Київський національний університет імені Тараса Шевченка

**Лабораторна робота №1**

**З навчального курсу «Розробка ПЗ під мобільні платформи»**

Виконав:

студент 4 курсу

факультету кібернетики

спеціальність «Комп’ютерні науки»

групи ТТП-42

Чебан Богдан Володимирович

**Київ 2024**

**Зміст**

1.Опис завдання

2.Короткий виклад теорії

3.Опис алгоритмів, які застосовуються для обчислень

3.1. Selection Sort

3.2. Quick Sort

3.3. Merge Sort

3.4. Insertion Sort

3.5. Bubble Sort

4.Опис роботи з програмою та скріншоти

5.Список використаних джерел

6. Код основної частини програми

**Опис завдання**

Метою роботи є формування базових знань та умінь розробляти для мобільного пристрою програми обчислювального характеру, використовувати програмні засоби роботи з файлами та візуалізувати дані. За бажанням студента погодженням з боку викладача, зміст роботи може бути змінений.

Загальні вимоги:

- Має бути документ (DOCX, PDF) з описом вимог до програмного продукту (ТЗ)

- Програма повинна реалізовувати запис/читання даних з файлу або локальної БД

- Програма повинна мати пункт з інформацією про розробника та його/її фото

- Повинні бути застосовані базові принципи UX-дизайну та ергономіки для мобільних пристроїв

- Частиною результату виконання Проєкту 1 є звіт – текстовий документ, студент має відправити в Classroom і який включає:

o Титульну сторінку

o Зміст

o Опис завдання

o Короткий виклад теорії,

o Опис алгоритмів, які застосовуються для обчислень

o Опис роботи з програмою та скрін-шоти

o Список використаних джерел (мінімум 2), оформлений за ДСТУ

o Код основної частини програми у додатку(ах)

**Опис структури , змісту і вимог до Лабораторної роботи 1**

**Тема**: Сортування масивів Демонстрація і порівняння різних методів

**Введення даних(Усі введені дані повинні мати можливість бути зчитаними з файлу):** Введення масиву

**Виведення даних:** Відсортований масив, кількість порівнянь при сортуванні (ітерацій) для кожного методу, визначення найефективнішого.

**Збереження даних(Усі введені користувачем дані повинні мати можливість бути збереженими у файл):**Введений масив клавіатури

**Графічна візуалізація:** Гістограма значень елементів масиву –початкового та кінцевого

**Примітка:** Мінімум 5 методів сортування

**Вимоги та їх виконання**

**1. Реалізація читання/запису даних з файлу або локальної БД**

**Реалізація у проекті:**

* **Читання з файлу:** Для зчитування масиву цілих чисел з текстового файлу використовується метод FileService.ReadIntArrayFromFileAsync. Цей метод дозволяє користувачеві завантажувати дані з обраного файлу, забезпечуючи коректну обробку числових значень.
* **Запис у файл:** Для збереження введеного масиву чисел застосовується метод FileService.SaveIntArrayToFileAsync. Цей функціонал дозволяє користувачу зберігати свої дані у текстовий файл, забезпечуючи можливість подальшого використання або аналізу.

**Відповідність вимогам:** Програма успішно реалізує функціонал читання та запису даних з файлів, що відповідає поставленим вимогам лабораторної роботи.

**2. Наявність пункту з інформацією про розробника та його/її фото**

**Реалізація у проекті:**

* **Сторінка "Про розробника":** В проектах AboutPage.xaml та AboutPage.xaml.cs створено окрему сторінку, яка містить детальну інформацію про розробника, включаючи фотографію (developer\_photo.png). Ця сторінка забезпечує користувачам можливість дізнатися більше про автора додатку.

**Відповідність вимогам:** Програма включає розділ з інформацією про розробника та його фото, що відповідає вимогам щодо надання інформації про автора проекту.

**3. Застосування базових принципів UX-дизайну та ергономіки для мобільних пристроїв**

**Реалізація у проекті:**

* **Інтерфейс користувача:** Для організації елементів інтерфейсу використовуються компоненти VerticalStackLayout та HorizontalStackLayout. Це забезпечує зрозуміле та інтуїтивне розташування елементів, сприяючи зручності використання додатку.
* **Елементи взаємодії:** У додатку передбачено зручні кнопки для завантаження та створення файлів, а також поле для введення масиву чисел. Це дозволяє користувачеві легко взаємодіяти з програмою та виконувати необхідні дії.
* **Візуалізація даних:** Гістограми, що відображаються на сторінці HistogramPage, забезпечують наочне представлення початкового та відсортованого масивів, що покращує сприйняття інформації користувачем.

**Відповідність вимогам:** Інтерфейс додатку розроблений з урахуванням основних принципів UX-дизайну та ергономіки, що робить його зручним та привабливим для користувачів.

**4. Тема - Сортування масивів: демонстрація та порівняння різних методів**

**Реалізація у проекті:**

* **Алгоритми сортування:** В додатку реалізовано п’ять методів сортування: BubbleSort, InsertionSort, MergeSort, QuickSort, SelectionSort. Кожен з цих алгоритмів має свою реалізацію, що дозволяє демонструвати їх особливості та ефективність.
* **Порівняння алгоритмів:** Клас SortManager відповідає за виконання сортування масиву всіма реалізованими методами та зберігання кількості ітерацій для кожного алгоритму. Це дозволяє порівнювати ефективність різних методів сортування на одних і тих же даних.

**Відповідність вимогам:** Додаток успішно демонструє та порівнює різні методи сортування масивів, що відповідає темі лабораторної роботи.

**5. Введення даних: можливість зчитування з файлу та введення масиву**

**Реалізація у проекті:**

* **Зчитування з файлу:** Користувач має можливість завантажити файл з масивом чисел за допомогою команди OpenFileCommand. Це дозволяє швидко імпортувати дані для подальшого сортування.
* **Введення масиву вручну:** Крім завантаження з файлу, користувач може вручну ввести масив чисел у відповідному полі вводу. Це забезпечує гнучкість у використанні додатку та можливість швидкого введення власних даних.

**Відповідність вимогам:** Обидва способи введення даних – зчитування з файлу та ручне введення масиву – успішно реалізовані у додатку.

**6. Виведення даних: відсортований масив, кількість порівнянь (ітерацій) для кожного методу, визначення найефективнішого методу**

**Реалізація у проекті:**

* **Виведення результатів сортування:** На головній сторінці MainPage відображається кількість ітерацій для кожного алгоритму сортування, а також визначається найефективніший метод. Це дозволяє користувачу легко зрозуміти ефективність різних методів на основі виконаних обчислень.
* **Обробка та збереження даних:** Клас SortManager виконує сортування масиву за всіма методами та зберігає кількість ітерацій для подальшого аналізу та відображення результатів.

**Відповідність вимогам:** Програма успішно виводить відсортований масив, кількість ітерацій для кожного методу сортування та визначає найефективніший алгоритм, що відповідає вимогам лабораторної роботи.

**7. Збереження даних: можливість збереження введеного масиву у файл**

**Реалізація у проекті:**

* **Запис у файл:** Користувач може зберегти введений масив чисел за допомогою команди CreateFileCommand, яка викликає метод FileService.SaveIntArrayToFileAsync. Це забезпечує збереження даних у текстовий файл для подальшого використання або аналізу.

**Відповідність вимогам:** Реалізована можливість збереження введених користувачем даних у файл повністю відповідає поставленим вимогам.

**8. Графічна візуалізація: гістограма значень елементів масиву – початкового та кінцевого**

**Реалізація у проекті:**

* **Візуалізація даних:** На сторінці HistogramPage відображаються гістограми початкового та відсортованого масивів за допомогою бібліотеки Microcharts.Maui. Це забезпечує наочне представлення розподілу елементів масиву до і після сортування.

**Відповідність вимогам:** Гістограми надають візуальне представлення даних, що відповідає вимогам щодо графічної візуалізації результатів сортування.

**9. Примітка: мінімум 5 методів сортування**

**Реалізація у проекті:**

* **Методи сортування:** В додатку реалізовано п’ять різних методів сортування: BubbleSort, InsertionSort, MergeSort, QuickSort, SelectionSort. Це забезпечує широкий спектр алгоритмів для демонстрації та порівняння їх ефективності.

**Відповідність вимогам:** Всі необхідні п’ять методів сортування успішно реалізовані у проекті, що повністю відповідає вимогам лабораторної роботи.

**Короткий виклад теорії**

**Сортування масивів** є однією з базових та найпоширеніших задач в інформатиці та програмуванні. Воно полягає у впорядкуванні елементів масиву за певним критерієм, зазвичай за зростанням або спаданням значень елементів. Сортування відіграє ключову роль у багатьох алгоритмах і застосуваннях, включаючи пошук, оптимізацію, статистику та організацію даних.

**Значення сортування:**

**Ефективний пошук:** Відсортовані дані дозволяють застосовувати швидкі алгоритми пошуку, такі як бінарний пошук, що значно зменшує час пошуку елементів.

**Організація даних:** Сортування допомагає впорядкувати дані для кращого сприйняття та аналізу.

**Попередня обробка:** У багатьох алгоритмах необхідно попередньо відсортувати дані для подальшої обробки.

**Класифікація алгоритмів сортування:**

Алгоритми сортування можна класифікувати за різними ознаками:

* **За складністю:**

**Прості алгоритми:** Bubble Sort, Insertion Sort, Selection Sort (часова складність O(n²)).

**Ефективні алгоритми:** Merge Sort, Quick Sort, Heap Sort (часова складність O(n log n)).

* **За методом сортування:**

**Порівняльні алгоритми:** Сортування здійснюється шляхом порівняння елементів (наприклад, Quick Sort).

**Непорівняльні алгоритми:** Використовують інші методи, такі як розподіл (наприклад, Counting Sort).

* **За стабільністю:**

**Стабільні алгоритми:** Зберігають відносний порядок елементів з однаковими ключами (наприклад, Merge Sort).

**Нестабільні алгоритми:** Можуть змінювати порядок однакових елементів (наприклад, Quick Sort).

* **За пам'яттєвою складністю:**

**Внутрішні сортування:** Виконуються в пам'яті та використовують O(1) або O(log n) додаткової пам'яті.

**Зовнішні сортування:** Використовуються для великих обсягів даних, які не вміщуються в оперативну пам'ять.

**Основні характеристики алгоритмів сортування:**

1. **Складність за часом:**

**Найгірший випадок (Worst-case):** Максимальний час виконання алгоритму при найбільш несприятливих вхідних даних.

**Середній випадок (Average-case):** Середній час виконання алгоритму для випадкових вхідних даних.

**Найкращий випадок (Best-case):** Мінімальний час виконання алгоритму при найбільш сприятливих вхідних даних.

1. **Складність за пам'яттю:**

Визначає кількість додаткової пам'яті, необхідної для виконання алгоритму.

**In-place алгоритми:** Використовують O(1) додаткової пам'яті (наприклад, Insertion Sort).

**Не in-place алгоритми:** Використовують O(n) додаткової пам'яті (наприклад, Merge Sort).

1. **Стабільність:**

**Стабільні алгоритми:** Зберігають відносний порядок елементів з однаковими ключами, що може бути важливим при сортуванні складних структур даних.

**Нестабільні алгоритми:** Не гарантують збереження порядку однакових елементів.

1. **Адаптивність:**

Алгоритм називається **адаптивним**, якщо його продуктивність покращується на вже частково відсортованих масивах.

Адаптивні алгоритми можуть ефективніше працювати на реальних даних, які часто мають певний порядок.

1. **Чутливість до розподілу даних:**

Деякі алгоритми можуть працювати швидше або повільніше в залежності від розподілу вхідних даних.

Наприклад, Quick Sort може працювати повільніше на відсортованих масивах без правильного вибору опорного елемента.

**Вибір алгоритму сортування:**

Вибір алгоритму залежить від:

**Розміру масиву:** Для невеликих масивів прості алгоритми можуть бути більш ефективними через малий накладний час.

**Характеру даних:** Якщо дані вже частково відсортовані або мають певний розподіл.

**Вимог до пам'яті:** Якщо є обмеження по використанню додаткової пам'яті.

**Вимог до стабільності:** Якщо потрібно зберегти відносний порядок елементів з однаковими ключами.

**Значення аналізу алгоритмів:**

**Теоретичний аналіз:** Дозволяє оцінити алгоритми за їх асимптотичною складністю та вибрати найбільш підходящий для конкретного завдання.

**Практичне тестування:** Реальні вимірювання часу виконання та використання ресурсів на практичних даних.

У даній роботі ми реалізуємо та порівнюємо п'ять різних алгоритмів сортування, аналізуємо їх ефективність та на практиці демонструємо їх особливості та відмінності.

**Опис алгоритмів, які застосовуються для обчислень**

У програмі реалізовано п'ять алгоритмів сортування:

**3.1. Selection Sort**

**Selection Sort** — простий алгоритм сортування, який працює шляхом послідовного вибору найменшого (або найбільшого) елемента з невідсортованої частини масиву і переміщення його в відсортовану частину.

**Алгоритм:**

1. Ініціалізуємо мінімальний індекс як i.
2. Проходимо по невідсортованій частині масиву, шукаючи мінімальний елемент.
3. Після знаходження мінімального елемента, міняємо його місцями з елементом на позиції i.
4. Збільшуємо i і повторюємо кроки 1-3 до кінця масиву.

**Характеристики:**

* **Часова складність**: O(n²)
* **Пам'яттєва складність**: O(1)
* **Стабільність**: Нестабільний

**3.2. Quick Sort**

**Quick Sort** — ефективний алгоритм сортування, який використовує підхід "розділяй і володарюй". Він вибирає опорний елемент (pivot) і розділяє масив на дві частини: елементи менші за опорний і елементи більші за опорний.

**Алгоритм:**

1. Вибрати опорний елемент (зазвичай останній елемент масиву).
2. Розподілити масив на дві частини: елементи менші за опорний і елементи більші або рівні опорному.
3. Рекурсивно застосувати Quick Sort до підмасивів.

**Характеристики:**

* **Часова складність**: O(n log n) у середньому випадку, O(n²) у найгіршому
* **Пам'яттєва складність**: O(log n) для рекурсії
* **Стабільність**: Нестабільний

**3.3. Merge Sort**

**Merge Sort** — ще один алгоритм, що використовує підхід "розділяй і володарюй". Він розділяє масив на дві половини, рекурсивно сортує кожну половину, а потім об'єднує їх в один відсортований масив.

**Алгоритм:**

1. Якщо масив містить більше одного елемента:
   * Розділити масив на дві половини.
   * Рекурсивно відсортувати кожну половину.
   * Об'єднати відсортовані половини.
2. Повернути відсортований масив.

**Характеристики:**

* **Часова складність**: O(n log n)
* **Пам'яттєва складність**: O(n)
* **Стабільність**: Стабільний

**3.4. Insertion Sort**

**Insertion Sort** — простий алгоритм сортування, який будує відсортований масив поступово, додаючи один елемент за раз і вставляючи його у відповідне місце.

**Алгоритм:**

1. Проходимо масив з другого елемента до кінця.
2. Для кожного елемента, порівнюємо його з елементами в відсортованій частині масиву.
3. Вставляємо елемент у правильне місце у відсортованій частині.

**Характеристики:**

* **Часова складність**: O(n²)
* **Пам'яттєва складність**: O(1)
* **Стабільність**: Стабільний

**3.5. Bubble Sort**

**Bubble Sort** — один з найпростіших алгоритмів сортування. Він повторно проходить через масив, порівнює сусідні елементи і міняє їх місцями, якщо вони в неправильному порядку.

**Алгоритм:**

1. Повторюємо до тих пір, поки масив не буде відсортований:
   * Проходимо по масиву.
   * Порівнюємо кожну пару сусідніх елементів.
   * Якщо вони в неправильному порядку, міняємо їх місцями.

**Характеристики:**

* **Часова складність**: O(n²)
* **Пам'яттєва складність**: O(1)
* **Стабільність**: Стабільний

**Опис роботи з програмою та скріншоти**

**Застосунок був створений з використанням .NET Multi-Platform App UI (.NET MAUI)** – це сучасна крос-платформна платформа від Microsoft, призначена для розробки мобільних, десктопних та веб-додатків за допомогою однієї кодової бази. .NET MAUI дозволяє розробникам використовувати мову програмування C# та розмітку XAML для створення інтуїтивно зрозумілих та ефективних інтерфейсів користувача.

**Архітектура та структура додатку**

Додаток розроблений за архітектурою **MVVM (Model-View-ViewModel)**, яка дозволяє розділити логіку бізнесу, інтерфейс користувача та взаємодію між ними. Це сприяє кращій організації коду, його повторному використанню та спрощує процес тестування.

**Model:** Відповідає за дані та бізнес-логіку додатку. У нашому проекті це класи, які реалізують алгоритми сортування та обробку даних.

**View:** Представляє собою інтерфейс користувача, створений за допомогою XAML. Це сторінки, які взаємодіють з користувачем, збирають вхідні дані та відображають результати.

**ViewModel:** Посередник між Model та View. Відповідає за обробку даних, виконання команд та оновлення інтерфейсу користувача через прив'язку даних (data binding).

**Переваги використання .NET MAUI у проекті:**

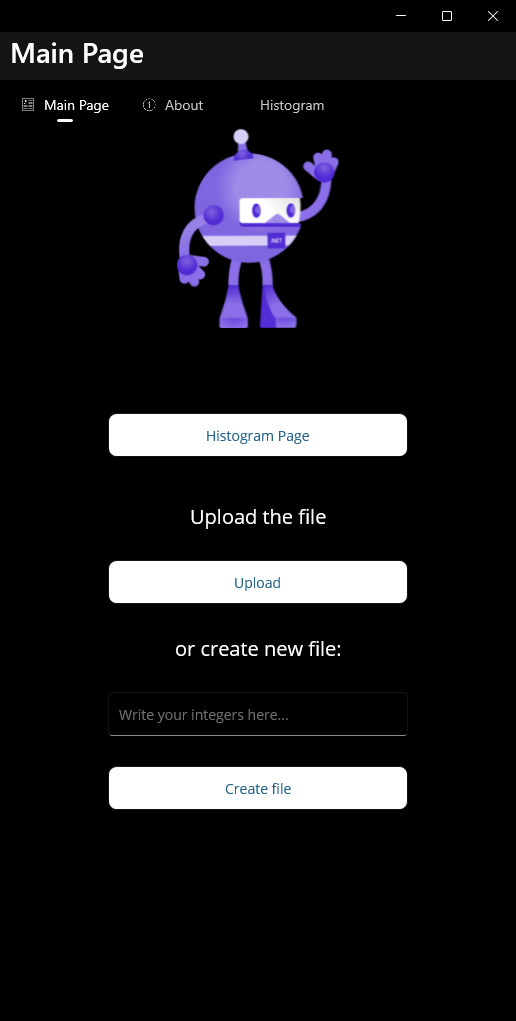
* **Крос-платформність:** Додаток може працювати на різних платформах, включаючи Android, iOS, Windows та macOS, без необхідності писати окремий код для кожної з них.
* **Єдина кодова база:** Використовуючи одну кодову базу для всіх платформ, розробка стає більш ефективною та менш витратною за часом.
* **Гнучкість та розширюваність:** .NET MAUI підтримує різноманітні контролі та бібліотеки, що дозволяє легко інтегрувати додатковий функціонал, та забезпечує високу гнучкість у дизайні інтерфейсу.
* **Підтримка MVVM-патерну:** Використання патерну Model-View-ViewModel (MVVM) сприяє кращій організації коду, підвищує його читабельність та спрощує тестування.
* **Інтеграція з екосистемою .NET:** .NET MAUI легко інтегрується з іншими технологіями та бібліотеками екосистеми .NET, такими як Entity Framework, LINQ та інші, що значно розширює можливості додатку.

**Інтерфейс програми** складається з кількох сторінок:

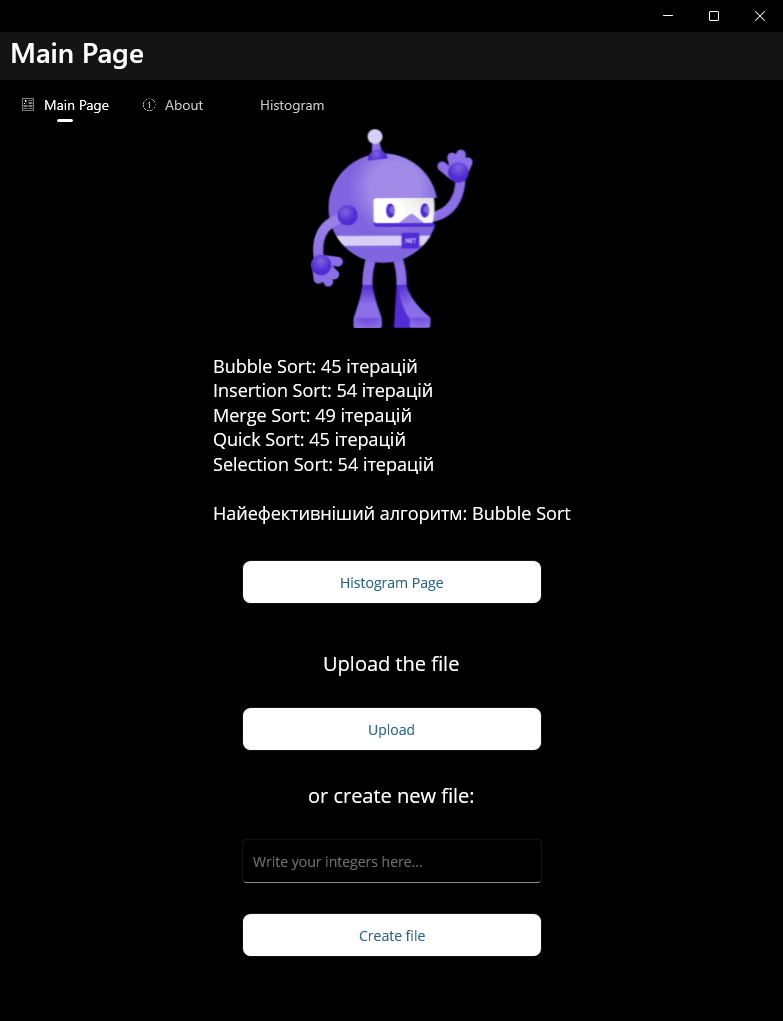
* **Головна сторінка**: дозволяє користувачу завантажити масив з файлу або ввести його вручну та зберегти у файл.
* **Сторінка з результатами**: відображає кількість ітерацій для кожного методу сортування та визначає найефективніший.
* **Сторінка з гістограмами**: відображає гістограми початкового та відсортованого масивів.
* **Сторінка "Про розробника"**: містить інформацію про розробника та його фото.

**Кроки роботи з програмою:**

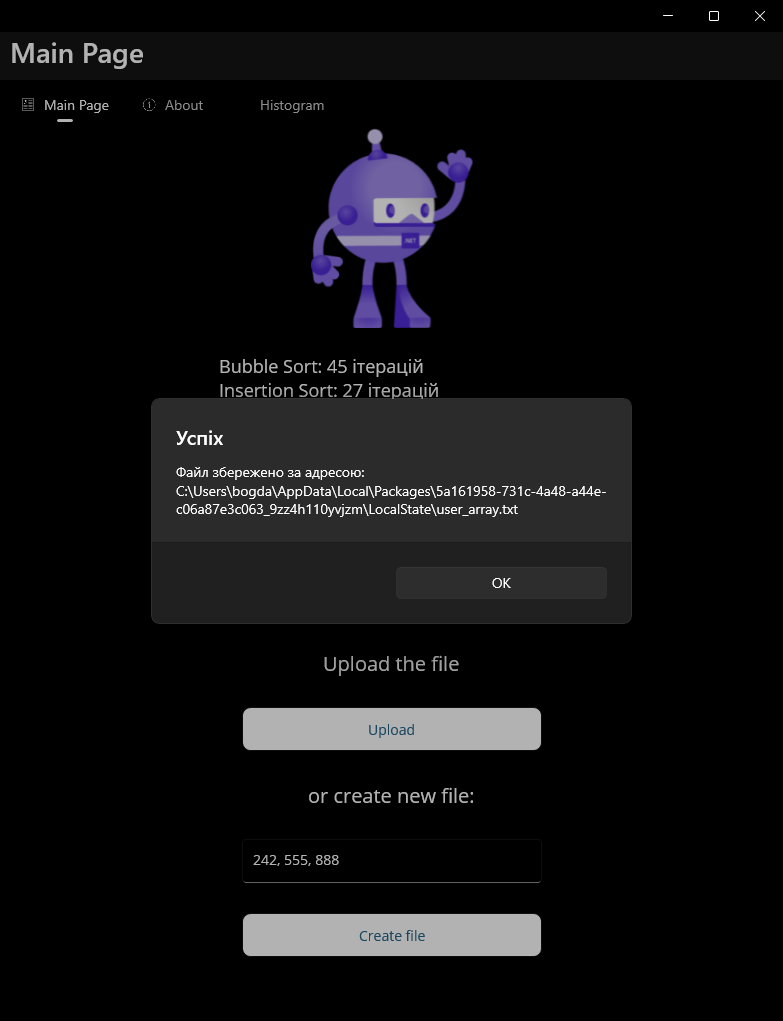
1. **Запуск програми**: відкривається головна сторінка з привітальним зображенням та опціями.



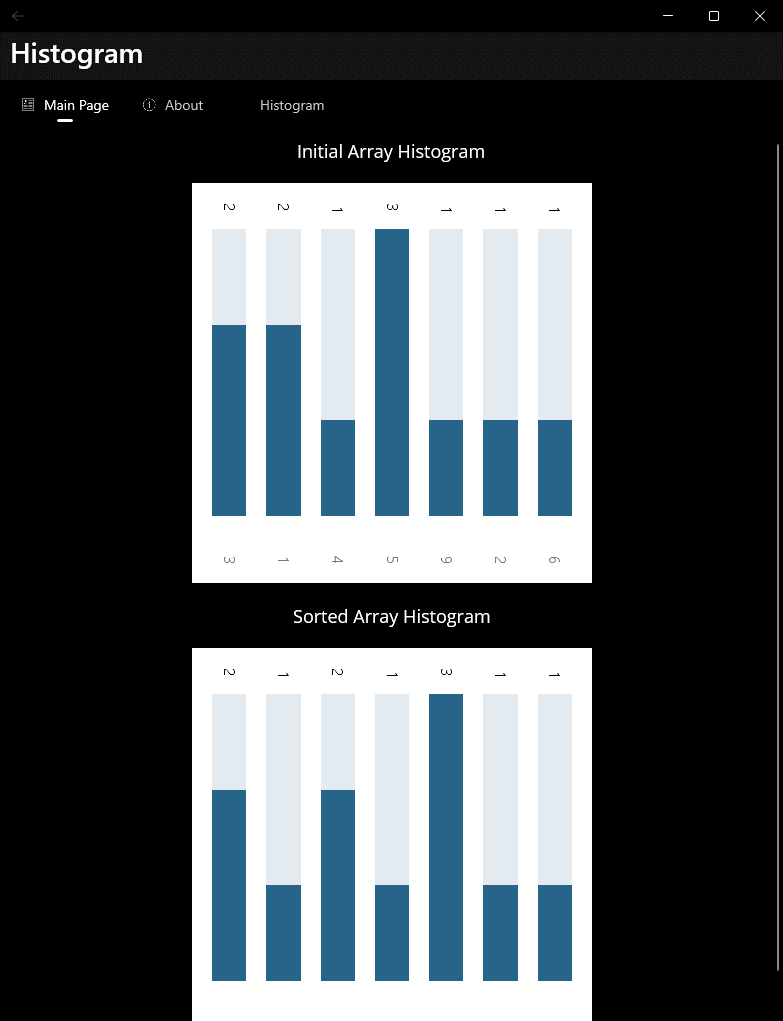
1. **Завантаження масиву з файлу**:
   * Натисніть кнопку **"Upload"**.
   * Виберіть текстовий файл з масивом чисел.
   * Програма зчитає масив та виконає сортування всіма методами.
   * Результати будуть відображені на екрані.



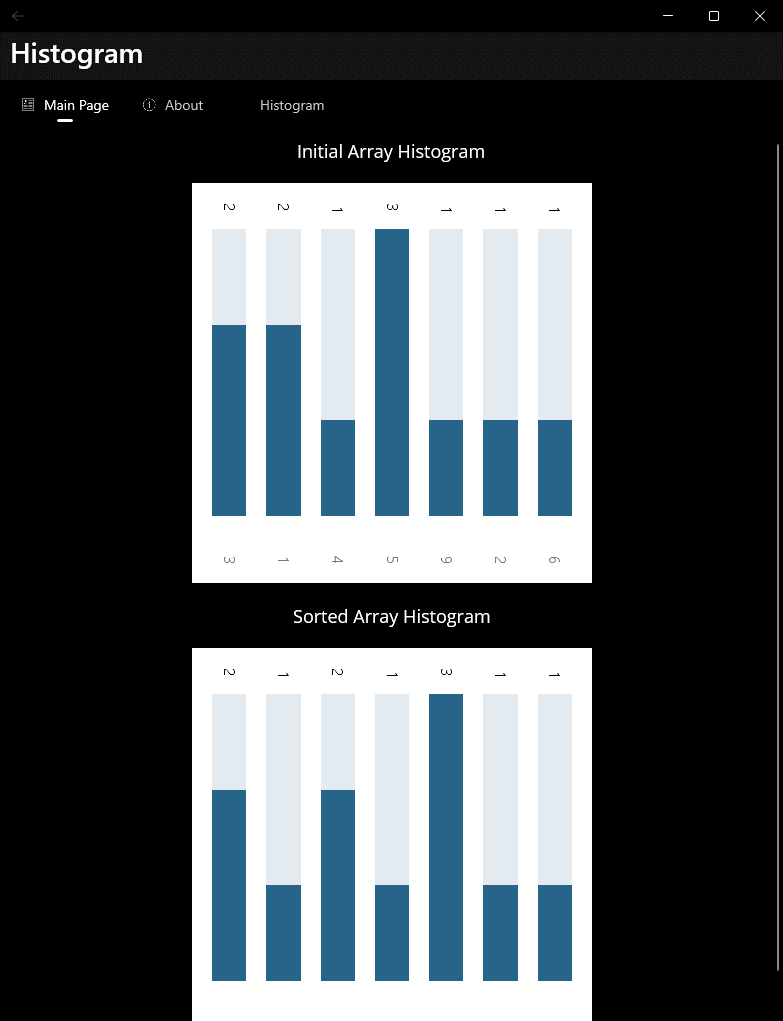
1. **Введення масиву вручну та збереження у файл**:
   * Введіть числа в поле вводу, розділені комами або пробілами.
   * Натисніть кнопку **"Create file"**.
   * Масив буде збережено у файл у внутрішньому сховищі програми.



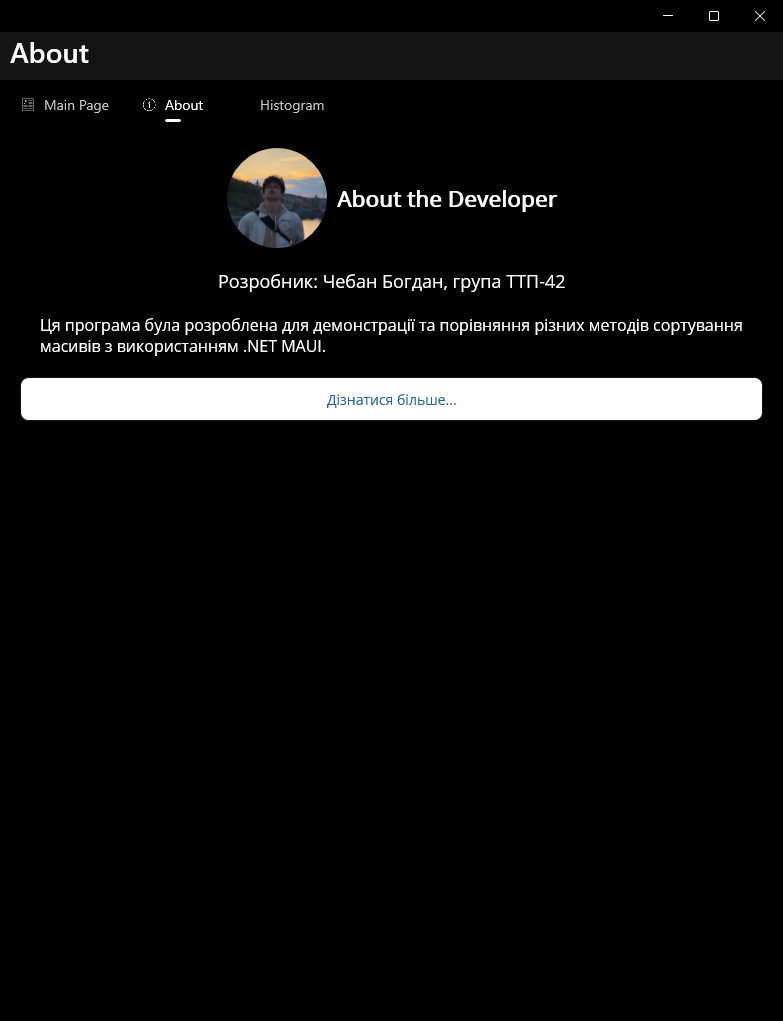
1. **Перегляд результатів сортування**:
   * Після сортування відображається кількість ітерацій для кожного методу.
   * Визначається найефективніший метод.



1. **Перегляд гістограм**:
   * Натисніть кнопку **"Histogram Page"**.
   * Відкриється сторінка з гістограмами початкового та відсортованого масивів.



1. **Перегляд інформації про розробника**:
   * Перейдіть на вкладку **"About"**.
   * Відобразиться інформація про розробника та його фото.



**Список використаних джерел**

1.**Кормен Т. Х.** Алгоритми: побудова і аналіз : пер. з англ. — К. : Діалектика, 2020. — 1296 с.

2.**Роберт Седжвік, Кевін Вейн.** Алгоритми на Java. Частина 1-4. — К. : BHV, 2019. — 864 с.

**Код основної частини програми у додатках**

У даному розділі наведено основний код програми, який реалізує функціонал сортування масивів, взаємодії з користувачем та графічної візуалізації.

**Додаток А: Класи алгоритмів сортування**

**A.1. Інтерфейс ISortAlgorithm**

namespace Laba1.Classes.Algorithms

{

public interface ISortAlgorithm

{

string Name { get; }

int Sort(int[] array);

}

}

A.2. Клас BubbleSort

namespace Laba1.Classes.Algorithms

{

public class BubbleSort : ISortAlgorithm

{

public string Name => "Bubble Sort";

public int Sort(int[] arr)

{

int iterations = 0;

int n = arr.Length;

bool swapped;

do

{

swapped = false;

for (int i = 1; i < n; i++)

{

iterations++;

if (arr[i - 1] > arr[i])

{

int temp = arr[i - 1];

arr[i - 1] = arr[i];

arr[i] = temp;

swapped = true;

}

}

n--;

} while (swapped);

return iterations;

}

}

}

**A.3. Клас InsertionSort**

namespace Laba1.Classes.Algorithms

{

public class InsertionSort : ISortAlgorithm

{

public string Name => "Insertion Sort";

public int Sort(int[] arr)

{

int iterations = 0;

int n = arr.Length;

for (int i = 1; i < n; i++)

{

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key)

{

iterations++;

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

iterations++;

}

return iterations;

}

}

}

**A.4. Клас MergeSort**

namespace Laba1.Classes.Algorithms

{

public class MergeSort : ISortAlgorithm

{

public string Name => "Merge Sort";

public int Sort(int[] array)

{

return MergeSortRecursive(array, 0, array.Length - 1);

}

private int Merge(int[] array, int lowIndex, int middleIndex, int highIndex)

{

int iterations = 0;

int[] tempArray = new int[highIndex - lowIndex + 1];

int left = lowIndex;

int right = middleIndex + 1;

int index = 0;

while (left <= middleIndex && right <= highIndex)

{

iterations++;

if (array[left] < array[right])

{

tempArray[index++] = array[left++];

}

else

{

tempArray[index++] = array[right++];

}

}

while (left <= middleIndex)

{

tempArray[index++] = array[left++];

}

while (right <= highIndex)

{

tempArray[index++] = array[right++];

}

for (int i = 0; i < tempArray.Length; i++)

{

array[lowIndex + i] = tempArray[i];

iterations++;

}

return iterations;

}

private int MergeSortRecursive(int[] array, int lowIndex, int highIndex)

{

int iterations = 0;

if (lowIndex < highIndex)

{

int middleIndex = (lowIndex + highIndex) / 2;

iterations += MergeSortRecursive(array, lowIndex, middleIndex);

iterations += MergeSortRecursive(array, middleIndex + 1, highIndex);

iterations += Merge(array, lowIndex, middleIndex, highIndex);

}

return iterations;

}

}

}

**A.5. Клас QuickSort**

namespace Laba1.Classes.Algorithms

{

public class QuickSort : ISortAlgorithm

{

public string Name => "Quick Sort";

public int Sort(int[] array)

{

return QuickSortRecursive(array, 0, array.Length - 1);

}

private void Swap(int[] arr, int i, int j)

{

if (i == j) return;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

private int Partition(int[] arr, int low, int high, out int iterations)

{

iterations = 0;

int pivot = arr[high];

int i = low;

for (int j = low; j < high; j++)

{

iterations++;

if (arr[j] <= pivot)

{

Swap(arr, i, j);

i++;

}

}

Swap(arr, i, high);

return i;

}

private int QuickSortRecursive(int[] arr, int low, int high)

{

int iterations = 0;

if (low < high)

{

int pivot = Partition(arr, low, high, out int partitionIterations);

iterations += partitionIterations;

iterations += QuickSortRecursive(arr, low, pivot - 1);

iterations += QuickSortRecursive(arr, pivot + 1, high);

}

return iterations;

}

}

}

**A.6. Клас SelectionSort**

namespace Laba1.Classes.Algorithms

{

public class SelectionSort : ISortAlgorithm

{

public string Name => "Selection Sort";

public int Sort(int[] arr)

{

int iterations = 0;

int n = arr.Length;

for (int i = 0; i < n - 1; i++)

{

int min = i;

for (int j = i + 1; j < n; j++)

{

iterations++;

if (arr[j] < arr[min])

{

min = j;

}

}

Swap(arr, min, i);

iterations++;

}

return iterations;

}

private void Swap(int[] arr, int i, int j)

{

if (i == j) return;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

}

**Додаток Б: Основні класи програми**

**B.1. Клас SortManager**

using Laba1.Classes.Algorithms;

using Laba1.Models;

using System.Collections.Generic;

using System.Linq;

namespace Laba1.Classes

{

public class SortManager

{

private List<ISortAlgorithm> \_algorithms;

public SortManager()

{

\_algorithms = new List<ISortAlgorithm>

{

new BubbleSort(),

new InsertionSort(),

new MergeSort(),

new QuickSort(),

new SelectionSort()

};

}

public List<SortResult> SortAll(int[] array)

{

var results = new List<SortResult>();

foreach (var algorithm in \_algorithms)

{

int[] arrayCopy = (int[])array.Clone();

int iterations = algorithm.Sort(arrayCopy);

results.Add(new SortResult

{

AlgorithmName = algorithm.Name,

Iterations = iterations,

SortedArray = arrayCopy

});

}

return results;

}

public SortResult GetMostEfficient(int[] array)

{

var results = SortAll(array);

return results.OrderBy(r => r.Iterations).FirstOrDefault();

}

}

}

**B.2. Клас FileService**

using System.IO;

using System.Threading.Tasks;

using Microsoft.Maui.Storage;

using System;

using System.Collections.Generic;

namespace Laba1.Classes.Services

{

public class FileService

{

public async Task<int[]> ReadIntArrayFromFileAsync(FileResult file)

{

if (file == null)

throw new ArgumentNullException(nameof(file));

using var stream = await file.OpenReadAsync();

using var reader = new StreamReader(stream);

string text = await reader.ReadToEndAsync();

string[] tokens = text.Split(new[] { ' ', '\r', '\n', ',', ';' }, StringSplitOptions.RemoveEmptyEntries);

List<int> integers = new List<int>();

foreach (var token in tokens)

{

if (int.TryParse(token, out int number))

integers.Add(number);

else

throw new FormatException("Файл містить нецілі числа.");

}

return integers.ToArray();

}

public async Task SaveIntArrayToFileAsync(int[] array, string filePath)

{

var text = string.Join(" ", array);

using var writer = new StreamWriter(filePath, false);

await writer.WriteAsync(text);

}

}

}

**B.3. Клас MainViewModel**

using System.ComponentModel;

using System.Threading.Tasks;

using System.Windows.Input;

using Laba1.Classes;

using Laba1.Classes.Helpers;

using Laba1.Classes.Services;

using Laba1.Models;

using Microsoft.Maui.Controls;

using System.Linq;

using System.IO;

using Microsoft.Maui.Storage;

using System.Collections.Generic;

namespace Laba1.ViewModels

{

public class MainViewModel : INotifyPropertyChanged

{

private readonly FileService \_fileService;

private readonly SortManager \_sortManager;

private string \_result;

public string Result

{

get => \_result;

set

{

\_result = value;

OnPropertyChanged(nameof(Result));

}

}

private string \_newArrayInput;

public string NewArrayInput

{

get => \_newArrayInput;

set

{

\_newArrayInput = value;

OnPropertyChanged(nameof(NewArrayInput));

}

}

public ICommand ShowHistogramPageCommand { get; }

public ICommand OpenFileCommand { get; }

public ICommand CreateFileCommand { get; }

private int[] \_initialArray;

private int[] \_sortedArray;

public MainViewModel()

{

\_fileService = new FileService();

\_sortManager = new SortManager();

ShowHistogramPageCommand = new Command(async () => await ShowHistogramPage());

OpenFileCommand = new Command(async () => await OpenFile());

CreateFileCommand = new Command(async () => await CreateFile());

}

private async Task ShowHistogramPage()

{

if (\_initialArray == null || \_sortedArray == null)

{

await Application.Current.MainPage.DisplayAlert("Помилка", "Спочатку виконайте сортування.", "OK");

return;

}

string initial = string.Join(",", \_initialArray);

string sorted = string.Join(",", \_sortedArray);

await Shell.Current.GoToAsync($"HistogramPage?initialArray={Uri.EscapeDataString(initial)}&sortedArray={Uri.EscapeDataString(sorted)}");

}

private async Task OpenFile()

{

try

{

var fileType = new FilePickerFileType(new Dictionary<DevicePlatform, IEnumerable<string>>

{

{ DevicePlatform.iOS, new[] { "public.text" } },

{ DevicePlatform.Android, new[] { "text/plain" } },

{ DevicePlatform.WinUI, new[] { ".txt" } },

{ DevicePlatform.macOS, new[] { "txt" } },

});

var file = await FilePicker.PickAsync(new PickOptions

{

FileTypes = fileType,

PickerTitle = "Виберіть текстовий файл з числами"

});

if (file != null)

{

\_initialArray = await \_fileService.ReadIntArrayFromFileAsync(file);

var results = \_sortManager.SortAll(\_initialArray);

var mostEfficient = \_sortManager.GetMostEfficient(\_initialArray);

\_sortedArray = results.FirstOrDefault()?.SortedArray;

Result = string.Join("\n", results.Select(r => $"{r.AlgorithmName}: {r.Iterations} ітерацій")) +

$"\n\nНайефективніший алгоритм: {mostEfficient?.AlgorithmName}";

}

}

catch (Exception ex)

{

await Application.Current.MainPage.DisplayAlert("Помилка", ex.Message, "OK");

}

}

private async Task CreateFile()

{

try

{

string input = NewArrayInput;

if (string.IsNullOrWhiteSpace(input))

{

await Application.Current.MainPage.DisplayAlert("Помилка", "Введіть числа.", "OK");

return;

}

if (!input.TryParseArray(out int[] integers))

{

await Application.Current.MainPage.DisplayAlert("Помилка", "Введені дані містять нецілі числа.", "OK");

return;

}

string filePath = Path.Combine(FileSystem.AppDataDirectory, "user\_array.txt");

await \_fileService.SaveIntArrayToFileAsync(integers, filePath);

await Application.Current.MainPage.DisplayAlert("Успіх", $"Файл збережено за адресою:\n{filePath}", "OK");

}

catch (Exception ex)

{

await Application.Current.MainPage.DisplayAlert("Помилка", ex.Message, "OK");

}

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string name) =>

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));

}

}

**B.4. Клас HistogramViewModel**

using System.ComponentModel;

using Laba1.Classes;

using Microcharts;

using SkiaSharp;

namespace Laba1.ViewModels

{

public class HistogramViewModel : INotifyPropertyChanged

{

private Chart \_initialChart;

public Chart InitialChart

{

get => \_initialChart;

set

{

\_initialChart = value;

OnPropertyChanged(nameof(InitialChart));

}

}

private Chart \_sortedChart;

public Chart SortedChart

{

get => \_sortedChart;

set

{

\_sortedChart = value;

OnPropertyChanged(nameof(SortedChart));

}

}

public void GenerateCharts(int[] initialArray, int[] sortedArray)

{

var initialBuilder = new ChartBuilder();

initialBuilder.FillEntriesUsingData(initialArray);

InitialChart = initialBuilder.BuildBarChart();

var sortedBuilder = new ChartBuilder();

sortedBuilder.FillEntriesUsingData(sortedArray);

SortedChart = sortedBuilder.BuildBarChart();

}

public event PropertyChangedEventHandler PropertyChanged;

protected void OnPropertyChanged(string name) =>

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(name));

}

}

**Додаток В: Основні файли інтерфейсу користувача**

**В.1. MainPage.xaml**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:vm="clr-namespace:Laba1.ViewModels"

x:Class="Laba1.Views.MainPage"

Title="Main Page">

<ContentPage.BindingContext>

<vm:MainViewModel />

</ContentPage.BindingContext>

<ScrollView>

<VerticalStackLayout

Spacing="25"

Padding="30,0"

VerticalOptions="Start">

<Image

Source="dotnet\_bot.png"

SemanticProperties.Description="Cute dot net bot waving hi to you!"

HeightRequest="200"

HorizontalOptions="Center" />

<Label Text="{Binding Result}"

FontSize="18"

HorizontalOptions="Center"

LineBreakMode="WordWrap" />

<Button WidthRequest="300"

Margin="10"

Text="Histogram Page"

Command="{Binding ShowHistogramPageCommand}" />

<VerticalStackLayout Spacing="10">

<Label

Margin="10"

Text="Upload the file"

SemanticProperties.HeadingLevel="Level1"

FontSize="20"

HorizontalOptions="Center" />

<Button WidthRequest="300"

Margin="10"

Text="Upload"

Command="{Binding OpenFileCommand}" />

<Label

Margin="10"

Text="or create new file:"

SemanticProperties.HeadingLevel="Level1"

FontSize="20"

HorizontalOptions="Center" />

<Entry WidthRequest="300"

Margin="10"

Placeholder="Write your integers here..."

Text="{Binding NewArrayInput}"

Keyboard="Numeric"

HorizontalOptions="Center" />

<Button WidthRequest="300"

Margin="10"

Text="Create file"

Command="{Binding CreateFileCommand}" />

</VerticalStackLayout>

</VerticalStackLayout>

</ScrollView>

</ContentPage>

**В.2. HistogramPage.xaml**

<?xml version="1.0" encoding="utf-8" ?>

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"

xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

xmlns:microchartsMaui="clr-namespace:Microcharts.Maui;assembly=Microcharts.Maui"

xmlns:vm="clr-namespace:Laba1.ViewModels"

x:Class="Laba1.Views.HistogramPage"

Title="Histogram">

<ContentPage.BindingContext>

<vm:HistogramViewModel />

</ContentPage.BindingContext>

<ScrollView>

<StackLayout Padding="10" Spacing="20">

<Label Text="Initial Array Histogram"

FontSize="18"

HorizontalOptions="Center" />

<microchartsMaui:ChartView

Chart="{Binding InitialChart}"

WidthRequest="400"

HeightRequest="400" />

<Label Text="Sorted Array Histogram"

FontSize="18"

HorizontalOptions="Center" />

<microchartsMaui:ChartView

Chart="{Binding SortedChart}"

WidthRequest="400"

HeightRequest="400" />

</StackLayout>

</ScrollView>

</ContentPage>

Наведений код є основною частиною програми та відповідає за реалізацію алгоритмів сортування, обробку введених даних, взаємодію з користувачем та візуалізацію результатів. Детальніший код можна знайти в репозиторії проекту або за запитом.