МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
Національний технічний університет України
Факультет інформатики та обчислювальної техніки
«Київський політехнічний інститут імені Ігоря Сікорського»
Кафедра інформаційних систем та технологій

Звіт
з комп'ютерного практикума 1
«Побудова і аналіз алгоритмів»
з дисципліни
«Теорія алгоритмів»

Бригада – 10
Варіант № 4

Перевірила:
ст. вик. Солдатова М. О.

Виконали:
Бойко Катерина, Гоголь Софія,
Павлова Софія, Хіврич Володимир
Студенти гр. ІС-12 , ФІОТ
1 курс

Київ 2022

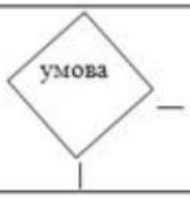
Комп’ютерний практикум 1

Тема: Побудова і аналіз алгоритмів.

Мета роботи: ознайомитись з роботами, що виконує програміст на кожному з етапів розв’язку задачі.

Теоретичні відомості

При створенні блок-схеми алгоритму застосовуються наступні позначення:

Блоки	Назва та призначення
	Початок або кінець алгоритму
	Блок введення даних
	Блок виведення даних на друк
	Арифметичний блок-використовується при обчисленні виразів; процес, присвоєння.
	Логічний блок – використовується для перевірки умови
	Блок модифікації – використовується для зміни в залежності від попередніх значень
	Блок звернення до підпрограм

Завдання

Постановка задачі:

Варіант 4:

В одномірному масиві з n елементів обчислити кількість елементів більше заданого числа.

Вхідні дані: кількість елементів масиву, число, з яким порівнюються елементи, масив чисел.

Вихідні елементи: кількість елементів, що більші за задане число, масив чисел.

Модель:

Основні величини:

Arrsize	Int	Розмір масиву
Arr	Int	Назва масиву
MaxValue	Int	Максимальний елемент, з яким будемо порівнювати
counter	int	Кількість ел-тів, що більше вказаного

Допоміжні величини:

variant	int	Обираємо, як будемо заповнювати масив

Блок-схема:

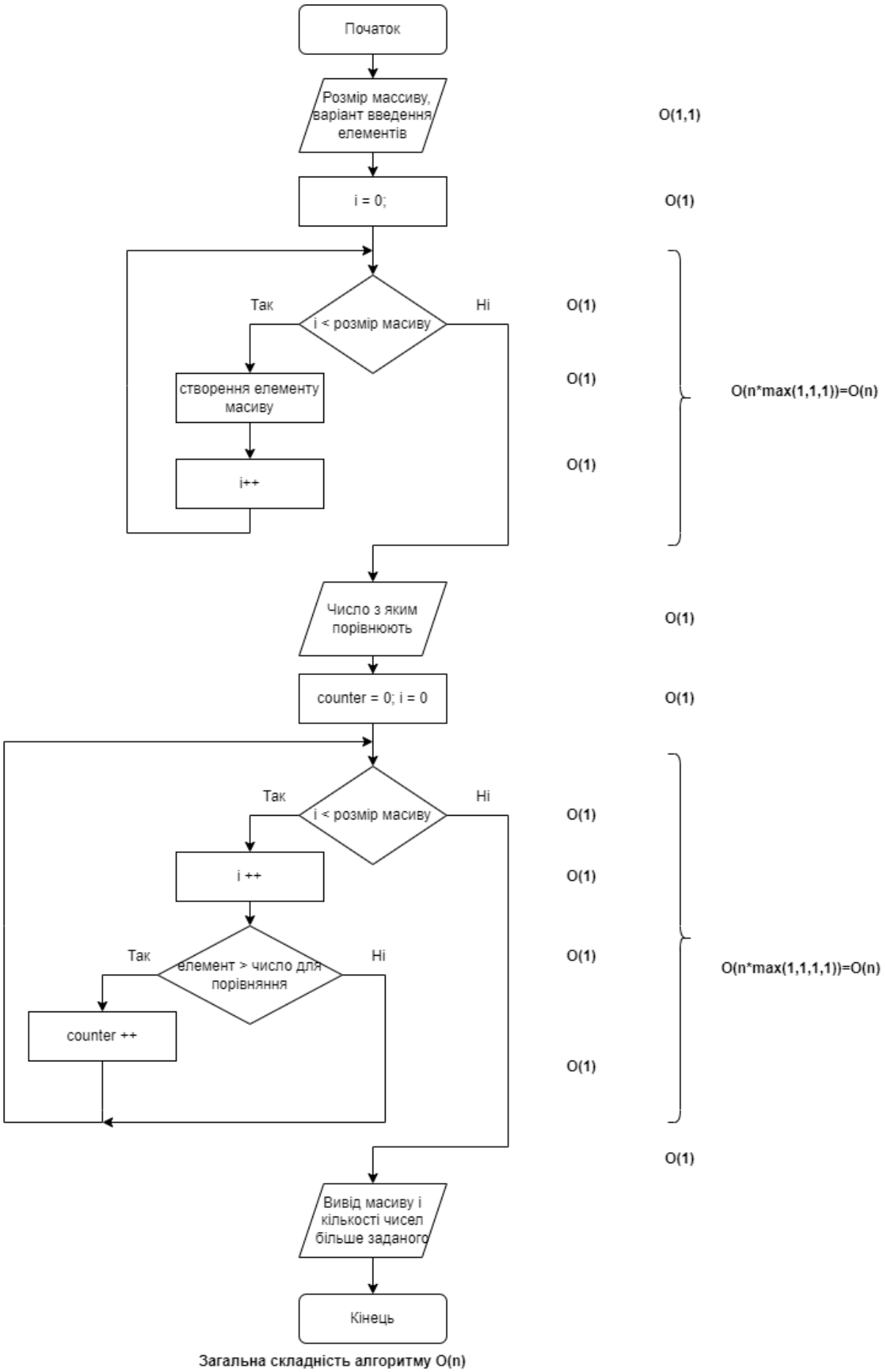


Рис. 1. Блок-схема алгоритму програми

Правильність алгоритму:

1. **Обґрунтування правомірності кожного кроку:**

   Блок 1 відповідає за введення початкових даних – без них алгоритм не працюватиме; блок 2 відповідає за присвоєння значень; блок 3 – відповідає за обрахунок кількості ел-тів більше за дане число із усього масиву; блок 4-6 забезпечують обчислення.

2. **Доведення кінцевості алгоритму:**

   Кінцевість алгоритму залежить тільки від блоків 3, 4, 5 і 6; початкове значення змінної «i» завжди менше довжини масиву, кількість ітерацій циклу завжди дорівнює довжині масиву – 1.

Код:

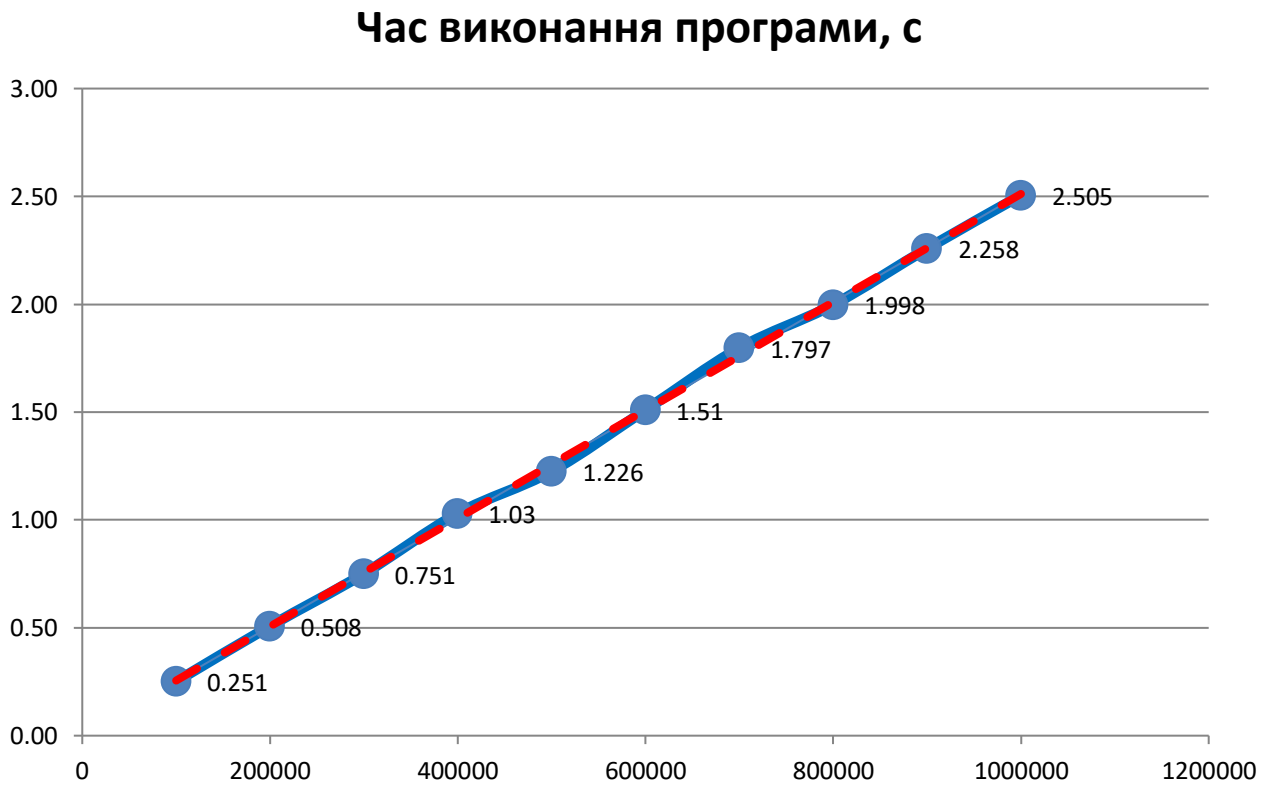
```
#include <iostream>
#include <ctime>
#include <fstream>
#include <Windows.h>
using namespace std;

//Функція генерація масиву
void Generation_Arr(int ArrSize, int Arr[]) {
	srand((time(NULL)));
	int variant;
	cout << "Введіть: \n '1' для автоматичної генерації \n '2' для ввода масиву власноруч \n '3' для вводу/виводу з файла " << endl;
	cin >> variant;
	switch (variant)
	{
	case 1:
		for (int i = 0; i < ArrSize; i++)
		{
			Arr[i] = rand() % 100;
		}
		break;
	case 2:
		for (int i = 0; i < ArrSize; i++)
		{
			cout << "a[" << i + 1 << "] = ";
			cin >> Arr[i];
			if (Arr[i] <= 0 || (Arr[i] - int(Arr[i]) != 0))
```

```
                {
                        cout << "* -------- * ПОМИЛКА * -------- * \n";
                        cout << "Введіть натуральне число!";
                        exit(0);
                }
        }
        break;
    case 3:
    {
        ifstream file_In("input.txt");
        if (!file_In)
        {
                cout << " * -------- * ПОМИЛКА * -------- * \n";
                cout << "Не знайдено файлу \"input.txt\"! ";
                exit(0);
        }
        for (int i = 0; i < ArrSize; i++)
        {
                file_In >> Arr[i];
        }
        file_In.close();
        break;
    }
    default:
        cout << "* -------- * ПОМИЛКА * -------- * \n";
        cout << "Введіть (1/2/3)!";
        exit(0);
    }
}

//Функція виводу масиву
void Output_Arr(int ArrSize, int Arr[]) {
    cout << "*--------* МАТРИЦЯ *--------*" << "\n";
    for (int i = 0; i < ArrSize; i++)
    {
        cout << Arr[i] << "\t";
    }
    cout << "\n\n";

    //Реалізація виводу масиву у файл
    ofstream file_Out;
    file_Out.open("output.txt");

    file_Out << "*--------* МАТРИЦЯ *--------*" << "\n";
    for (int i = 0; i < ArrSize; i++)
    {
        file_Out << Arr[i] << "\t";
```

```
	}
	file_Out << "\n";
	file_Out.close();
}

//Функція пошуку вказаних елементів
void Search_minValue(int ArrSize, int Arr[], int MaxValue) {
	int min = 100, counter = 0;

	for (int i = 0; i < ArrSize; i++)
	{
		if (Arr[i] > MaxValue)
		{
			counter++;
		}
	}
	cout << "*----------------------------*" << "\n";
	cout << "Кількість елементів, що більше " << MaxValue << " = " << counter <<
endl;
	cout << "\n";

	//Реалізація виводу результату у файл
	ofstream file_Out;
	file_Out.open("output.txt", ios::app);
	file_Out << "Кількість елементів, що більше " << MaxValue << " = " << counter <<
endl;
	file_Out.close();
}

int main() {
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	cout << "\t * -------- * Комп'ютерний практикум 1 * -------- *\n";
	cout << "\t\t\t Бригада 10. Варіант 4  \n";
	cout << "\t\t\t Реалізація алгоритму \n";

	int ArrSize, MaxValue;

	cout << "\n\nВведіть розмір масиву: ";
	cin >> ArrSize;

	int* Arr = new int[ArrSize]();

	//Обмеження на вхідні дані
	if (ArrSize <= 0 || (ArrSize - int(ArrSize) != 0))
	{
```

```
            cout << " * -------- * ПОМИЛКА * -------- * \n";
            cout << "Введіть натуральне число! ";
            exit(0);
        }
        Generation_Arr(ArrSize, Arr);
        Output_Arr(ArrSize, Arr);

        cout << "Введіть число для порівняння: ";
        cin >> MaxValue;

        //Обмеження на вхідні дані
        if (MaxValue <= 0 || (MaxValue - int(MaxValue) != 0))
        {
            cout << "* -------- * ПОМИЛКА * -------- * \n";
            cout << "Введіть натуральне число!";
            exit(0);
        }
        Search_minValue(ArrSize, Arr, MaxValue);
}
```

Перевірка обчислюваної складності програми:



Діагр. 1. Діаграма часу виконання програми

Червоний пунктирний графік – теоретична складність

Синій графік – реальна складність

Перевірка обчислювальної складності програми підтвердила аналітичну оцінку O(n).

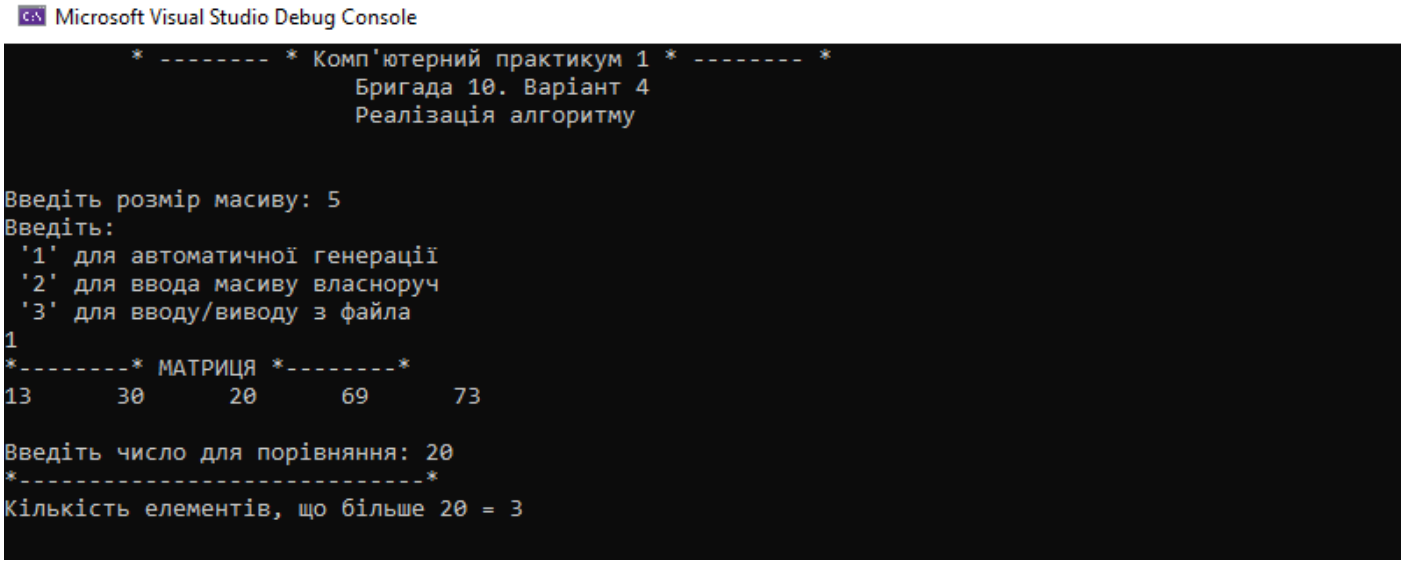
Перевірка правильності програми:

Вхідні дані				Результат	Призначення тесту
ArrSize	variant	Arr	MaxValue		
5	1	Випадковий	50	Невідомий, але легко обчислити	Перевірка ефективності та обчислювальної складності алгоритму із замірюванням часу виконання програми та побудовою профілю програми.
g	-	-	-	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
5	2	{1, 2, 3, 4, 5}	3	Відомий	Загальна перевірка працездатності алгоритму та коректності результату
5	2	{0, -2, g, 4.4566, 5}	-	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
5	3	{1, 2, 3, 4, 5}	3	Відомий	Загальна перевірка працездатності алгоритму та коректності результату

5	3	Невідомий	-	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
5	g	-	-	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)
5	1	Випадковий	g	Відомий	Перевірка реакції на некоректні вхідні дані (додавання повідомлень про помилки)

Табл. 1. Таблиця тестування програми

Скріншоти роботи програми:



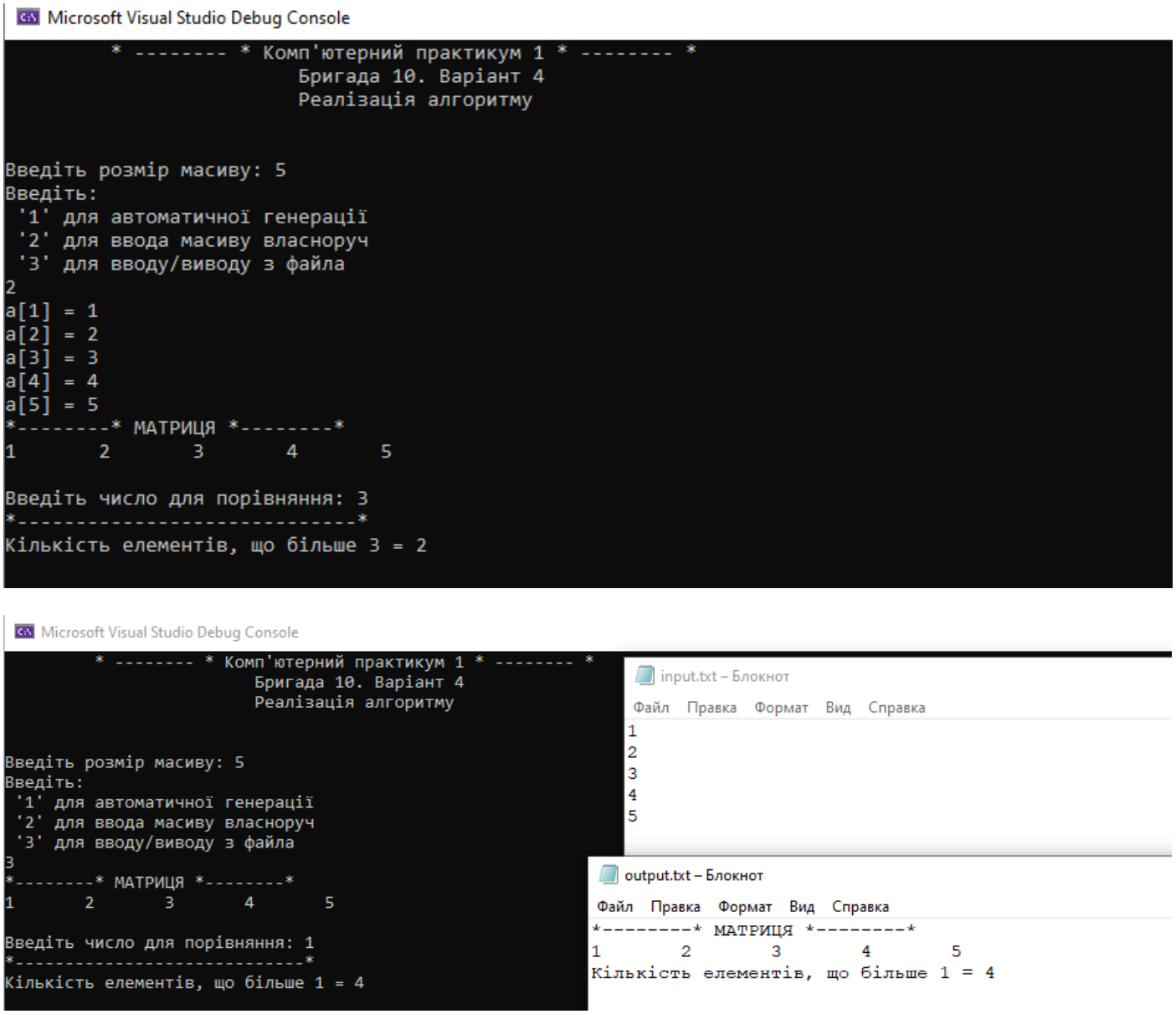


Рис. 2. Результати виконання завдання за різних типів виконання програми

Висновки:

У ході виконання комп’ютерного практикуму ми знайомились з видами робіт, що виконує програміст на кожному з етапів розв’язку задачі. Ми навчились правильному підходу до створення програм, а саме: формулювати задачу, будувати модель, розробляти алгоритм, перевіряти його на правильність та складність, реалізовувати алгоритм, перевіряти програму на ефективність та розв'язувати за допомогою створеної програми поставлені математичні задачі.

Відповіді на контрольні запитання:

1. З яких етапів складається процес створення комп'ютерної програми для вирішення довільної практичної задачі?

Процес створення комп'ютерної програми для вирішення довільної практичної задачі складається з наступних етапів:

1) Постановка задачі
2) Побудова моделі
3) Розробка алгоритму
4) Правильність алгоритму
5) Аналіз алгоритму та його складності
6) Реалізація алгоритму
7) Перевірка програми
8) Складання таблиці тестування
9) Наведення скріншотів роботи програми

2. Що саме має з'ясувати розробник програми на етапі постановки задачі?

На етапі постановки задачі розробник програми має чітко з'ясувати, що дано і що потрібно знайти.

3. Що робить розробник програми на етапі побудови моделі? Які фактори впливають на вибір структури моделі?

На етапі побудови моделі розробник визначає, які структури даних та які математичні залежності будуть використані у його майбутній програмі.

Перед тим, як використовувати модель, розробник перевіряє:

- Чи вся важлива інформація задачі добре описана математичними об'єктами?
- Чи існує математична величина, що асоціюється з шуканим результатом?
- Чи були виявлені які-небудь корисні відношення між об'єктами моделі?
- Чи може він працювати з моделлю?
- Чи зручно з нею працювати?

На вибір структури моделі впливають наступні фактори:

1) Обмеженість наших знань відносно невеликою кількістю структур;
2) Зручність представлення;
3) Простота обчислень;
4) Корисність різних операцій, пов’язаних з структурами, що розглядаються.

4. Якими міркуваннями має керуватися розробник програми на етапі розробки алгоритму? Чи потрібно перевіряти або доводити правильність алгоритму, якщо так, то з якою метою?

На етапі розробки алгоритму розробник має керуватися простотою розуміння та перекладу алгоритму на програмний код і наладки, а також ефективністю використаних комп'ютерних ресурсів. Також програма має виконуватися по можливості швидко.

Правильність алгоритму потрібно доводити з метою розуміння, чи буде досягнута поставлена задача у результаті виконання запропонованого алгоритму.

5. Навіщо виконується аналіз алгоритму та його складності?

Аналіз алгоритму та його складності виконується з метою:

- отримання оцінок або границь для обсягу пам’яті або часу роботи;
- порівняння алгоритмів (за певним критерієм);
- виявлення найбільш ефективних алгоритмів (за набором критеріїв);
- оцінки доцільності подальшого покращення алгоритму.

6. Існує три аспекти перевірки програми: на правильність, на ефективність реалізації, на обчислювальну складність. Розкрийте суть кожної з перевірок.

Перевірка на правильність підтверджує, що програма робить саме те, для чого вона була призначена.

При перевірці правильності програми звертають увагу на:

- Міри безпеки
- Локальні перевірки
- Перевірка усіх частин програми
- Тестові дані

- Пошук скритих помилок

Перевірка на ефективність направлена на відшукання способів змусити правильну програму працювати швидше або витрачати менше пам’яті.

Методи підвищення ефективності програми поділяються на:

- Машинно-незалежні:
  - арифметичні операції
  - надлишкові обчислення
  - порядок в логічних виразах
  - виключення, розгортання та спрощення циклів
  - профілі виконання
- Машинно-залежні:
  - налаштування компілятора
  - доступний обсяг оперативної

Перевірка на обчислювальну складність зводиться до експериментального аналізу складності алгоритму або до експериментального порівняння двох чи більше алгоритмів, що розв’язують одну і ту ж саму задачу.

7. Навіщо виконується вимірювання часу виконання програми? Які чинники на нього впливають?

Вимірювання часу виконання програми виконується з метою оцінки ефективності використання ресурсів комп'ютера та оцінки складності алгоритму.
Чим складніший програмний код, тобто чим складніші математичні операції виконує програма, тим більше часу потрібно комп'ютеру для їх виконання.