МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

(повна назва кафедри, циклової комісії)

**КУРСОВА РОБОТА**

з дисципліни “Основи програмування. Частина 2. Модульне програмування”

на тему: “Упорядкування масивів”

Студента 1 курсу, групи ІП-23

Гончаренка Івана Юрійовича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник

Доцент Муха І.П

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Національна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |  |
| --- | --- | --- | --- |
| Члени комісії |  |  | Старший викладач Головченко М. М. |
|  | (підпис) |  | (посада, вчене звання, науковий ступінь, прізвище та ініціали) |
|  |  |  | Асистент Вовк Є. А. |
|  | (підпис) |  | (посада, вчене звання, науковий ступінь, прізвище та ініціали) |

Київ- 2023 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ІПЗ"

Курс 1 Група ІП-23 Семестр 2

ЗАВДАННЯ

на курсову роботу студента

Гончаренка Івана Юрійовича

(прізвище, ім’я, по батькові)

1. Тема роботи Упорядкування масивів

2. Строк здачі студентом закінченої роботи 31.05.2023

3. Вихідні дані до роботи Технічне завдання (Додаток А)

4. Зміст розрахунково-пояснювальної записки (перелік питань, які підлягають розробці)

Постановка задачі, Теоретичні відомості, Опис алгоритмів, Опис програмного забезпечення, Тестування програмного забезпечення, Інструкція користувача, Аналіз результатів

5. Перелік графічного матеріалу ( з точним зазначенням обов’язкових креслень )

6. Дата видачі завдання 21.03.2023

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва етапів курсової роботи | Термін виконання етапів роботи | Підписи керівника, студента |
| 1. | Отримання теми курсової роботи | 21.02.2023 |  |
| 2. | Підготовка ТЗ | 21.03.2023 |  |
| 3. | Пошук та вивчення літератури з питань курсової роботи | 4.04.2023 |  |
| 4. | Розробка сценарію роботи програми | 11.04.2023 |  |
| 6. | Узгодження сценарію роботи програми з керівником | 15.04.2023 |  |
| 5. | Розробка (вибір) алгоритму рішення задачі | 20.04.2023 |  |
| 6. | Узгодження алгоритму з керівником | 25.04.2023 |  |
| 7. | Узгодження з керівником інтерфейсу користувача | 27.04.2023 |  |
| 8. | Розробка програмного забезпечення | 15.05.2023 |  |
| 9. | Налагодження розрахункової частини програми | 17.05.2023 |  |
| 10. | Розробка та налагодження інтерфейсної частини програми | 19.05.2023 |  |
| 11. | Узгодження з керівником набору тестів для контрольного прикладу | 20.05.2023 |  |
| 12. | Тестування програми | 25.05.2023 |  |
| 13. | Підготовка пояснювальної записки | 29.05.2023 |  |
| 14. | Здача курсової роботи на перевірку | 31.05.2023 |  |
| 15. | Захист курсової роботи | 07.06.2023 |  |

Студент

(підпис)

Керівник Муха І. П.

(підпис) (прізвище, ім’я, по батькові)

"\_\_\_" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 95 сторінок, 25 рисунків, 21 таблиця, 7 посилань.

Мета роботи: коректна реалізація програмного забезпечення для упорядкування масивів за методом сортування злиттям (Д. фон Неймана), методом швидкого сортування та методом інтроспективного сортування

Вивчено методи сортування злиттям, швидкого сортування та інтроспективного сортування для впорядкування масивів

Виконана програмна реалізація алгоритму сортування злиттям (Д. фон Неймана), швидкого сортування та інтроспективного сортування для сортування масивів цілих чисел, згенерованих випадковим чином.

У першому розділі описане завдання, яке необхідно виконати. У другому розділі описані теоретичні відомості, необхідні для виконання курсової роботи. У 3 розділі описані алгоритми сортування, а також основний алгоритм програми. У розділі №4 зображена діаграма класів програмного забезпечення, та стандартні й користувацькі методи, які використовуються в ньому. У розділі №5 здійснено тестування програми в різних варіантах використання користувачем. У розділі №6 описана інструкція користування програмним забезпеченням та технічні вимоги до нього. У розділі №7 здійснено аналіз результатів, а також наведено графіки практичної складності алгоритмів.

МАСИВ, СОРТУВАННЯ, ШВИДКЕ СОРТУВАННЯ, ІНТРОСПЕКТИВНЕ СОРТУВАННЯ, СОРТУВАННЯ ЗЛИТТЯМ, РОЗДІЛЯЙ І ВОЛОДАРЮЙ, РЕКУРСІЯ, ПІДМАСИВ, ІНДЕКС, ОПОРНИЙ ЕЛЕМЕНТ.

Зміст

[Вступ 6](#__RefHeading___Toc8225_3874632622)

[1 Постановка задачі 7](#__RefHeading___Toc8227_3874632622)

[2 Теоретичні відомості 8](#__RefHeading___Toc8229_3874632622)

[3 Опис алгоритмів 11](#__RefHeading___Toc8231_3874632622)

[3.1. Загальний алгоритм 11](#__RefHeading___Toc8233_3874632622)

[3.2. Алгоритм методу сортування злиттям (Д. фон Неймана) 12](#__RefHeading___Toc8235_3874632622)

[3.3. Алгоритм методу швидкого сортування 14](#__RefHeading___Toc8237_3874632622)

[3.4. Алгоритм методу інтроспективного сортування 15](#__RefHeading___Toc8239_3874632622)

[4 Опис програмного забезпечення 18](#__RefHeading___Toc8241_3874632622)

[4.1. Діаграма класів програмного забезпечення 18](#__RefHeading___Toc8243_3874632622)

[4.2. Опис методів частин програмного забезпечення 19](#__RefHeading___Toc8245_3874632622)

[4.2.1. Стандартні методи 19](#__RefHeading___Toc8247_3874632622)

[4.2.2. Користувацькі методи 22](#__RefHeading___Toc8249_3874632622)

[5 Тестування програмного забезпечення 30](#__RefHeading___Toc8251_3874632622)

[5.1. План тестування 30](#__RefHeading___Toc8253_3874632622)

[5.2. Приклади тестування 31](#__RefHeading___Toc8255_3874632622)

[6 Інструкція користувача 41](#__RefHeading___Toc8257_3874632622)

[6.1. Робота з програмою 41](#__RefHeading___Toc8259_3874632622)

[6.2. Формат вхідних та вихідних даних 48](#__RefHeading___Toc8261_3874632622)

[6.3. Системні вимоги 48](#__RefHeading___Toc8263_3874632622)

[7 Аналіз результатів 49](#__RefHeading___Toc8265_3874632622)

[Висновки 62](#__RefHeading___Toc8267_3874632622)

[Перелік посилань 63](#__RefHeading___Toc8269_3874632622)

[Додаток А Технічне завдання 64](#__RefHeading___Toc8271_3874632622)

[Додаток Б Тексти програмного коду 67](#__RefHeading___Toc8273_3874632622)

Вступ

Алгоритми сортування є досить важливими для багатьох задач, які використовуються в реальному світі, а також мають застосування в інших алгоритмах, таких як алгоритми пошуку, алгоритми в структурах даних, алгоритми баз даних та ін.

Алгоритми сортування широко використовуються в онлайн-торгівлі. Коли користувач хоче впорядкувати товари в певному порядку, наприклад, у порядку зростання ціни, то сортування великої кількості товарів може зайняти дуже багато часу. Оскільки швидкість роботи вебсайту чи мобільного застосунку прямо впливає на бажання ним користуватися, то для інтернет-магазинів дуже важливим є вибір максимально швидкого алгоритму сортування, який знизить час очікування користувача до мінімуму.

Платформи соціальних мереж використовують алгоритми сортування для персоналізації стрічок користувачів і підвищення їхньої активності. Ці алгоритми аналізують вподобання, поведінку та взаємодію користувачів, щоб визначити релевантність і рейтинг контенту. Сортуючи та представляючи дописи, історії чи рекомендації на основі індивідуальних вподобань, соціальні медіа-платформи можуть надавати кожному користувачеві індивідуальний досвід. Це гарантує, що користувачі бачать контент, який, швидше за все, їх зацікавить, підвищуючи їхню залученість і задоволеність.

Отже, алгоритми сортування є досить важливими в сучасному світі, а їхнє пришвидшення дає перевагу бізнесам та іншим, більш комплексним і складним алгоритмам.

# Постановка задачі

Розробити програмне забезпечення для упорядкування масивів наступними методами:

а) метод сортування злиттям (Д. фон Неймана);

б) метод швидкого сортування;

в) метод інтроспективного сортування.

Вхідними даними для даної роботи є масив цілих чисел, який заданий у вигляді:

,

де – масив цілих чисел, – числа масиву.Програмне забезпечення повинно обробляти масив, згенерований випадковим чином, розмірність якого знаходиться в межах від 100 до 50000.

Вихідними даними для даної роботи є масив відсортованих у порядку зростання чисел виду

,

де – масив цілих чисел, – числа масиву, причому . Отриманий масив, а також (за бажанням користувача) дані для оцінки якості алгоритму (кількість порівнянь, кількість перестановок і час сортування) виводяться на екран, .

# Теоретичні відомості

Масив із n цілих чисел можна задати таким чином:

|  |
| --- |
| (2.1) |

де:

– масив цілих чисел, – числа масиву

Тоді відсортований масив можна отримати одним із наступних методів.

* 1. Метод сортування злиттям (Д. фон Неймана) [1]

Алгоритм сортування зливанням (merge sort) побудований за парадигмою «розділяй і володарюй». Спочатку ми розділяємо вхідний масив на два підмасиви, кожен по n/2 елементів.

Далі Рекурсивно впорядковуємо два підмасиви за допомогою сортування зливанням.

Рекурсія «вичерпується», коли послідовність для сортування має довжину 1 ­- у цьому випадку нема що робити, адже будь-яка послідовність завдовжки 1 вже впорядкована.

Після цього зливаємо два впорядковані підмасиви в один для отримання впорядкованого масиву, допоки не отримаємо остаточний відсортований масив.

Це відбувається за допомогою допоміжної процедури де A — масив, р, q, r - такі його індекси, що . Процедура передбачає, що підмасиви i вже впорядковані. Вона зливає їх в один впорядкований підмасив, що заміщує поточний підмасив .

* 1. Метод швидкого сортування [1]

Метод швидкого сортування, як і сортування злиттям, побудований за парадигмою «розділяй і володарюй». Основою методу швидкого сортування є процедура , яка переміщує опорний елемент у правильну позицію відсортованого масиву, розміщуючи менші елементи ліворуч від опорного елемента, а більші ­- праворуч.

Нехай опорний елемент — останній елемент масиву (2.1), причому . Тоді в результаті виконання ми отримаємо:

де:

– масив цілих чисел, – числа масиву, ­- опорний елемент масиву, причому

Логіка процедури така: спочатку, ми починаємо з крайнього лівого елемента масиву і відстежуємо індекс менших (або рівних) елементів від опорного як . Під час обходу масиву , якщо ми знаходимо менший елемент, ніж опорний, то обмінюємо місцями поточний елемент з елементом . Інакше, ми ігноруємо поточний елемент. Наприклад, якщо масив , , то

Отже внаслідок виконання процедури опорний елемент отримує індекс 4, а менші елементи розташовуються справа від опорного, а більші ­- зліва. Далі алгоритм швидкого сортування рекурсивно викликається для

* 1. Метод інтроспективного сортування

Метод інтроспективного сортування — це гібридний алгоритм, тобто використовує декілька алгоритмів сортування у своїй основі. Інтроспективне сортування починається з виклику процедури методу швидкого сортування. Далі обирається один із трьох методів:

1. Якщо максимальна глибина рекурсії швидкого сортування може перевищити певну межу, що складає , де n — кількість елементів масиву, то викликається метод пірамідального сортування
2. Якщо розмір підмасиву достатньо малий, викликається метод сортування вставками ­- . Максимальна довжина підмасиву для виклику сортування вставками визначена як
3. Інакше викликається метод швидкого сортування

# Опис алгоритмів

Перелік всіх основних змінних та їхнє призначення наведено в таблиці 3.1.

Таблиця 3.1 – Основні змінні та їхні призначення

|  |  |
| --- | --- |
| Змінна | Призначення |
| unsortedArray | Початковий масив |
| sortedArray | Впорядкований масив |
| min\_number | Нижня межа генерування |
| max\_number | Верхня межа генерування |
| sorting \_data | Зберігання додаткової інформації про масив (розмірність, метод сортування, кількість порівнянь, кількість перестановок, час сортування) |

## Загальний алгоритм

1. ПОЧАТОК
2. Обрати розмірність масиву n (за замовчуванням — 100)
3. Обрати діапазон генерування чисел
   1. Обрати нижню межу генерування (за замовчуванням — 0)
   2. Обрати верхню межу генерування (за замовчуванням — 100)
4. Натиснути кнопку генерування масиву
   1. Цикл проходу від 0 до n – 1
      1. Згенерувати число з поточним індексом
      2. Додати число до масиву
   2. Вивести згенерований масив
5. Обрати метод сортування (за замовчуванням — сортування злиттям)
6. Натиснути кнопку сортування масиву
   1. ЯКЩО обраний метод сортування злиттям, ТО сортувати масив відповідно до цього методу (підрозділ 3.2)
   2. ЯКЩО обраний метод швидкого сортування, ТО сортувати масив відповідно до цього методу (підрозділ 3.3)
   3. ЯКЩО обраний метод інтроспективного сортування, ТО сортувати масив відповідно до цього методу (підрозділ 3.4)
   4. Зберегти відсортований масив
   5. Вивести відсортований масив
   6. Зберегти додаткові дані про сортування
7. [За бажанням користувача] Обрати відображення процесу сортування у вигляді анімації
   1. ЯКЩО розмірність масиву не перевищує 300, ТО
      1. Вивести на графік початковий масив
      2. Запустити обраний алгоритм сортування
         1. ЯКЩО значення масиву змінюються, ТО змінити ці значення на графіку
         2. Чекати обраний час
8. [За бажанням користувача] Зберегти дані в обраний файл
9. КІНЕЦЬ

## Алгоритм методу сортування злиттям (Д. фон Неймана)[2]

Функція

1. ПОЧАТОК
2. ЯКЩО , ТО повернути
3. Обчислити значення:
4. Викликати функцію сортування злиттям для індексів масиву, що не перевищують mid:
5. Викликати функцію сортування злиттям для індексів масиву, що перевищують mid:
6. Викликати функцію злиття:
7. КІНЕЦЬ

Допоміжна функція

1. ПОЧАТОК
2. Створити підмасив L, скопіювавши елементи масиву arr з індексами від begin до mid
3. Створити підмасив R, скопіювавши елементи масиву arr з індексами від mid + 1 до end
4. Ініціалізувати змінні i = 0, j = 0, k = 0
5. ЦИКЛ проходу елементів обох підмасивів ПОКИ не пройдено один із підмасивів
   1. ЯКЩО елемент лівого підмасиву з індексом менше від елемента правого підмасиву з індексом , ТО
      1. Присвоїти значення масиву значення
      2. Збільшити на 1
   2. ІНАКШЕ
      1. Присвоїти значення масиву значення
      2. Збільшити на 1
6. ЦИКЛ проходу елементів лівого масиву від до довжини підмасиву
   * 1. Присвоїти значення масиву значення
     2. Збільшити на 1
     3. Збільшити на 1
7. ЦИКЛ проходу елементів правого масиву від до довжини підмасиву
   * 1. Присвоїти значення масиву значення
     2. Збільшити на 1
     3. Збільшити на 1
8. КІНЕЦЬ

## Алгоритм методу швидкого сортування[3]

Функція

1. ПОЧАТОК
2. ЯКЩО ТО
   1. Ініціалізувати змінну значенням, що повертає допоміжна функція
   2. Викликати функцію швидкого сортування для лівої частини масиву відносно :
   3. Викликати функцію швидкого сортування для правої частини масиву відносно :
3. КІНЕЦЬ

Допоміжна функція

1. ПОЧАТОК
2. Обрати опорний елемент як останній елемент підмасиву:
3. Ініціалізувати змінну значенням
4. ЦИКЛ проходу всіх елементів підмасиву
   1. ЯКЩО поточний елемент масиву менший за опорний, ТО
      1. збільшити на 1
      2. обміняти місцями поточний елемент та елемент з індексом
5. Обміняти місцями останній елемент масиву з елементом цього масиву з індексом
6. Повернути значення
7. КІНЕЦЬ

## Алгоритм методу інтроспективного сортування[4]

Початкова функція

1. ПОЧАТОК
2. Ініціалізувати змінну значенням
3. Викликати функцію
4. КІНЕЦЬ

Функція

1. ПОЧАТОК
2. ЯКЩО довжина масиву не перевищує 16, ТО викликати функцію сортування вставками для цього масиву:
3. ІНАКШЕ ЯКЩО значення дорівнює 0, ТО викликати функцію пірамідального сортування:
4. ІНАКШЕ ініціалізувати змінну медіанним значенням серед першого, середнього та останнього елемента масиву
5. Обміняти місцями останній елемент масиву та елемент масиву з індексом
6. Ініціалізувати змінну значенням, що повертає допоміжна функція
7. Викликати функцію інтроспективного сортування для лівої частини масиву відносно :
8. Викликати функцію швидкого сортування для правої частини масиву відносно :
9. КІНЕЦЬ

Функція

1. ПОЧАТОК
2. ЦИКЛ проходу всіх елементів масиву для індексів від до
   1. Ініціалізувати змінну значенням поточного елемента
   2. Ініціалізувати змінну індексом поточного елемента, зменшеного на 1
   3. ЦИКЛ ПОКИ і
      1. Присвоїти значення елемента наступному елементу
      2. Зменшити j на 1
      3. Присвоїти значення змінної елементу
3. КІНЕЦЬ

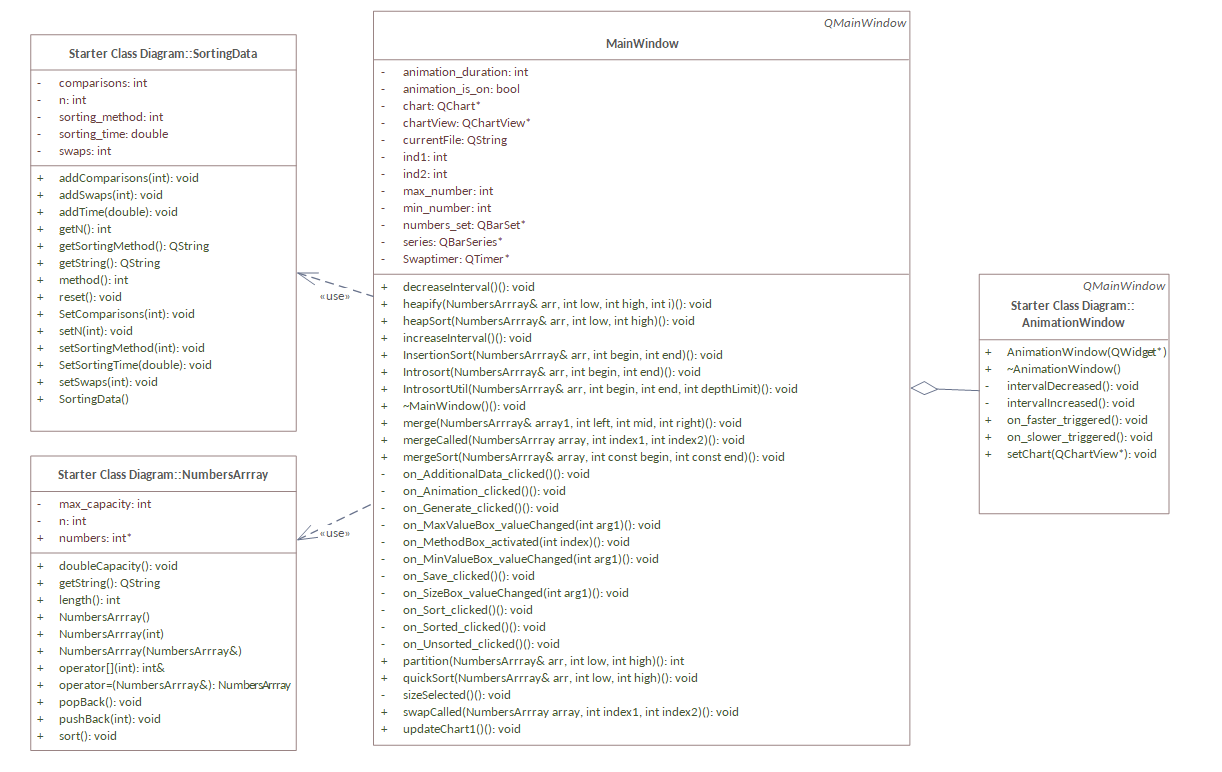
Функція

1. ПОЧАТОК
2. ЦИКЛ проходу елементів масиву від половини довжини масиву до
   1. Викликати допоміжну функцію для поточного елемента масиву
3. ЦИКЛ проходу елементів масиву від індексу до
   1. Обміняти значення поточного елемента масиву з елементом масиву з індексом
   2. Викликати допоміжну функцію
4. КІНЕЦЬ
5. Функція
6. ПОЧАТОК
7. Визначити найбільший елемент серед кореня , його правого та лівого нащадка та
8. ЯКЩО кореневий елемент не більший за своїх нащадків та , ТО
   1. Обміняти місцями та найбільший з його нащадків
   2. Викликати функцію , де ­- найбільший елемент з , та
9. КІНЕЦЬ

# Опис програмного забезпечення

## Діаграма класів програмного забезпечення

Діаграма класів зображена на рисунку 4.1

Рисунок 4.1 – Діаграма класів

## Опис методів частин програмного забезпечення

### Стандартні методи

У таблиці 4.1 наведено стандартні функції, методи та класи мови програмування C++ та фреймворку QT

Таблиця 4.1 – Стандартніметоди[5]

| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | QApplication | exec | Вхід до основного циклу програми | – | int | <QApplication> |
| 2 | QString | number | Перетворення числа в об’єкт класу QString | int, int | QString | <QString> |
| 3 | QFile | open | Відкриття файлу | QIODeviceBase::OpenMode, QFileDevice::Permissions | bool | <QFile> |
| 4 | QFile | close | Закриття файлу | – | void | <QFile> |
| 5 | QFileDialog | getSaveFileName | Повернення імені файлу, вибраного користувачем | QWidget\*, const QString, const QString, const QString, QString\*, QFileDialog::Options | QString | <QFileDialog> |
| 6 | QTextStream | operator<< | Додавання інформації в потік QTextStream | const QString & | QTextStream | <QTextStream> |
| 7 | QPalette | setColor | Встановлення кольору | Qpalette::ColorRole, const QColor& | void | <QPalette> |
| 8 | QBarSet | remove | Вилучення декількох значень | const int, const int | void | <QBarSet> |
| 9 | QBarSet | operator<< | Додавання значення | const qreal & | QBarSet & | <QBarSet> |
| 10 | QBarSet | replace | Заміна значення | const int, const qreal value | void | <QBarSet> |
| 11 | QChart | update | Оновлення графіка | – | void | <QChart> |
| 12 | QChart | addSeries | Додавання ряду даних | QAbstractSeries\* | void | <QChart> |
| 13 | QChart | setTitle | Встановлення заголовоку | const QString & | void | <QChart> |
| 14 | QChart | setAnimationOptions | Встановлення параметрів анімації | QChart::AnimationOptions | void | <QChart> |
| 15 | QChart | createDefaultAxes | Додавання осей за замовчуванням | – | void | <QChart> |
| 16 | QBarSeries | append | Додавання множини значень до ряду | const Qlist<QbarSet\*>& | bool | <QBarSeries> |
| 19 | QValueAxis | setRange | Встановлення меж осі | qreal, qreal | void | <QValueAxis> |
| 20 | QDebug | qDebug() | Створення об’єкта класу QDebug для запису інформації налагодження | const char\* | – | <QDebug> |
| 21 | QDebug | operator<< | Запис інформації для налагодження | const QString & | QDebug & | <QDebug> |
| 22 | QObject | connect | Підключення сигналу до слоту | const QObject\*, PointerToMemberFunction, const QObject\*, PointerToMemberFunction, | QMetaObject::Connection | <QObject> |
| 23 | QTimer | setInterval | Встановити інтервал | int | void | <QTimer> |
| 24 | QTimer | timeout | Сигнал закінчення таймера | – | void | <QTimer> |
| 25 | QTimer | start | Запуск таймера | int | void | <QTimer> |
| 26 | QTimer | stop | Зупинення таймера | – | void | <QTimer> |
| 27 | ­- | srand | Створення початкової точки генерації чисел | unsigned int | void | <cstdlib> |
| 28 | ­- | time | Повернення часу | time\_t | time\_t\* | <ctime> |
| 29 | ­- | swap | Обміняти значення | T&, T& | void | <string\_view> |
| 30 | std::queue<int> | push | Додавання елемента в чергу | const int& | void | <queue> |
| 31 | std::queue<int> | pop | Видалення елемента з черги | – | void | <queue> |
| 32 | std::queue<int> | front | Отримання поточного елемента черги | – | void | <queue> |
| 33 | std::chrono::high\_resolution\_clock | now | Отримання поточного часу | – | static std::chrono::time\_point<std::chrono::high\_resolution\_clock> | <chrono> |
| 34 | std::chrono::duration<microseconds> | duration\_cast | Перетворення типу std::chrono::duration в інший тип | const std::chrono::duration<Rep,Period>& | std::chrono::microseconds& | <chrono> |

### Користувацькі методи

У таблиці 4.2 наведено користувацькі функції, методи та класи

Таблиця 4.2 – Користувацькі методи

| № п/п | Назва класу | Назва функції | Призначення функції | Опис вхідних параметрів | Опис вихідних параметрів | Заголовний файл |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | NumbersArrray | doubleCapacity | Динамічне подвоєння розмірності масиву | – | void | numbersarray.h |
| 2 | NumbersArrray | getString | Отримання чисел у форматі рядка | – | QString | numbersarray.h |
|  | NumbersArrray | length | Отримання довжини масиву | – | int | numbersarray.h |
| 3 | NumbersArrray | NumbersArrray | Конструктор за замовчуванням | – | – | numbersarray.h |
| 4 | NumbersArrray | NumbersArrray | Конструктор з параметрами | int | – | numbersarray.h |
| 5 | NumbersArrray | NumbersArrray | Конструктор копіювання | NumbersArrray& | – | numbersarray.h |
| 6 | NumbersArrray | operator[] | Звертання до елементів масиву | int | int& | numbersarray.h |
| 7 | NumbersArrray | operator= | Копіювання елементів до іншого об’єкта | NumbersArrray& | NumbersArrray | numbersarray.h |
| 8 | NumbersArrray | popBack | Видалення останнього елемента | – | void | numbersarray.h |
| 9 | NumbersArrray | pushBack | Додавання елемента в кінець | int | void | numbersarray.h |
| 10 | SortingData | addComparisons | Додавання кількості порівнянь | int | void | sortingdata.h |
| 11 | SortingData | addSwaps | Додавання кількості перестановок | int | void | sortingdata.h |
| 12 | SortingData | addTime | Додавання часу | double | void | sortingdata.h |
| 13 | SortingData | getN | Отримання розмірності масиву | – | int | sortingdata.h |
| 14 | SortingData | getSortingMethod | Отримання методу сортування у рядковому форматі | – | QString | sortingdata.h |
| 15 | SortingData | getString | Отримання даних про сортування у рядковому форматі | – | QString | sortingdata.h |
| 16 | SortingData | method | Отримання методу сортування у числовому форматі | – | int | sortingdata.h |
| 17 | SortingData | reset | Стерти дані сортування | – | void | sortingdata.h |
| 18 | SortingData | setComparisons | Встановити кількість порівнянь | int | void | sortingdata.h |
| 19 | SortingData | setN | Встановити кількість чисел | int | void | sortingdata.h |
| 20 | SortingData | setSorrtingMethod | Встановити метод сортування | int | void | sortingdata.h |
| 21 | SortingData | SetSortingTime | Встановити час сортування | double | void | sortingdata.h |
| 22 | SortingData | setSwaps | Встановити кількість перестановок | int | void | sortingdata.h |
| 23 | SortingData | SortingData | Конструктор за замовчуванням | – | – | sortingdata.h |
| 24 | MainWindow | MainWindow | Конструктор головного вікна | QWidget \* | – | mainwindow.h |
| 25 | MainWindow | ~MainWindow | Деструктор головного вікна | – | – | mainwindow.h |
| 26 | MainWindow | on\_AdditionalData\_clicked | Обробка натискання на кнопку “Додаткові дані” | – | void | mainwindow.h |
| 27 | MainWindow | on\_Animation\_clicked | Обробка натискання на кнопку “Анімація” | – | void | mainwindow.h |
| 28 | MainWindow | on\_Generate\_clicked | Обробка натискання на кнопку “Згенерувати масив випадкових чисел” | – | void | mainwindow.h |
| 29 | MainWindow | on\_MaxValueBox\_valueChanged | Обробка вибору верхньої межі генерування чисел | int | void | mainwindow.h |
| 30 | MainWindow | on\_MinValueBox\_valueChanged | Обробка вибору нижньої межі генерування чисел | int | void | mainwindow.h |
| 31 | MainWindow | on\_Save\_clicked | Обробка натискання на кнопку “Зберегти результати” | – | void | mainwindow.h |
| 32 | MainWindow | on\_SizeBox\_valueChanged | Обробка вибору розмірності масиву | int | void | mainwindow.h |
| 33 | MainWindow | on\_Sort\_clicked | Обробка натискання на кнопку “Сортувати” | – | void | mainwindow.h |
| 34 | MainWindow | on\_Sorted\_clicked | Обробка натискання на кнопку “Відсортований масив” | – | void | mainwindow.h |
| 35 | MainWindow | on\_Unsorted\_clicked | Обробка натискання на кнопку “Початковий масив” | – | void | mainwindow.h |
| 36 | MainWindow | swapCalled | Сигнал про виклик процедури перестановки | NumbersArrray, int, int | void | mainwindow.h |
| 37 | MainWindow | mergeCalled | Сигнал про виклик процедури злиття | NumbersArrray, int, int | void | mainwindow.h |
| 38 | MainWindow | updateChart1 | Оновлення графіка з анімацією | – | void | mainwindow.h |
| 39 | MainWindow | decreaseInterval | Зменшення інтервалу анімації | – | void | mainwindow.h |
| 40 | MainWindow | increseInterval | Збільшення інтервалу анімації | – | void | mainwindow.h |
| 41 | MainWindow | partition | Допоміжна функція швидкого сортування | NumbersArrray&, int, int | int | mainwindow.h |
| 42 | MainWindow | quickSort | Функція швидкого сортування | NumbersArrray&, int, int | void | mainwindow.h |
| 43 | MainWindow | merge | Допоміжна функція сортування злиттям | NumbersArrray&, int, int | void | mainwindow.h |
| 44 | MainWindow | mergeSort | Функція сортування злиттям | NumbersArrray&, int, int | void | mainwindow.h |
| 45 | MainWindow | heapify | Допоміжна функція пірамідального сортування | NumbersArrray&, int, int, int | void | mainwindow.h |
| 46 | MainWindow | heapSort | Функція пірамідального сортування | NumbersArrray&, int, int | void | mainwindow.h |
| 47 | MainWindow | InsertionSort | Функція сортування вставками | NumbersArrray&, int, int | void | mainwindow.h |
| 48 | MainWindow | Introsort | Функція початку інтроспективного сортування | NumbersArrray&, int, int | void | mainwindow.h |
| 49 | MainWindow | IntrosortUtil | Функція інтроспективного сортування | NumbersArrray&, int, int, int | void | mainwindow.h |
| 50 | AnimationWindow | AnimationWindow | Конструктор вікна анімації | QWidget\* | – | animationwindow.h |
| 51 | AnimationWindow | ~AnimationWindow | Деструктор вікна анімації | – | – | animationwindow.h |
| 52 | AnimationWindow | intervalDecreased | Сигнал зменшення інтервалу | – | void | animationwindow.h |
| 53 | AnimationWindow | intervalIncreased | Сигнал збільшення інтервалу | – | void | animationwindow.h |
| 54 | AnimationWindow | on\_faster\_triggered | Обробка натискання кнопки “Прискорити анімацію” | – | void | animationwindow.h |
| 55 | AnimationWindow | on\_slower\_triggered | Обробка натискання кнопки “Уповільнити анімацію” | – | void | animationwindow.h |

# Тестування програмного забезпечення

## План тестування

Розробимо набір тестів для перевірки коректності роботи програми. Тестування знаходяться в таблицях 5.1 ­- 5.15

1. Тестування коректності генерації масиву
2. Тестування коректності роботи методів 1,2,3.
   1. Перевірка коректності роботи методу 1.
   2. Перевірка коректності роботи методу 2.
   3. Перевірка коректності роботи методу 3.
3. Тестування сортування при відсутності згенерованого масиву
4. Тестування побудови графіків
   1. Тестування коректності виводу початкового масиву на графік при наявності відсортованого масиву
   2. Тестування коректності виводу початкового масиву на графік при відсутності відсортованого масиву
   3. Тестування відображення графіка при зміні розмірності масиву
   4. Тестування відображення графіка при зміні діапазону чисел
   5. Тестування роботи при масиві з розмірністю більше 300
5. Тестування роботи анімації
   1. Тестування коректності анімації для методу 1
   2. Тестування коректності анімації для методу 2
   3. Тестування коректності анімації для методу 3
   4. Тестування анімації після повторного натискання на кнопку
   5. Тестування анімації після перервання анімації закриттям вікна
6. Тестування збереження даних у файл
   1. Тестування коректності збережених даних

Тестування при відсутності вибраного файлу

## Приклади тестування

Побудуємо таблиці для відображення результатів тестування

Таблиця 5.1 ‑ Тестування коректності генерації масиву

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність генерації масиву випадковим чином (кількість елементів та межі) |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 100  Діапазон від 0 до 10 |
| Схема проведення тесту | Заповнити вхідні дані та натиснути кнопку “Згенерувати масив випадкових чисел” |
| Очікуваний результат | Виведення згенерованого масиву з 10 елементів, кожен з яких не більше 10 і не менше 0 |
| Стан програми після проведення випробувань | Виведення елементів масиву:  9 10 9 2 6 5 2 4 0 3 |

Таблиця 5.2 ‑ Перевірка коректності роботи методу 1 ‑ сортування злиттям (Д. фон Неймана)

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність сортування згенерованого масиву методом сортування злиттям |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Згенерований масив цілих чисел:  9 10 9 2 6 5 2 4 0 3 |
| Схема проведення тесту | Згенерувати масив цілих чисел та натиснути кнопку “Сортувати” |
| Очікуваний результат | Виведення відсортованого масиву з 10 елементів |
| Стан програми після проведення випробувань | Виведення елементів масиву:  0 2 2 3 4 5 6 9 9 10 |

Таблиця 5.3 ‑ Перевірка коректності роботи методу 2 ‑ швидкого сортування

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність сортування згенерованого масиву методом швидкого сортування |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Згенерований масив цілих чисел:  1 2 0 8 5 6 3 5 1 4 |
| Схема проведення тесту | Згенерувати масив цілих чисел, обрати метод швидкого сортування та натиснути кнопку “Сортувати” |
| Очікуваний результат | Виведення відсортованого масиву з 10 елементів |
| Стан програми після проведення випробувань | Виведення елементів масиву:  0 1 1 2 3 4 5 5 6 8 |

Таблиця 5.4 ‑ Перевірка коректності роботи методу 3 ‑ інтроспективного сортування

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність сортування згенерованого масиву методом інтроспективного сортування |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Згенерований масив цілих чисел:  8 0 5 9 3 2 1 1 1 2 |
| Схема проведення тесту | Згенерувати масив цілих чисел, обрати метод інтроспективного сортування та натиснути кнопку “Сортувати” |
| Очікуваний результат | Виведення відсортованого масиву з 10 елементів |
| Стан програми після проведення випробувань | Виведення елементів масиву:  0 1 1 1 2 2 3 5 8 9 |

Таблиця 5.5 ‑ Тестування сортування при відсутності згенерованого масиву

|  |  |
| --- | --- |
| Мета тесту | Перевірити результат роботи програми, якщо початковий масив пустий |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 100  Діапазон від 0 до 10  Масив не згенерований |
| Схема проведення тесту | Натиснути кнопку “Сортувати” |
| Очікуваний результат | Виведення помилки |
| Стан програми після проведення випробувань | Виведення вікна з помилкою:  “Масив для сортування не згенерований” |

Таблиця 5.6 ‑ Тестування коректності виводу початкового масиву на графік при наявності відсортованого масиву

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність виведення невпорядкованого масиву на графік при наявності відсортованого |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Діапазон від 0 до 10  Початковий масив:  9 5 2 0 5 7 2 9 4 5  Впорядкований масив:  0 2 2 4 5 5 5 7 9 9 |
| Схема проведення тесту | Згенерувати масив, обравши межі від 0 до 10, натиснути кнопку “Сортувати”, натиснути кнопку “Анімація” |
| Очікуваний результат | Виведення початкового масиву у вигляді графіка на екран |
| Стан програми після проведення випробувань | Виведення графіка, що відповідає початковому масиву |

Таблиця 5.7 ‑ Тестування коректності виводу початкового масиву на графік при відсутності відсортованого масиву

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність виведення невпорядкованого масиву на графік при відсутності відсортованого |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Діапазон від 0 до 10  Початковий масив:  7 10 5 9 4 2 8 4 9 6 |
| Схема проведення тесту | Згенерувати масив, обравши межі від 0 до 10, натиснути кнопку “Анімація” |
| Очікуваний результат | Виведення початкового масиву у вигляді графіка на екран |
| Стан програми після проведення випробувань | Виведення графіка, що відповідає початковому масиву |

Таблиця 5.8 ‑ Тестування відображення графіка при зміні розмірності масиву

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність відображення горизонтальної осі |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Діапазон від 0 до 10  Початковий масив:  9 7 7 2 4 0 2 6 10 3  Зміна кількості чисел до 100  Новий початковий масив:  5 6 8 6 7 2 6 8 2 7 6 6 7 1 5 9 10 4 0 10 |
| Схема проведення тесту | Згенерувати масив на 10 елементів, натиснути кнопку “Анімація”, закрити вікно анімації, згенерувати масив на 100 елементів, натиснути кнопку “Анімація” |
| Очікуваний результат | Горизонтальна вісь змінена і всі числа відображаються на графіку |
| Стан програми після проведення випробувань | Всі 20 чисел присутні на графіку й відображені коректно |

Таблиця 5.9 ‑ Тестування відображення графіка при зміні діапазону чисел

|  |  |
| --- | --- |
| Мета тесту | Перевірити правильність відображення вертикальної осі |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Діапазон від 0 до 10  Початковий масив:  8 1 9 5 2 6 4 3 5 8  Зміна діапазону: від 20 до 50  Новий початковий масив:  29 20 29 23 35 44 22 21 38 22 |
| Схема проведення тесту | Згенерувати масив з початковим діапазоном, натиснути кнопку “Анімація”, закрити вікно анімації, згенерувати масив на з новим діапазоном, натиснути кнопку “Анімація” |
| Очікуваний результат | Вертикальна вісь змінена і всі числа відображаються на графіку |
| Стан програми після проведення випробувань | Вертикальна вісь змінюється від 20 до 50, відображаючи елементи масиву коректно |

Таблиця 5.10 ‑ Тестування при масиві з розмірністю більше 300

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність роботи програми при масиві більшому, ніж максимальний для виведення анімації |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 400  Діапазон від 0 до 100  Масив згенеровано |
| Схема проведення тесту | Згенерувати масив на 400 елементів, натиснути кнопку “Анімація” |
| Очікуваний результат | Анімація не відображається |
| Стан програми після проведення випробувань | Виведення помилки “Розмірність масиву занадто велика і перевищує 300” |

Таблиця 5.11 ‑ Тестування коректності анімації для методу 1 ­- сортування злиттям

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність анімації сортування злиттям |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Згенерований масив на 10 елементів |
| Схема проведення тесту | Згенерувати масив на 10 елементів, натиснути кнопку “Анімація” |
| Очікуваний результат | Анімація відображається правильно, відповідно до методу сортування злиттям |
| Стан програми після проведення випробувань | Виведення вікна з анімацією. Анімація працює коректно: спочатку зливаються перші два елементи, далі зливаються з третім, четвертий з п’ятим і тд. |

Таблиця 5.12 ‑ Тестування коректності анімації для методу 2 ­- швидкого сортування

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність анімації швидкого сортування |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Згенерований масив на 10 елементів |
| Схема проведення тесту | Згенерувати масив на 10 елементів, обрати метод швидкого сортування, натиснути кнопку “Анімація” |
| Очікуваний результат | Анімація відображається правильно, відповідно до методу швидкого сортування |
| Стан програми після проведення випробувань | Виведення вікна з анімацією. Анімація працює коректно: числа, менші за опорний розташовуються зліва, більші справа, і тд. за рекурсією |

Таблиця 5.13 ‑ Тестування коректності анімації для методу 3 ­- інтроспективного сортування

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність анімації інтроспективного сортування |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Згенерований масив на 10 елементів |
| Схема проведення тесту | Згенерувати масив на 10 елементів, обрати метод інтроспективного сортування, натиснути кнопку “Анімація” |
| Очікуваний результат | Анімація відображається правильно, відповідно до методу інтроспективного сортування |
| Стан програми після проведення випробувань | Виведення вікна з анімацією. Анімація працює коректно: розмірність масиву не перевищує 16, тому сортування відбувається сортуванням вставками |

Таблиця 5.14 ‑ Тестування анімації після повторного натискання на кнопку

|  |  |
| --- | --- |
| Мета тесту | Перевірити коректність роботи програми після повторного натискання на кнопку “Анімація” |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Згенерований масив на 100 елементів |
| Схема проведення тесту | Згенерувати масив на 100 елементів, натиснути кнопку “Анімація”, натиснути кнопку ще раз |
| Очікуваний результат | Відкривається додаткове вікно або анімація починається з початку |
| Стан програми після проведення випробувань | Виведення вікна з анімацією. Анімація починається з початку |

Таблиця 5.15 ‑ Тестування анімації після перервання анімації закриттям вікна

|  |  |
| --- | --- |
| Мета тесту | Перевірити, чи анімація відобразиться з початку після переривання |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Згенерований масив на 100 елементів |
| Схема проведення тесту | Згенерувати масив на 100 елементів, натиснути кнопку “Анімація”, закрити вікно до того часу, поки анімація завершиться |
| Очікуваний результат | Анімація починається з початку |
| Стан програми після проведення випробувань | Виведення вікна з анімацією. Анімація починається з початку |

Таблиця 5.16 ‑ Тестування коректності збережених даних

|  |  |
| --- | --- |
| Мета тесту | Перевірити, чи результати коректно зберігаються у файл |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Діапазон від 0 до 100  Початковий масив:  42 38 81 45 44 45 57 39 7 75  Метод швидкого сортування  Упорядкований масив:  7 38 39 42 44 45 45 57 75 81 |
| Схема проведення тесту | Згенерувати масив на 10 елементів, натиснути кнопку “Сортувати”, натиснути кнопку “Зберегти результати”, обрати ім’я файлу |
| Очікуваний результат | Дані збережено коректно |
| Стан програми після проведення випробувань | Файл збережено. Вміст файлу:  Кількість чисел (n): 10  Метод сортування: швидке сортування  Кількість порівнянь: 29  Кількість перестановок: 19  Час сортування: 1 мкс  Початковий масив: 42 38 81 45 44 45 57 39 7 75  Відсортований масив: 7 38 39 42 44 45 45 57 75 81 |

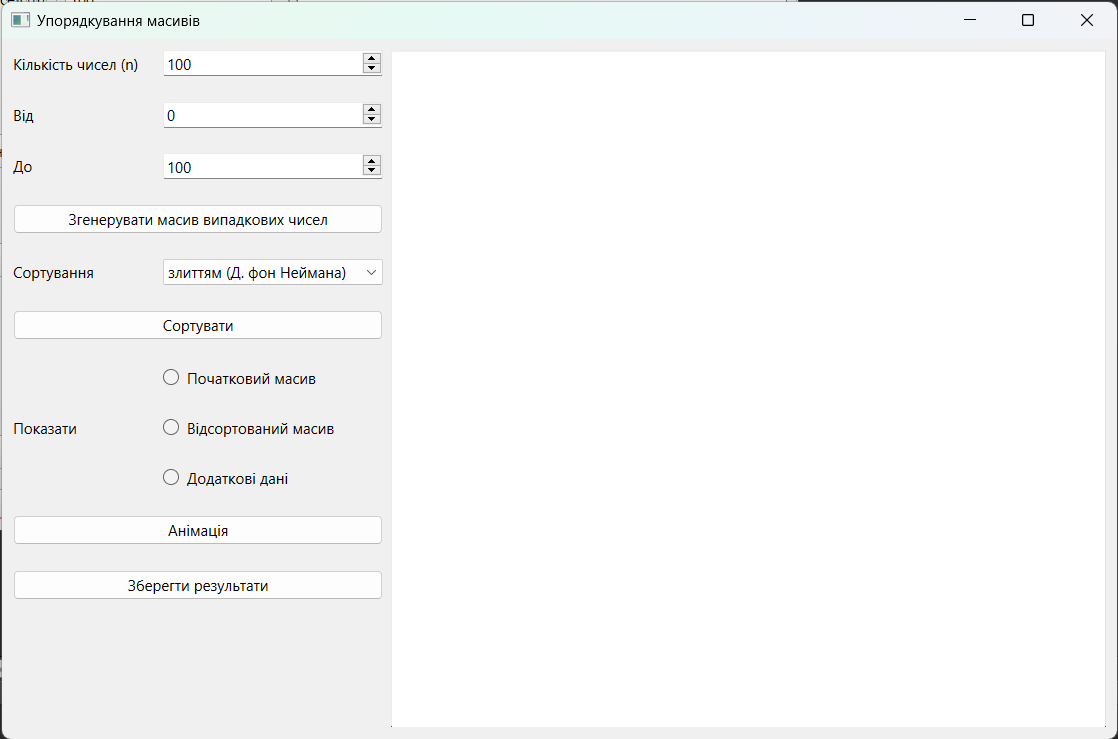
Таблиця 5.17 ‑ Тестування при відсутності вибраного файлу

|  |  |
| --- | --- |
| Мета тесту | Перевірити роботу програми, якщо файл не вибрано і вікно вибору закрите |
| Початковий стан програми | Відкрите вікно програми |
| Вхідні дані | Кількість чисел: 10  Діапазон від 0 до 100  Початковий масив:  42 38 81 45 44 45 57 39 7 75  Метод швидкого сортування  Упорядкований масив:  7 38 39 42 44 45 45 57 75 81 |
| Схема проведення тесту | Згенерувати масив на 10 елементів, натиснути кнопку “Сортувати”, натиснути кнопку “Зберегти результати”, закрити вікно вибору |
| Очікуваний результат | Виведення помилки |
| Стан програми після проведення випробувань | Виведення вікна з помилкою: “Неможливо зберегти файл” |

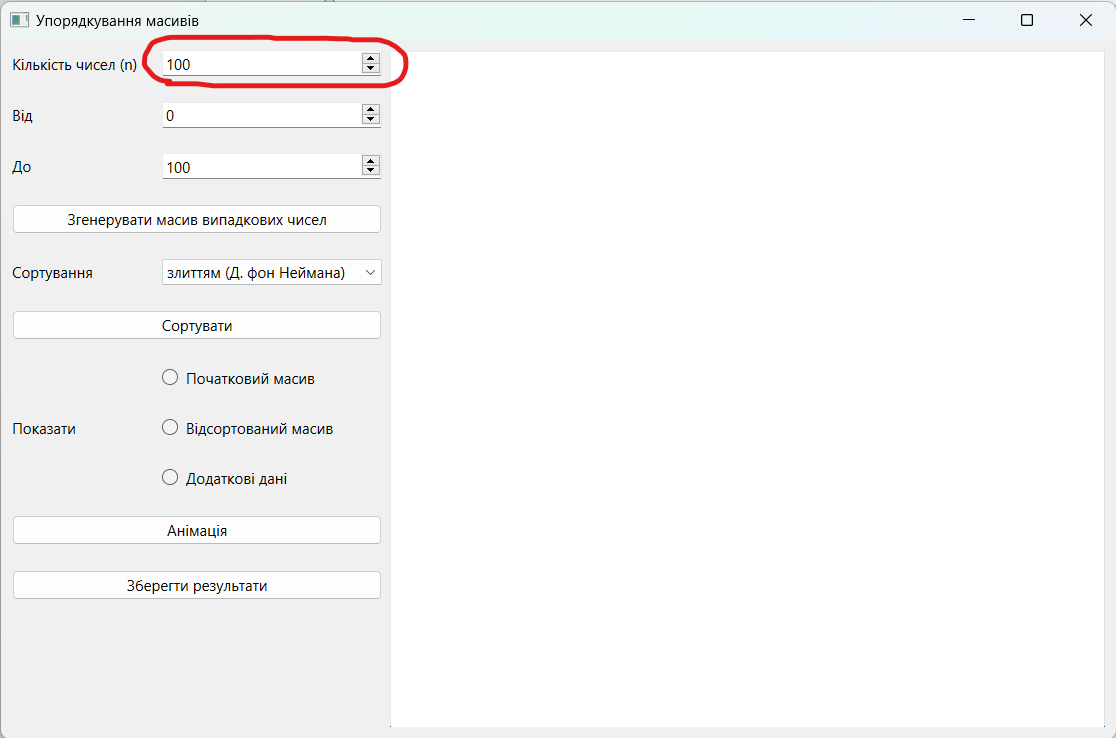
# Інструкція користувача

## Робота з програмою

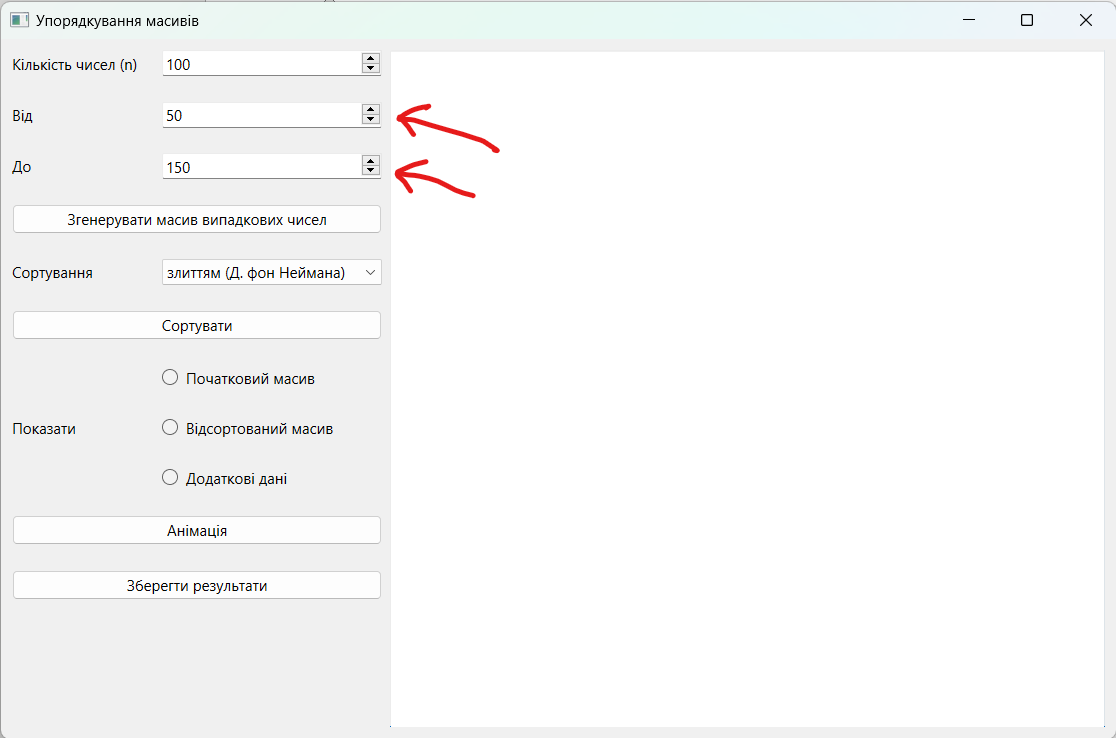
Після запуску виконавчого файлу з розширенням \*.exe, відкривається головне вікно програми (Рисунок 6.1).

Рисунок 6.1 – Головне вікно програми

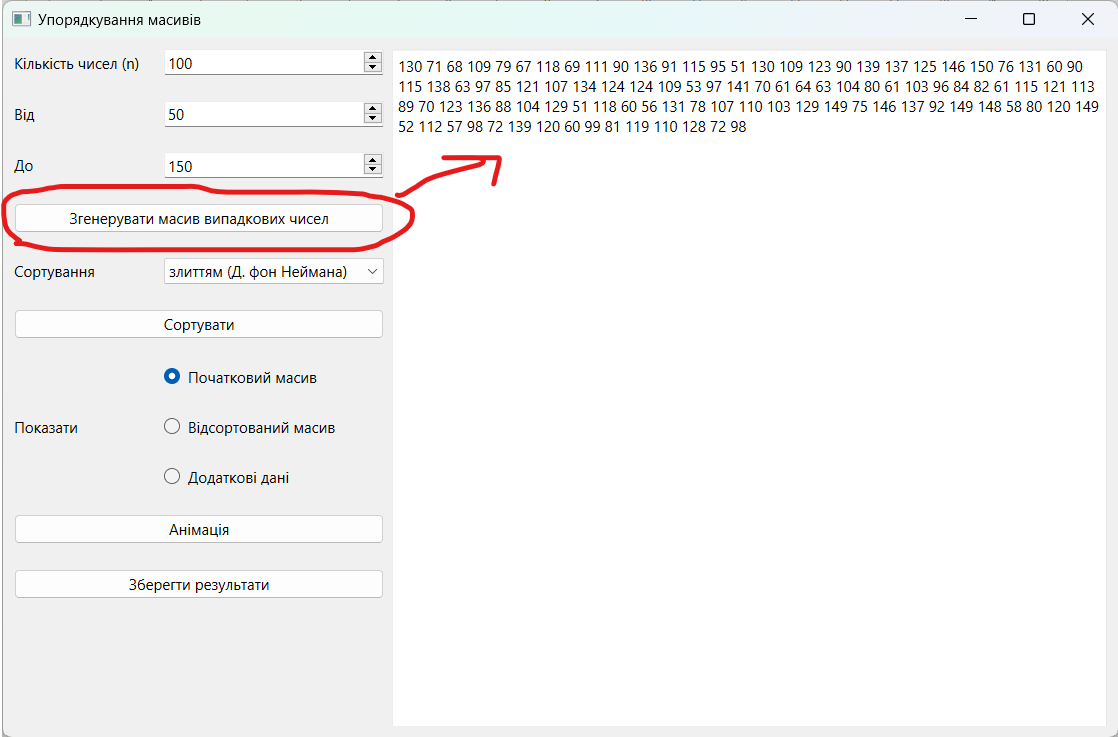
Далі за допомогою лічильника з назвою «Кількість чисел (n)» шляхом натиску на стрілки або введенням числа з клавіатури можна змінити розмірність масиву, який буде згенерований. За замовчуванням розмірність масиву дорівнює 100 (рисунок 6.2):

Рисунок 6.2 – Вибір розмірності масиву

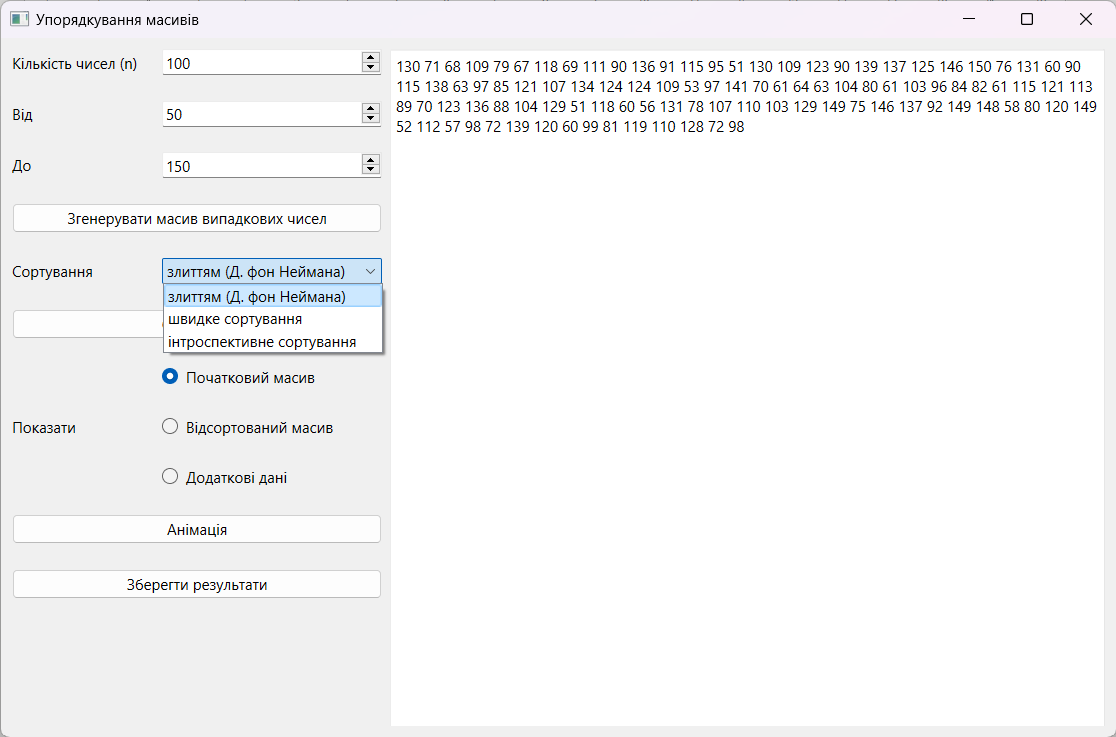
Далі за допомогою лічильників з назвами «Від» та “До” шляхом натиску на стрілки або введенням числа з клавіатури можна змінити межі генерування масиву (рисунок 6.3):

Рисунок 6.3 – Вибір діапазону генерування

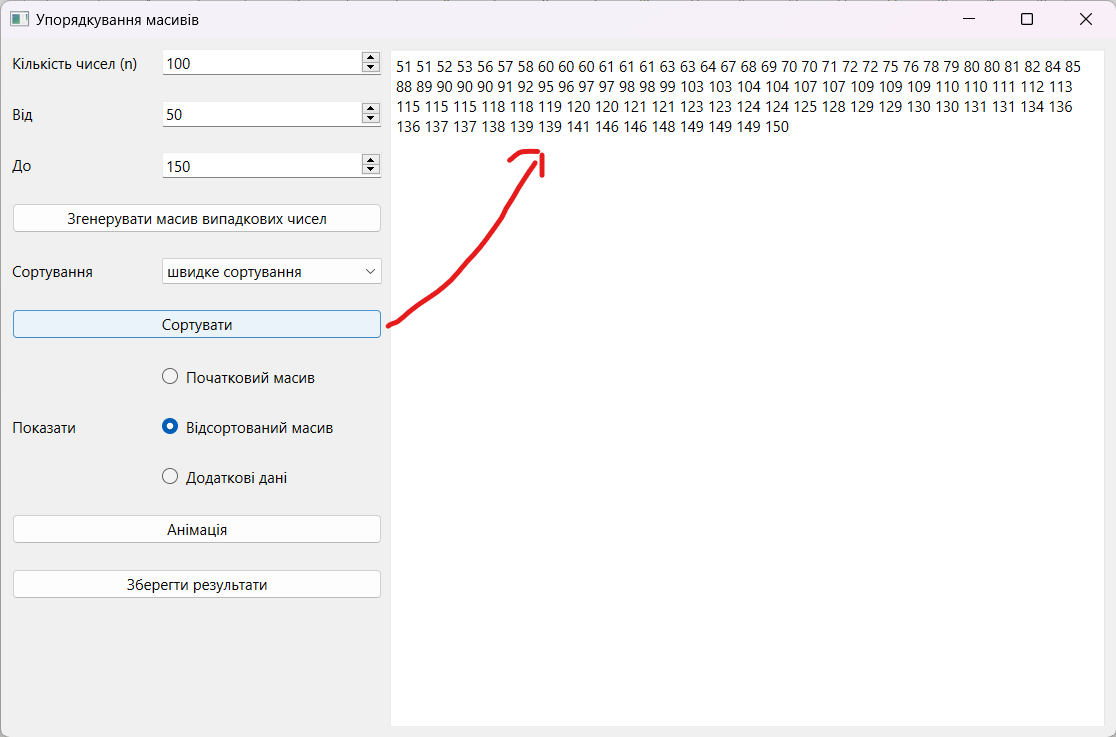
За допомогою кнопки “Згенерувати масив випадкових чисел” генеруємо випадковий масив з обраною розмірністю та діапазоном. Після натискання кнопки справа з’являється згенерований масив (рисунок 6.3):

Рисунок 6.4 – Генерування масиву

Далі за бажанням можна обрати метод сортування серед таких: сортування злиттям, швидке сортування або інтроспективне сортування (рисунок 6.5).

Рисунок 6.5 – Вибір методу сортування

Натиснемо кнопку “Сортувати”, щоб упорядкувати наш згенерований масив обраним методом, після сортування впорядкований масив з’явиться праворуч замість початкового (рисунок 6.6).

Рисунок 6.6 – Сортування масиву

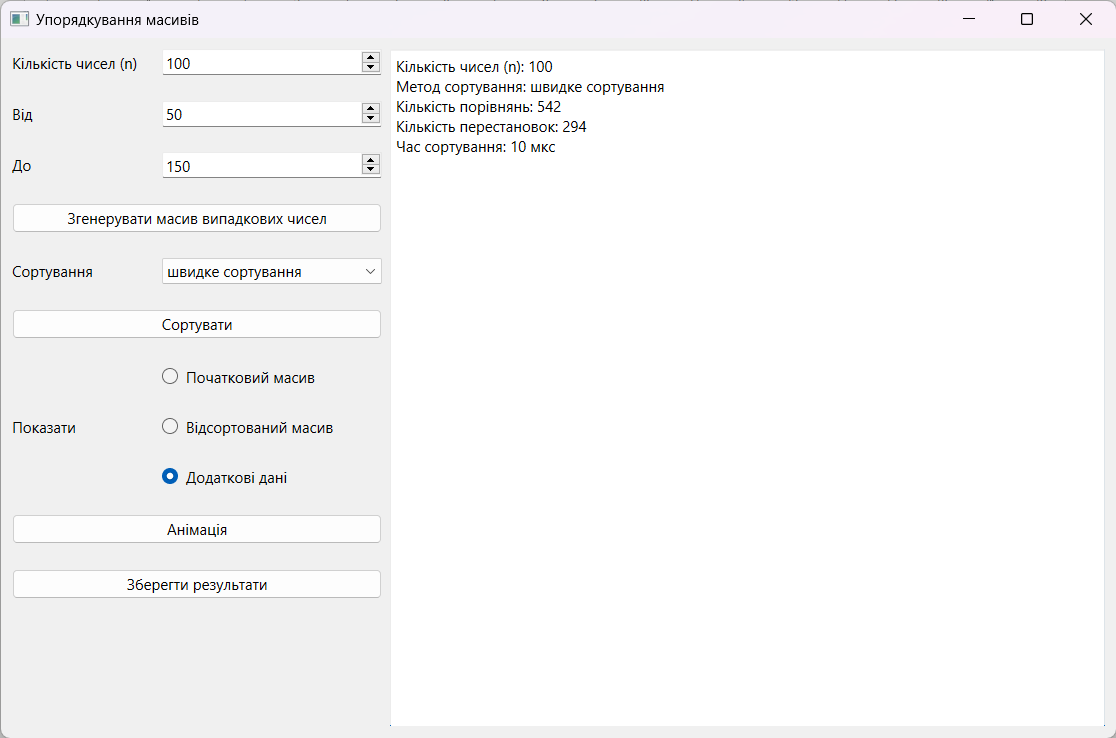
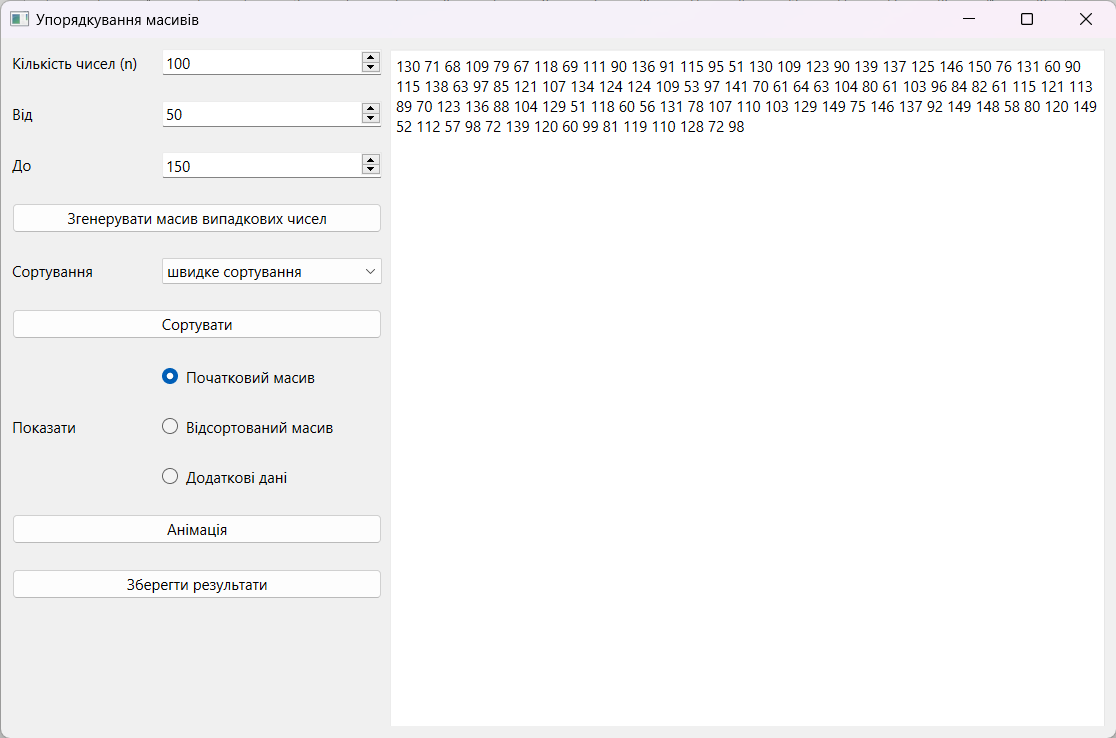
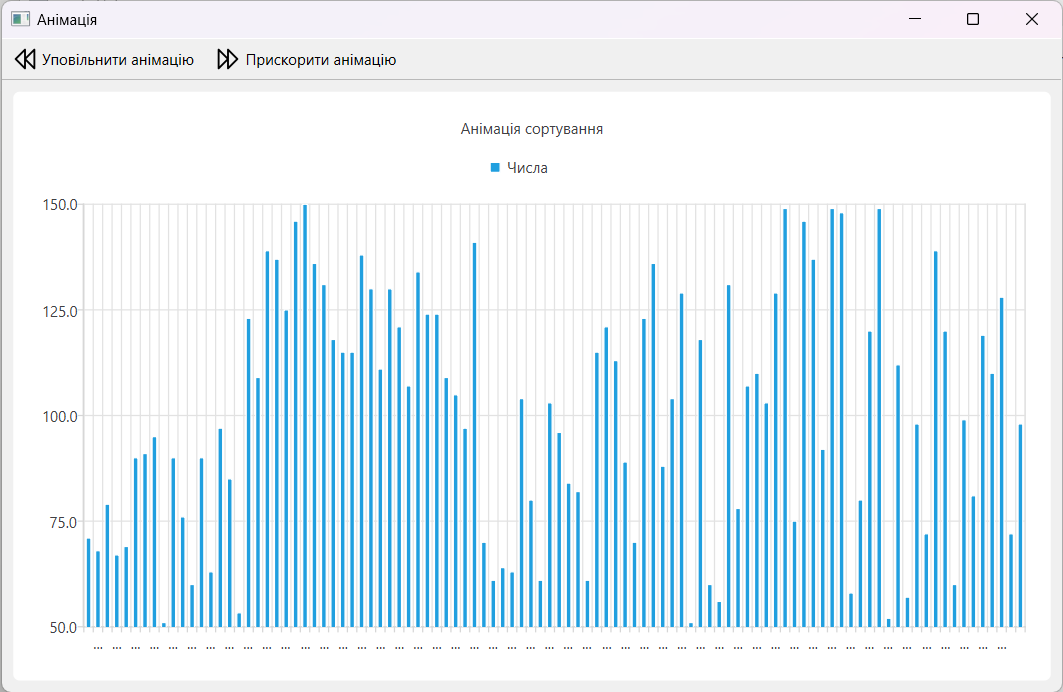
Як ви могли помітити, серед варіантів вибору з розділу “Показати” автоматично обрався пункт “Відсортований масив”. За бажанням його можна змінити і праворуч виведеться потрібна інформація (рисунок 6.7, рисунок 6.8).

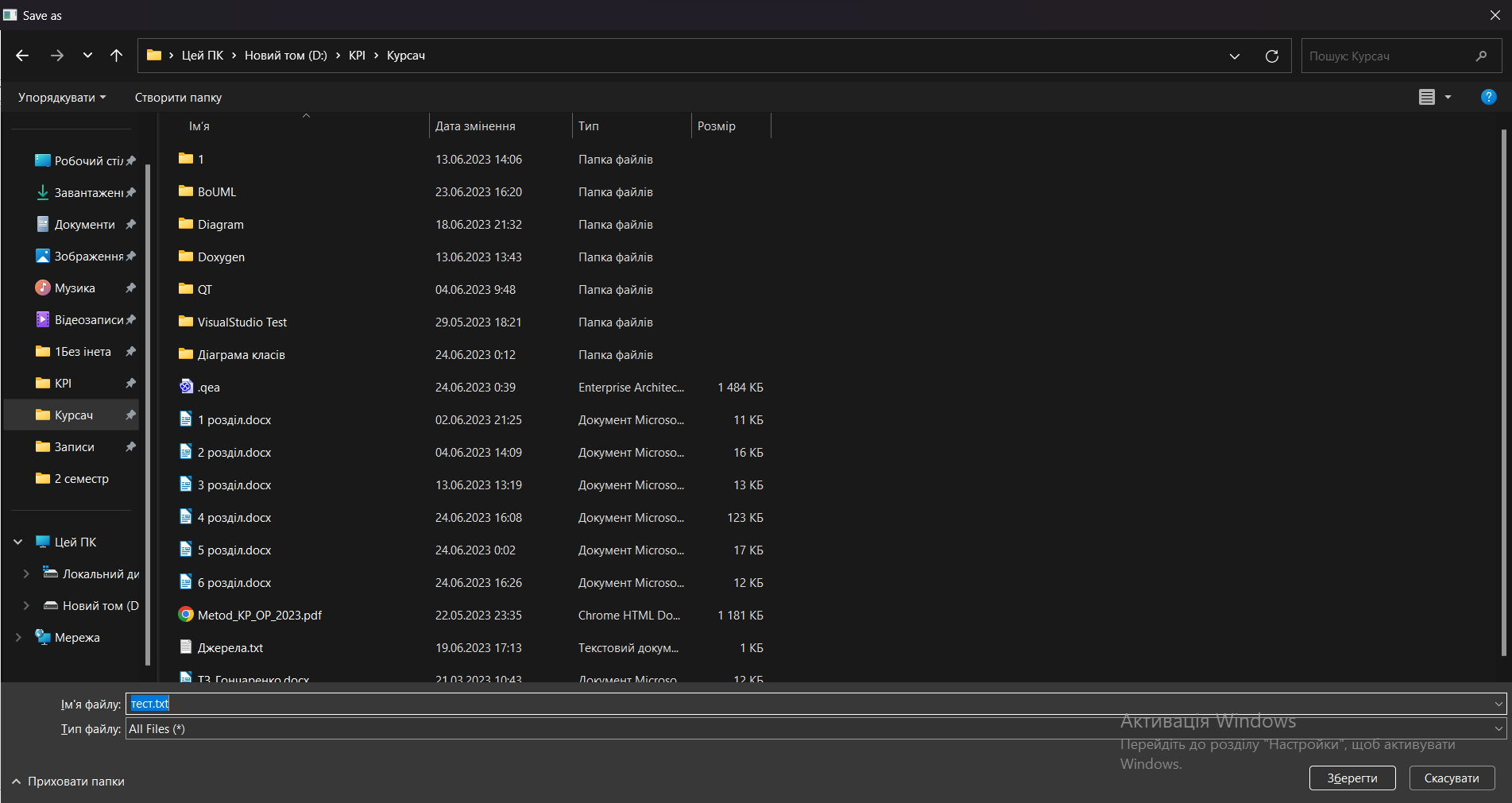
Рисунок 6.7 – Додаткові дані про сортування

Рисунок 6.8 – Невідсортований масив

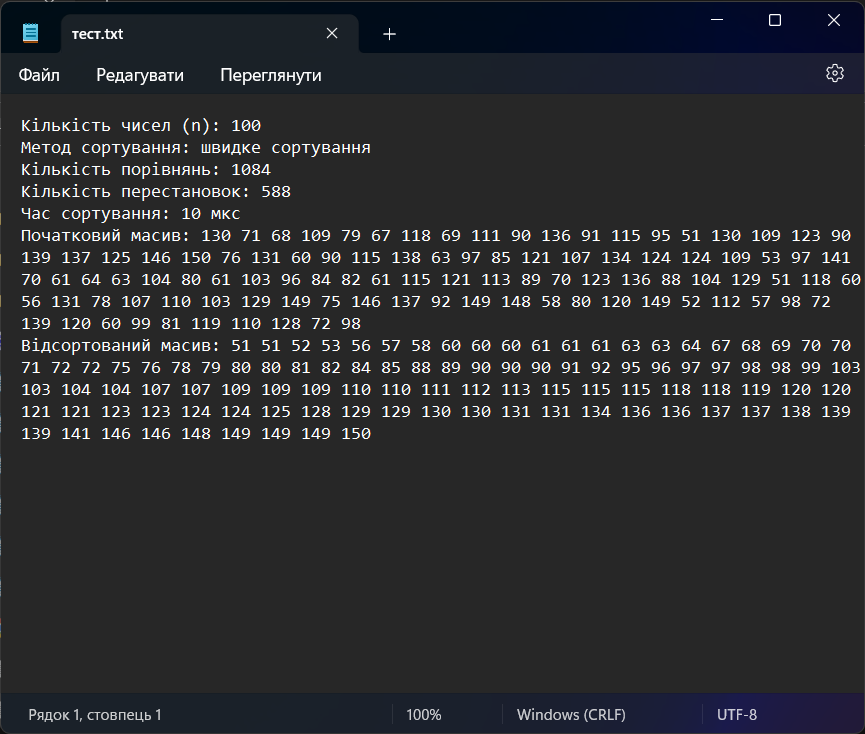
Далі, якщо вам цікаво, як працює той чи інший алгоритм, можна переглянути його анімацію сортування, натиснувши кнопку “Анімація” та зменшити або збільшити її швидкість (рисунок 6.9). За бажанням можна змінити алгоритм сортування і просто натиснути кнопку “Анімація” ще раз.

Рисунок 6.9 – Вікно з анімацією

Якщо ви бажаєте зберегти результати тестування у вигляді текстового файлу, у вас теж є така можливість. Для цього потрібно натиснути кнопку “Зберегти результати” і обрати ім’я та розташування файлу (рисунок 6.10).

Рисунок 6.10 – Збереження файлу

В результаті (рисунок 6.11) у файлі збережеться інформація про сортування, яку можна переглянути, натиснувши на кнопку “Додаткові дані” у розділі “Показати” (рисунок 6.7).

Рисунок 6.11 – Отриманий текстовий файл

За бажанням ці дії можна виконувати будь-яку кількість разів, не виходячи з програми, а просто змінюючи дані.

## Формат вхідних та вихідних даних

Користувачем на вхід програми подається розмірність масиву та діапазон генерування, тобто вводиться за допомогою лічильників і обробляється програмою.

Результатом виконання програми є відсортований масив, дані про сортування або анімація сортування

## Системні вимоги

Системні вимоги до програмного забезпечення наведені в таблиці 6.1.

Таблиця 6.1 – Системні вимоги програмного забезпечення

|  |  |  |
| --- | --- | --- |
|  | Мінімальні | Рекомендовані |
| Операційна система | Windows® XP/Windows Vista/Windows 7/ Windows 8/Windows 10 (з останніми обновленнями) | Windows 11 |
| Процесор | Intel® Pentium® ІІІ  1.0 GHz або  AMD Athlon™ 1.0 GHz | Intel(R) Core(TM) i5-11400H |
| Оперативна пам'ять | 256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/  Windows 8/Windows 10) | 8 GB RAM |
| Відеоадаптер | Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог) | |
| Дисплей | 1024х768 | 1920х1080 або краще |

Продовження таблиці 6.1

|  |  |
| --- | --- |
| Прилади введення | Клавіатура, комп’ютерна миша |
| Додаткове програмне забезпечення | Microsoft Visual C++ 2015-2022 Redistributable |

# Аналіз результатів

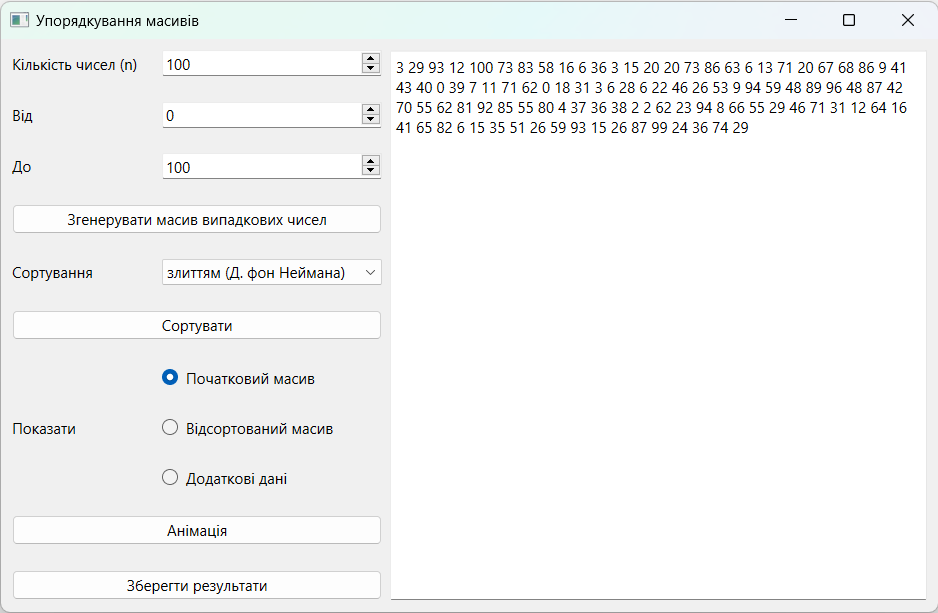
Головною задачею курсової роботи була реалізація програми для впорядкування масивів такими методами: сортування злиттям, швидке сортування, інтроспективне сортування

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувач неправильно використовував програму.

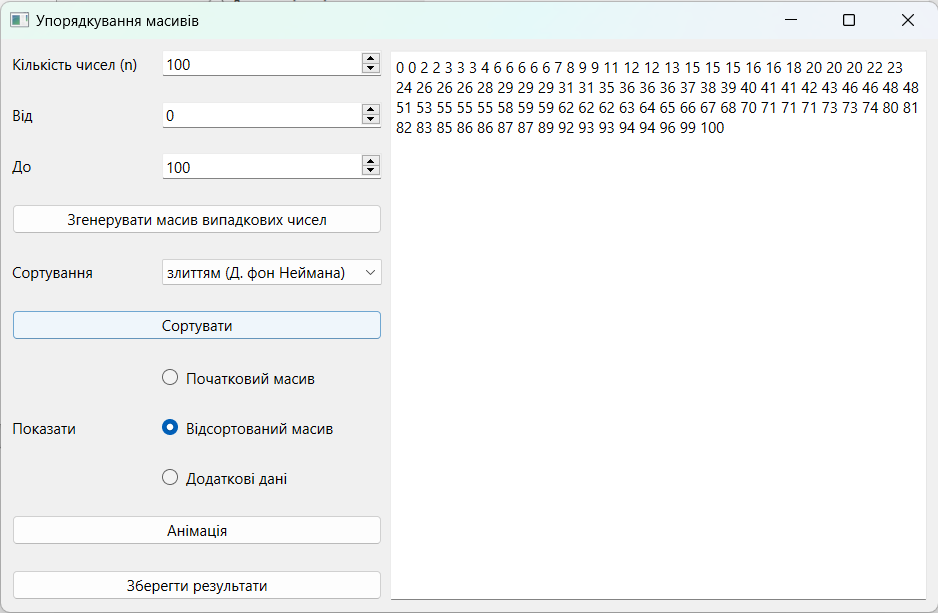
Для перевірки та доведення достовірності результатів виконання програмного забезпечення скористаюся онлайн-інструментом на сайті <https://onlinetools.com/number/sort-numbers> [6]:

а) Метод сортування злиттям (Д. фон Неймана)

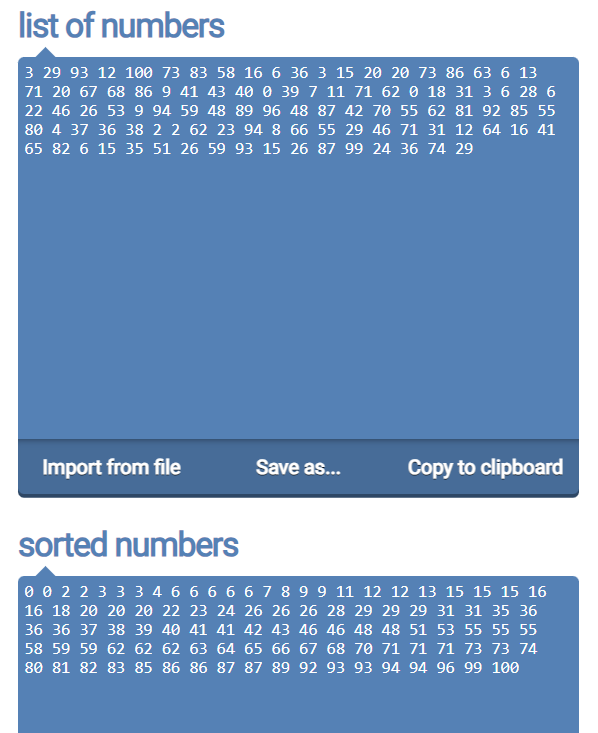
Початковий масив на 100 елементів, згенерований випадковим чином, наведено на рисунку 7.1:

Рисунок 7.1 – Початковий масив випадкових чисел

Результат сортування злиттям цього масиву наведено на рисунку 7.2:

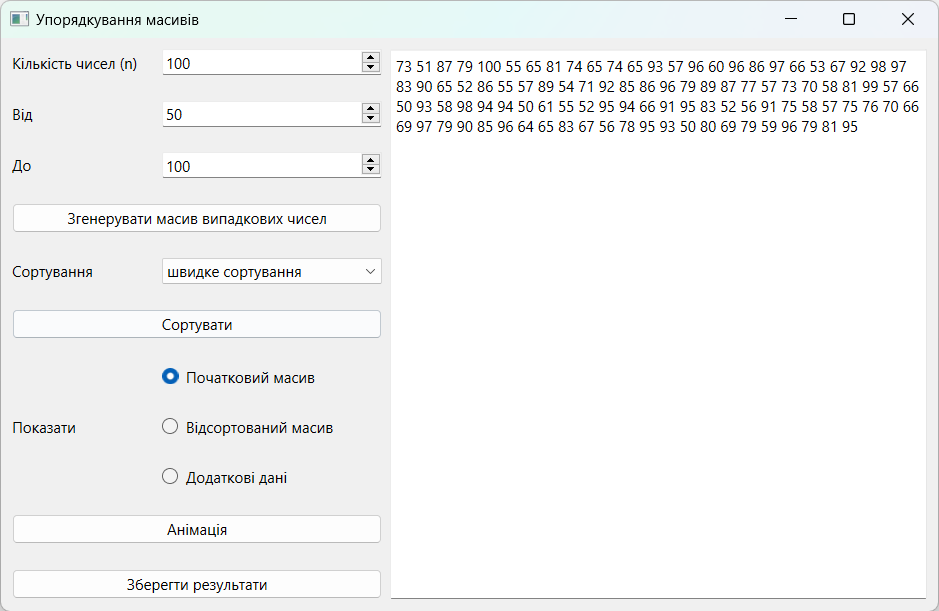
Рисунок 7.2 – Масив, упорядкований сортуванням злиттям

Оскільки результат виконання збігається з результатом на [сайті](https://onlinetools.com/number/sort-numbers?input=3 29 93 12 100 73 83 58 16 6 36 3 15 20 20 73 86 63 6 13 71 20 67 68 86 9 41 43 40 0 39 7 11 71 62 0 18 31 3 6 28 6 22 46 26 53 9 94 59 48 89 96 48 87 42 70 55 62 81 92 85 55 80 4 37 36 38 2 2 62 23 94 8 66 55 29 46 71 31 12 64 16 41 65 82 6 15 35 51 26 59 93 15 26 87 99 24 36 74 29 &ascending-order=true&descending-order=false&input-separator= &output-separator= &skip-duplicate-numbers=false) (рисунок 7.3), то даний метод працює правильно.

Рисунок 7.3 – Перевірка сортування на сайті[6]

б) Метод швидкого сортування.

Перевіримо упорядкування іншого масиву (рисунок 7.4) методом швидкого сортування (рис. 7.5):

Рисунок 7.4 – Новий масив

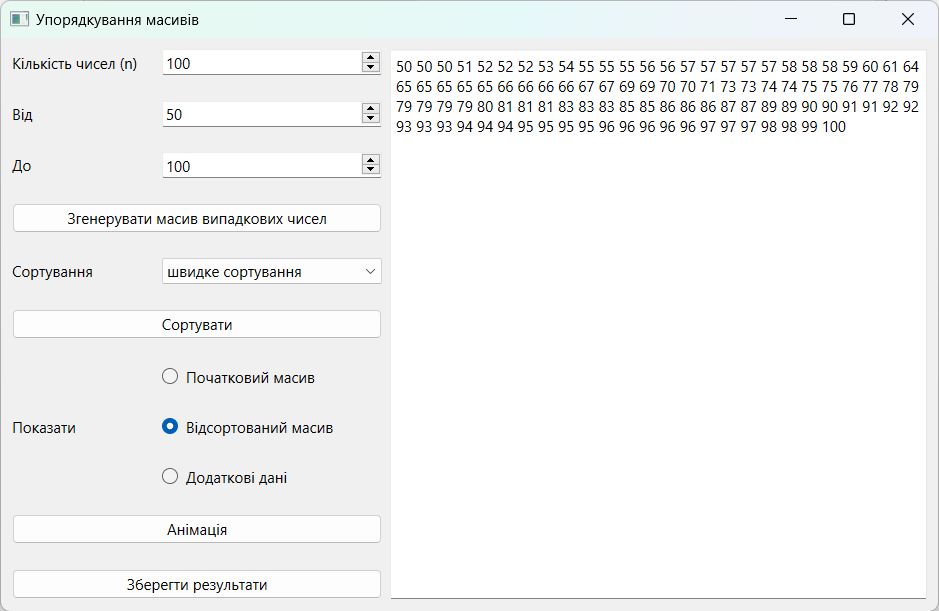


Рисунок 7.5 – Результат швидкого сортування

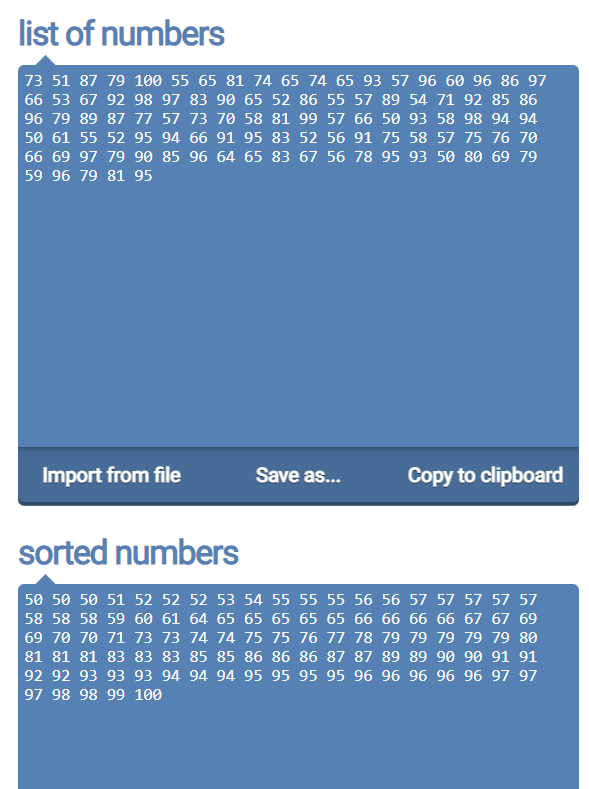
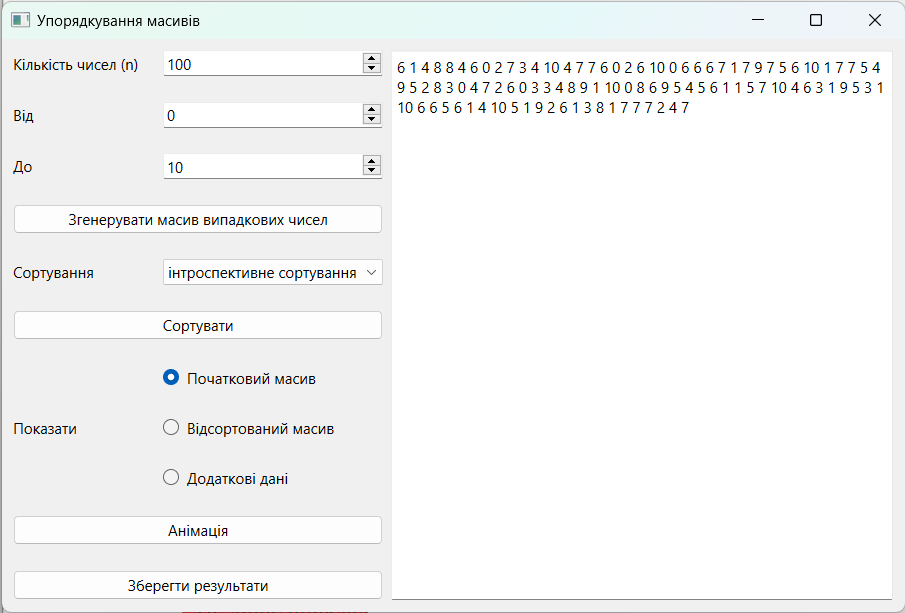
Оскільки результат виконання збігається з результатом на [сайті](https://onlinetools.com/number/sort-numbers?input=73 51 87 79 100 55 65 81 74 65 74 65 93 57 96 60 96 86 97 66 53 67 92 98 97 83 90 65 52 86 55 57 89 54 71 92 85 86 96 79 89 87 77 57 73 70 58 81 99 57 66 50 93 58 98 94 94 50 61 55 52 95 94 66 91 95 83 52 56 91 75 58 57 75 76 70 66 69 97 79 90 85 96 64 65 83 67 56 78 95 93 50 80 69 79 59 96 79 81 95&ascending-order=true&descending-order=false&input-separator= &output-separator= &skip-duplicate-numbers=false) (рисунок 7.6), то даний метод працює правильно.

Рисунок 7.6 – Перевірка сортування на сайті[6]

в) Метод швидкого сортування.

Тепер перевіримо упорядкування іншого масиву (рис. 7.7) методом інтроспективного сортування (рис. 7.8):

Рисунок 7.7 – Новий масив

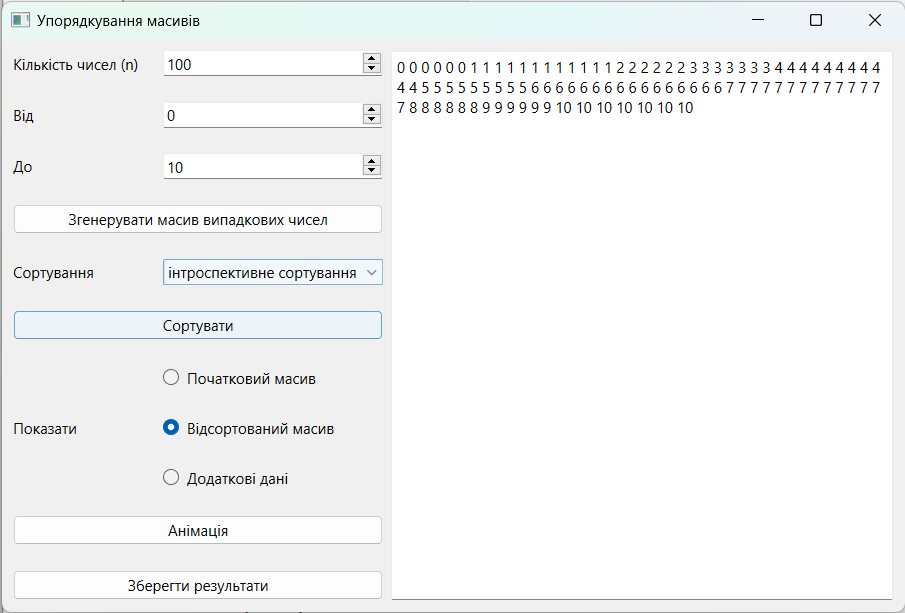
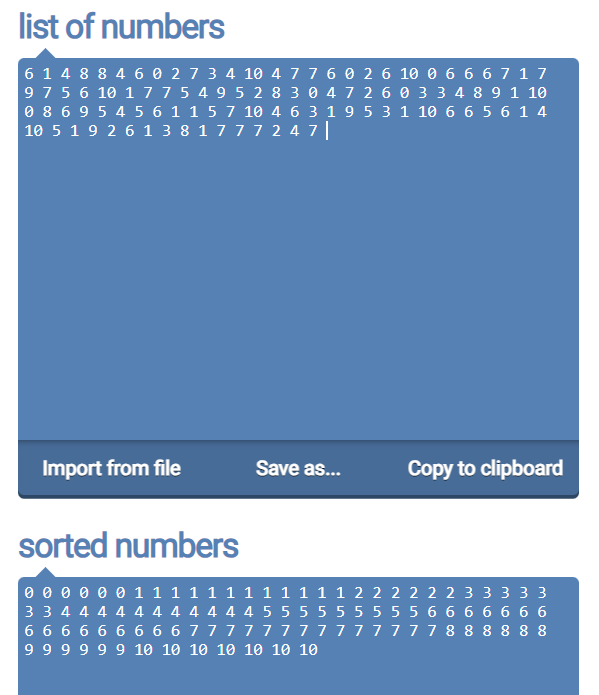


Рисунок 7.8 – Результат інтроспективного сортування

Оскільки результат виконання збігається з результатом на [сайті](https://onlinetools.com/number/sort-numbers?input=6 1 4 8 8 4 6 0 2 7 3 4 10 4 7 7 6 0 2 6 10 0 6 6 6 7 1 7 9 7 5 6 10 1 7 7 5 4 9 5 2 8 3 0 4 7 2 6 0 3 3 4 8 9 1 10 0 8 6 9 5 4 5 6 1 1 5 7 10 4 6 3 1 9 5 3 1 10 6 6 5 6 1 4 10 5 1 9 2 6 1 3 8 1 7 7 7 2 4 7 &ascending-order=true&descending-order=false&input-separator= &output-separator= &skip-duplicate-numbers=false) (рисунок 7.9), то даний метод працює правильно.

Рисунок 7.9 – Перевірка сортування на сайті[6]

Визначимо асимптотичну складність алгоритмів

1. Алгоритм сортування злиттям[7]

Часова складність:

Використання додаткової пам’яті:

1. Алгоритм швидкого сортування

У найкращому випадку:

У найгіршому випадку:

Використання додаткової пам’яті:

1. Алгоритм інтроспективного сортування

У найкращому та середньому випадку алгоритм використовує швидке сортування, тому часова складність складає:

У найгіршому випадку він перемикається на пірамідальне сортування, яке має часову складність , тому й часова складність інтроспективного сортування складає .

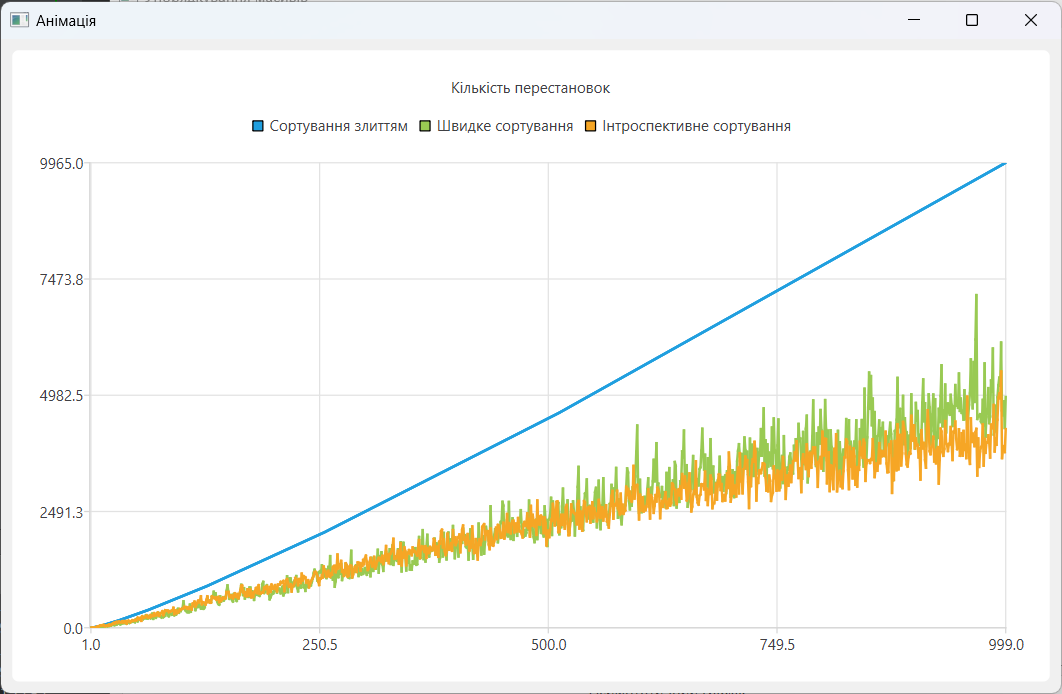
Перевіримо ефективність програми для деяких розмірностей масиву

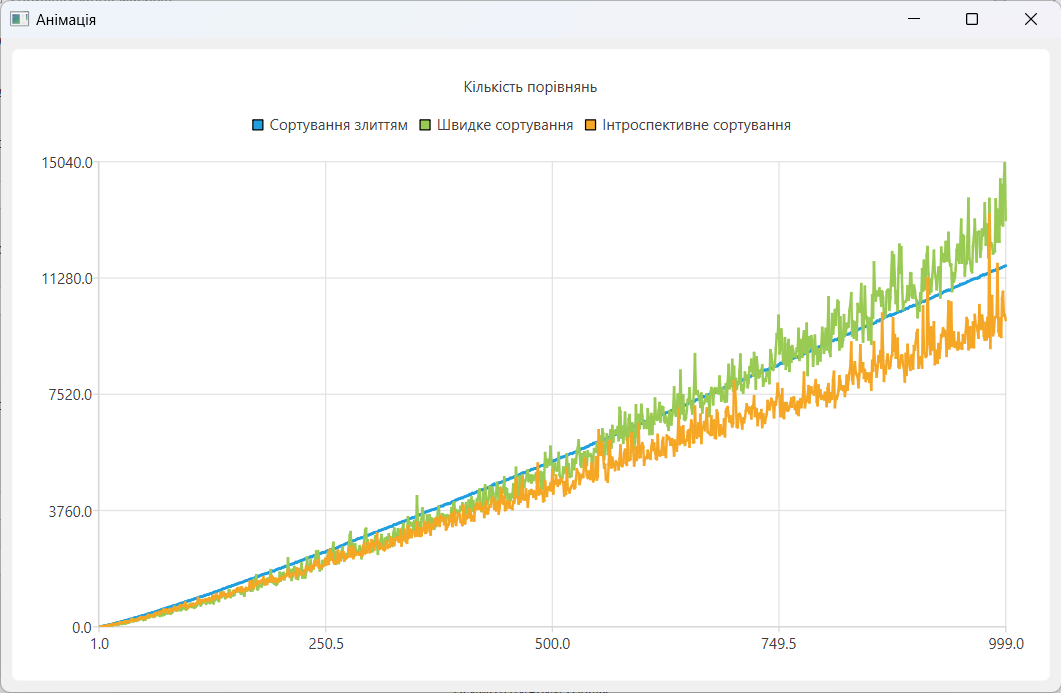
Результати тестування ефективності алгоритмів сортування масивів наведено в таблиці 7.1:

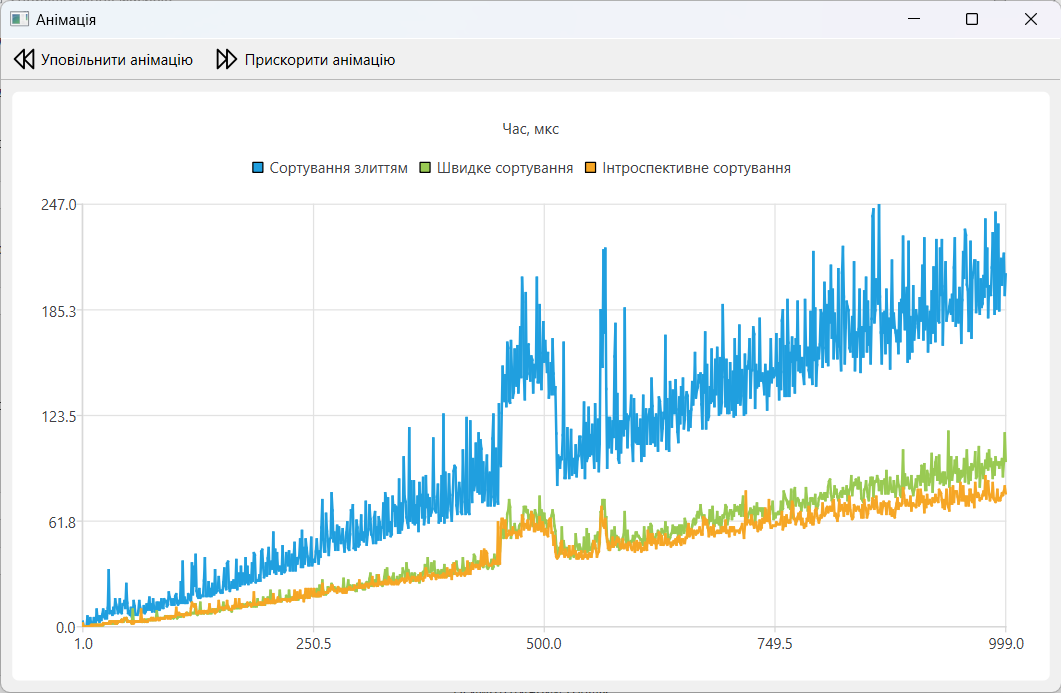
Таблиця 7.1 – Тестування ефективності методів

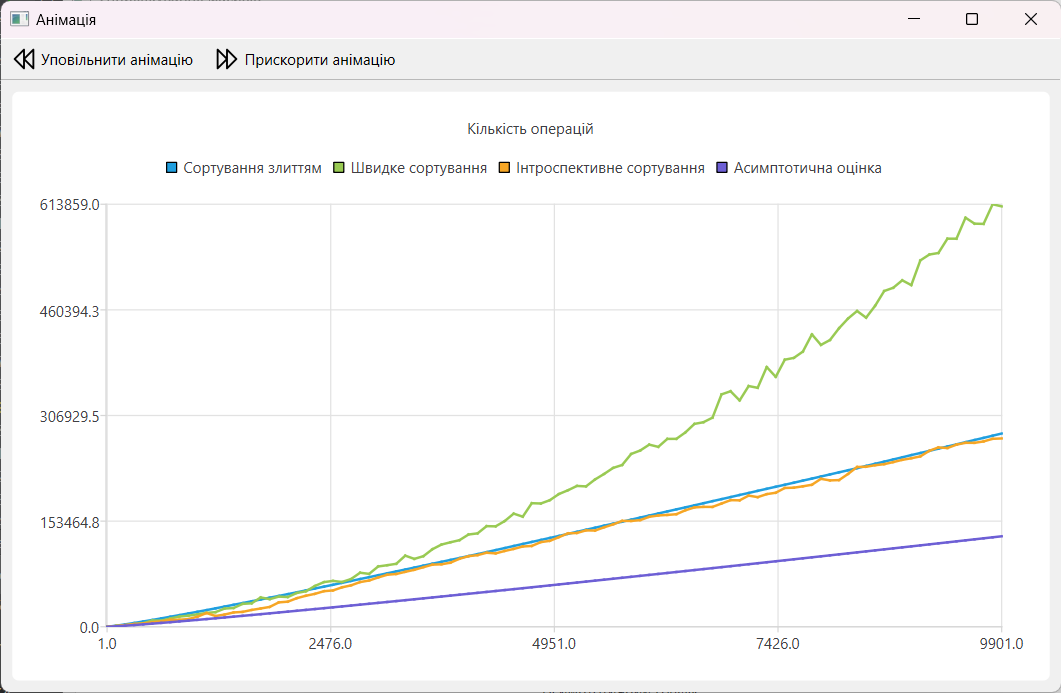
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Розмірність масиву | Метод сортування | Теоретична складність | Практична складність | | |
| Кількість порівнянь | Кількість перестановок | Час (мкс) |
| 1000 | Злиттям | 9966 | 11704 | 9976 | 303 |
| Швидке | 13009 | 4838 | 127 |
| Інтроспективне | 9287 | 3783 | 87 |
| 10000 | Злиттям | 132877 | 150266 | 133616 | 3854 |
| Швидке | 576465 | 46340 | 2250 |
| Інтроспективне | 230131 | 45860 | 1229 |
| 100000 | Злиттям | 1660964 | 1832933 | 1668928 | 27384 |
| Швидке | 50329261 | 572072 | 135425 |
| Інтроспективне | 2882362 | 427350 | 13066 |
| 500000 | Злиттям | 9465784 | 10313413 | 9475712 | 147197 |
| Швидке | 1242001121 | 2394555 | 3.15951e+06 |
| Інтроспективне | 15977143 | 2145423 | 68906 |
| 1000000 | Злиттям | 19931569 | 21620222 | 19951424 | 309019 |
| Швидке | 663620749 | 5109800 | 1.24587e+07 |
| Інтроспективне | 32974179 | 4162002 | 145199 |

Візуалізація залежності критеріїв від розмірності масиву наведено на рисунках 7.10, 7.11, 7.12, 7.13:

Рисунок 7.10 – Графік залежності кількості перестановок для трьох алгоритмів

Рисунок 7.11 – Графік залежності кількості порівнянь для трьох алгоритмів

Рисунок 7.12 – Графік залежності часу сортування для трьох алгоритмів

Рисунок 7.13 – Графік залежності кількості операцій сортування для трьох алгоритмів та теоретична складність

За результатами тестування можна зробити такі висновки:

а) Всі розглянуті методи дозволяють сортувати великі та надвеликі масиви цілих чисел

б) Складність всіх розглянутих методів дорівнює у всіх випадках, окрім швидкого сортування в найгіршому випадку, де часова складність складає

в) З розглянутих методів найоптимальнішим для практичного використання є метод інтроспективного сортування, адже він виконується найшвидше та не потребує додаткової пам’яті, на відміну від сортування злиттям, де просторова складність складає

Висновки

У ході виконання курсової роботи були дослідженні та реалізовані такі методи для швидкісного впорядкування масивів: сортування злиттям, швидке сортування та інтроспективне сортування. Для виконання курсової роботи були описані 7 розділів, у яких показано повний цикл розробки ПЗ.

У першому розділі було детально описано постановку задачі, що дозволило визначити подальший шлях розробки програми.

Після аналізу літератури у другому розділі було описано теоретичну частину алгоритмів, що дозволило зрозуміти, що роблять алгоритми та як вони працюють на поверхневому рівні. У третьому розділі детально описано, як саме працює загальний алгоритм програми та методи сортування. Це дозволило сформувати основу для кодування майбутньої програми.

Після створення програми, була створена діаграма класів, яка описує атрибути та методи класів програми. Також було описано ці методи детальніше в таблицях опису стандартних і користувацьких методів. Разом із діаграмою, вони увійшли в основу четвертого розділу.

Для того, щоб перевірити програму на коректність роботи згідно з технічним завданням та загальним алгоритмом, описаним у третьому розділі, у п’ятому розділі було описано план тестування, а також результати цих тестів на створеному програмному забезпеченні. За результатами тестів було визначено, що програма працює коректно, а можливі помилки обробляються програмою.

Далі в шостому розділі було описано інструкцію користувача задля опису можливостей програми та ознайомлення користувачів з інтерфейсом, а також технічні вимоги до програми.

В останньому, сьомому розділі, було проаналізовано результати роботи програми та перевірено їх з результатами в онлайн-інструменті. Згідно з перевірками, всі три методи працюють коректно, а результати збігаються. Задля оцінки ефективності алгоритмів, було оформлено результати теоретичної та практичної складності алгоритмів у вигляді таблиці, а також графіків для наочного сприйняття.

Перелік посилань

1. Кормен, Томас Г. Вступ до алгоритмів : Переклад з англійської третього видання : [укр.] = Introduction to Algorithms : Third Edition : [пер. з англ.] / Томас Г. Кормен, Чарлз Е. Лейзерсон, Роналд Л. Рівест, Кліфорд Стайн, —К. : К. I. С., 2019. — 1288 с
2. Merge sort – data structure and algorithms tutorials. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/merge-sort/> (дата звернення: 25.06.2023).
3. Quicksort – data structure and algorithm tutorials. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/quick-sort/> (дата звернення: 25.06.2023).
4. Introsort – c++’s sorting weapon. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/introsort-cs-sorting-weapon/> (дата звернення: 25.06.2023).
5. Qt documentation. Qt Documentation. URL: https://doc.qt.io/ (дата звернення: 25.06.2023).
6. Sort a List of Numbers – Online Number Tools. Online tools. URL: <https://onlinetools.com/number/sort-numbers> (дата звернення: 25.06.2023).
7. Analysis of Merge sort algorithm, 2013. YouTube. URL: <https://youtu.be/0nlPxaC2lTw> (дата звернення: 25.06.2023).

Додаток А Технічне завдання

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра

інформатики та програмної інженерії

Затвердив

Керівник \_\_Головченко М.М.\_\_\_\_

«21» березня 2023 р.

Виконавець:

Студент \_\_*Гончаренко І.Ю.\_\_\_\_*

«21» березня 2023 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Упорядкування масивів»

з дисципліни:

«Основи програмування»

Київ 2023

* 1. *Мета*: Метою курсової роботи є коректна реалізація програмного забезпечення для упорядкування масивів за методом сортування злиттям (Д. фон Неймана), методом швидкого сортування та методом інтроспективного сортування
  2. *Дата початку роботи*: «21» березня 2023 р.
  3. *Дата закінчення роботи*: «25» червня 2023 р.
  4. *Вимоги до програмного забезпечення*.

1) Функціональні вимоги:

* Можливість задання розмірності масиву (від 100 до 50000 елементів)
* Можливість генерації масиву випадковим чином у заданому діапазоні
* Можливість відображення згенерованого масиву
* Можливість вибору методу сортування (метод сортування злиттям (Д. фон Неймана), метод швидкого сортування, метод інтроспективного сортування)
* Можливість сортування масиву обраним методом
* Можливість відображення відсортованого масиву
* Можливість відображення даних для оцінки якості алгоритму (кількість порівнянь, кількість перестановок, час сортування)
* Можливість збереження результатів у вигляді текстового файлу
* Можливість відображення процесу сортування у вигляді анімації для масиву до 300 елементів

2) Нефункціональні вимоги:

* Можливість запускати програмне забезпечення на операційній системі Windows 11.
* Все програмне забезпечення та супроводжуюча технічна документація повинні задовольняти наступним ДЕСТам:

ГОСТ 29.401 - 78 - Текст програми. Вимоги до змісту та оформлення.

ГОСТ 19.106 - 78 - Вимоги до програмної документації.

ГОСТ 7.1 - 84 та ДСТУ 3008 - 2015 - Розробка технічної документації.

* 1. *Стадії та етапи розробки*:

1. Об'єктно-орієнтований аналіз предметної області задачі (до 26.06.2023 р.)
2. Об'єктно-орієнтоване проектування архітектури програмної системи (до 26.06.2023 р.)
3. Розробка програмного забезпечення (до 26.06.2023 р.)
4. Тестування розробленої програми (до 26.06.2023 р.)
5. Розробка пояснювальної записки (до 26.06.2023 р.).
6. Захист курсової роботи (до 26.06.2023 р.).
   1. *Порядок контролю та приймання*. Поточні результати роботи над КР регулярно демонструються викладачу. Своєчасність виконання основних етапів графіку підготовки роботи впливає на оцінку за КР відповідно до критеріїв оцінювання.

Додаток Б Тексти програмного коду

(Вид носія даних)

*Електронний*

(Обсяг програми (документа), арк., Кб)

*28 арк, 73,5 Кб*

*студента групи ІП-23 І курсу*

*Гончаренка І.Ю.*

(Найменування програми (документа))

*Тексти програмного коду програмного забезпечення вирішення задачі упорядкування масивів*

main.cpp:

#include "mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

return a.exec();

}

mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "animationwindow.h"

class NumbersArrray;

#include "sortingdata.h"

#include <cstdlib>

#include <chrono>

#include <QApplication>

#include <QTimer>

#include <QFile>

#include <QFileDialog>

#include <QMessageBox>

#include <QtWidgets/QMainWindow>

#include <QtCharts/QChartView>

#include <QtCharts/QBarSeries>

#include <QtCharts/QBarSet>

#include <QtCharts/QLegend>

#include <QtCharts/QBarCategoryAxis>

#include <QtCharts/QHorizontalStackedBarSeries>

#include <QtCharts/QLineSeries>

#include <QValueAxis>

#ifdef \_WIN32

#include <Windows.h>

#else

#include <unistd.h>

#endif

using std::swap;

using namespace std::chrono;

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow(QWidget \*parent = nullptr);

void quickSort(NumbersArrray& arr, int low, int high);

int partition(NumbersArrray& arr, int low, int high);

void merge(NumbersArrray& array1, int left, int mid, int right);

void mergeSort(NumbersArrray& array, int const begin, int const end);

void InsertionSort(NumbersArrray& arr, int begin, int end);

void IntrosortUtil(NumbersArrray& arr, int begin, int end, int depthLimit);

void Introsort(NumbersArrray& arr, int begin, int end);

void heapify(NumbersArrray& arr, int low, int high, int i);

void heapSort(NumbersArrray& arr, int low, int high);

~MainWindow();

signals:

void swapCalled(NumbersArrray array, int index1, int index2);

void mergeCalled(NumbersArrray array, int index1, int index2);

public slots:

void updateChart1();

void decreaseInterval();

void increaseInterval();

private slots:

void on\_SizeBox\_valueChanged(int arg1);

void on\_MaxValueBox\_valueChanged(int arg1);

void on\_MinValueBox\_valueChanged(int arg1);

void on\_Generate\_clicked();

void on\_MethodBox\_activated(int index);

void on\_Sort\_clicked();

void on\_Unsorted\_clicked();

void on\_Sorted\_clicked();

void on\_AdditionalData\_clicked();

void on\_Animation\_clicked();

void on\_Save\_clicked();

void on\_asymtoticButton\_clicked();

private:

Ui::MainWindow \*ui;

AnimationWindow \*animation\_window;

QString currentFile = "";

QTimer \*Swaptimer;

QBarSet \*numbers\_set;

QBarSeries \*series;

QChart \*chart;

QChartView \*chartView;

int min\_number, max\_number, animation\_duration;

int ind1, ind2;

bool animation\_is\_on;

};

#endif // MAINWINDOW\_H

animationwindow.h:

#ifndef ANIMATIONWINDOW\_H

#define ANIMATIONWINDOW\_H

#include <QMainWindow>

namespace Ui {

class AnimationWindow;

}

class AnimationWindow : public QMainWindow

{

Q\_OBJECT

signals:

void intervalIncreased();

void intervalDecreased();

public:

explicit AnimationWindow(QWidget \*parent = nullptr);

~AnimationWindow();

private slots:

void on\_slower\_triggered();

void on\_faster\_triggered();

private:

Ui::AnimationWindow \*ui;

};

#endif // ANIMATIONWINDOW\_H

numbersarray.h:

#ifndef NUMBERSARRAY\_H

#define NUMBERSARRAY\_H

#include "mainwindow.h"

class NumbersArrray

{

int n, max\_capacity;

int\* numbers;

public:

NumbersArrray();

NumbersArrray(int n1);

NumbersArrray(NumbersArrray& other);

int length();

QString getString();

void pushBack(int value);

void popBack();

void doubleCapacity();

bool isEmpty();

int& operator[] (int index);

NumbersArrray operator=(const NumbersArrray& other);

};

#endif // NUMBERSARRAY\_H

sortingdata.h:

#ifndef SORTINGDATA\_H

#define SORTINGDATA\_H

#include "mainwindow.h"

#include <QString>

#include "numbersarray.h"

class SortingData

{

int swaps, comparisons, n, sorting\_method;

double sorting\_time;

public:

SortingData();

QString getString();

QString getSortingMethod();

int method();

int getN();

int getComparisons();

int getSwaps();

int getTime();

void setSwaps(int new\_swaps);

void SetComparisons(int new\_comparisons);

void SetSortingTime(double sorting\_time1);

void setN(int new\_number);

void setSortingMethod(int method\_number);

void addSwaps(int number);

void addComparisons(int number);

void addTime(double time\_in\_ms);

void reset();

};

#endif // SORTINGDATA\_H

mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

#include "numbersarray.h"

#include <queue>

SortingData sorting\_data;

NumbersArrray unsortedArray, sortedArray, animationArray;

std::queue<int> animationQueue, valuesQueue;

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

ui->asymtoticButton->setVisible(false);

min\_number = 0, max\_number = 100, animation\_duration = 500;

animation\_is\_on = false;

numbers\_set = new QBarSet("Числа");

series = new QBarSeries();

chart = new QChart();

chartView = new QChartView(chart);

Swaptimer = new QTimer(this);

Swaptimer->setSingleShot(false);

animation\_window = new AnimationWindow(this);

QObject::connect(Swaptimer, &QTimer::timeout, this, &MainWindow::updateChart1);

QObject::connect(animation\_window, &AnimationWindow::intervalDecreased, this, &MainWindow::decreaseInterval);

QObject::connect(animation\_window, &AnimationWindow::intervalIncreased, this, &MainWindow::increaseInterval);

}

MainWindow::~MainWindow()

{

delete ui;

}

void clear\_queue( std::queue<int> &q )

{

std::queue<int> empty;

swap( q, empty );

}

void MainWindow::swapCalled(NumbersArrray array, int index1, int index2)

{

qDebug() << "swap called";

animationQueue.push(index1);

animationQueue.push(index2);

valuesQueue.push(array[index1]);

valuesQueue.push(array[index2]);

}

void MainWindow::mergeCalled(NumbersArrray array, int index1, int index2)

{

qDebug() << "merge called";

int size = index2 - index1 + 1;

animationQueue.push(index1);

animationQueue.push(index2);

for (int i = 0; i < size; i++)

{

valuesQueue.push(array[index1 + i]);

}

}

void MainWindow::updateChart1()

{

int i1 = animationQueue.front();

qDebug() << "front:" << animationQueue.front();

animationQueue.pop();

int i2 = animationQueue.front();

qDebug() << "front:" << animationQueue.front();

animationQueue.pop();

if (sorting\_data.method() == 0)

{

qDebug() << "chart merged";

qDebug() << "============" ;

int size = i2 - i1 + 1;

int v;

for (int i = 0; i < size; i++)

{

v = valuesQueue.front();

valuesQueue.pop();

numbers\_set->replace(i1 + i, v);

}

}

else if (sorting\_data.method() == 1 || sorting\_data.method() == 2)

{

qDebug() << "chart updated";

qDebug() << "============" ;

int v1 = valuesQueue.front();

valuesQueue.pop();

int v2 = valuesQueue.front();

valuesQueue.pop();

numbers\_set->replace(i1, v2);

numbers\_set->replace(i2, v1);

}

chart->update();

if (animationQueue.empty()) Swaptimer->stop();

}

void MainWindow::decreaseInterval()

{

if (animation\_duration > 100)

{

animation\_duration -= 100;

Swaptimer->setInterval(animation\_duration);

}

}

void MainWindow::increaseInterval()

{

animation\_duration += 100;

Swaptimer->setInterval(animation\_duration);

}

void MainWindow::merge(NumbersArrray& array1, int left, int mid, int right)

{

int const subArrayOne = mid - left + 1;

int const subArrayTwo = right - mid;

NumbersArrray leftArray(subArrayOne), rightArray(subArrayTwo);

for (int i = 0; i < subArrayOne; i++)

leftArray[i] = array1[left + i];

for (int j = 0; j < subArrayTwo; j++)

rightArray[j] = array1[mid + 1 + j];

int indexOfSubArrayOne = 0, indexOfSubArrayTwo = 0;

int indexOfMergedArray = left;

sorting\_data.addComparisons(1);

while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo)

{

sorting\_data.addComparisons(1);

if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo])

{

array1[indexOfMergedArray] = leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

sorting\_data.addSwaps(1);

}

else

{

array1[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

sorting\_data.addSwaps(1);

}

indexOfMergedArray++;

}

sorting\_data.addComparisons(1);

while (indexOfSubArrayOne < subArrayOne) {

array1[indexOfMergedArray] = leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

indexOfMergedArray++;

sorting\_data.addSwaps(1);

}

sorting\_data.addComparisons(1);

while (indexOfSubArrayTwo < subArrayTwo) {

array1[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

indexOfMergedArray++;

sorting\_data.addSwaps(1);

}

if (animation\_is\_on) mergeCalled(array1, left, right);

}

void MainWindow::mergeSort(NumbersArrray& array, int const begin, int const end)

{

if (begin >= end)

return;

int mid = begin + (end - begin) / 2;

mergeSort(array, begin, mid);

mergeSort(array, mid + 1, end);

merge(array, begin, mid, end);

}

int MainWindow::partition(NumbersArrray& arr, int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++) {

sorting\_data.addComparisons(1);

if (arr[j] < pivot) {

i++;

if (animation\_is\_on) swapCalled(arr, i, j);

swap(arr[i], arr[j]);

sorting\_data.addSwaps(1);

}

}

if (animation\_is\_on) swapCalled(arr, i + 1, high);

swap(arr[i + 1], arr[high]);

sorting\_data.addSwaps(1);

return (i + 1);

}

void MainWindow::quickSort(NumbersArrray& arr, int low, int high)

{

if (low < high) {

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void MainWindow::heapify(NumbersArrray& arr, int low, int high, int i)

{

int largest = i;

int l = 2 \* i + 1 - low;

int r = 2 \* i + 2 - low;

sorting\_data.addComparisons(3);

if (l <= high && arr[l] > arr[largest])

largest = l;

if (r <= high && arr[r] > arr[largest])

largest = r;

if (largest != i)

{

sorting\_data.addSwaps(1);

if (animation\_is\_on) swapCalled(arr, i, largest);

swap(arr[i], arr[largest]);

heapify(arr, low, high, largest);

}

}

void MainWindow::heapSort(NumbersArrray& arr, int low, int high)

{

sorting\_data.addComparisons(1);

for (int i = (high - low + 1) / 2 - 1 + low; i >= low; i--)

{

sorting\_data.addComparisons(1);

heapify(arr, low, high, i);

}

sorting\_data.addComparisons(1);

for (int i = high; i > low; i--)

{

sorting\_data.addSwaps(1);

sorting\_data.addComparisons(1);

if (animation\_is\_on) swapCalled(arr, low, i);

swap(arr[low], arr[i]);

heapify(arr, low, i - 1, low);

}

}

void MainWindow::InsertionSort(NumbersArrray& arr, int begin, int end)

{

int i, j;

sorting\_data.addComparisons(1);

for (i = begin + 1; i <= end; i++)

{

j = i;

sorting\_data.addComparisons(1);

while (j > begin && arr[j - 1] > arr[j])

{

sorting\_data.addComparisons(1);

sorting\_data.addSwaps(1);

if (animation\_is\_on) swapCalled(arr, j - 1, j);

swap(arr[j], arr[j - 1]);

j--;

}

}

}

int MedianOfThree(NumbersArrray& arr, const int& low, const int& high)

{

int mid = (low + high) / 2;

if (arr[low] < arr[mid] && arr[mid] < arr[high])

return mid;

if (arr[low] < arr[high] && arr[high] <= arr[mid])

return high;

if (arr[mid] <= arr[low] && arr[low] < arr[high])

return low;

if (arr[mid] < arr[high] && arr[high] <= arr[low])

return high;

if (arr[high] <= arr[low] && arr[low] < arr[mid])

return low;

if (arr[high] <= arr[mid] && arr[mid] <= arr[low])

return mid;

}

void MainWindow::IntrosortUtil(NumbersArrray& arr, int begin, int end, int depthLimit)

{

int size = end - begin + 1;

if (size < 16)

{

sorting\_data.addComparisons(1);

InsertionSort(arr, begin, end);

return;

}

sorting\_data.addComparisons(2);

if (depthLimit == 0)

{

heapSort(arr, begin, end);

return;

}

int pivot = MedianOfThree(arr, begin, end);

if (animation\_is\_on) swapCalled(arr, pivot, end);

swap(arr[pivot], arr[end]);

sorting\_data.addSwaps(1);

int partitionPoint = partition(arr, begin, end);

IntrosortUtil(arr, begin, partitionPoint-1, depthLimit - 1);

IntrosortUtil(arr, partitionPoint + 1, end, depthLimit - 1);

return;

}

/\* Implementation of introsort\*/

void MainWindow::Introsort(NumbersArrray& arr, int begin, int end)

{

int depthLimit = 2 \* log(end - begin + 1);

// qDebug() << "end - begin + 1 =" << end - begin + 1;

// qDebug() << "log(end - begin + 1) =" << log(end - begin + 1);

// qDebug() << "depthLimit:" << depthLimit;

IntrosortUtil(arr, begin, end, depthLimit);

return;

}

void MainWindow::on\_MinValueBox\_valueChanged(int arg1)

{

min\_number = arg1;

ui->MaxValueBox->setMinimum(arg1);

}

void MainWindow::on\_MaxValueBox\_valueChanged(int arg1)

{

max\_number = arg1;

}

void MainWindow::on\_SizeBox\_valueChanged(int arg1)

{

sorting\_data.setN(arg1);

ui->textBrowser->setPlainText("Кількість чисел для генерації: " + QString::number(arg1));

}

void MainWindow::on\_Generate\_clicked()

{

srand((unsigned) time(nullptr));

NumbersArrray Generated(sorting\_data.getN());

for (int i = 0; i < sorting\_data.getN(); i++)

{

Generated.pushBack( min\_number + rand() % (max\_number - min\_number + 1) );

}

ui->Unsorted->setChecked(true);

sortedArray = Generated;

unsortedArray = Generated;

ui->textBrowser->setPlainText( unsortedArray.getString() );

}

void MainWindow::on\_MethodBox\_activated(int index)

{

sorting\_data.setSortingMethod(index);

}

void MainWindow::on\_Sort\_clicked()

{

if (!unsortedArray.isEmpty())

{

sortedArray = unsortedArray;

sorting\_data.reset();

animation\_is\_on = false;

auto start = high\_resolution\_clock::now();

switch(sorting\_data.method()) {

case 0:

mergeSort(sortedArray, 0, sortedArray.length() - 1);

break;

case 1:

quickSort(sortedArray, 0, sortedArray.length() - 1);

break;

case 2:

Introsort(sortedArray, 0, sortedArray.length() - 1);

break;

}

auto end = high\_resolution\_clock::now();

auto time\_taken = duration\_cast<microseconds>(end - start);

sorting\_data.addTime(time\_taken.count());

ui->Sorted->setChecked(true);

ui->textBrowser->setPlainText(sortedArray.getString());

}

else

{

QMessageBox::warning(this, "Помилка", "Масив для сортування не згенерований");

}

}

void MainWindow::on\_Unsorted\_clicked()

{

ui->textBrowser->setPlainText(unsortedArray.getString());

}

void MainWindow::on\_Sorted\_clicked()

{

ui->textBrowser->setPlainText(sortedArray.getString());

}

void MainWindow::on\_AdditionalData\_clicked()

{

ui->textBrowser->setPlainText(sorting\_data.getString());

}

void MainWindow::on\_Animation\_clicked()

{

if (unsortedArray.length() <= 300)

{

animation\_is\_on = true;

numbers\_set->remove(0, numbers\_set->count());

for (int i = 0; i < unsortedArray.length(); i++)

{

\*numbers\_set << unsortedArray[i];

}

clear\_queue(animationQueue);

clear\_queue(valuesQueue);

animationArray = unsortedArray;

series->append(numbers\_set);

chart->addSeries(series);

chart->setTitle("Анімація сортування");

chart->setAnimationOptions(QChart::AllAnimations);

QValueAxis \*axisY = new QValueAxis;

axisY->setRange(min\_number, max\_number);

chart->createDefaultAxes();

chart->setAxisY(axisY, series);

chartView->setRenderHint(QPainter::Antialiasing);

QPalette pal = qApp->palette();

pal.setColor(QPalette::Window, QRgb(0xffffff));

pal.setColor(QPalette::WindowText, QRgb(0x404040));

animation\_window->setCentralWidget(chartView);

animation\_window->show();

ui->Animation->setEnabled(false);

Swaptimer->start(animation\_duration);

switch (sorting\_data.method()) {

case 0:

mergeSort(animationArray, 0, animationArray.length() - 1);

break;

case 1:

quickSort(animationArray, 0, animationArray.length() - 1);

break;

case 2:

Introsort(animationArray, 0, animationArray.length() - 1);

break;

default:

break;

}

ui->Animation->setEnabled(true);

}

else

{

animation\_is\_on = false;

QMessageBox::warning(this, "Помилка", "Розмірність масиву занадто велика і перевищує 300");

}

}

void MainWindow::on\_Save\_clicked()

{

QString fileName = QFileDialog::getSaveFileName(this, "Save as");

QFile file(fileName);

if (!file.open(QFile::WriteOnly | QFile::Text))

{

QMessageBox::warning(this, "Помилка", "Неможливо зберегти файл");

return;

}

currentFile = fileName;

setWindowTitle(fileName);

QTextStream out(&file);

out << sorting\_data.getString() << "\n"

<< "Початковий масив: " << unsortedArray.getString() << "\n"

<< "Відсортований масив: " << sortedArray.getString();

file.close();

}

void MainWindow::on\_asymtoticButton\_clicked()

{

QLineSeries \*series1 = new QLineSeries();

QLineSeries \*series2 = new QLineSeries();

QLineSeries \*series3 = new QLineSeries();

QLineSeries \*series4 = new QLineSeries();

for (int i = 1; i < 10000; i+=100)

{

sorting\_data.reset();

srand((unsigned) time(nullptr));

NumbersArrray Generated(i);

for (int j = 0; j < i; j++)

{

Generated.pushBack( min\_number + rand() % (max\_number - min\_number + 1) );

}

NumbersArrray CopyArr = Generated;

NumbersArrray Copy2Arr = Generated;

// auto start = high\_resolution\_clock::now();

mergeSort(Generated, 0, Generated.length() - 1);

// auto end = high\_resolution\_clock::now();

// auto time\_taken = duration\_cast<microseconds>(end - start);

series1->append(i, sorting\_data.getComparisons() + sorting\_data.getSwaps());

// series1->append(i, sorting\_data.getSwaps());

// series1->append(i, sorting\_data.getComparisons());

// series1->append(i, time\_taken.count());

sorting\_data.reset();

// start = high\_resolution\_clock::now();

quickSort(CopyArr, 0, CopyArr.length() - 1);

// end = high\_resolution\_clock::now();

// time\_taken = duration\_cast<microseconds>(end - start);

qDebug() << sorting\_data.getComparisons();

series2->append(i, sorting\_data.getSwaps() + sorting\_data.getComparisons());

// series2->append(i, sorting\_data.getSwaps());

// series2->append(i, sorting\_data.getComparisons());

// series2->append(i, time\_taken.count());

sorting\_data.reset();

// start = high\_resolution\_clock::now();

Introsort(Copy2Arr, 0, Copy2Arr.length() - 1);

// end = high\_resolution\_clock::now();

// time\_taken = duration\_cast<microseconds>(end - start);

series3->append(i, sorting\_data.getSwaps() + sorting\_data.getComparisons());

// series3->append(i, sorting\_data.getSwaps());

// series3->append(i, sorting\_data.getComparisons());

// series3->append(i, time\_taken.count());

series4->append(i, (int) (i \* log2(i)));

}

chart->addSeries(series1);

chart->addSeries(series2);

chart->addSeries(series3);

chart->addSeries(series4);

series1->setName("Сортування злиттям");

series2->setName("Швидке сортування");

series3->setName("Інтроспективне сортування");

series4->setName("Асимптотична оцінка");

// chart->setTitle("Кількість перестановок");

// chart->setTitle("Кількість порівнянь");

chart->setTitle("Кількість операцій");

chart->createDefaultAxes();

chartView->setRenderHint(QPainter::Antialiasing);

animation\_window->setCentralWidget(chartView);

animation\_window->show();

}

animationwindow.cpp:

#include "animationwindow.h"

#include "ui\_animationwindow.h"

AnimationWindow::AnimationWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::AnimationWindow)

{

ui->setupUi(this);

}

AnimationWindow::~AnimationWindow()

{

delete ui;

}

void AnimationWindow::on\_slower\_triggered()

{

emit intervalIncreased();

}

void AnimationWindow::on\_faster\_triggered()

{

emit intervalDecreased();

}

numbersarray.cpp:

#include "numbersarray.h"

NumbersArrray::NumbersArrray()

{

n = 0; max\_capacity = 1;

numbers = new int[1];

}

NumbersArrray::NumbersArrray(int n1)

{

n = 0;

max\_capacity = n1;

numbers = new int[n1];

}

NumbersArrray::NumbersArrray(NumbersArrray& other)

{

n = other.n;

max\_capacity = other.max\_capacity;

numbers = new int[other.n];

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

{

numbers[i] = other[i];

}

}

void NumbersArrray::doubleCapacity()

{

max\_capacity \*= 2;

int\* temporary\_array = new int[max\_capacity];

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

{

temporary\_array[i] = numbers[i];

}

delete[] numbers;

numbers = temporary\_array;

}

int NumbersArrray::length()

{

return n;

}

bool NumbersArrray::isEmpty()

{

return (n == 0);

}

void NumbersArrray::pushBack(int value)

{

if (n == max\_capacity)

{

doubleCapacity();

}

numbers[n] = value;

n++;

}

int& NumbersArrray::operator[](int index)

{

return numbers[index];

}

NumbersArrray NumbersArrray::operator=(const NumbersArrray& other)

{

if (other.max\_capacity > max\_capacity)

{

max\_capacity = other.max\_capacity;

delete[] numbers;

numbers = new int[max\_capacity];

}

n = other.n;

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

{

numbers[i] = other.numbers[i];

}

return \*this;

}

QString NumbersArrray::getString()

{

QString s;

for (int i = 0; i < n; i++) s += QString::number(numbers[i]) + " ";

return s;

}

sortingdata.cpp:

#include "sortingdata.h"

SortingData::SortingData()

{

swaps = 0; comparisons = 0; sorting\_time = 0.0;

n = 100; sorting\_method = 0;

}

int SortingData::getN()

{

return n;

}

QString SortingData::getString()

{

QString result;

result += "Кількість чисел (n): " + QString::number(n) + "\n";

result += "Метод сортування: " + getSortingMethod() + "\n";

result += "Кількість порівнянь: " + QString::number(comparisons) + "\n";

result += "Кількість перестановок: " + QString::number(swaps) + "\n";

result += "Час сортування: " + QString::number(sorting\_time) + " мкс";

return result;

}

QString SortingData::getSortingMethod()

{

switch (sorting\_method) {

case 0:

return "злиттям (Д. фон Неймана)";

break;

case 1:

return "швидке сортування";

break;

case 2:

return "інтроспективне сортування";

break;

default:

break;

}

}

int SortingData::getComparisons()

{

return comparisons;

}

int SortingData::getSwaps()

{

return swaps;

}

int SortingData::getTime()

{

return sorting\_time;

}

int SortingData::method()

{

return sorting\_method;

}

void SortingData::SetComparisons(int new\_comparisons)

{

comparisons = new\_comparisons;

}

void SortingData::setSwaps(int new\_swaps)

{

swaps = new\_swaps;

}

void SortingData::SetSortingTime(double sorting\_time1)

{

sorting\_time = sorting\_time1;

}

void SortingData::setN(int new\_number)

{

n = new\_number;

}

void SortingData::setSortingMethod(int method\_number)

{

sorting\_method = method\_number;

}

void SortingData::addComparisons(int number)

{

comparisons += number;

}

void SortingData::addSwaps(int number)

{

swaps += number;

}

void SortingData::addTime(double time\_in\_ms)

{

sorting\_time += time\_in\_ms;

}

void SortingData::reset()

{

swaps = 0; comparisons = 0; sorting\_time = 0;

}