НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



**ЗВІТ**

**Про виконання лабораторної роботи № 9**

**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**: Бінарний пошук.

**Мета роботи**: Навчитися застосовувати алгоритм бінарного пошуку при розв’язуванні задач та перевірити його ефективність на різних масивах даних. Експериментально визначити складність алгоритму.

**Теоретичні відомості**

Бінарний, або двійковий пошук – алгоритм пошуку елементу у відсортованому масиві. Це класичний алгоритм, ще відомий як метод дихотомії (ділення навпіл).

Якщо елементи масиву впорядковані, задача пошуку суттєво спрощується, тому що ми можемо передбачити розміщення елементу відносно інших. Згадайте, наприклад, як Ви шукаєте слово у словнику. Якщо перша знайдена буква слова стоїть після шуканої у алфавіті, ми гортаємо сторінки назад, якщо до – вперед. Стандартний метод пошуку в упорядкованому масиві – це метод поділу відрізка навпіл, причому відрізком є відрізок індексів 1..n.

Дійсно, нехай масив A впорядкований за зростанням і m (k < m < l) – деякий індекс. Нехай Buffer = A[m]. Тоді якщо Buffer > b, далі елемент необхідно шукати на відрізку k..m-1, а якщо Buffer < b – на відрізку m+1..l. Для цього можна зменшувати умовні межі підмасиву відповідно до того, чи обраний елемент більший чи менший за шуканий. Пошук елементу зупиняється якщо елемент знайдений, поточний індекс виходить за межі заданого або межі умовного масиву співпадають. Для того, щоб збалансувати кількість обчислень в тому і іншому випадку, індекс m необхідно обирати так, щоб довжина відрізків k..m, m..l була (приблизно) рівною.

Описану стратегію пошуку називають бінарним пошуком. b – елемент, місце якого необхідно знайти. Крок бінарного пошуку полягає у порівнянні шуканого елемента з середнім елементом Buffer = A[m] в діапазоні пошуку [k..l]. Алгоритм закінчує роботу при Buffer = b (тоді m – шуканий індекс). Якщо Buffer > b, пошук продовжується ліворуч від m, а якщо Buffer < b – праворуч від m. При l < k пошук закінчується, і елемент не знайдено.

**Постановка завдання**

Використовуючи алгоритм бінарного пошуку, знайдіть елемент *b* у масиві *А* з кількістю елементів від 10 до 100, розташованих за зростанням.

11.   Дано одновимірний масив цілих чисел A[і], де і =10, ,…,20. Знайти позиції елементів масиву, які при діленні на 10 мають остачу 3.

**Покроковий опис алгоритму лінійного пошуку:**

L1. Дано A = {x1,…, xn}. Цикл за індексом проходження i = 1..n. X - елемент пошуку.low – індекс встановлений на 1-му елементі масиву. High – індекс встановлений на останньому елементі масиву. Mid – індекс який вказує на середину між low і high.

L2. mid = low + (high - low + 1) / 2.

L3 Цикл поки low <= high

L4.Якщо A[mid] < X то low = mid + 1

L5. Якщо A[mid] > X то high = mid – 1

L6. Якщо A[mid] = X вивести mid

L7. Кінець. Вихід.

**Код програми**

**mainwindow.cpp**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <ctime>

#include <vector>

#include <algorithm>

#include <QMessageBox>

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()));

}

MainWindow::~MainWindow()

{

*delete* ui;

}

void MainWindow::MySlot()

{

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

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

std::vector<int> vect;

std::vector<int> vectIndexes;

std::vector<int> vectIndexes\_2;

QString text = ui->textEdit->toPlainText();

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

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

int count = valuestr.size();

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

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

bool ok;

int value = valuestr[j].toInt(&ok);

*if*(ok){

vect.push\_back(value);

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

}

}

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

vectIndexes.resize(vect.size());

std::iota(vectIndexes.begin(),vectIndexes.end(),0);

std::vector InputVect = vect;

std::sort(vect.begin(),vect.end(),[](*auto* a, *auto* b){*return* (a % 10) < (b % 10);});

*for* (size\_t i = 0;i < vect.size(); ++i) {

*for*(size\_t j = 0; j < vectIndexes.size(); ++j){

*if*(vect[i] == InputVect[vectIndexes[j]]){

vectIndexes\_2.push\_back(vectIndexes[j]);

vectIndexes.erase(vectIndexes.begin()+j);

j--;

}

}

}

size\_t iter = 0;

size\_t iterComa = 0;

QString line;

*for* (*const* *auto*& a : vect) {

line += QString::number(a);

*if*(iterComa != vect.size()-1){

line += ", ";

}

iter++;

iterComa++;

}

ui->textEdit\_sort->append(line);

int CompNum = 0;

int count = 0;

*while*(!vect.empty()){

bool flag = *false*;

size\_t l = 0;

size\_t r = vect.size();

size\_t mid;

*while*((l <= r) && (flag != *true*) && !vect.empty()){

mid = (l+r) / 2;

*if*( vect[mid] % 10 == 3){

ui->textEdit\_result->append("Element found: index = " + QString::number(vectIndexes\_2[mid]+1) + " ;");

flag = *true*;

vectIndexes\_2.erase(vectIndexes\_2.begin() + mid);

vect.erase(vect.begin() + mid);

count++;

}

*if*( vect[mid] % 10 > 3){

r = mid-1;

}*else* {

l = mid+1;

}

CompNum++;

}

*if*(flag == *false*){

*break*;

}

}

*if*(count == 0){

ui->textEdit\_result->append("Element not found;");

}

ui->lineEdit->setText(QString::number(CompNum));

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

ui->textEdit->clear();

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

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

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

std::vector<int> vect;

ui->textEdit->clear();

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

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

vect.clear();

srand(time(*nullptr*));

*for*(*auto* i = 0; i < ui->spinBox->value(); ++i){

vect.push\_back(rand() % 500);

}

size\_t iter = 0;

size\_t iterComa = 0;

QString line;

*for* (*const* *auto*& a : vect) {

line += QString::number(a);

*if*(iterComa != vect.size()-1){

line += ", ";

}

iter++;

iterComa++;

}

ui->textEdit->append(line);

}

}

**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*);

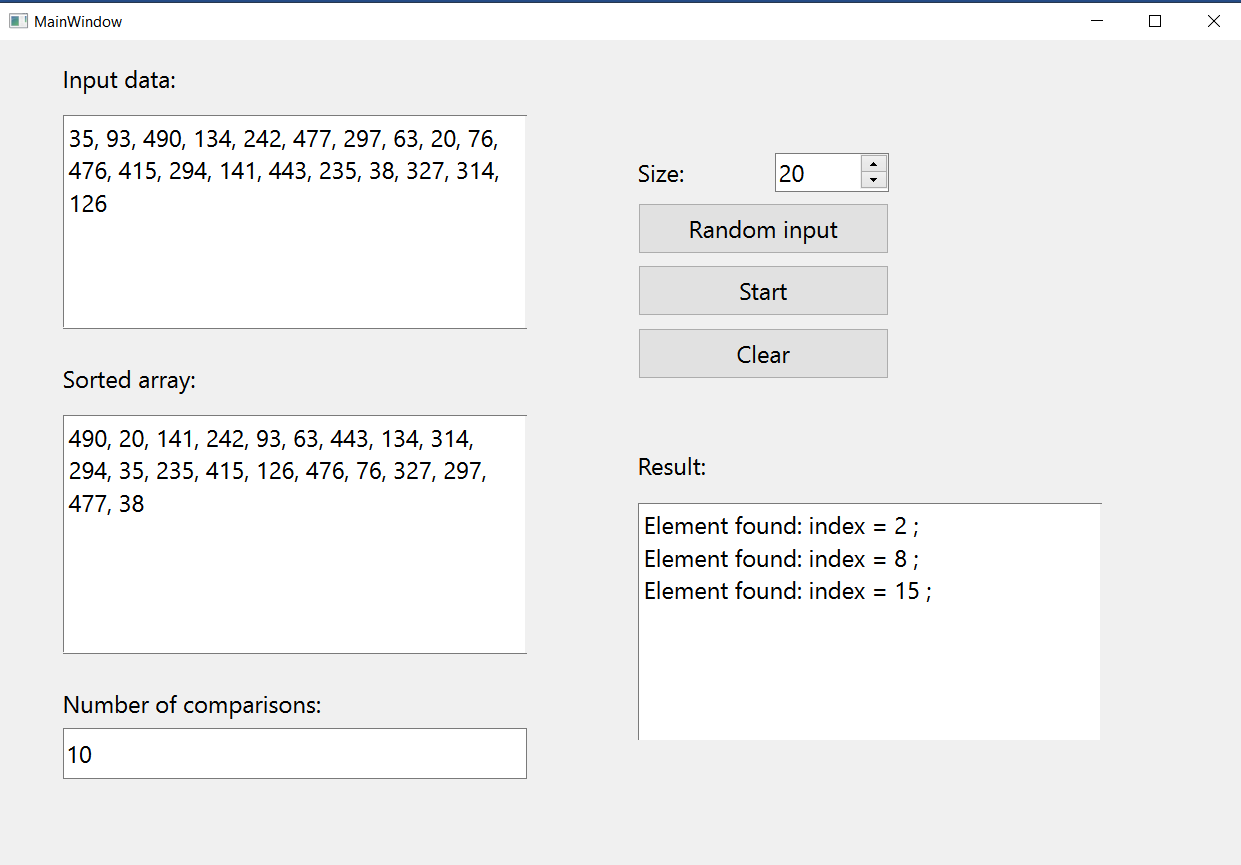
MainWindow w;

w.show();

*return* a.exec();

}

**Результат програми**

****

### Висновок

На цій лабораторній роботі я розробив програму, яка здійснює пошук елемента бінарним методом. Це класичний алгоритм, ще відомий як метод дихотомії. Для пошуку я використовував відсорований одновимірний масив цілих чисел. Складність алгоритму дорівнює *O*(log(*n)*), де *n* — кількість елементів для сортування.