НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



**ЗВІТ**

**Про виконання лабораторної роботи № 4**

**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 швидкого сортування.

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

**Теоретичні відомості**

Швидке сортування (англійською «Quick Sort») — алгоритм сортування, який не потребує додаткової пам’яті і виконує *O*(*n*∙log(*n*)) операцій порівнянь.

Ідея алгоритму полягає в перестановці елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин вудбувається рекурсивно. Алгоритм швидкого сортування може бути реалізований як на масиві, так і на двонапрямленому списку.

В класичному варіанті алгоритму з вхідного масиву обирався один елемент у якості базового, і весь масив розбивався на дві частини за принципом: в першій частині — ті, що не більші базового елементу, в другій частині — ті, що не менші базового елемента.

Час роботи алгоритму щвидкого сортування залежить від збалансованості, що характеризує розбиття. Збалансованість, у свою чергу залежить від того, який елемент обрано як базовий. Якщо розбиття збалансоване, то асимптотично алгоритм працює так само швидко як і алгоритм сортування злиттям. У найгіршому випадку асимптотична поведінка алгоритму настільки ж погана, як і в алгоритму сортування включенням.

* Найгірше розбиття. Найгірша поведінка має місце у тому випадку, коли процедура, що виконує розбиття, породжує одну підзадачу з (*n* – 1) елементом, а другу — з 0 елементами. Нехай таке незбалансоване розбиття виникає при кожному рекурсивному виклику. Для самого розбиття потрібен час *Θ*(*n*). Тоді рекурентне співвідношення для часу роботи, можна записати наступним чином: *T*(*n*) = *T*(*n* – 1) + *T*(0) + *Θ*(*n*) = *T*(*n* ­– 1) + *Θ*(*n*). Розв’язком такого співвідношення є: *T*(*n*) = *Θ*(*n*2).
* Найкраще розбиття. В найкращому випадку процедура поділу ділить задачу на дві підзадачі, розмір кожної з яких не перевищує (*n* / 2). Час роботи описується нерівністю: *T*(*n*) ≤ 2∙*T*(*n* / 2) + *Θ*(*n*). Тоді: *T*(*n*) = *O*(*n*∙log(*n*)) — асимптотично найкращий час.
* Середній випадок. Математичне очікування часу роботи алгоритму на всіх можливих вхідних масивах є O(n∙log(n)), тобто середній випадок ближчий до найкращого.

**Покроковий опис роботи алгоритму**

***Алгоритм QS.***

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. ; pi – елемент, що поставлений на своє місце; low – індекс , з якого починається підмасив; high - індекс , на який закінчується підмасив.*

QS1. low = 0; high = n;

[QS2. Для даних low і high, якщо low < high виконуємо кроки QS3-QS8, інакше повертаємось в батьківстку функцію;

QS3. Вибираєм Rhigh як опорний елемент.

QS4. i = low – 1.

QS5. Цикл за індексом проходження. При j = low..high-1 повторюємо крок QS5.

QS6. Rj < Rhigh ? , то: i++ , Rj ⬄ Ri .

QS7. Rhigh ⬄Ri+1.

QS8. low = low; high = i, QS2 – QS9;

QS9. low = i +2; high = high, QS2-QS9;]

QS10. Кінець. Вихід.

Кроки в квадратних дужках виконуються рекурсивно.

**Постановка завдання**

14. Задано перелік міст. Упорядкувати за алфавітом лише ті міста, довжина назв яких не перевищує 8.

**Код програми**

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <vector>

#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\_start,SIGNAL(clicked()),*this*,SLOT(MySlot()));

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

}

int **partition** (std::string \*arr, int low, int high,QTextEdit\* textEdit, int count)

{

std::string pivot = arr[high];

int i = (low - 1);

*for* (int j = low; j <= high- 1; j++)

{

*if* (arr[j] < pivot)

{

i++;

std::string t = arr[j];

arr[j] = arr[i];

arr[i] = t;

}

}

std::string t = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = t;

textEdit->append("");

textEdit->append("Pivot = " + QString::fromStdString(arr[i+1]));

std::string res = "";

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

res += arr[i];

*if*(i < count-1){

res += ", ";

}

}

textEdit->append(QString::fromStdString(res));

*return* (i + 1);

}

void **quickSort**(std::string \* arr, int low, int high,QTextEdit\* textEdit, int count)

{

*if* (low < high)

{

int pi = partition(*arr*, low, high,*textEdit*, count);

quickSort(*arr*, low, pi - 1,*textEdit*, count);

quickSort(*arr*, pi + 1, high,*textEdit*, count);

}

}

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

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

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

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

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

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

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

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

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

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

std::vector<QString> city = {"Lviv","Kharkow","Vrotslav","Dnipro","Singapur","Istanbul","Shanghay","Paris","Mykolaiv","Madrid","London","Vadodara",

"Monrovia","Sydney","Zanzibar","Uzhorod","Seoul","Dortmund","Barselona","Osaka","Damaskus","Rome","Campinas","Tokyo",

"Belgrade","Mumbai","Mariupol","Dhaka","Sorocaba","Karachi","Kolkata","Santiago","Budapest","Lagos","Lahore","Tangshan",

"Varshava","Jakarta","Bradford","Chicago","Kyiv","Katowice","Kawasaki","Lima","Semarang","Tehran","Wuhan","Kingston",

"Pune","Hannover","Dallas","Florence","Miami","Tashkent","Florida","Hamilton","Atlanta","Contagem","Dalian","Jinan"};

srand(time(NULL));

QString data = "";

int count = rand() % 15 + 7;

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

int index;

*while*(1){

index = rand() % city.size();

*if*(city[index] != "0"){

*break*;

}

}

data += city[index];

city[index] = "0";

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

data += ", ";

}

}

ui->textEdit\_enter->append(data);

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

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

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

std::vector<QString> city;

QString enter\_data = ui->textEdit\_enter->toPlainText();

QStringList enter\_cities = enter\_data.split(',');

enter\_data.clear();

*for* (int i = 0;i < enter\_cities.length(); i++) {

enter\_data += enter\_cities[i];

}

enter\_cities.clear();

enter\_cities = enter\_data.split(" ");

*for* (int i = 0;i < enter\_cities.length(); i++) {

*if*(enter\_cities[i].length() == 8){

city.push\_back(enter\_cities[i]);

}

}

*//quicksort*

std::string city\_arr[city.size()];

*for* (std::size\_t i = 0;i < city.size(); i++) {

city\_arr[i] = city[i].toStdString();

}

quickSort(*city\_arr*,0,city.size()-1,*ui->textEdit\_steps*,city.size());

std::string res = "";

*for* (std::size\_t i = 0;i < city.size(); i++) {

res += city\_arr[i];

*if*(i < city.size()-1){

res += ", ";

}

}

ui->textEdit\_result->append(QString::fromStdString(res));

}

}

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

{

*delete* ui;

}

**mainwindow.h**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

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*);

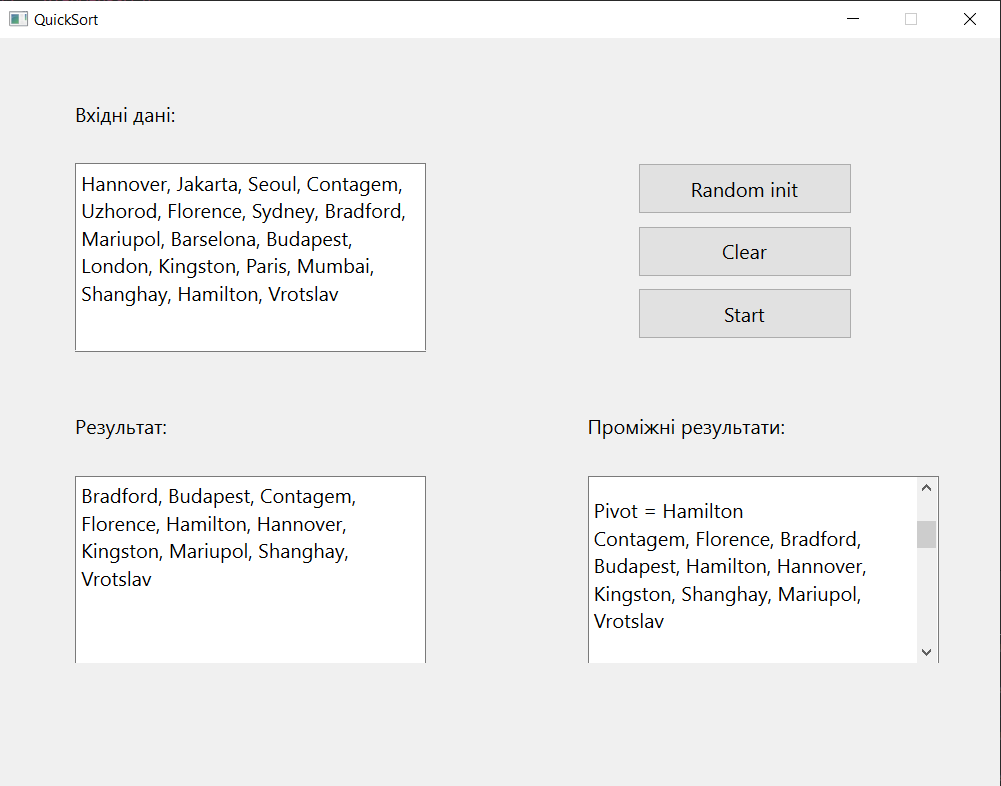
MainWindow w;

w.show();

*return* a.exec();

}

**Результат програми**



### Висновок

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