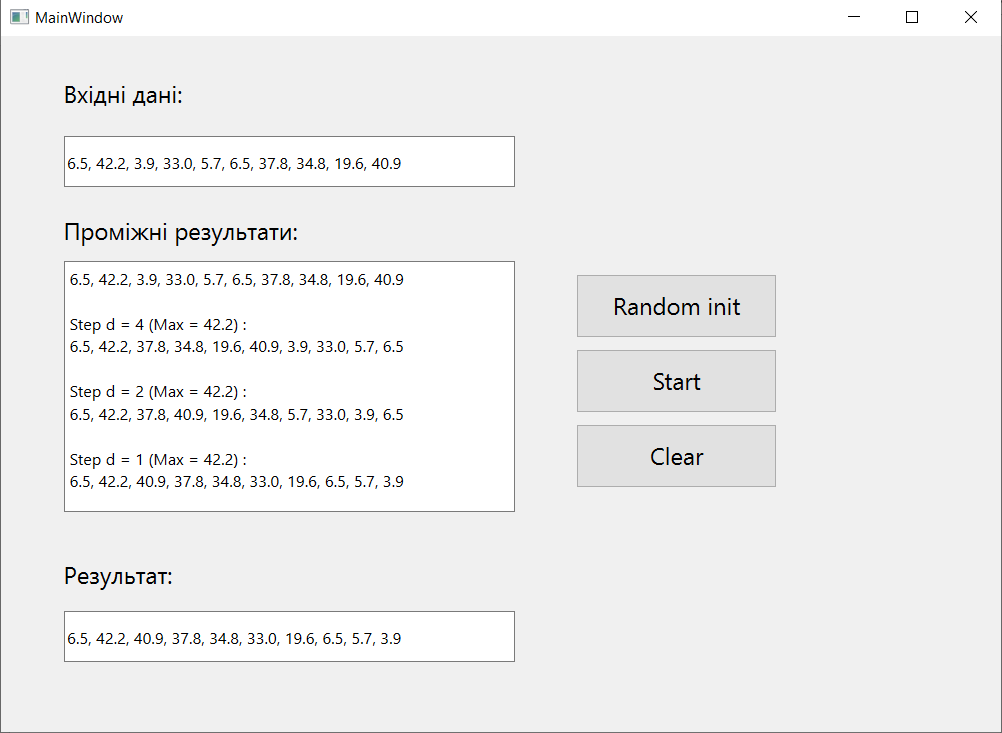
w.show();

*return* a.exec();

}

**Результат програми**





### Висновок

На даній лабораторній роботі я вивчив алгоритм сортування Шела, здійснив програмну реалізацію алгоритму сортування Шела, дослідив швидкодію алгоритму сортування Шела (складність методу О**.**