Додаток 1

Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 8 з дисципліни
«Алгоритми та структури даних-1.
Основи алгоритмізації»

«Дослідження алгоритмів пошуку
та сортування»
Варіант 26

Виконав студент ІП-13 Паламарчук Олександр Олександрович
(шифр, прізвище, ім'я, по батькові)

Перевірла Вечерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2021

Лабораторна робота 8

Дослідження алгоритмів пошуку та сортування

Мета – дослідити алгоритми пошуку та сортування, набути практичних навичок використання цих алгоритмів під час складання програмних специфікацій.

Варіант 26

Розробити алгоритм та написати програму, яка складається з наступних дій:

1. Опису змінної індексованого типу (двовимірний масив) згідно з варіантом.
2. Ініціювання змінної, що описана в п.1 даного завдання.
3.Створення нової змінної індексованого типу (одновимірний масив) та її ініціювання значеннями, що обчислюються згідно з варіантом.

26	6 x 5	Дійсний	Із середнього арифметичного від'ємних значень елементів рядків двовимірного масиву. Відсортувати методом бульбашки за спаданням.

◆ Постановка задачі

Ініціювати першу змінну індексованого типу (двовимірний масив) розміру 6x5, що містить дійсні числа, випадковими числами за допомогою функції random(), для цього розробимо функцію *initializeMatrix*(). Ініціювати другу змінну індексованого типу (одновимірний масив) із середнього арифметичного від'ємних значень елементів рядків двовимірного масиву. Для цього розробимо функцію getResArr(). Відсортувати методом бульбашки за спаданням другий масив, для цього розробимо функцію bubbleSort(). Вихідним даним є результуючий масив (другий), що складається з шести дійсних значень.

◆ Побудова математичної моделі

Складемо таблицю змінних

Змінна	Тип	Призначення
matrix[][]	Індексований	Проміжне значення

resArr[]	Індексований	Кінцеве дане
sum	Дійсний	Проміжне значення
counter	Натуральний	Проміжне значення
replacement	Логічний	Проміжне значення
temp	Дійсний	Проміжне значення
resultArray[]	Індексовний	Проміжне значення
resultMatrix[][]	Індексований	Проміжне значення
result[]	Індексований	Проміжне значення
random	Дійсний	Проміжне значення

Складемо таблицю функцій

Назва функції	**Синтаксис**	**Призначення**
Random	*random*()	Генерація випадкового дійсного числа з діапазону [0; 1)

Складемо таблицю операцій

Назва операції	**Синтаксис**	**Призначення**
Довжина масиву	*length*	Повертає довжину масива (значення цілого типу)

matrix[][] - змінна індексованого типу розміру 6x5(згідно з умовою), що містить дійсні числа. ***resArr[] -*** змінна індексованого типу, що мість 6 дійсних чисел. ***resultArray[] -*** змінна індексованого типу, що мість 6 дійсних чисел. ***resultMatrix[][] -*** змінна індексованого типу розміру 6x5, що містить дійсні числа. ***result[] -*** змінна індексованого типу, що мість 6 дійсних чисел. Для вирішення задачі використаємо допоміжні алгоритми (підпрограми), виклики яких мають вигляд: *initializeMatrix*(), *getResArr*(***matrix[][]***), bubbleSort(***matrix[][]***), де ***matrix[][] -*** це формальний параметр функцій.

Термінальна гілка функції *initializeMatrix*() буде мати вигляд "**повернути *resultMatrix[][]***", *getResArr*(***matrix[][]***) "**повернути *resultArray[]***",

bubbleSort(***matrix[][]***) “**повернути *result***”.

Функція *random*() генерує випадкове дійсне число з діапазону [0;1), за допомогою певних математичних дій можемо задати потрібний нам діапазон.

Операція *length* повертає кількість елементів, що містить масив. Приклад застосування ***array***[].*length*.

◆ Розв’язання

Програмні специфікації запишемо у псевдокоді, графічні схемі у вигляді блок-схеми, та у вигляді коду.

1. Основна програма

Крок 1. Визначимо основні дії

Крок 2. Ініціалізація матриці

Крок 3. Сортування масиву

2. Підпрограма *initializeMatrix*()

Крок 1. Визначимо основні дії

Крок 2. Генерація випадкового числа

Крок 3. Заповнення матриці

3. Підпрограма *bubbleSort*(***matrix[][]***)

Крок 1. Визначимо основні дії

Крок 2. Ініціалізація ***result***[]

Крок 3. Ініціалізаця значення ***replacement***

Крок 4. Присвоєння значення false змінній ***replacement***

Крок 5. Перевірка умови сортування

Крок 6. Збільшення ***counter*** на одиницю

4. Підпрограма *getResArr*(***matrix[][]***)

Крок 1. Визначимо основні дії

Крок 2. Перевірка умови для елементів матриці

Крок 3. Перевірка умови для ***counter***

◆ Псевдокод алгоритму основної програми

Крок 3.

Початок

Ввід

matrix[][] = *initializeMatrix*()

resArr[] = *bubblesort*(***matrix***)

Вивід ***resArr***[]

Кінець

◆ Псевдокод алгоритму *initializeMatrix*()

Крок 3

initializeMatrix()

Повторити

Для *i* **від** 0 **до** 6 **крок** 1

Повторити

Для *j* **від** 0 **до** 5 **крок** 1

random **=** *random()* ***** 200 - 100

resultMatrix[*i*][*j*] = ***random***

Все повторити

Все повторити

Повернути ***resultMatrix***[][]**;**

Кінець *initializeMatrix*()

◆ Псевдокод алгоритму *bubbleSort*(***matrix***[][])

Крок 5

bubbleSort(***matrix[][]***)

result[] = *getResArr*(***matrix[][]***)

replacement = true

Повторити

Поки ***replacement***

replacement **=** false

Повторити

Для *i* **від** 0 **до** ***result[]***.*length* - 1 - ***counter,*** **крок** 1

Якщо ***result***[*i*] < ***result***[*i* + 1]

то

temp = ***result***[*i*]

result[*i*] = ***result***[*i* + 1]

r***esult***[*i* + 1] = ***temp***

replacement = true

Інакше

Все якщо

Все повторити

counter = ***counter*** + 1

Все повторити

Повернути *result[]*

Кінець *bubbleSort*(***matrix[][]***)

◆ Псевдокод алгоритму *getResArr*(***matrix[][]***)

Крок 3

getResArr(***matrix[][]***)

Повторити

Для *i* **від** 0 **до** 6 **крок** 1

sum = 0

counter = 0

Повторити

Для *j* **від** 0 **до** 5 **крок** 1

Якщо *matrix*[*i*][*j*] < 0

То

sum += ***matrix***[*i*][*j*]

counter += 1

Інакше

Все якщо

Все повторити

Якщо *counter* == 0

То

counter = 1

Інакше

Все якщо

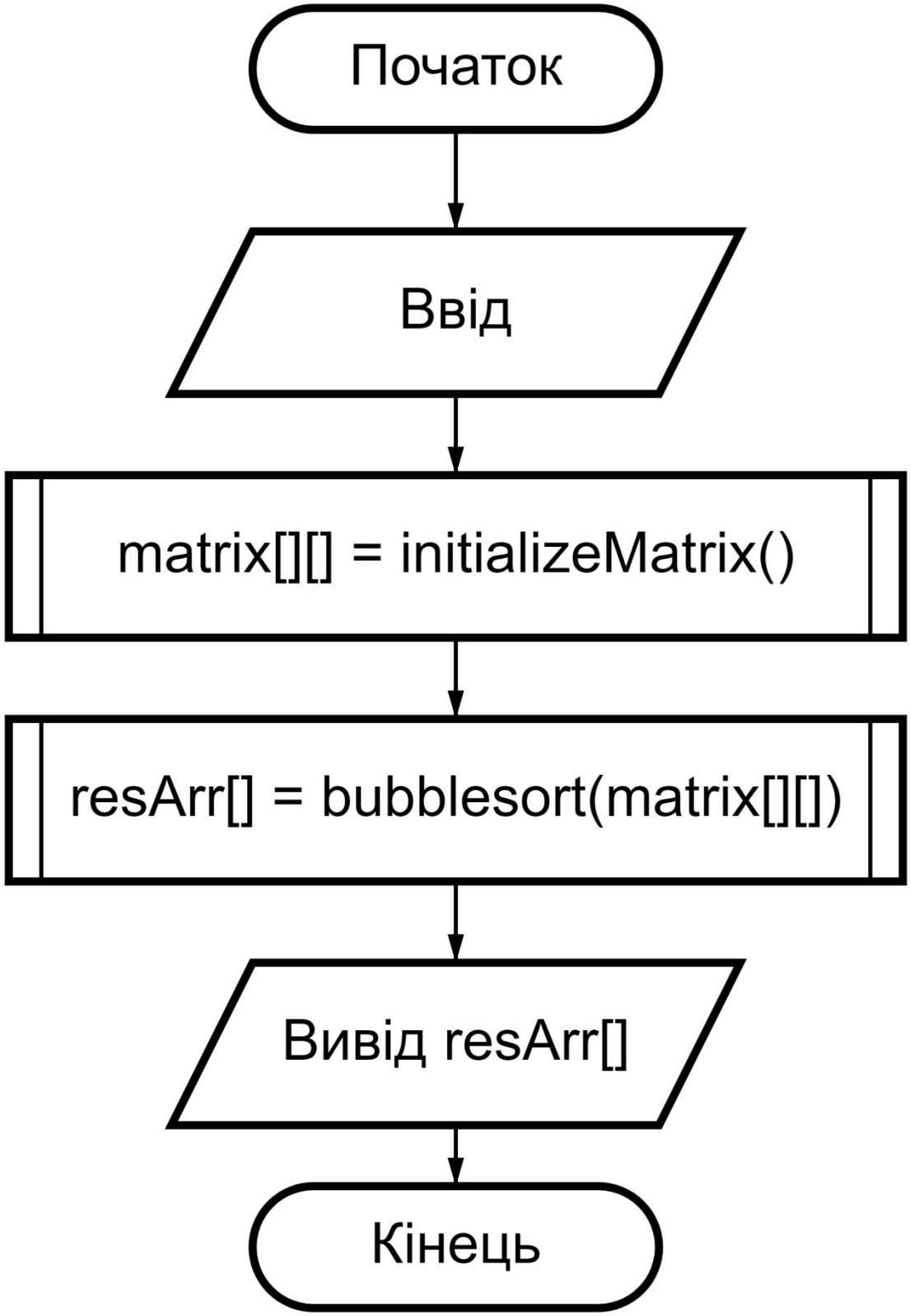
resultArray[*i*] = ***sum*** / ***counter***

Все повторити

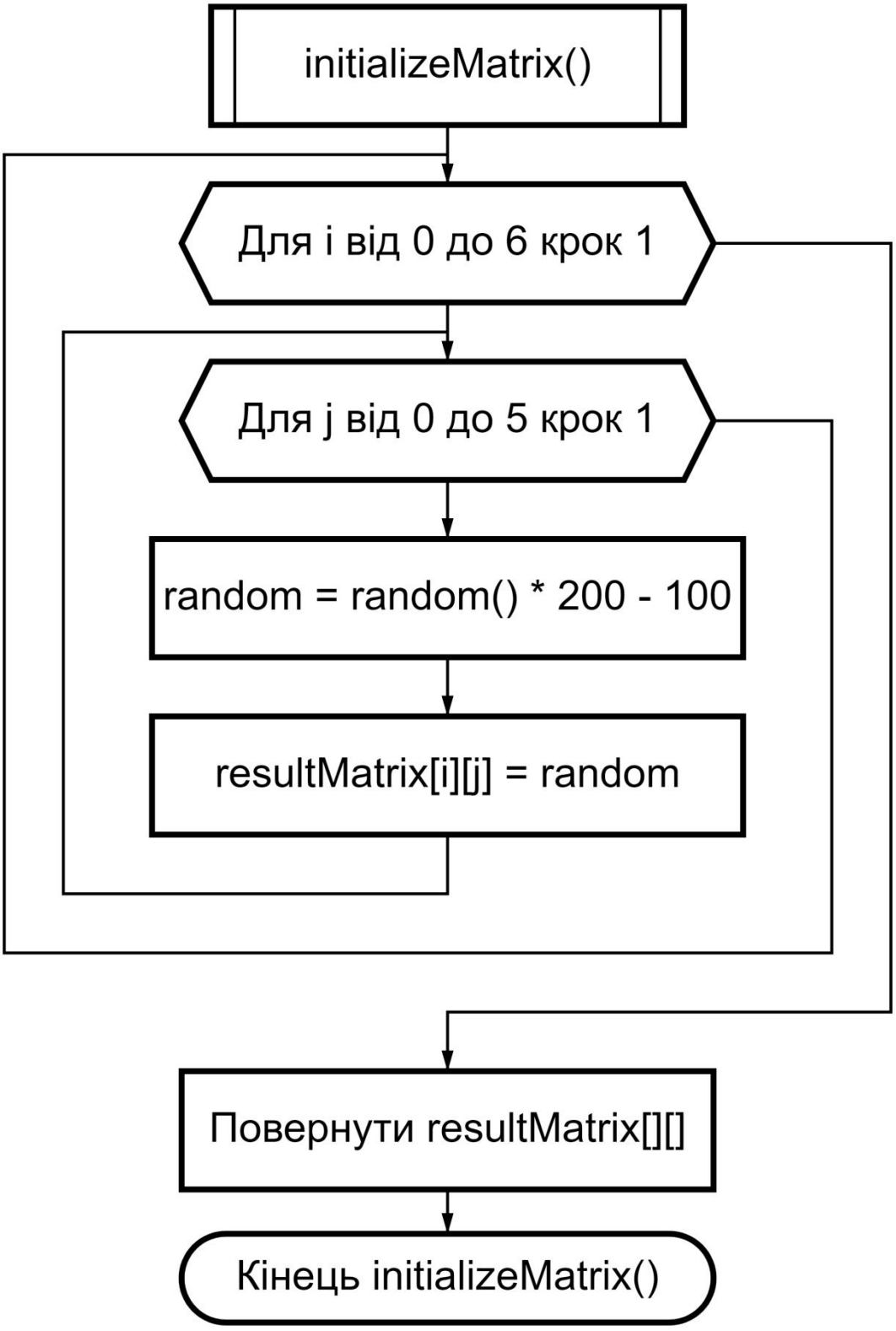
Повернути *resultArray[][]*

Кінець *getResArr*(***matrix[][]***)

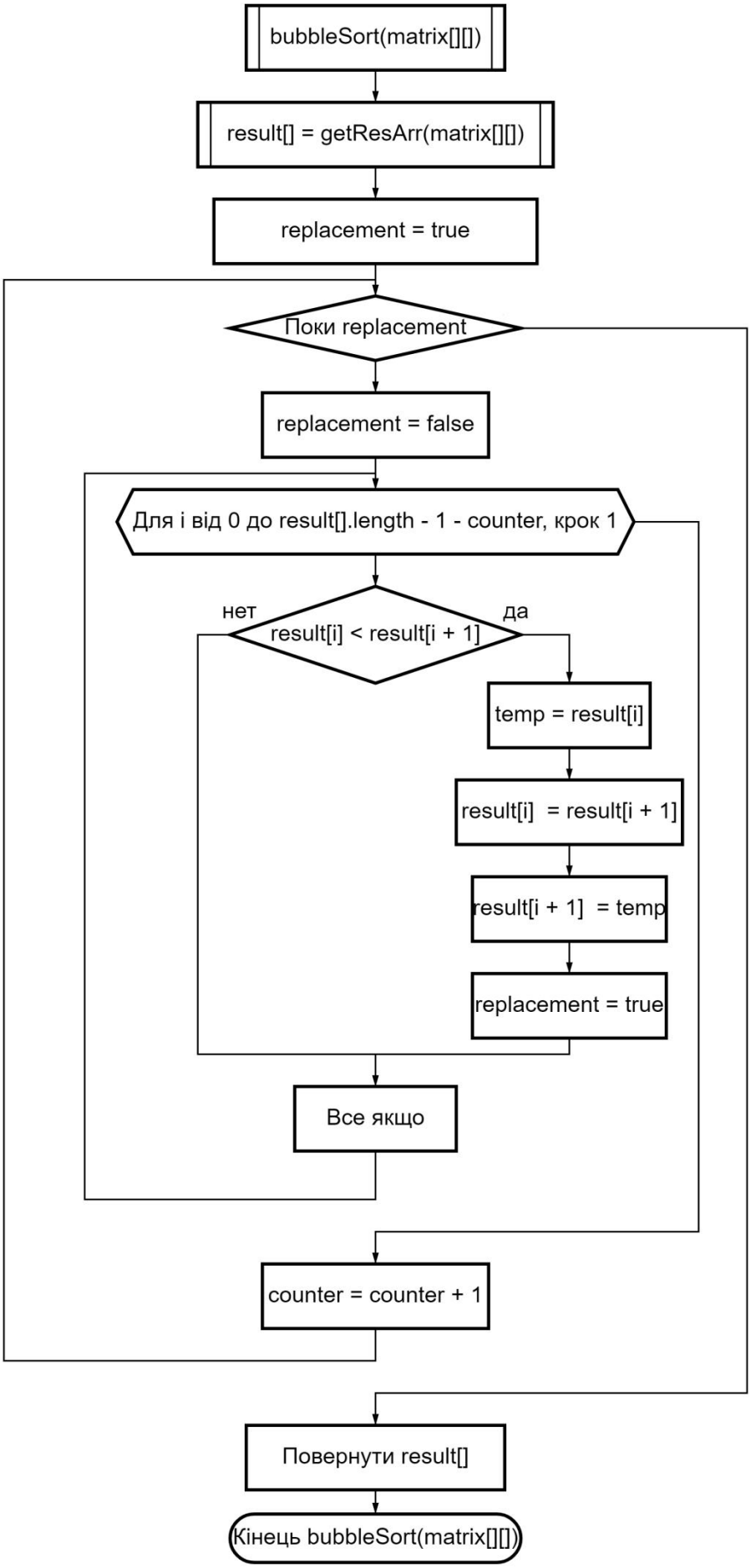
◆ Блок-схема алгоритму основної програми



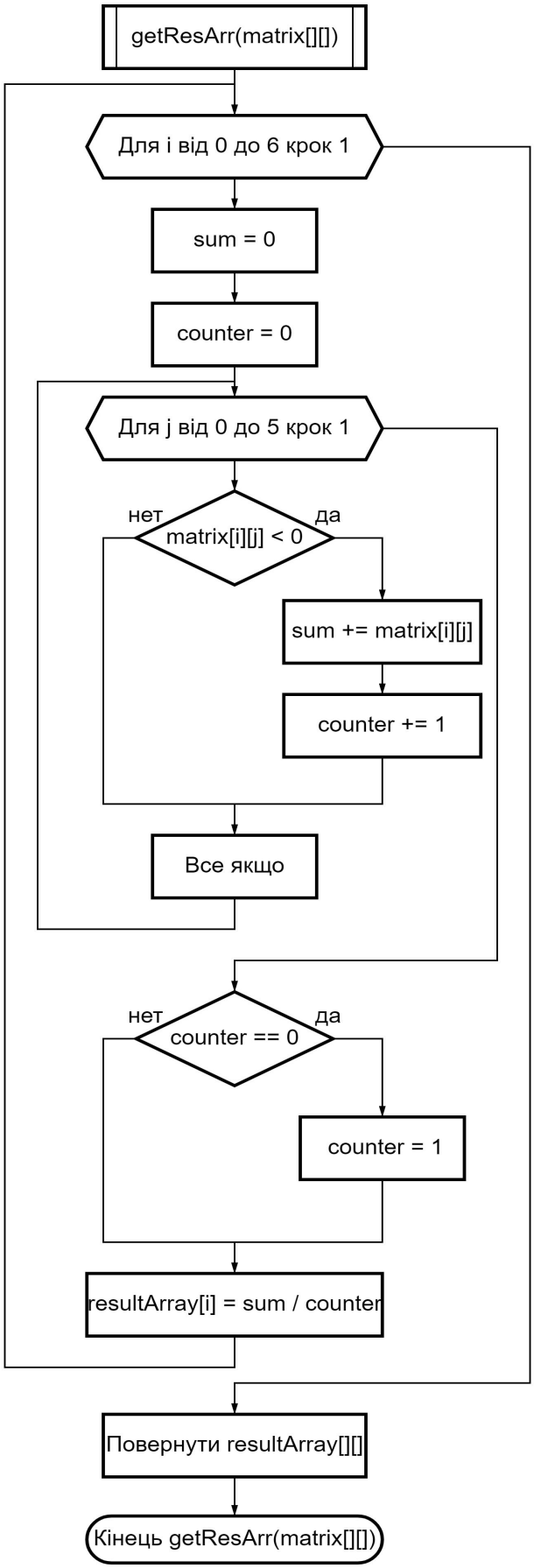
◆ **Блок-схема алгоритму** *initializeMatrix*()



◆ **Блок-схема алгоритму** *bubbleSort*(***matrix***[][])


bubbleSort(matrix[][])
result[] = getResArr(matrix[][])
replacement = true
Поки replacement
replacement = false
Для і від 0 до result[].length - 1 - counter, крок 1
нет
result[i] < result[i + 1]
да
temp = result[i]
result[i] = result[i + 1]
result[i + 1] = temp
replacement = true
Все якщо
counter = counter + 1
Повернути result[]
Кінець bubbleSort(matrix[][])

◆ Блок-схема алгоритму *getResArr*(***matrix[][]***)



◆ **Код програми**

Main.java

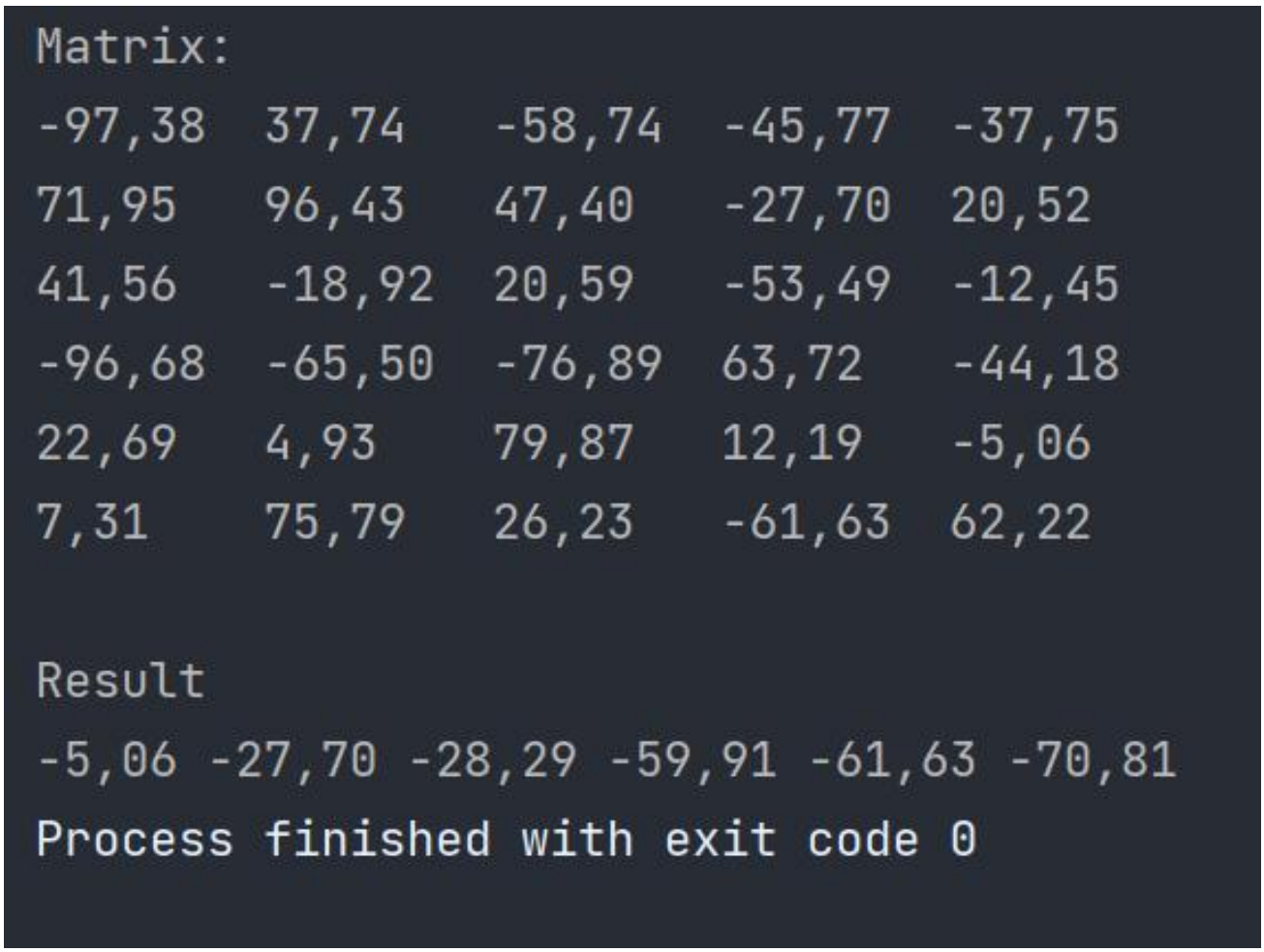
```java
import static java.lang.Math.*;

public class Main {
    public static void main(String[] args) {
        float[][] matrix;
        float[] resArr;
        matrix = initializeMatrix();
        resArr = bubbleSort(matrix);
        display(matrix, resArr);
    }
    private static float[][] initializeMatrix(){
        float[][] resultMatrix = new float[6][5];
        for(int i = 0; i < 6; i++){
            for(int j = 0; j < 5; j++){
                float random = (float)random() * 200 - 100;
                resultMatrix[i][j] = random;
            }
        }
        return resultMatrix;
    }
    private static float[] getResArr(float[][] matrix){
        float[] resultArray = new float[6];
        for(int i = 0; i < 6; i++){
            float sum = 0;
            int counter = 0;
            for(int j = 0; j < 5; j++){
                if(matrix[i][j] < 0){
                    sum += matrix[i][j];
                    counter += 1;
                }
```

```
ava ×
            }
            if(counter == 0) counter = 1;
            resultArray[i] = sum/counter;
        }
        return resultArray;
    }
    private static float[] bubbleSort(float[][] matrix){
        float[] result = getResArr(matrix);
        int counter = 0;
        boolean replacement = true;
        while (replacement) {
            replacement = false;
            for (int i = 0; i < result.length - 1 - counter; i++) {
                if (result[i] < result[i + 1]) {
                    float temp = result[i];
                    result[i] = result[i + 1];
                    result[i + 1] = temp;
                    replacement = true;
                }
            }
            counter++;
        }
        return result;
    }
    private static void display(float[][] matrix, float[] array){
        System.out.println("Matrix: ");
        for(int i = 0; i < 6; i++){
            for(int j = 0; j < 5; j++){
                System.out.printf("%3.2f\t", matrix[i][j]);
            }
```

```java
            for (int i = 0; i < result.length - 1 - counter; i++) {
                if (result[i] < result[i + 1]) {
                    float temp = result[i];
                    result[i] = result[i + 1];
                    result[i + 1] = temp;
                    replacement = true;
                }
            }
            counter++;
        }
        return result;
    }
    private static void display(float[][] matrix, float[] array){
        System.out.println("Matrix: ");
        for(int i = 0; i < 6; i++){
            for(int j = 0; j < 5; j++){
                System.out.printf("%3.2f\t", matrix[i][j]);
            }
            System.out.println();
        }
        System.out.println("\nResult");
        for(double i : array) System.out.printf("%5.2f ",i);
    }
}
```

◆ **Результат роботи програми**



◆ Висновок

На лабораторній роботі було декомпозовано задачу на такі етапи: основна програма, визначимо основні дії, ініціалізація матриці, сортування масиву, підпрограма *initializeMatrix*(), визначимо основні дії, генерація випадкового числа, заповнення матриці, підпрограма *bubbleSort*(***matrix[][]***), визначимо основні дії, ініціалізація ***result***[], Ініціалізаця значення ***replacement,*** присвоєння значення false змінній ***replacement***, перевірка умови сортування, збільшення ***counter*** на одиницю, підпрограма *getResArr*(***matrix[][]***), визначимо основні дії, перевірка умови для елементів матриці, перевірка умови для ***counter***. Було досліджено алгоритми пошуку та сортування та набуто практичних навичок використання цих алгоритмів під час складання програмних специфікацій.