МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

Інститут **ІКНІ**

Кафедра **ПЗ**

ЗВІТ

До лабораторної роботи №1

На тему: «Метод сортування бульбашкою»

З дисципліни: «Алгоритми та структури даних»

Лектор : доцент каф.ПЗ

Коротєєва Т.О.

Виконала: ст.гр.ПЗ-23

Кохман О.В.

Прийняв: аспірант каф.ПЗ

Франко А.В.

«____»_____2022 р.

∑____ _ _ _ _ _ _ _ _ .

Львів – 2022

Тема роботи: Метод сортування бульбашкою.

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

Теоретичні відомості

Алгоритм сортування бульбашкою (англійською «Bubble Sort») відноситься до класу алгоритмів сортування вибіркою. Алгоритм працює наступним чином — у заданому наборі даних (списку чи масиві) порівнюються два сусідні елементи. Якщо один з елементів не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи обмінюються місцями. Прохід по масиву продовжується до тих пір, доки дані не будуть відсортованими. Алгоритм отримав свою назву від того, що процес сортування згідно нього нагадує поведінку бульбашок повітря у резервуарі з водою: найлегша бульбашка піднімається до гори першою. Оскільки для роботи з елементами масиву алгоритм використовує лише порівняння, це сортування на основі порівнянь.
Складність алгоритму у найгіршому випадку дорівнює $O(n^2)$, де n — кількість елементів для сортування. Даний алгоритм має низьку ефективність у випадках, коли n є досить великим, за винятком рідких конкретних випадків, коли заздалегідь відомо, що масив з самого початку буде добре відсортований.
Одним із способів підвищення ефективності алгоритму є використання прапорця перестановок , який дозволяє швидше завершити процес сортування.

Покроковий опис роботи алгоритму сортування бульбашкою:

Алгоритм B:

Алгоритм сортування одновимірного режиму в порядку зростання

Задано одновимірний масив array, size – довжина масиву, first – перший індекс елементу, з якого починається сортування , last – останній індекс елементу, яким закінчується сортування, i – індекс проходження по масиву, temp – тимчасова змінна для свапу елементів.

B1: цикл, який повторюється при size>=0, після кожного проходження size--; Повторювати B2 крок.

B2: цикл за індексом проходження i, який повторюється поки i<=size. Якщо array[i - 1] > array[i], то свапаємо елементи.

В3: Вихід.

Алгорим BF:

Задано одновимірний масив array, size – довжина масиву, first – перший індекс елементу, з якого починається сортування , last – останній індекс елементу, яким закінчується сортування, i – індекс проходження по масиву, temp – тимчасова змінна для свапу елементів, isSorted – змінна для перевірки чи посортований масив.

В1: цикл, який повторюється при size>=0, після кожного проходження size--; Повторювати В2 крок. Перевірка чи isSorted дорівнює true, якщо так , то вийти з циклу.

В2: цикл за індексом проходження i, який повторюється поки i<=size. Якщо array[i - 1] > array[i], то свапаємо елементи і присвоюємо isSorted false.

В3: Вихід.

Індивідуальне завдання

1.Відвідати лекцію, вислухати та зрозуміти пояснення лектора. Прочитати та зрозуміти методичні вказівки, рекомендовані джерела та будь-які інші матеріали, що можуть допомогти при виконанні лабораторної роботи. Відвідати лабораторне заняття, вислухати та зрозуміти рекомендації викладача.

2.Встановити та налаштувати середовище розробки. Для виконання лабораторних робіт може бути використане будь-яке існуюче IDE

3.Написати віконний додаток на мові програмування C або C++. Реалізована програма повинна виконувати наступну послідовність дій:

1) запитуватиме в користувача кількість цілих чотирьохбайтових знакових чисел - елементів масиву , сортування якого буде пізніше здійснено;

2) виділятиме для масиву стільки пам’яті, скільки необхідно для зберігання вказаної кількості елементів, але не більше;

3) ініціалізовуватиме значення елементів масиву за допомогою стандартної послідовності псевдовипадкових чисел;

4) засікатиме час початку сортування масиву з максимально можливою точністю;

5) сортуватиме елементи масиву в неспадному порядку за допомогою алгоритму сортування бульбашкою;

6) засікатиме час закінчення сортування масиву з максимально можливою точністю;

7) здійснюватиме перевірку упорядкованості масиву;

8) повідомлятиме користувачу результат перевірки упорядкованості масиву та загальний час виконання сортування з максимально можливою точністю;

9) звільнятиме усю виділену раніше пам’ять.

9 варіант: Задано одномірний масив дійсних чисел. Впорядкувати елементи, розташовані між першим і останнім від'ємним елементом по зростанню.

4.Оформити звіт про виконання лабораторної роботи.

5.Захистити звіт про виконання лабораторної роботи. Процедура захисту передбачає демонстрацію роботи програми, перевірку оформлення звіту та відповіді на будь-яку кількість будь-яких запитань викладача, що так чи інакше стосуються теми лабораторної роботи.

Код програми

Файл: MyForm.h

```
#pragma once
#include "Sort.h"
#include "SortWithFlag.h"
namespace Project1 {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		MyForm(void)
		{
			InitializeComponent();
		}

	protected:
		~MyForm()
```

```
		{
			if (components)
			{
				delete components;
			}
		}
	private: System::Windows::Forms::Button^ button1;
	protected:
	private: System::Windows::Forms::TextBox^ textBox1;
	private: System::Windows::Forms::RichTextBox^ richTextBox1;
	private: System::Windows::Forms::Label^ label1;
	private: System::Windows::Forms::TextBox^ textBox2;
	private: System::Windows::Forms::TextBox^ textBox3;
	private: System::Windows::Forms::Label^ label2;
	private: System::Windows::Forms::Label^ label3;
	private: System::Windows::Forms::TextBox^ textBox4;
	private: System::Windows::Forms::Label^ label4;
	private: System::Windows::Forms::TextBox^ textBox5;
	private: System::Windows::Forms::Label^ label5;
	private: System::Windows::Forms::RichTextBox^ richTextBox2;
	private: System::Windows::Forms::Label^ label6;
	private: System::Windows::Forms::TextBox^ textBox6;
	private: System::Windows::Forms::Label^ label7;
	private: System::Windows::Forms::TextBox^ textBox7;
	private: System::Windows::Forms::Label^ label8;
	private: System::Windows::Forms::Label^ label9;
	private: System::Windows::Forms::TextBox^ textBox8;
	private: System::Windows::Forms::TextBox^ textBox9;
	private: System::Windows::Forms::TextBox^ textBox10;
	private: System::Windows::Forms::Label^ label10;
	private:
		System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code

// another code //

		int size = 0;
#pragma endregion
	private: System::Void textBox1_TextChanged(System::Object^ sender,
System::EventArgs^ e) {
		size = System::Convert::ToInt16(textBox1->Text);
	}
private: System::Void buttonClick(System::Object^ sender, System::EventArgs^ e) {
	System::Windows::Forms::Button^ button = (Button^)sender;
	Sort* sort = new Sort(size);
	sort->randomNumbers();
	textBox10 << sort;
	sort->firstLast();
	SortWithFlag* sortWithFlag = new SortWithFlag(sort);
	sortWithFlag->firstLast();
		richTextBox1 << sort;
		sort->increaseCount();
		DateTime start = DateTime::Now;
		textBox2->Text = start.ToString("hh.mm.ss.fff tt");
		sort->bubbleSort(richTextBox1);
		DateTime end = DateTime::Now;
		textBox3->Text = end.ToString("hh:mm:ss.fff tt");
		TimeSpan inverval = end - start;
		textBox4->Text = inverval.Seconds.ToString() + "." +
inverval.Milliseconds.ToString();
		sort->isOrdered();
```

```
            textBox5->Text = sort->getIsChecked().ToString();
            richTextBox2 << sortWithFlag;
            sortWithFlag->increaseCount();
            DateTime start2 = DateTime::Now;
            textBox6->Text = start.ToString("hh.mm.ss.fff tt");
            sortWithFlag->bubbleSort(richTextBox2);
            DateTime end2 = DateTime::Now;
            textBox7->Text = end2.ToString("hh:mm:ss.fff tt");
            TimeSpan inverval2 = end2 - start2;
            textBox8->Text = inverval2.Seconds.ToString() + "." +
inverval2.Milliseconds.ToString();
            sortWithFlag->isOrdered();
            textBox9->Text = sortWithFlag->getIsChecked().ToString();
}
};
}
```

Файл: Sort.h

```
#ifndef _SORT_H
#define _SORT_H
#pragma once
#include <iostream>
#include <random>
using namespace std;
class Sort {
public:
	int length;
	double* array;
	int count = 0;
	int first;
	int last;
	bool isChecked = false;
public:
	Sort();
	Sort(int _length);
	~Sort();
	friend void operator<<(System::Windows::Forms::TextBox^ textBox, const
Sort* array);
	friend void operator<<(System::Windows::Forms::RichTextBox^ richTextBox,
const Sort* array);
	void bubbleSort(System::Windows::Forms::RichTextBox^ richTextBox);
	void randomNumbers();
	void increaseCount();
	void isOrdered();
	bool getIsChecked();
	void firstLast();
};
#endif
```

Файл: Sort.cpp

```
#include "Sort.h"
Sort::Sort() : length(0), array(0) {}
Sort::Sort(int _length) : length(_length) {
	array = new double[length];
}
void Sort::firstLast() {
	for (int i = 0; i < length; i++) {
		if (array[i] < 0) {
			first = i;
			break;
```

```
			}
		}
		for (int i = length - 1; i >= 0; i--) {
			if (array[i] < 0) {
				last = i;
				break;
			}
		}
 }
Sort::~Sort() {
		delete[] array;
}
void Sort::increaseCount() {
		count++;
}
void operator<<(System::Windows::Forms::TextBox^ textBox, const Sort* array) {
		for (int i = 0; i < array->length; i++) {
			textBox->Text += array->array[i].ToString("#,0.00") + " ";
		}
}
void operator<<(System::Windows::Forms::RichTextBox^ richTextBox, const Sort*
array) {
		for (int i = 0; i < array->length; i++) {
			richTextBox->Text += array->array[i].ToString("#,0.00") + " ";
		}
		richTextBox->Text += System::Convert::ToString("-------step " + array-
>count + " \n");
}
void Sort::randomNumbers() {
		random_device random_device;
		mt19937 generator(random_device());
		uniform_real_distribution<double> distribution(-200.0 , 200.0);
		for (int i = 0; i < length; i++) {
			array[i] = distribution(generator);
		}
}
void Sort::bubbleSort(System::Windows::Forms::RichTextBox^ richTextBox) {
		if (first == last) {
			richTextBox << this;
			return;
		}
		int size = last;
		double temp = 0;
		while (size >= 0) {
			for (int i = first + 1 ; i <= size; i++) {
				if (array[i - 1] > array[i]) {
					temp = array[i - 1];
					array[i - 1] = array[i];
					array[i] = temp;
				}
			}
			size--;
			richTextBox << this;
			this->increaseCount();
		}
}
void Sort::isOrdered() {
		for (int i = first; i + 1 <= last; i++) {
			if (array[i] > array[i + 1]) {
				isChecked = false;
			}
		}
		isChecked = true;
```

```
}
bool Sort::getIsChecked() {
	return isChecked;
}
```

Файл: SortWithFlag.h

```
#ifndef SORTWITHFLAG_H
#define SORTWITHFLAG_H
#pragma once
#include "Sort.h"
class SortWithFlag : public Sort {
public:
	SortWithFlag(const Sort* sort);
	~SortWithFlag();
	void bubbleSort(System::Windows::Forms::RichTextBox^ richTextBox);
};
#endif
```

Файл: SortWithFlag.cpp

```
#include "SortWithFlag.h"
SortWithFlag::~SortWithFlag()  {
	delete[] array;
};
void SortWithFlag::bubbleSort(System::Windows::Forms::RichTextBox^ richTextBox) {
	int size = last;
	double temp = 0;
	bool isSorted = true;
	while (size >= 0) {
		for (int i = first + 1; i <= size; i++) {
			if (array[i - 1] > array[i]) {
				temp = array[i - 1];
				array[i - 1] = array[i];
				array[i] = temp;
				isSorted = false;
			}
		}
		if (isSorted) {
			break;
		}
		else {
			isSorted = true;
		}
		size--;
		richTextBox << this;
		this->increaseCount();
	}
}
SortWithFlag::SortWithFlag(const Sort* sort) {
	length = sort->length;
	array = new double[length];
	for (int i = 0; i < length; i++) {
		array[i] = sort->array[i];
	}
}
```

Протокол роботи

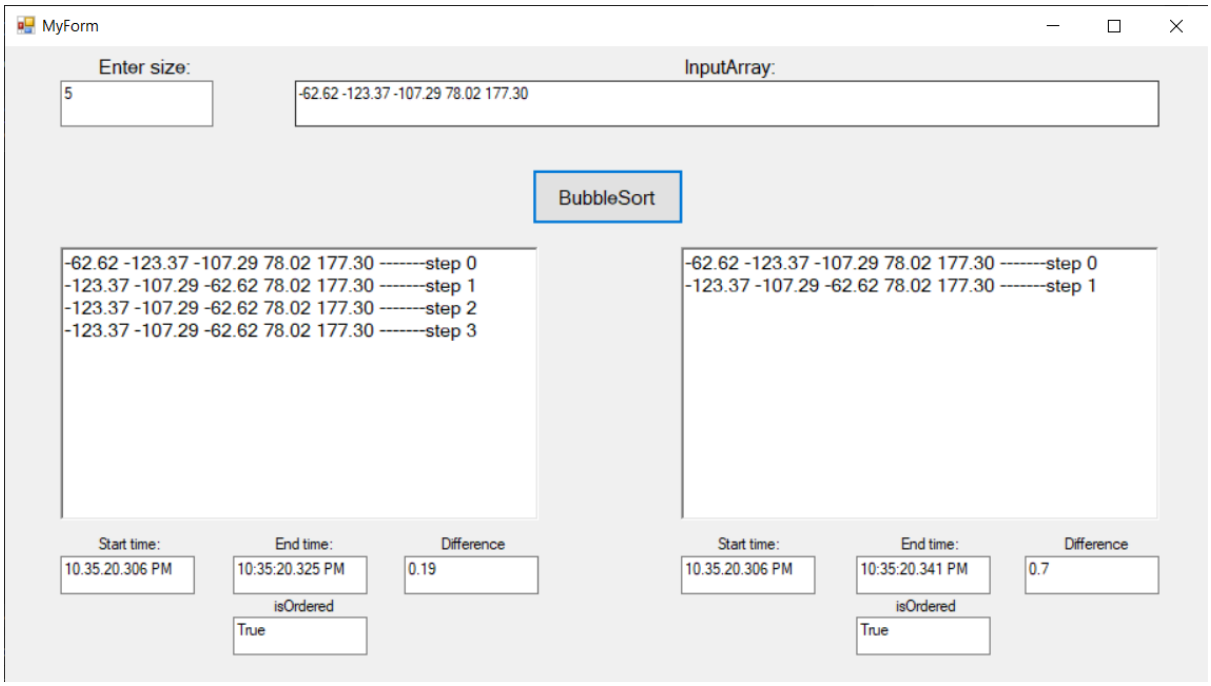

Рис. 1 Форма проекту з виведеним результатом

Висновок

На цій лабораторній роботі я навчилась працювати з алгоритмом сортування бульбашкою та з сортування бульбашкою з прапорцем. Також створила форму, де вивела усі результати та дізналась про алгоритмічну складність алгоритму, що дорівнює у найгіршому випадку O(n^2), у найкращому – O(n), у середньому – O(n^2). При сортуванні з прапорцем кількість ітерацій може змершитись, бо сортування закінчується при вже посортованому масиві, що зменшує кількість ітерацій.