МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

КУРСОВА РОБОТА

з дисципліни «Основи програмування 2. Модульне програмування»

на тему «Упорядкування масивів»

Студента 1 курсу, групи ІП-23
Зубарева Миколи Костянтиновича
Спеціальності 121 «Інженерія програмного забезпечення»

Керівник
Ст. Викладач Головченко М. М.

Кількість балів: _____
Національна оцінка _____

Члени комісії

_____	ст. вик. Головченко М.М.
(підпис)	(посада, вчене звання, науковий ступінь, прізвище та ініціали)
_____	асистент Вовк С.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

Зубарева Миколи Костянтиновича

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

Вступ, постановка задачі, теоретичні відомості, опис алгоритмів, опис програмного забезпечення, результати тестування програмного забезпечення, інструкція користувача, висновок, перелік посилань

5. Перелік графічного матеріалу (з точним зазначенням обовязкових креслень)

Скріншоти тестування спроєктованого програмного забезпечення

6. Дата видачі завдання 06.03.2023

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	06.03.2023	
2.	Підготовка ТЗ	27.03.2023	
3.	Пошук та вивчення літератури з питань курсової роботи	27.03.2023	
4.	Розробка сценарію роботи програми	05.04.2023	
5.	Узгодження сценарію роботи програми з керівником	11.04.2023	
6.	Розробка (вибір) алгоритму рішення задачі	14.04.2023	
7.	Узгодження алгоритму з керівником	18.04.2023	
8.	Узгодження з керівником інтерфейсу користувача	18.04.2023	
9.	Розробка програмного забезпечення	22.04.2023	
10.	Налагодження розрахункової частини програми	3.05.2023	
11.	Розробка та налагодження інтерфейсної частини програми	11.05.2023	
12.	Узгодження з керівником набору тестів для контрольного прикладу	16.05.2023	
13.	Тестування програми	20.05.2023	
14.	Підготовка пояснювальної записки	25.05.2023	
15.	Здача курсової роботи на перевірку	31.05.2023	
16.	Захист курсової роботи	07.06.2023	

Студент Зубарев М.К.

(підпис)

Керівник _____ Головченко М. М.

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

"___" _____20__ р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 74 сторінок, 21 рисунок, 15 таблиць, 3 посилання.

Мета роботи: розробка якісного та ефективного програмного забезпечення для наочної демонстрації алгоритмів упорядкування масиву (блочне сортування, сортування підрахунком, порозрядне сортування)

Курсова робота присвячена вивченню та порівнянню трьох методів сортування масивів: блочного сортування, сортування підрахунком та порозрядного сортування. В роботі детально розглянуті особливості кожного з методів.

У першому розділі описано постановку задачі.

У другому розділі описані теоретичні відомості про кожний з описаних вище алгоритмів.

У третьому розділі описані алгоритми за допомогою псевдокоду (алгоритми блочного сортування, сортування підрахунком та порозрядне сортування).

У четвертому розділі описане програмне забезпечення за допомогою діаграми класів та таблиці з використаними методами та класами.

У пятому розділі описано процес тестування кінцевого програмного забезпечення.

У шостому розділі наведена інструкція користувача.

У сьомому розділі описано тестування алгоритмів сортування.

ЗМІСТ

ВСТУП

Упорядкування масивів є важливою операцією у багатьох областях програмування та аналізу даних. Існує безліч методів сортування, і кожен з них має свої переваги та обмеження. У даній курсовій роботі досліджується ефективність трьох методів сортування: блочного сортування, сортування підрахунком та порозрядного сортування.

Впродовж останніх десятиліть було виконано значну кількість досліджень та вдосконалень у сфері сортування масивів. Метод блочного сортування базується на розбитті масиву на блоки та подальшому сортуванні кожного блоку. Сортування підрахунком використовує підрахунок кількості елементів з певним значенням для упорядкування масиву. Порозрядне сортування розбиває елементи масиву на окремі розряди та сортує їх послідовно.

Метою даної роботи є детальний аналіз та порівняння цих трьох методів сортування. Розглянемо їх алгоритми, переваги та обмеження, а також проведемо аналіз часової складності для кожного з методів. Додатково, буде розроблено програмне забезпечення для оцінки ефективності цих методів у реальних сценаріях сортування масивів.

Ця курсова робота сприятиме розумінню різних методів сортування та надасть цінні відомості для вибору найефективнішого методу у конкретних завданнях сортування масивів.

1. ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення для упорядкування масиву, використовуючи наступні алгоритми сортування:

a) блочне сортування;

b) сортування підрахунком;

c) порозрядне сортування;

Вхідними даними для даної роботи є масив не менше 100 та не більше 50000 елементів. Програма надає користувачу можливість вибрати необхідний метод сортування, генерувати випадковий масив, зберігати результати на диск, подивитись анімацію сортування, а також можна подивитись графік характеристик алгоритмів, які відображають їх практичну складність.

Вихідними даними для даної роботи являється результат сортування масиву заданим методом. Ця курсова робота дозволяє отримати глибше розуміння трьох методів сортування масивів та надає корисну інформацію для вибору найефективнішого методу в конкретних завданнях сортування даних.

2. ТЕОРЕТИЧНІ ВІДОМОСТІ

Методи сортування блочного, сортування підрахунком та порозрядного сортування є ефективними алгоритмами сортування, які застосовуються для упорядкування масивів даних.

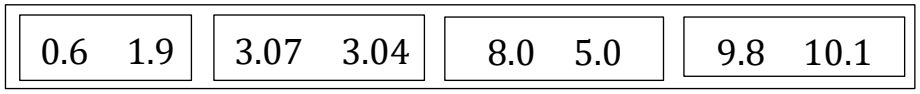
1. Сортування блочного (Bucket Sort):

Нехай дано масив Input array. Розмір цього масиву 8 елементів

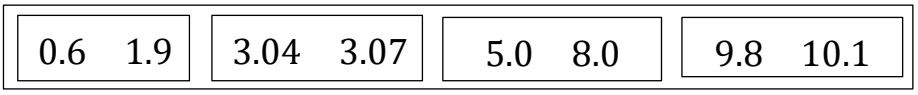
Input array

9.8 0.6 10.1 1.9 3.07 3.04 8.0 5.0

Unsorted buckets



Sorted buckets



Output array

0.6 1.9 3.04 3.07 5.0 8.0 9.8 10.1

Цей метод сортування розподіляє елементи масиву в окремі "корзини" або "блоки" відповідно до їх значень. Кожен блок може бути відсортований окремо, застосовуючи будь-який інший алгоритм сортування або рекурсивно використовуючи той самий алгоритм сортування блочного. Після сортування блоків вони зєднуються, утворюючи впорядкований масив. Отже, загальна часова складність блочного сортування буде O(n + klogk). Зазвичай вважається, що якщо кількість блоків k є постійною або залежить від n (наприклад, k = O(n)), то ця складність може бути спрощена до O(n).

2. Сортування підрахунком (Counting Sort):

Нехай дано масив з 5 елементів. Застосуємо для нього алгоритм сортування підрахунком.

Input array

3 1 1 5 2

0	2	1	1	0	1	Кількість входжень
0	1	2	3	4	5	Значення елементів

0	2	3	4	4	5	Номер позиції
0	1	2	3	4	5	Значення елементів

1	1	2	3	5	Номер елементу
0	1	2	3	4	Номер позиції

Output array

1 1 2 3 5

Цей метод сортування використовує підрахунок кількості входжень кожного елемента у масиві. Створюється допоміжний масив, в якому зберігаються кількості входжень кожного елемента. Обчислюємо кумулятивну суму підрахованих значень, щоб встановити правильні позиції кожного елемента у відсортованому масиві. Проходимося по вхідному масиву ще раз і розміщуємо кожен елемент у відповідній позиції в відсортованому масиві за допомогою кумулятивної суми. Отже, загальна часова складність сортування підрахунком складається з двох частин: O(n + k). Важливо зазначити, що часова складність не залежить від значень елементів у вхідному масиві, але залежить від діапазону цих значень.

3. Порозрядне сортування (Radix Sort):

Цей метод сортування базується на розрядному представленні чисел. Він сортує елементи масиву, спочатку порівнюючи їх за найменш значущим розрядом, потім застосовуючи сортування до наступного розряду. Розглянемо кроки алгоритму більш детально:

- Прохід по розрядам: Алгоритм виконує проходи по розрядам елементів, починаючи з найменш значущого розряду до найбільш значущого.
- Підрахунок кількості елементів: На кожному проході по розряду проходимося по всім елементам масиву та підраховуємо, скільки елементів мають певне значення на поточному розряді.

- Кумулятивна сума: Обчислюємо кумулятивну суму підрахованих значень, щоб встановити правильні позиції кожного елемента у відсортованому масиві на поточному розряді.
- Побудова відсортованого масиву: Проходимося по вхідному масиву ще раз і розміщуємо кожен елемент у відповідній позиції в відсортованому масиві на поточному розряді.

Зазвичай використовуються стабільні сортування на кожному кроці для збереження порядку елементів з однаковими значеннями розрядів. Отже, загальна часова складність порозрядного сортування становить O(d * (n + k)), де d - кількість розрядів, n - кількість елементів у масиві, а k - кількість можливих значень на кожному розряді. Зауважимо, що якщо d та k є константами або малими значеннями, то часова складність може спроститись до O(n), що робить порозрядне сортування досить ефективним для великих масивів з невеликим діапазоном значень.

3. ОПИС АЛГОРИТМІВ

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

Таблиця 3.1. — основні змінні та їхні призначення

Змінна	Призначення
array	Масив для сортування
max_value	Максимальний елемент масиву
min_value	Мінімальний елемент масиву
size	Розмір масиву
sorted_array	Результат сортування

3.1. Загальний алгоритм

1. ПОЧАТОК
2. Ввести size масиву
3. Натиснути кнопку «Згенерувати».
4. Вивести згенерований array у новому вікні.
5. Обрати алгоритм сортування.
6. Натиснути кнопку «Сортувати».
7. ЯКЩО масив не згенеровано, ТО видати повідомлення про те, що треба спочатку треба згенерувати масив.
8. ЯКЩО обрано алгоритм блочного сортування, ТО відсортувати масив згідно блочного алгоритму сортування (підрозділ 3.2.)
9. ЯКЩО обрано алгоритм сортування підрахунком, ТО відсортувати масив згідно алгоритму сортування підрахунком (підрозділ 3.3.)
10. ЯКЩО обрано алгоритм порозрядного сортування, ТО відсортувати масив згідно алгоритму порозрядного сортування (підрозділ 3.4.)
11. Вивести sorted_array у новому вікні.
12. КІНЕЦЬ

3.2 Алгоритм блочного сортування(*array*)

1. ПОЧАТОК
2. Знайдення максимального значення max_val та мінімального значення min_val у масиві array.

3. Обчислення діапазону range_val, який є різницею між max_val та min_val.
4. Визначення кількості блоків size як довжину масиву array.
5. Обчислення розміру кожного блоку bucket_size як частку range_val на size.
6. Створення порожніх блоків buckets за допомогою спискового зображення з size порожніх списків.
7. Розподілення елементів масиву array по блоках:
   7.1. Для кожного числа num у масиві array:
      7.1.1. Обчислення індексу блоку index_b за допомогою формули (num - min_val) // range_val.
      7.1.2. Якщо index_b дорівнює num_buckets, зменшити його на 1.
      7.1.3. Додавання числа num до блоку з індексом index_b в buckets.
8. Зібрання відсортованих елементів з блоків:
   8.1. Оголошення порожнього масиву sorted_array.
   8.2. Для кожного блоку bucket у buckets:
      8.2.1. Сортування блоку bucket за допомогою сортування вставкою:
         8.2.1.1. Для кожного індексу i від 1 до довжини блоку bucket:
            8.2.1.1.1. Запамятати елемент key з позиції i.
            8.2.1.1.2. Ініціалізація лічильника j як i - 1.
            8.2.1.1.3. Допоки j не стане відємним або bucket[j] більше key:
               8.2.1.1.3.1. Зсунути bucket[j] на наступну позицію.
               8.2.1.1.3.2. Зменшити значення j на 1.
            8.2.1.1.4. Присвоїти key на позицію j + 1 у блоку bucket.
      8.2.2. Додавання елементів блоку bucket до sorted_array.
9. Повернення відсортованого масиву sorted_array.
10. КІНЕЦЬ.

3.3 Алгоритм сортування підрахунком(*array*)

1. ПОЧАТОК
2. Отримання розміру масиву size як довжину array.
3. Створення порожнього масиву output довжиною size та заповнення його нулями.

4. Знаходження максимального значення max_value у масиві array.
5. Ініціалізація порожнього масиву count довжиною (max_value + 1).
6. Обчислення кількості кожного елементу у масиві array та збереження їх у масиві count:
   6.1. Для кожного індексу i від 0 до size - 1:
      6.1.1. Збільшення значення count[array[i]] на 1.
7. Обчислення накопиченої кількості елементів у масиві count:
   7.1. Для кожного індексу i від 1 до max_value:
      7.1.1. Додавання значення count[i - 1] до count[i].
8. Ініціалізація лічильника i як size - 1.
9. Поки i не стане меншим за 0, виконувати наступне:
   9.1. Присвоєння output[count[array[i]] - 1] значення array[i].
   9.2. Зменшення значення count[array[i]] на 1.
   9.3. Зменшення значення i на 1.
10. Присвоєння відсортованого масиву output змінній sorted_array.
11. Повернення відсортованого масиву sorted_array.
12. КІНЕЦЬ.

3.4 Алгоритм порозрядного сортування(*array*)

1. ПОЧАТОК
2. Оголошення функції countingSort, яка приймає масив array та позицію place як вхідні параметри.
1. Отримання розміру масиву size як довжину array.
2. Створення порожнього масиву output довжиною size та заповнення його нулями.
3. Знаходження максимального значення max_value у масиві array.
4. Ініціалізація порожнього масиву count довжиною (max_value + 1).
5. Обчислення кількості елементів у масиві array на основі позиції place та збереження їх у масиві count:
   5.1. Для кожного індексу i від 0 до size - 1:
      5.1.1. Обчислення index як цілочисельного частку від ділення array[i] на place.

5.1.2. Збільшення значення count[index % 10] на 1.

6. Обчислення накопиченої кількості елементів у масиві count:
   6.1. Для кожного індексу i від 1 до 9:
      6.1.1. Додавання значення count[i - 1] до count[i].
7. Ініціалізація лічильника i як size - 1.
8. Поки i не стане меншим за 0, виконувати наступне:
   8.1. Обчислення index як цілочисельного частку від ділення array[i] на place.
   8.2. Присвоєння output[count[index % 10] - 1] значення array[i].
   8.3. Зменшення значення count[index % 10] на 1.
   8.4. Зменшення значення i на 1.
9. Створення порожнього масиву sorted_array.
10. Копіювання елементів з масиву output до sorted_array.
11. Повернення відсортованого масиву sorted_array.

12. Оголошення функції radix_sort, яка приймає масив array як вхідний параметр.
13. Знаходження максимального елемента max_element у масиві array.
14. Створення копії масиву sorted_array, яка буде використовуватись для збереження відсортованих елементів.
15. Застосування алгоритму countingSort для сортування елементів на основі позицій значень:
   15.1. Ініціалізація змінної place як 1.
   15.2. Поки max_element // place більше 0, виконувати наступне:
      15.2.1. Присвоєння sorted_array результату виклику функції countingSort з параметрами sorted_array та place.
      15.2.2. Збільшення значення place на 10.
16. Повернення відсортованого масиву sorted_array.
17. КІНЕЦЬ

4. ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Діаграма класів програмного забезпечення зображена на рисунку 4.1

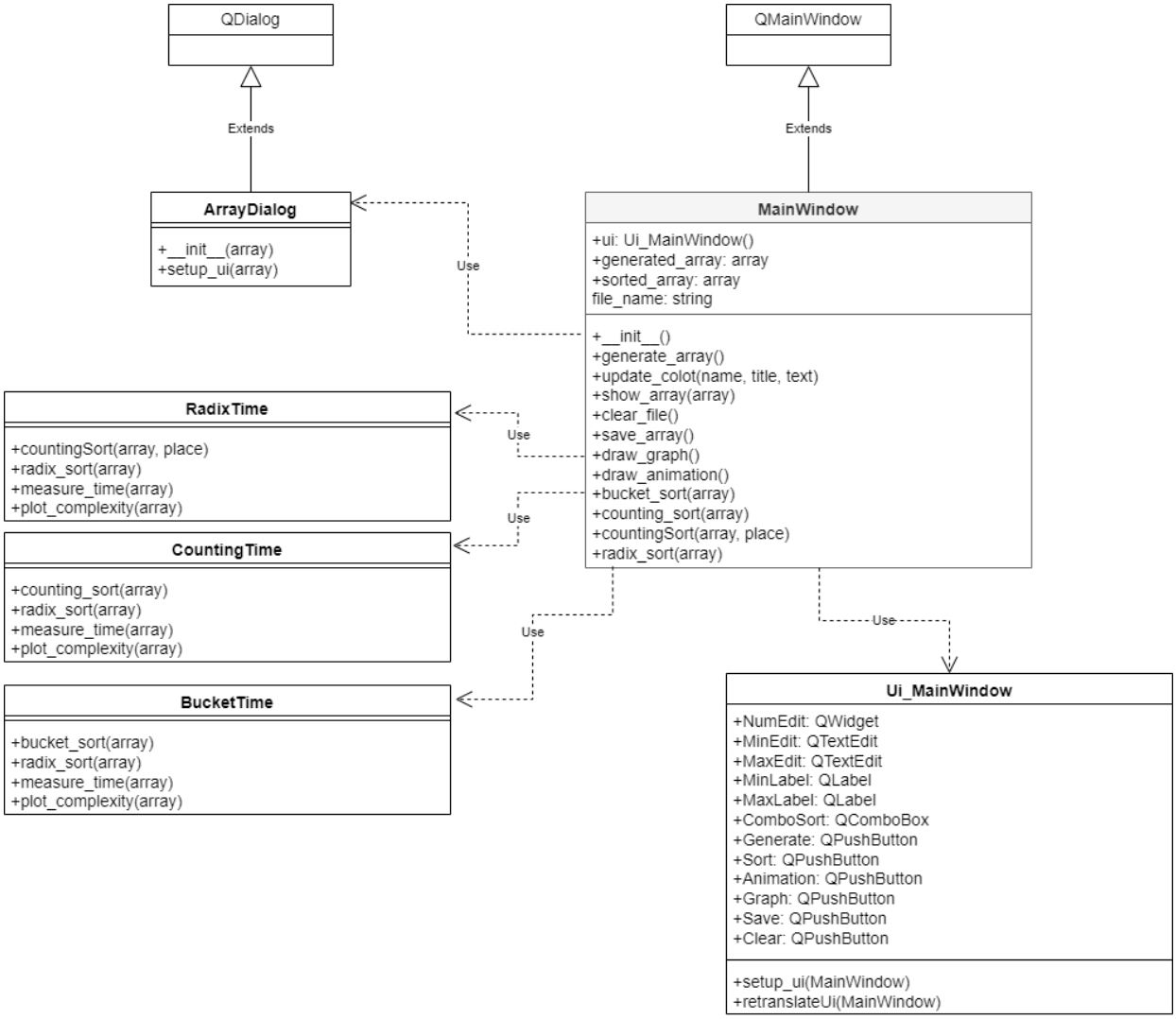


Рисунок 4.1 – Діаграма класів

У таблиці 4.1 наведено користувацькі функції, які були застосовані для реалізації задачі.

Таблиця 4.1 - Користувацькі функції

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	MainWindow	__init__	Ініціалізація атрибутів класу	Немає	Немає
2	MainWindow	update_color	Оновлення кольору QMessageBox	Назва, заголовок,	Немає

				текст повідомлення	
3	MainWindow	generate_array	Генерація масиву	Немає	Немає
4	MainWindow	show_array	Відображення масиву у новому вікні	Масив, який буде відображено	Немає
5	MainWindow	sort_array	Сортує масив заданим алгоритмом	Немає	Немає
6	MainWindow	draw_animation	Виводить анімацію заданого алгоритма	Немає	Немає
7	MainWindow	clear_file	Очистка файлу	Немає	Немає
8	MainWindow	bucket_sort	Алгоритм блочного сортування	Масив для сортування	Відсортований масив
9	MainWindow	counting_sort	Алгоритм сортування підрахунком	Масив для сортування	Відсортований масив
10	MainWindow	countingSort	Внутрішня функція	Масив та позицію	Немає
11	MainWindow	radix_sort	Алгоритм сортування	Масив для сортування	Відсортований масив
12	MainWindow	draw_graph	Відображання характеристик заданого алгоритма	Немає	Багаторядковий string
13	ArrayDialog	__init__	Ініціалізація атрибутів класу	Масив, який буде виведено у вікно	Немає
14	ArrayDialog	setup_ui	Виведення масиву у вікні	Масив	
15	BucketTime	__init__	Ініціалізація атрибутів класу	Немає	Немає

16	BucketTime	measure_time	Вираховування середнього часу	Масив, щоб дізнатись час сортування	Середній час
17	BucketTime	plot_complexity	Вивід графіку часової складності певного алгоритму	Масив	Немає
18	BucketTime	bucket_sort	Алгоритм блочного сортування	Масив для сортування	Відсортований масив та кількість ел. операцій
19	RadixTime	radix_sort	Алгоритм порозрядного сортування	Масив для сортування	Відсортований масив та кількість елементарних операцій
20	RadixTime	countingSort	Внутрішня функція	Масив для сортування та розряд	Відсортований масив та кількість елементарних операцій
21	CountingTime	counting_sort	Алгоритм сортування підрахунком	Масив для сортування	Відсортований масив та кількість елементар

					них операцій
22	Ui_MainWindow	setup_ui	Встановлює вигляд головного вікна	MainWindow	Немає
23	Ui_MainWindow	retranslateUi	Налаштовує тексти елементів інтерфейсу	MainWindow	Немає

4.2 Стандартні функції

У таблиці 4.2 наведено стандартні функції, які були використані для реалізації поставленої задачі

Таблиця 4.2 – Стандартні функції

№ п/п	Назва классу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1.	PySide6.QtWidgets.QApplication	__init__	Ініціалізація додатку	Список аргументів командного рядка	Немає
2.	PySide6.QtWidgets.QApplication	exec	Запуск головного циклу додатку	Немає	Немає
3.	PySide6.QtWidgets.QMainWindow	__init__	Створення головного вікна	Немає	Немає
4.	PySide6.QtWidgets.QMainWindow	setWindowTitle	Встановлення заголовку вікна	Заголовок	Немає
5.	PySide6.QtWidgets.QMainWindow	resize	Зміна розміру вікна	Ширина,висота	Немає

6.	PySide6.QtWidgets.QMainWindow	centralWidget	Встановленя центрального віджета	QWidget	Немає
7.	PySide6.QtWidgets.QWidget	__init__	Створення віджета	Батьківський віджет	Немає
8.	PySide6.QtWidgets.QWidget	show	Відображення віджета	Немає	Немає
9.	PySide6.QtWidgets.QLabel	__init__	Створення віджета	Текст	Немає
10.	PySide6.QtWidgets.QLabel	setGeometry	Встановлення розміру	Висота, ширина	Немає
11.	PySide6.QtWidgets.QTextEdit	__init__	Створення поля для вводу	Текст	Немає
12.	PySide6.QtWidgets.QTextEdit	setGeometry	Встановлення розміру	Висота, ширина	Немає
13.	PySide6.QtWidgets.QTextEdit	text	Отримання тексту з поля для вводу	Текст	Немає
14.	PySide6.QtWidgets.QTextEdit	setText	Встановлення тексту у полі для вводу	Текст	Немає
15.	PySide6.QtWidgets.QPushButton	__init__	Створення кнопки	Текст, батьківськ13ий клас	Немає

16.	PySide6.QtWidgets.QPushButton	clicked	Сигнал про натискання кнопки	Немає	Немає
17.	PySide6.QtWidgets.QMessageBox	critical	Відображення повідомлення про помилку	Батьківський відмет, заголовок, текст	Немає
18.	PySide6.QtWidgets.QMessageBox	information	Відображення інформаційного повідомлення	Батьківський віджет, заголовок, текст	Немає
19.	PySide6.QtWidgets.QComboBox	__init__	Створення випадаючого списку	Батьківський віджет	Немає
20.	PySide6.QtWidgets.	addItem	Додавання елементу до списку	Текст	Немає

5. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

В розробці якісного програмного забезпечення тестування відіграє важливу роль і є одним з ключових етапів процесу. Головна мета тестування полягає в перевірці відповідності програмного забезпечення вимогам, визначеним у технічному завданні. Це дозволяє забезпечити впевненість у якості програми та відсутності непередбачуваних проблем, які мають бути виявлені й вирішені розробником, а не користувачем програми.

Тестування програмного забезпечення включає дві основні складові: перевірку основного функціоналу програми та випробування її реакції на виняткові ситуації. Для випробування реакції на виняткові ситуації розроблено план тестування, який ми зараз опрацюємо.

5.1 План тестування

1. Тестування правильності введених значень.
    1.1. Тестування при введені некоректних даних.
    1.2. Тестування при введені замалих та завеликих значень розміру масиву.
    1.3. Тестування при введенні замалих та завеликих значень для проміжку генерації випадкових значень.
2. Тестування коректності роботи алгоритмів.
    2.1. Тестування коректності роботи алгоритму блочного сортування.
    2.2. Тестування коректності роботи алгоритму сортування підрахунком.
    2.3. Тестування коректності роботи алгоритму порозрядного сортування.
3. Тестування коректності збереження масиву у файл.
    3.1. Можливість збереження масиву у файл.
    3.2. Можливість очищення файлу.
4. Тестування можливості відображення характеристик алгоритмів, які відображають їх практичну складність
5. Можливість відображення процесу сортування у вигляді анімацій.

5.2 Приклади тестування

Тестування роботи програми за планом тестування, наведеним вище. Результати зображені у таблицях і рисунках.

Таблиця 5.1 — тестування при введенні некоректної розмірності масиву

Мета тесту	Перевірити можливість введення некоректних даних
Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Розмірність масиву: 2.6
Схема проведення тесту	Генерація масиву розмірністю не цілочисельного числа
Очікуваний результат	Повідомлення про помилку
Стан програми після проведення випробувань	Видано помилку «Please enter a valid number of characters»

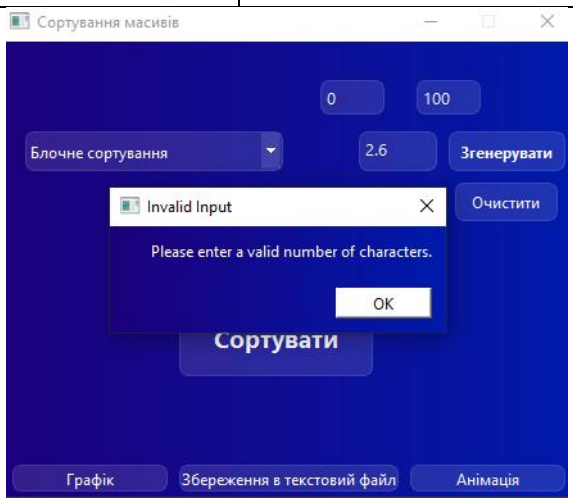


Рисунок 5.1 – Результат виконання програми

Таблиця 5.2 — Тестування при введенні замалих та завеликих значень для розміру масиву

Мета тесту	Перевірити можливість введення замалих та завеликих значень для розмірності масиву
Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Розмір масиву: або менше 10, або більше 50000

Схема проведення тесту	Заповнення масиву завеликими або замалими значеннями
Очікуваний результат	Повідомлення про помилку
Стан програми після проведення випробувань	Видано помилку «Please enter number more than 100 and less than 50000»

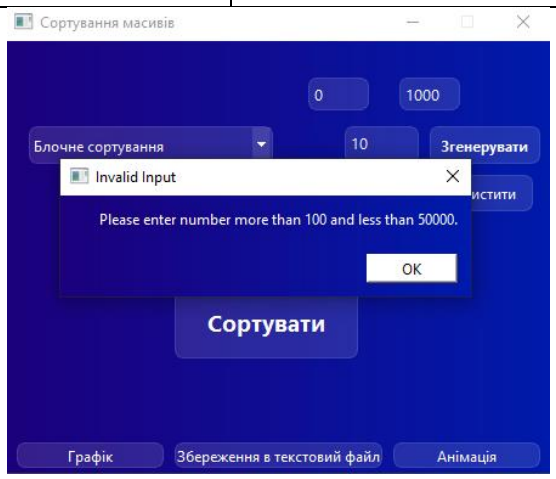


Рисунок 5.2 – Результат роботи програми

Таблиця 5.3 — Тестування при введенні замалих та завеликих значень для діапазону генерації випадкових значень.

Мета тесту	Перевірити неможливість введення початкового значення діапазону більшим ніж кінцеве.
Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Початок діапазону: 10000 Кінець діапазону: 100
Схема проведення тесту	Заповнення початкового поля діапазону генерації масиву числом більшим ніж кінцевого
Очікуваний результат	Помилка.
Стан програми після проведення випробувань	Видано помилку «Minimal value shoulb be less then maximum value»

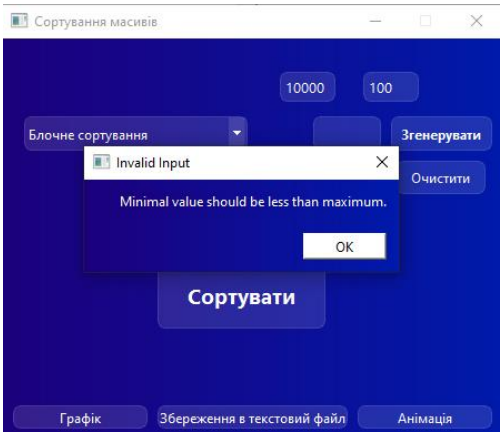


Рисунок 5.3 – Результат роботи програми

Таблиця 5.4 — тестування коректності роботи алгоритму блочного сортування

Мета тесту	Перевірити правильність сортування масиву за допомогою алгоритму блочного сортування
Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Згенерован масив
Схема проведення тесту	Заповнити поле розмірності масиву, вказати діапазон генерації та натиснути кнопку «Сортувати»
Очікуваний результат	Масив – результат сортування початкого масиву
Стан програми після проведення випробувань	Виведено вікно з масивом, який було відсортовано за допомогою блочного сортування.

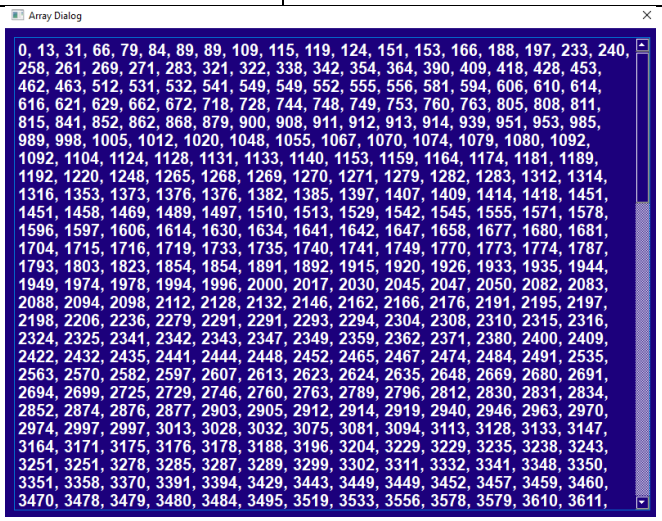


Рисунок 5.4 – Результат роботи програми

Таблиця 5.5 — тестування коректності роботи алгоритму сортування підрахунком

Мета тесту	Перевірити правильність сортування масиву за допомогою алгоритму сортування підрахунком
Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Згенерован масив
Схема проведення тесту	Заповнити поле розмірності масиву, вказати діапазон генерації та натиснути кнопку «Сортувати»
Очікуваний результат	Масив – результат сортування початкого масиву
Стан програми після проведення випробувань	Виведено вікно з масивом, який було відсортовано за допомогою сортування підрахунком.

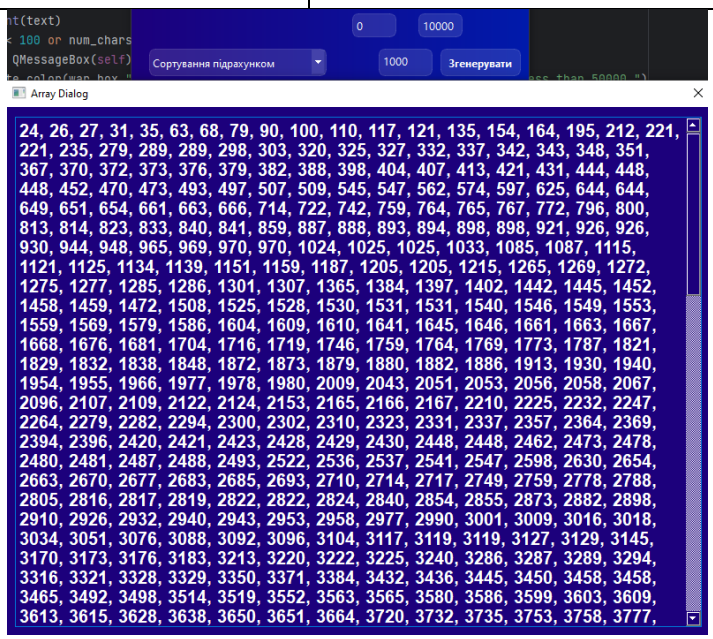


Рисунок 5.5 – Результат роботи програми

Таблиця 5.6 — тестування коректності роботи алгоритму порозрядного сортування

Мета тесту	Перевірити правильність сортування масиву за допомогою алгоритму порозрядного сортування

Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Згенерован масив
Схема проведення тесту	Заповнити поле розмірності масиву, вказати діапазон генерації та натиснути кнопку «Сортувати»
Очікуваний результат	Масив – результат сортування початкого масиву
Стан програми після проведення випробувань	Виведено вікно з масивом, який було відсортовано за допомогою блочного сортування.

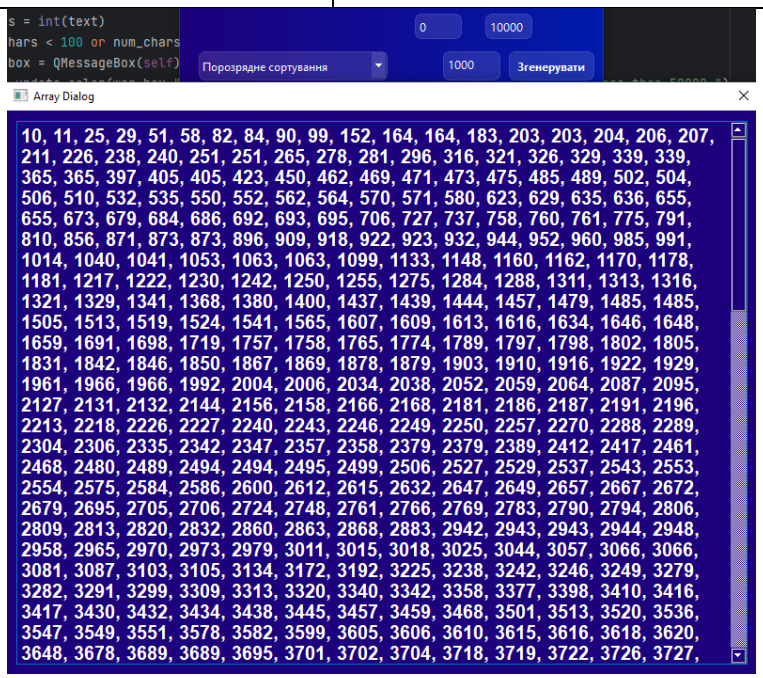


Рисунок 5.6 – Результат роботи програми

Таблиця 5.7 — тестування коректності збереження масиву у файл

Мета тесту	Перевірити коректність процесу зберігання масиву у файл
Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Згенерована масив
Схема проведення тесту	Натиснути кнопку «Збереження в текстовий файл» і ввести назву файлу
Очікуваний результат	Текстовий файл із інформацією про масиви

Стан програми після проведення випробувань	У робочій директорії з’явився файл з розширенням .txt, який містить у собі згенерований масив та відсортований

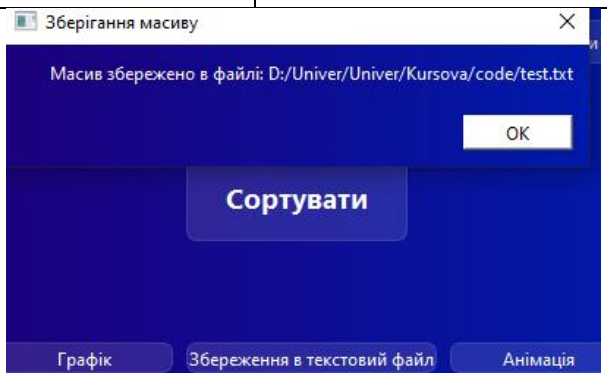


Рисунок 5.7 – Результат роботи програми

Таблиця 5.8 — тестування коректності очищення файлу

Мета тесту	Перевірити коректність процесу очищення файлу
Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Згенерована масив
Схема проведення тесту	Натиснути кнопку «Очистити»
Очікуваний результат	Повідомлення про очищення файлу
Стан програми після проведення випробувань	Видано повідомлення «Файл успішно очіщено» та директорія файлу

Рисунок 5.8 – Результат роботи програми

Таблиця 5.9 — тестування відображення характеристик алгоритмів, які відображають їх практичну складність

Мета тесту	Перевірити коректність відображення характеристик алгоритмів, які відображають їх практичну складність(графік часової складності та кількість елементарних операцій)

Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Згенерован масив
Схема проведення тесту	Заповнити поля розмірностей масиву, заповнити діапазон генерації, обрати алгоритм сортування та натиснути кнопку «Графік»
Очікуваний результат	Вікно із статистикою по заданому алгоритму алгоритму (кількість елементарних операцій, графік часової складності)
Стан програми після проведення випробувань	Виведено вікно з інформацією про кількість елементарних операцій та графік часової складності

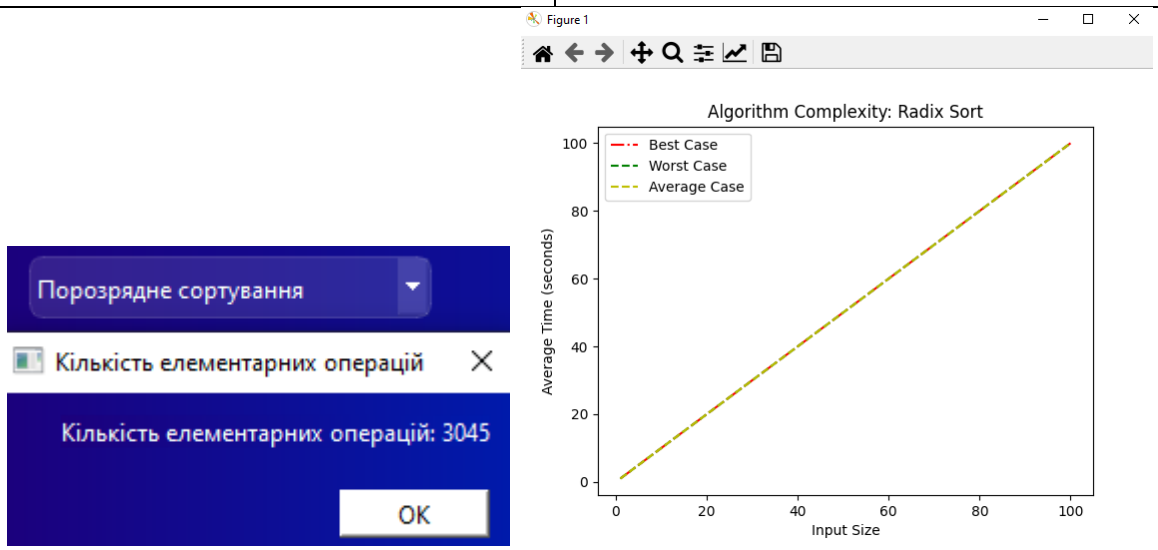


Рисунок 5.9 – Результат роботи програми

Таблиця 5.10 — тестування відображення процесу сортування у вигляді анімацій.

Мета тесту	Перевірити коректність відображення процесу сортування у вигляді анімацій
Початковий стан програми	Відкрите головне вікно програми
Вхідні дані	Згенерован масив
Схема проведення тесту	Заповнити поля розмірностей масиву, заповнити діапазон генерації, обрати

	алгоритм сортування та натиснути кнопку «Анімація»
Очікуваний результат	Вікно з гістограмою сортування масиву
Стан програми після проведення випробувань	Виведено вікно з анімацією

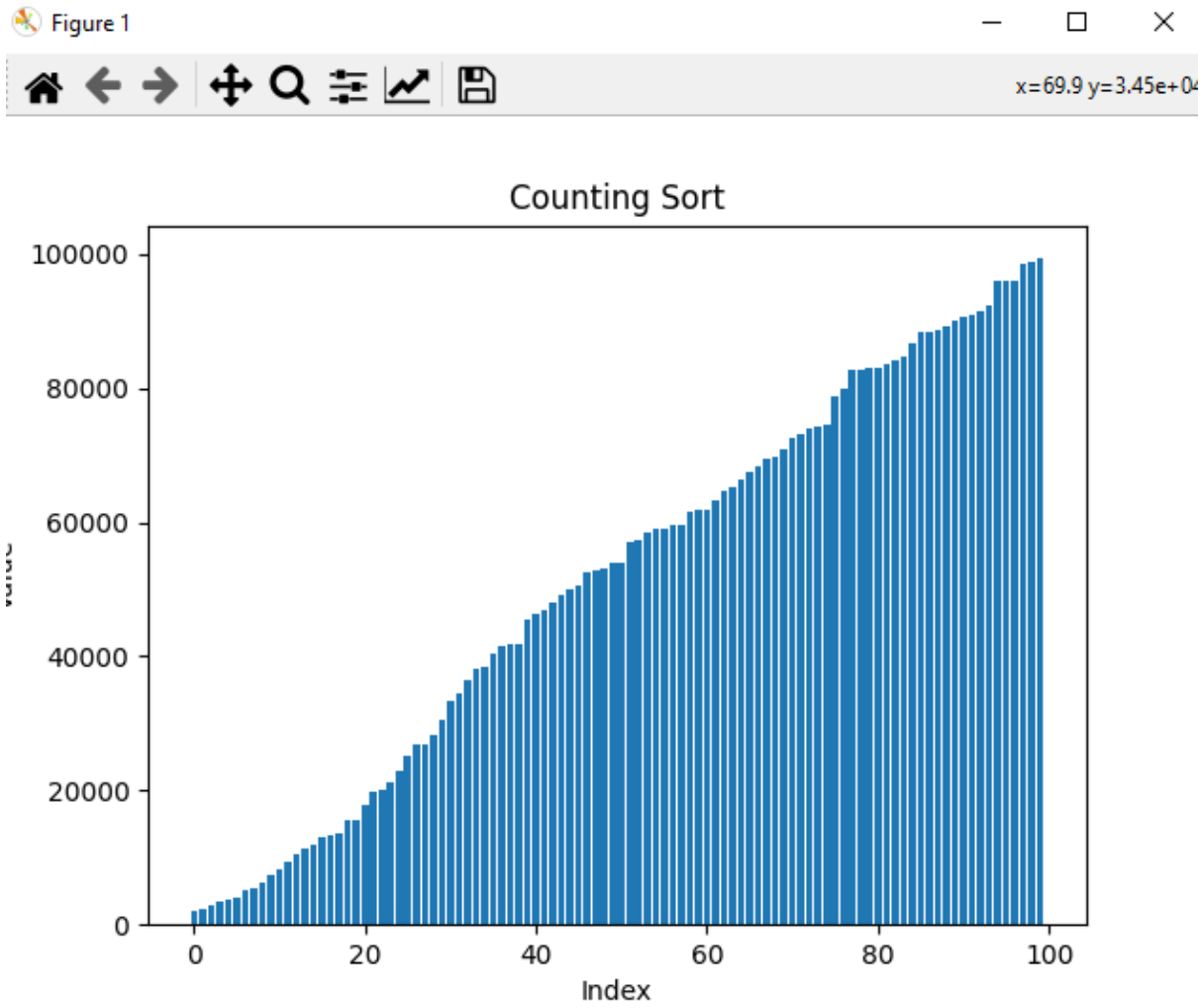


Рисунок 5.10 – Результат роботи програми

6. ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Робота з програмою

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

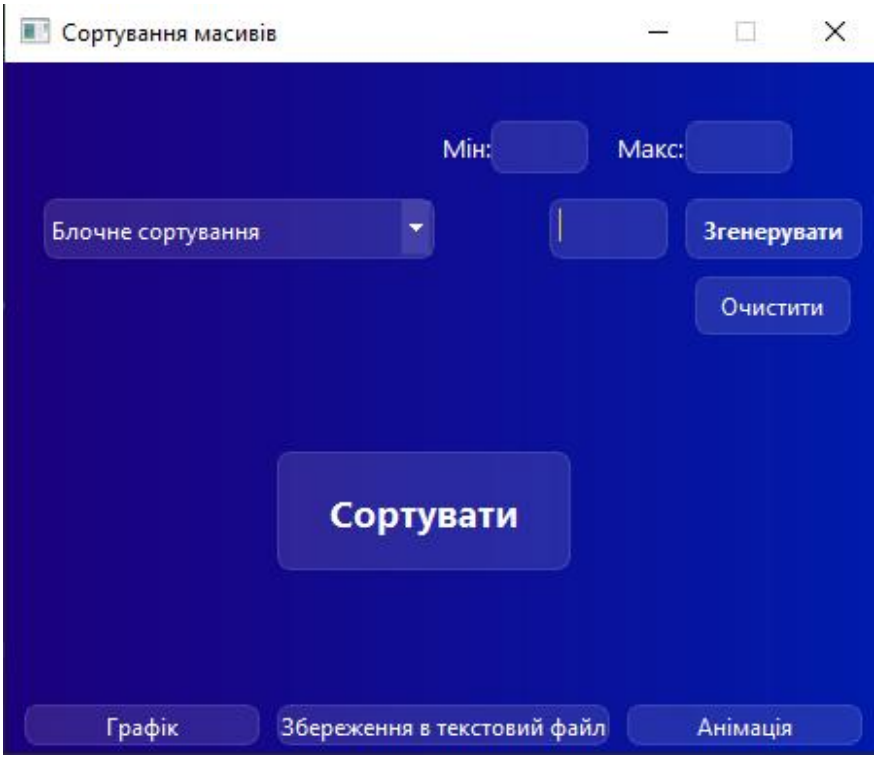


Рисунок 6.1 – Головне вікно програми

Вводимо з клавіатури у текстові поля дані про мінімальне значення діапазону, максимальне значення та розмір масиву(Рисунок 6.2)

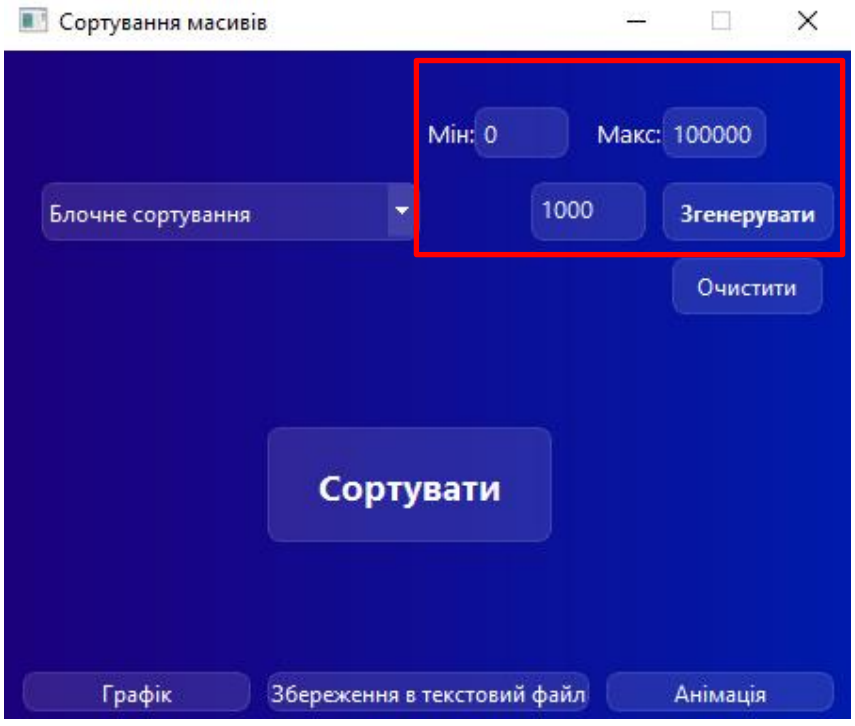


Рисунок 6.2 – Введення значень

Після введення даних у текстові поля, треба натиснути кнопку «Згенерувати» для генерації масиву заданого розміру у заданому діапазоні(Рисунок 6.3)

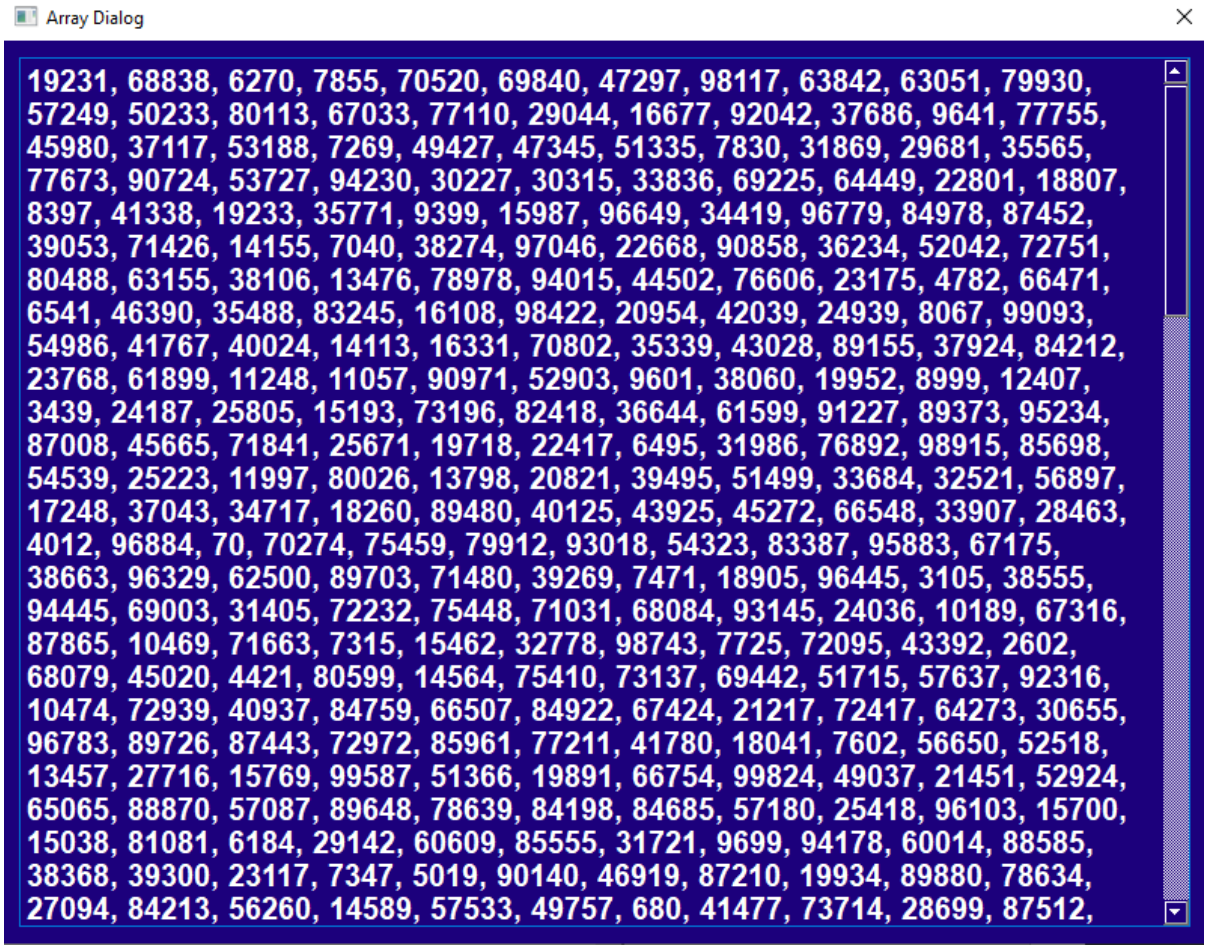


Рисунок 6.3 – Вікно зі згенерованим масивом

Після генерації масиву, треба обрати алгоритм сортування з комбо-бокса. За замовчуванням стоїть алгоритм блочного сортування, проте можна обрати будь-який інший з трьох запропонованих(Рисунок 6.4)

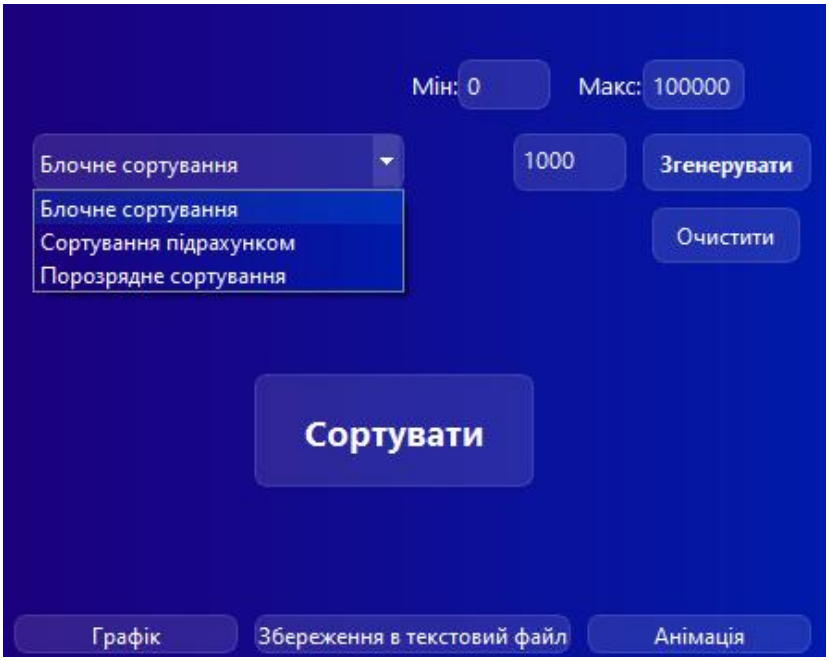


Рисунок 6.4 – Вибір алгоритму сортування

Після того як ви обрали алгоритм сортування потрібно натиснути кнопку «Сортувати», щоб відсортувати масив заданим алгоритмом.

Сортування алгоритмом блочного сортування(Рисунок 6.5)

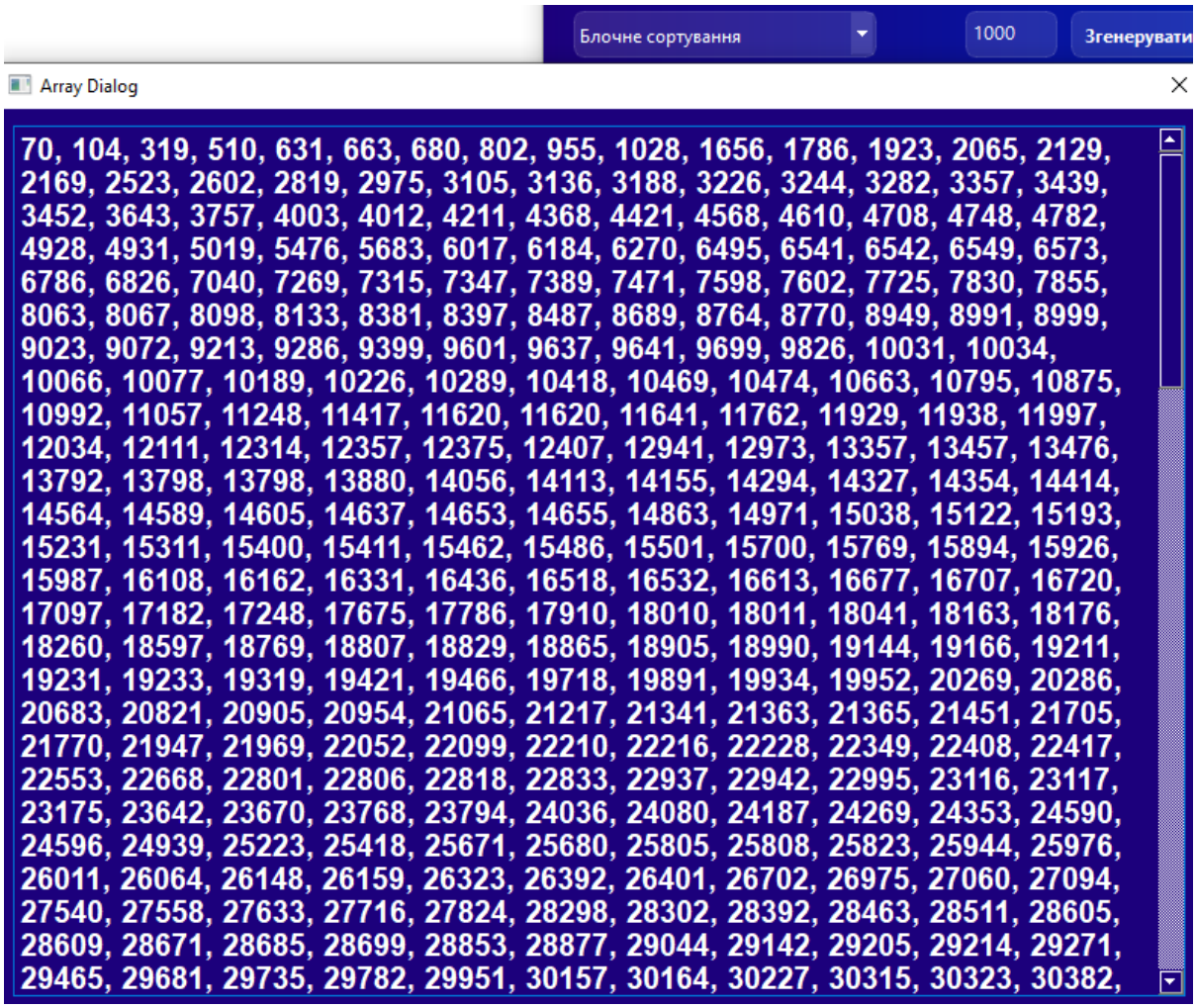


Рисунок 6.5 – Блочне сортування

Сортування алгоритмом сортування підрахунком(Рисунок 6.6)

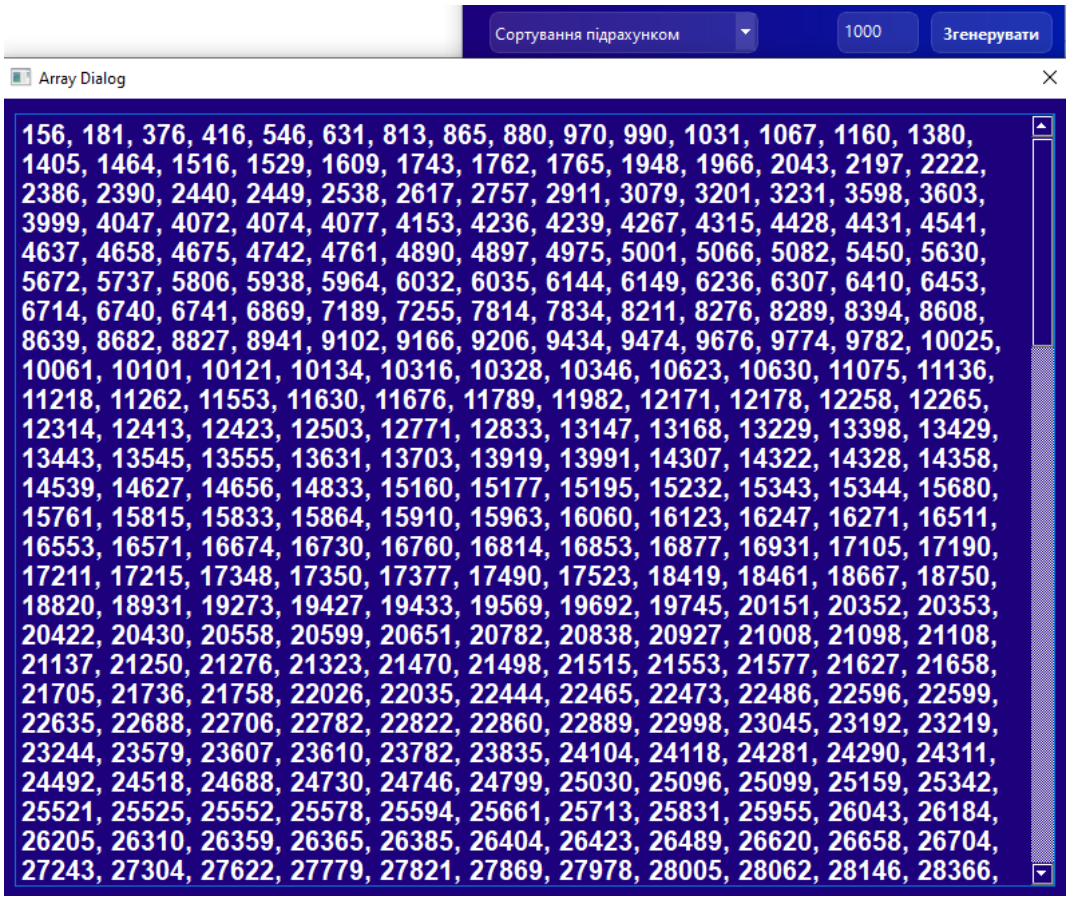


Рисунок 6.6 – Сортування підрахунком

Сортування алгоритмом порозрядного сортування(Рисунок 6.7)

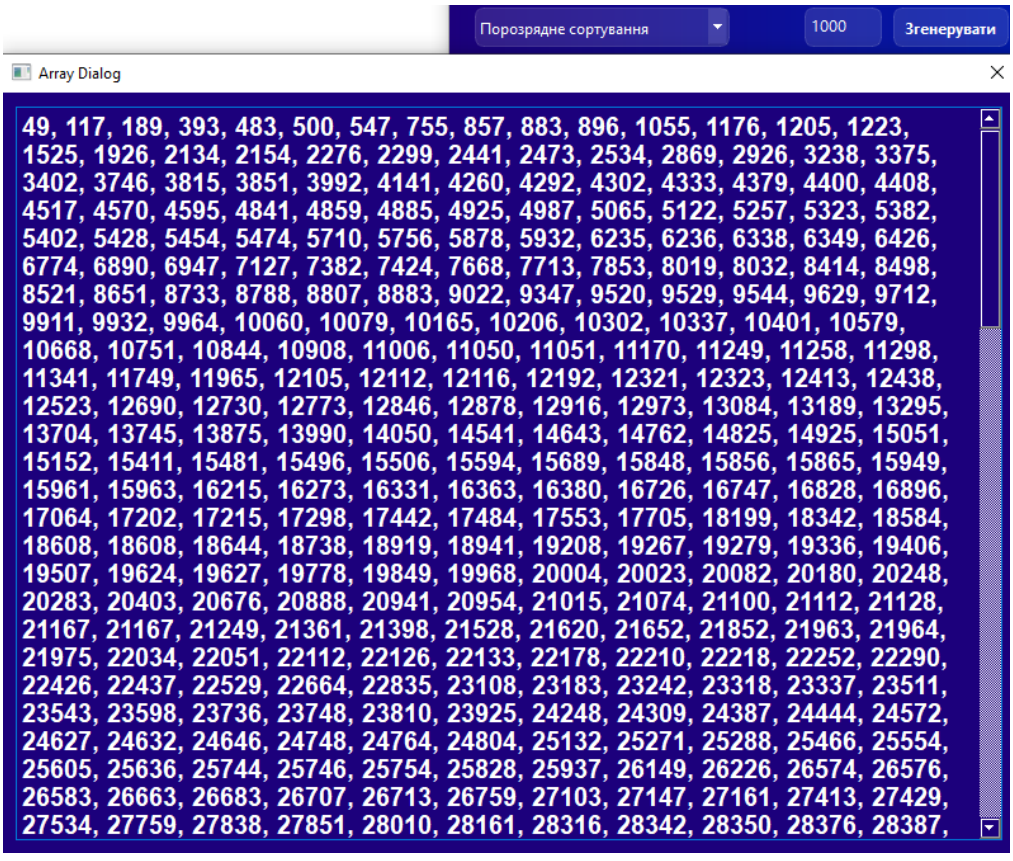


Рисунок 6.7 – Порозрядне сортування

Щоб відобразити характеристики алгоритмів, які відображають їх практичну складність треба натиснути на кнопку «Графік». На рисунку ви побачите кількість елементарних операцій кожного алгоритму(Рисунок 6.8)

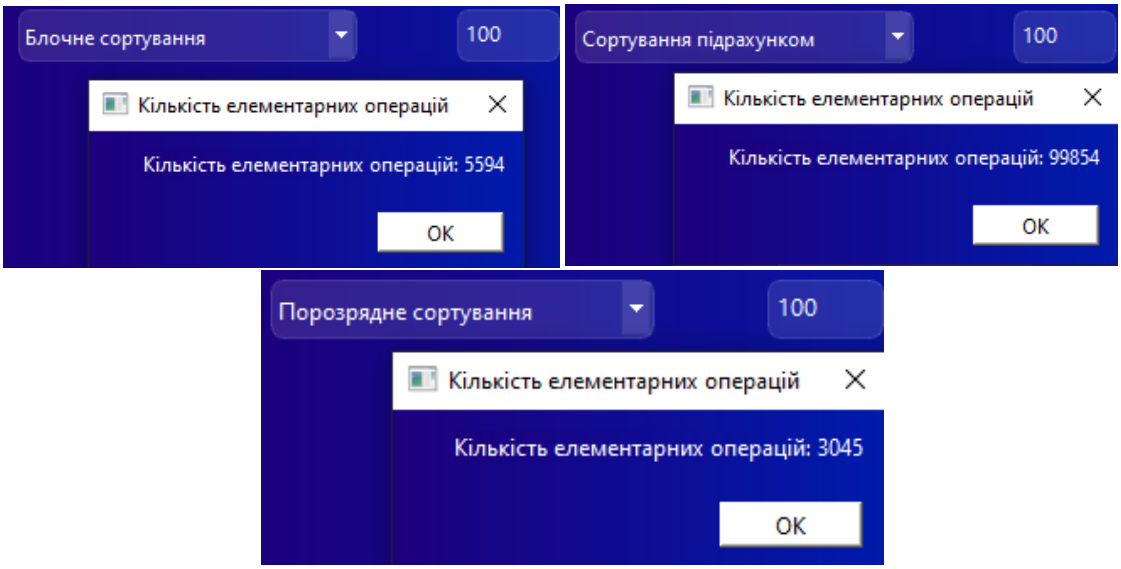


Рисунок 6.8 – Кількість елементарних операцій кожного алгоритма

За допомогою кнопки «Збереження в текстовий файл» можна зберегти невідсортований та відсортований масиви у текстовий файл(Рисунок 6.9)

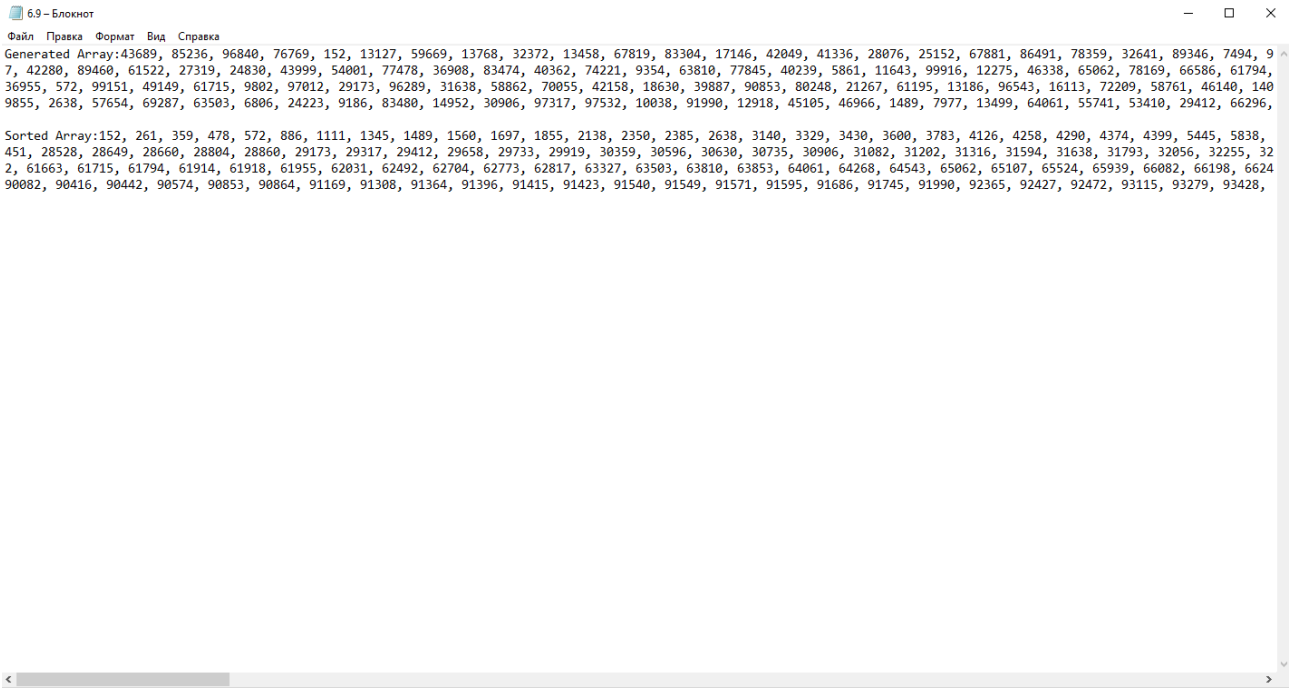


Рисунок 6.9 – Файл з масивами

Для того, щоб подивитись анімацію потрібно натиснути на кнопку «Анімація». Після цього відобразиться гістограма та буде відображатися анімація. Після закінчення анімації отримуємо гістограму відсортованого масиву(Рисунок 6.10)

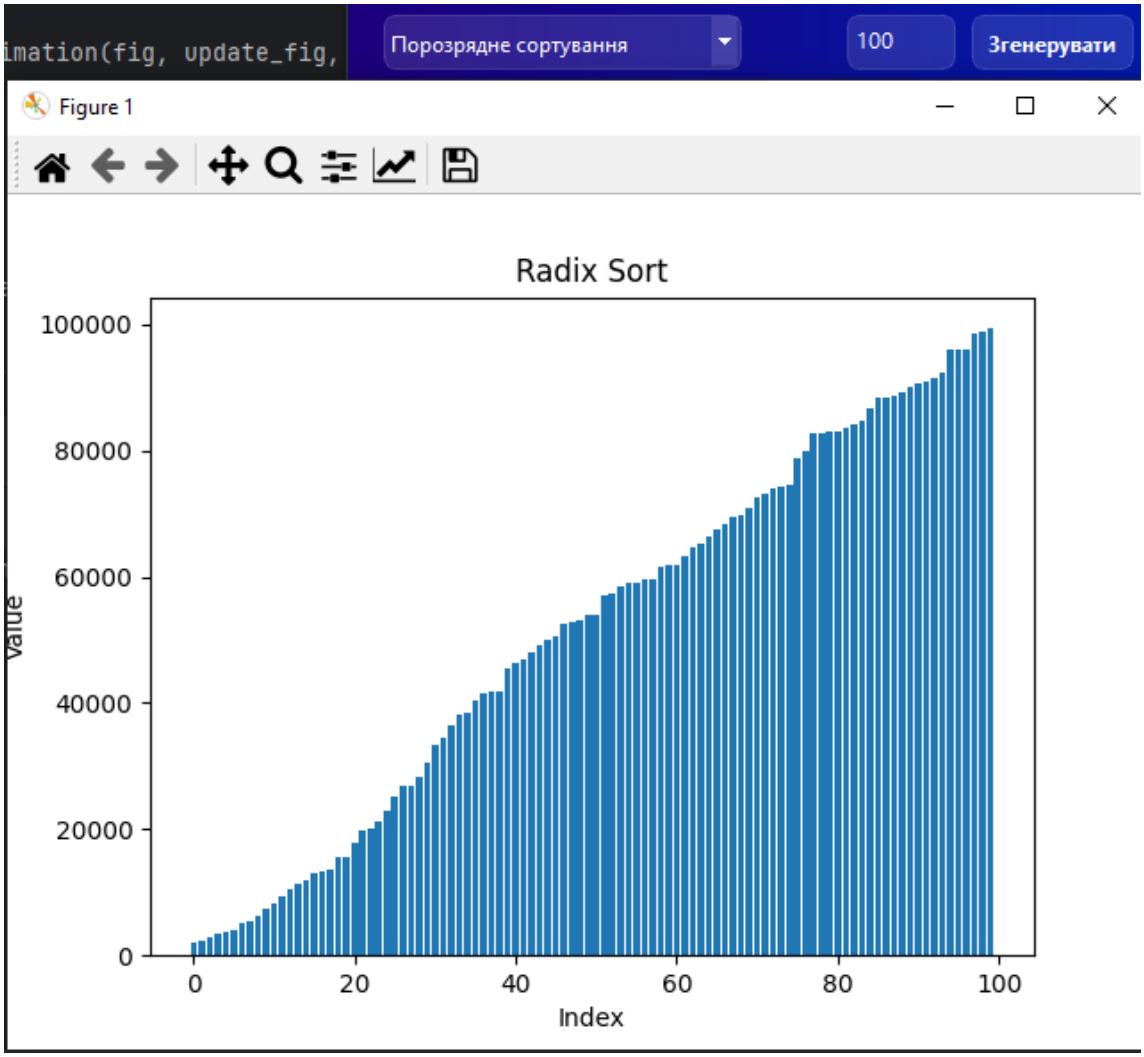


Рисунок 6.10 – Анімація сортування

6.2 Формат вхідних та вихідних даних

На вхід програми подається розмірність масиву, діапазон значень для генерації масиву, алгоритм сортування.

Результатом виконання програми є розв’язок задачі 8-puzzle, а також всі кроки до вирішення і час роботи програми, які видаються у текстовому полі.

6.3 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 7/ Windows 8/Windows 10/ Windows11 (з останніми обновленнями)	Windows 10/ Windows11 (з останніми обновленнями)
Процесор	Intel® Core® i5-3470K 3.20 GHz або AMD FX-8300 3.3 GHz	Intel® Core® i5-10400F 2.90 GHz або AMD Ryzen 5 1600 3.2 GHz
Оперативна пам'ять	1 GB	8 GB
Відеоадаптер	NVIDIA GeForce 940MX (або сумісний аналог)	
Дисплей	1600x1024	1920x1080
Прилади введення	Клавіатура, комп’ютерна миша	
Додаткове програмне забезпечення	Python 3.7 PySide6	

7. АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програмного забезпечення для упорядкування масиву заданими алгоритмами: блочне сортування, сортування підрахунком та порозрядне сортування.

Ми бачили, що масиви сортувалися у правильній послідовності, тому додаткових порівнянь проводити не треба.

Під час тестування було виявлено, що програма працює коректно, коли користувачем подаються коректні дані. Всі дані, які вводяться користувачем перевіряються на коректність і лише потім обробляються програмою.

Для проведення тестування ефективності програми згенеревано декілька масивів у діапазоні [-10000; 10000]. Результат тестування ефективності алгоритмів у таблиці 7.1:

Таблиця 7.1 – Тестування ефективності алгоритмів

Розмір масиву	Параметри тестування	Алгоритм		
		Блочне сортування	Сортування підрахунком	Порозрядне сортування
100	Кількість елементарних операцій	4836	19985	2436
1000	Кількість елементарних операцій	489740	23965	24036
10000	Кількість елементарних операцій	50014450	59999	240036
50000	Кількість елементарних операцій	1254026236	220000	1500045

На основі проведеного тестування можна зробити висновок, що порозрядне сортування є найефективнішим алгоритмом серед розглянутих в контексті кількості елементарних операцій. Однак, при виборі алгоритму сортування для конкретної задачі слід ретельно розглянути інші фактори та вимоги, такі як стабільність сортування, обробка великих обсягів даних, пам'яті, особливості вхідних даних та інші.

ВИСНОВОК

У ході виконання курсової роботи були досліджені та реалізовані алгоритми блочного сортування, сортування підрахунком та порозрядного сортування для упорядкування масивів.

Дані методи були описані у теоретичній частині документації і у псевдокоді програми. Після розробки алгоритмів було описано програмне забезпечення для розв’язання поставленої задачі.

Також було написано детальну інструкцію з використання програми користувачем і також проведено аналіз описаних алгоритмів. Результати аналізу показали наступні висновки: блочне сортування виявилося менш ефективним порівняно з іншими алгоритмами у розглянутому тестуванні. Воно вимагало найбільшої кількості елементарних операцій для сортування масивів різних розмірів. Сортування підрахунком показало гарні результати ефективності. Воно вимагало меншої кількості елементарних операцій порівняно з блочним сортуванням. Порозрядне сортування виявилося найефективнішим серед розглянутих алгоритмів. Воно дозволяє швидко упорядкувати масив, розглядаючи його по розрядам. Цей алгоритм показує хорошу продуктивність, особливо при великих розмірах масиву.

Результатом курсової роботи є програмне забезпечення, яке виконує поставлені задачі, воно пройшло повний цикл розробки. Отже, розробку можна вважати успішною та завершеною.

ПЕРЕЛІК ПОСИЛАНЬ

1. Алгоритм блочного сортування. URL: https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BA%D0%BE%D0%BC%D1%96%D1%80%D0%BA%D0%B0%D0%BC%D0%B8
2. Алгоритм сортування підрахунком. URL: https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%96%D0%B4%D1%80%D0%B0%D1%85%D1%83%D0%BD%D0%BA%D0%BE%D0%BC
3. Алгоритм порозрядного сортування URL: https://uk.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0%B0_%D1%80%D0%BE%D0%B7%D1%80%D1%8F%D0%B4%D0%B0%D0%BC%D0%B8

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив

Керівник Головченко Максим Миколайович

«___»_____201_ р.

Виконавець:

Студент Зубарев Микола Костянтинович

«___» _____201_ р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Упорядкування масивів»

з дисципліни:

«Основи програмування»

Київ 2023

1. *Мета*: Метою курсової роботи є розробка ефективного програмного забезпечення для сортування масивів різними методами(блочне сортування, сортування підрахунком, порозрядне сортування)
2. *Дата початку роботи*: «7» березня 2023 р.
3. *Дата закінчення роботи*: «___»_____ 202_ р.
4. *Вимоги до програмного забезпечення*.
   1) Функціональні вимоги:
      - Можливість задавати розмір масиву
      - Можливість генерувати масив від 100 елементів до 50000 у заданому діапазоні значень
      - Можливість перевірки коректності введених даних
      - Можливість обирати метод сортування (блочне сортування, сортування підрахунком, порозрядне сортування)
      - Можливість сортування масиву обраним методом
      - Можливість відображення результатів сортування
      - Можливість відображення процесу сортування у вигляді анімацій(до 300 елементів)
      - Можливість збереження результатів сортування у текстовий файл
      - Можливість відображення характеристик алгоритмів, які відображають їх практичну складність
   2) Нефункціональні вимоги:
      - Можливість запускати програмне забезпечення на операційній системі Windows 10 та вище.
      - Виконання курсової роботи виконується на мові програмування Python
      - 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.

5. *Стадії та етапи розробки*:

1) Об'єктно-орієнтований аналіз предметної області задачі (до__.__.202_ р.)
2) Об'єктно-орієнтоване проектування архітектури програмної системи (до __.__.202_р.)
3) Розробка програмного забезпечення (до __.__.202_р.)
4) Тестування розробленої програми (до __.__.202_р.)
5) Розробка пояснювальної записки (до __.__.202_ р.).
6) Захист курсової роботи (до __.__.202_ р.).

6. *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

Тексти програмного коду програмного забезпечення

вирішення задачі упорядкування масивів

(Найменування програми (документа))

Електронний

(Вид носія даних)

18,6 Мб

(Обсяг програми (документа), арк.,

студента групи ІП-23 І курсу

Зубарева М.К.

main.py

```
from PySide6.QtWidgets import QApplication
from code import MainWindow

if __name__ == "__main__":
    app = QApplication([])
    window = MainWindow()
    window.show()
    app.exec()
```

mas.py

```
from PySide6.QtCore import (QCoreApplication, QPropertyAnimation, QDate,
QDateTime, QLocale,
    QMetaObject, QObject, QPoint, QRect, QEasingCurve, QTime, QUrl, Qt)
from PySide6.QtGui import (QBrush, QColor, QShowEvent, QConicalGradient,
QCursor,
    QFont, QFontDatabase, QGradient, QIcon,
    QPalette, QPixmap, QRadialGradient, QTransform)
from PySide6.QtWidgets import (QApplication, QComboBox, QMainWindow,
QPushButton,QMessageBox,QFileDialog,QVBoxLayout, QDialog,
    QTextEdit, QWidget, QLabel)



class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        if not MainWindow.objectName():
            MainWindow.setObjectName(u"MainWindow")
        MainWindow.setFixedSize(450, 355)  # Set fixed size here
        #MainWindow.resize(450, 355)

        MainWindow.setStyleSheet(u"background-color: qlineargradient(spread:pad,
x1:0, y1:0, x2:1, y2:0, stop:0 rgba(28, 0, 124, 255), stop:1 rgba(0, 26, 171, 255));")
```

```
        self.centralwidget = QWidget(MainWindow)
        self.centralwidget.setObjectName(u"centralwidget")
        self.NumEdit = QTextEdit(self.centralwidget)
        self.NumEdit.setObjectName(u"NumEdit")
        self.NumEdit.setGeometry(QRect(280, 70, 61, 31))
        font = QFont()
        font.setPointSize(10)
        self.NumEdit.setFont(font)
        self.NumEdit.setStyleSheet(u"QTextEdit {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:30px;\n"
"height:20px; \n"
"}\n"
"")
        self.MinEdit = QTextEdit(self.centralwidget)
        self.MinEdit.setObjectName(u"NumEdit")
        self.MinEdit.setGeometry(QRect(250, 30, 50, 27))
        font = QFont()
        font.setPointSize(10)
        self.MinEdit.setFont(font)
        self.MinEdit.setStyleSheet(u"QTextEdit {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:30px;\n"
"height:20px; \n"
"}\n"
```

```
"")
        self.MaxEdit = QTextEdit(self.centralwidget)
        self.MaxEdit.setObjectName(u"NumEdit")
        self.MaxEdit.setGeometry(QRect(350, 30, 55, 27))
        font = QFont()
        font.setPointSize(10)
        self.MaxEdit.setFont(font)
        self.MaxEdit.setStyleSheet(u"QTextEdit {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:30px;\n"
"height:20px; \n"
"}\n"
"")
        self.MinLabel = QLabel(self.centralwidget)
        self.MinLabel.setObjectName(u"MinLabel")
        self.MinLabel.setGeometry(QRect(225, 30, 25, 27))
        font = QFont()
        font.setPointSize(10)
        self.MinLabel.setFont(font)
        self.MinLabel.setStyleSheet(u"QLabel {\n"
"color: white;\n"
"background-color: rgba(255,255,255,0);\n"
"border:0px solid rgba(255,255,255,40);\n"
"width:30px;\n"
"height:20px; \n"
"}\n"
"")
```

```
        self.MaxLabel = QLabel(self.centralwidget)
        self.MaxLabel.setObjectName(u"MaxLabel")
        self.MaxLabel.setGeometry(QRect(315, 30, 35, 27))
        font = QFont()
        font.setPointSize(10)
        self.MaxLabel.setFont(font)
        self.MaxLabel.setStyleSheet(u"QLabel {\n"
"color: white;\n"
"background-color: rgba(255,255,255,0);\n"
"border:px solid rgba(255,255,255,40);\n"
"width:30px;\n"
"height:20px; \n"
"}\n"
"")


        self.ComboSort = QComboBox(self.centralwidget)
        self.ComboSort.addItem("")
        self.ComboSort.addItem("")
        self.ComboSort.addItem("")
        self.ComboSort.setObjectName(u"ComboSort")
        self.ComboSort.setGeometry(QRect(20, 70, 201, 31))
        self.ComboSort.setStyleSheet(u"QComboBox {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:30px;\n"
"height:20px; \n"
"}\n"
"QComboBox QAbstractItemView{\n"
"color: white\n"
```

```
"}\n"
"")
        self.Generate = QPushButton(self.centralwidget)
        self.Generate.setObjectName(u"Generate")
        self.Generate.setGeometry(QRect(350, 70, 91, 31))
        font1 = QFont()
        font1.setBold(True)
        self.Generate.setFont(font1)
        self.Generate.setStyleSheet(u"QPushButton {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:30px;\n"
"height:20px; \n"
"}\n"
"QPushButton:hover{\n"
"background-color: rgba(255,255,255,40)\n"
"}\n"
"QPushButton:pressed{\n"
"background-color: rgba(255,255,255,70)\n"
"}\n"
"")
        #self.Generate.clicked.connect(self.generate_numbers)

        self.Sort = QPushButton(self.centralwidget)
        self.Sort.setObjectName(u"Sort")
        self.Sort.setGeometry(QRect(140, 200, 151, 61))
        font2 = QFont()
        font2.setPointSize(14)
        font2.setBold(True)
```

```
        self.Sort.setFont(font2)
        self.Sort.setStyleSheet(u"QPushButton {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:30px;\n"
"height:20px; \n"
"}\n"
"QPushButton:hover{\n"
"background-color: rgba(255,255,255,40)\n"
"}\n"
"QPushButton:pressed{\n"
"background-color: rgba(255,255,255,70)\n"
"}\n"
"")
        #self.Sort.clicked.connect(self.show_message)  # Connect the button to the function


        self.Animation = QPushButton(self.centralwidget)
        self.Animation.setObjectName(u"Animation")
        self.Animation.setGeometry(QRect(320, 330, 121, 21))
        self.Animation.setStyleSheet(u"QPushButton {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:60px;\n"
"height:20px; \n"
"}\n"
```

```
"QPushButton:hover{\n"
"background-color: rgba(255,255,255,40)\n"
"}\n"
"QPushButton:pressed{\n"
"background-color: rgba(255,255,255,70)\n"
"}\n"
"")
        self.Graph = QPushButton(self.centralwidget)
        self.Graph.setObjectName(u"Graph")
        self.Graph.setGeometry(QRect(10, 330, 121, 21))
        self.Graph.setStyleSheet(u"QPushButton {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:10px;\n"
"height:20px; \n"
"}\n"
"QPushButton:hover{\n"
"background-color: rgba(255,255,255,40)\n"
"}\n"
"QPushButton:pressed{\n"
"background-color: rgba(255,255,255,70)\n"
"}\n"
"")
        self.Save = QPushButton(self.centralwidget)
        self.Save.setObjectName(u"Save")
        self.Save.setGeometry(QRect(140, 330, 171, 21))
        self.Save.setStyleSheet(u"QPushButton {\n"
"color: white;\n"
"background-color: rgba(255,255,255,30);\n"
```

```
"border:1px solid rgba(255,255,255,40);\n"
"border-radius: 7px;\n"
"width:50px;\n"
"height:20px; \n"
"}\n"
"QPushButton:hover{\n"
"background-color: rgba(255,255,255,40)\n"
"}\n"
"QPushButton:pressed{\n"
"background-color: rgba(255,255,255,70)\n"
"}\n"
"")
        self.Clear = QPushButton(self.centralwidget)
        self.Clear.setObjectName(u"Clear")
        self.Clear.setGeometry(QRect(355, 110, 80, 30))
        self.Clear.setStyleSheet(u"QPushButton {\n"
 "color: white;\n"
 "background-color: rgba(255,255,255,30);\n"
 "border:1px solid rgba(255,255,255,40);\n"
 "border-radius: 7px;\n"
 "width:10px;\n"
 "height:20px; \n"
 "}\n"
 "QPushButton:hover{\n"
 "background-color: rgba(255,255,255,40)\n"
 "}\n"
 "QPushButton:pressed{\n"
 "background-color: rgba(255,255,255,70)\n"
 "}\n"
 "")
        MainWindow.setCentralWidget(self.centralwidget)
```

```
        self.retranslateUi(MainWindow)

        QMetaObject.connectSlotsByName(MainWindow)
    # setupUi

    def retranslateUi(self, MainWindow):
        MainWindow.setWindowTitle(QCoreApplication.translate("MainWindow",
u"\u0421\u043e\u0440\u0442\u0443\u0432\u0430\u043d\u043d\u044f
\u043c\u0430\u0441\u0438\u0432\u0456\u0432", None))
        self.ComboSort.setItemText(0, QCoreApplication.translate("MainWindow",
u"\u0411\u043b\u043e\u0447\u043d\u0435
\u0441\u043e\u0440\u0442\u0443\u0432\u0430\u043d\u043d\u044f", None))
        self.ComboSort.setItemText(1, QCoreApplication.translate("MainWindow",
u"\u0421\u043e\u0440\u0442\u0443\u0432\u0430\u043d\u043d\u044f
\u043f\u0456\u0434\u0440\u0430\u0445\u0443\u043d\u043a\u043e\u043c", None))
        self.ComboSort.setItemText(2, QCoreApplication.translate("MainWindow",
u"\u041f\u043e\u0440\u043e\u0437\u0440\u044f\u0434\u043d\u0435
\u0441\u043e\u0440\u0442\u0443\u0432\u0430\u043d\u043d\u044f", None))
        self.MinLabel.setText(QCoreApplication.translate("MainWindow", u"Мін:",
None))
        self.MaxLabel.setText(QCoreApplication.translate("MainWindow", u"Макс:",
None))

        self.Generate.setText(QCoreApplication.translate("MainWindow",
u"\u0417\u0433\u0435\u043d\u0435\u0440\u0443\u0432\u0430\u0442\u0438",
None))
        self.Sort.setText(QCoreApplication.translate("MainWindow",
u"\u0421\u043e\u0440\u0442\u0443\u0432\u0430\u0442\u0438", None))
        self.Animation.setText(QCoreApplication.translate("MainWindow",
u"\u0410\u043d\u0456\u043c\u0430\u0446\u0456\u044f", None))
```

```
        self.Graph.setText(QCoreApplication.translate("MainWindow",
u"\u0413\u0440\u0430\u0444\u0456\u043a", None))
        self.Save.setText(QCoreApplication.translate("MainWindow",
u"\u0417\u0431\u0435\u0440\u0435\u0436\u0435\u043d\u043d\u044f \u0432
\u0442\u0435\u043a\u0441\u0442\u043e\u0432\u0438\u0439
\u0444\u0430\u0439\u043b", None))
        self.Clear.setText(QCoreApplication.translate("MainWindow",
u"\u041e\u0447\u0438\u0441\u0442\u0438\u0442\u0438", None))
    # retranslateUi
```

code.py

```
        from PySide6.QtCore import (QCoreApplication, QPropertyAnimation,
QDate, QDateTime, QLocale,
    QMetaObject, QObject, QPoint, QRect, QEasingCurve,
    QSize, QTime, QUrl, Qt)
from PySide6.QtGui import (QBrush, QColor, QShowEvent, QConicalGradient,
QCursor,
    QFont, QFontDatabase, QGradient, QIcon,
    QImage, QKeySequence, QLinearGradient, QPainter,
    QPalette, QPixmap, QRadialGradient, QTransform)
from PySide6.QtWidgets import (QApplication, QComboBox, QMainWindow,
QPushButton,QMessageBox,QFileDialog,QVBoxLayout, QDialog,
    QTextEdit, QWidget)
from PySide6 import QtCharts
import time
from anim import AnimationDialog, SortAnim, CountAnim
from graphic import BucketTime, CountingTime, RadixTime
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy as np
from random import randint
from mas import Ui_MainWindow
```

```
from dialog import ArrayDialog

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.Generate.clicked.connect(self.generate_array)
        self.ui.Sort.clicked.connect(self.sort_array)
        self.ui.Clear.clicked.connect(self.clear_arrays)
        self.ui.Save.clicked.connect(self.save_array)
        self.ui.Graph.clicked.connect(self.draw_graph)
        #self.ui.Animation.clicked.connect(self.draw_animation)
        self.ui.Animation.clicked.connect(self.draw_animation)



        self.generated_array = []
        self.sorted_array = []
        self.file_name = ""


    def update_color(self, name, title, text):
        name = QMessageBox(self)
        #name.setFixedSize(500, 500)
        name.setWindowTitle(f"{title}")
        name.setText(f"{text}")
        name.setTextInteractionFlags(Qt.TextSelectableByMouse)
        name.setStyleSheet("QLabel {color: white;}""QMessageBox QPushButton
{background-color: white;}")
        name.exec_()
```

```
    def generate_array(self):
        if not self.generated_array:
            min_text = self.ui.MinEdit.toPlainText()
            max_text = self.ui.MaxEdit.toPlainText()

            if min_text == '' or max_text == '':
                war_box = QMessageBox(self)
                self.update_color(war_box, "Invalid Input", "Please enter values for minimum and maximum values.")
                return

            min_value = int(min_text)
            max_value = int(max_text)

            if min_value >= max_value:
                war_box = QMessageBox(self)
                self.update_color(war_box, "Invalid Input", "Minimal value should be less than maximum.")
                self.ui.MinEdit.clear()
                self.ui.MaxEdit.clear()
                return
            # if min_value < 0 or max_value < 0:
            #     self.ui.MinEdit.clear()
            #     self.ui.MaxEdit.clear()
            #     war_box = QMessageBox(self)
            #     self.update_color(war_box, "Invalid Input", "Please enter positive values.")
            #     return

            text = self.ui.NumEdit.toPlainText()
```

```
            try:
                num_chars = int(text)
                if num_chars < 100 or num_chars > 50000:
                    war_box = QMessageBox(self)
                    self.update_color(war_box,"Invalid Input", "Please enter number more than 100 and less than 50000.")
                    self.ui.NumEdit.clear()
                    #QMessageBox.warning(self, "Invalid Input", "Please enter a valid number of characters.")
                else:
                    self.generated_array = [randint(min_value, max_value) for _ in range(num_chars)]
                    self.show_array(self.generated_array)
            except ValueError:
                war_box = QMessageBox(self)
                self.update_color(war_box,"Invalid Input", "Please enter a valid number of characters.")
                #QMessageBox.warning(self, "Invalid Input", "Please enter a valid number of characters.")
        else:
            gen_box = QMessageBox
            self.update_color(gen_box, "Array Generated", f"Your array has already been generated")
            self.show_array(self.generated_array)

    def counting_animation(self):
        window = AnimationDialog()
        window.counting_sort_animation(self.generated_array)

    def bucket_animation(self):
        app = SortAnim()
```

```
        app.block_sort_animation(self.generated_array)
        #
        # window = CustomDialog()
        # window.bucket_sort_animation(self.generated_array)

    def clear_arrays(self):
        # Очистка массивов
        self.generated_array = []
        self.sorted_array = []
        self.clear_file()
        msg = QMessageBox(self)
        self.update_color(msg, "Очищення масива", "Масиви очищено")

    def show_array(self, array):
        dialog = ArrayDialog(array)
        dialog.exec()

    def sort_array(self):
        if not self.generated_array:
            war_box = QMessageBox(self)
            self.update_color(war_box, "Array Not Generated", "Please generate an array
first.")
            #QMessageBox.warning(self, "Array Not Generated", "Please generate an
array first.")
            return

        if not self.sorted_array:
            # Получение выбранного метода сортировки из QComboBox
            selected_sorting_method = self.ui.ComboSort.currentText()

            # Применение выбранного метода сортировки
```

```
            if selected_sorting_method == "Блочне сортування":
                self.sorted_array = self.bucket_sort(self.generated_array)
            elif selected_sorting_method == "Сортування підрахунком":
                self.sorted_array = self.counting_sort(self.generated_array)
            elif selected_sorting_method == "Порозрядне сортування":
                self.sorted_array = self.radix_sort(self.generated_array)
            else:
                self.sorted_array = self.generated_array
        self.show_array(self.sorted_array)

    def draw_animation(self):
        if not self.generated_array:
            war_box = QMessageBox(self)
            self.update_color(war_box, "Array Not Generated", "Please generate an array
first.")
            return
        #window = CustomDialog()
        select_method = self.ui.ComboSort.currentText()
        if select_method == "Блочне сортування":
            self.bucket_animation()
            #window.bucket_sort_animation(self.generated_array)
        elif select_method == "Сортування підрахунком":
            anim = CountAnim()
            anim.counting_sort_animation(self.generated_array)
            #print("Counting Sort")
        elif select_method == "Порозрядне сортування":
            print("Radix Sort")

    def clear_file(self):
        if self.file_name:
            try:
```

```
            open(self.file_name, 'w').close()
            msg = QMessageBox(self)
            self.update_color(msg, "Очищення файлу", "Файл успішно очищено:
{}".format(self.file_name))
        except IOError:
            err = QMessageBox()
            self.update_color(err, "Помилка", "Помилка при очищенні файлу.")
    def save_array(self):
        self.file_name, _ = QFileDialog.getSaveFileName(self, "Save Array", "", "Text
Files (*.txt)")
        if self.file_name:
            try:
                with open(self.file_name, 'w') as file:
                    file.write('Generated Array:')
                    for i, element in enumerate(self.generated_array):
                        if i == len(self.generated_array) - 1:
                            file.write(str(element))
                        else:
                            file.write(str(element) + ', ')
                    file.write('\n\n')
                    file.write("Sorted Array:")
                    for i, element in enumerate(self.sorted_array):
                        if i == len(self.sorted_array) - 1:
                            file.write(str(element))
                        else:
                            file.write(str(element) + ', ')
                msg = QMessageBox(self)
                self.update_color(msg, "Зберігання масиву", "Масив збережено в файлі:
{}".format(self.file_name))
            except IOError:
                err = QMessageBox()
```

```
            self.update_color(err, "Помилка", "Помилка при збереженні масиву в файлі.")

    def draw_graph(self):
        if not self.generated_array:
            war_box = QMessageBox(self)
            self.update_color(war_box, "Array Not Generated", "Please generate an array first.")
            #QMessageBox.warning(self, "Array Not Generated", "Please generate an array first.")
            return

        # Получение выбранного метода сортировки из QComboBox
        selected_sorting_method = self.ui.ComboSort.currentText()

        # Применение выбранного метода сортировки
        if selected_sorting_method == "Блочне сортування":
            graph = BucketTime()
            arr, operations = graph.bucket_sort(self.generated_array)
            msg = QMessageBox(self)
            self.update_color(msg, "Кількість елементарних операцій", f"Кількість елементарних операцій: {operations}")
            #graph.plot_complexity(self.generated_array)
        elif selected_sorting_method == "Сортування підрахунком":
            graph = CountingTime()
            arr, operations = graph.counting_sort(self.generated_array)
            msg = QMessageBox(self)
            self.update_color(msg, "Кількість елементарних операцій", f"Кількість елементарних операцій: {operations}")
            #graph.plot_complexity(self.generated_array)
        elif selected_sorting_method == "Порозрядне сортування":
```

```
            graph = RadixTime()
            arr, operations = graph.radix_sort(self.generated_array)
            msg = QMessageBox(self)
            self.update_color(msg, "Кількість елементарних операцій", f"Кількість елементарних операцій: {operations}")
            #graph.plot_complexity(self.generated_array)

    # def radix_graph(self):
    #     graph = RadixTime()
    #     graph.plot_complexity(self.generated_array)
    #
    # def bucket_graph(self):
    #     graph = BucketTime()
    #     graph.plot_complexity(self.generated_array)
    #
    # def counting_graph(self):
    #     graph = CountingTime()
    #     sizes = range(1, len(self.generated_array) + 1)
    #     graph.plot_complexity(sizes)

    def bucket_sort(self, array):
        # Знайдемо максимальне та мінімальне значення у масиві
        max_val = max(array)
        min_val = min(array)
        range_val = max_val - min_val

        # Визначимо кількість блоків та розмір кожного блоку
        size = len(array)
        bucket_size = range_val / size

        # Створимо порожні блоки
```

```
    buckets = [[] for _ in range(size)]

    # Розподілим елементи у масиві по блоках
    for num in array:
        if min_val < 0:
            index_b = int((num - min_val) // range_val * (size - 1))
        else:
            index_b = int((num - min_val) // range_val * size)
        if index_b == size:
            index_b -= 1
        buckets[index_b].append(num)

    # Зіберемо відсортовані елементи з блоків
    sorted_array = []
    for bucket in buckets:
        # Використовуємо сортування вставкою у кожному блоку
        for i in range(1, len(bucket)):
            key = bucket[i]
            j = i - 1
            while j >= 0 and bucket[j] > key:
                bucket[j + 1] = bucket[j]
                j -= 1
            bucket[j + 1] = key

        #sorted_array.extend(bucket)
        sorted_array += bucket

    return sorted_array

def counting_sort(self, array):
    size = len(array)
```

```
    output = [0] * size

    # Find the maximum value in the array
    max_value = max(array)
    min_value = min(array)
    range_array = max_value - min_value

    # Initialize count array
    count = [0] * (range_array + 1)

    # Считаем количество каждого элемента в массиве
    for i in range(size):
        count[array[i] - min_value] += 1

    # Вычисляем накопленное количество
    for i in range(1, range_array + 1):
        count[i] += count[i - 1]

    # Восстанавливаем отсортированный массив
    i = size - 1
    while i >= 0:
        output[count[array[i] - min_value] - 1] = array[i]
        count[array[i] - min_value] -= 1
        i -= 1

    sorted_array = output

    return sorted_array

def countingSort(self, array, place):
    size = len(array)
```

```
output = [0] * size

min_value = min(array)
adjusted_array = [num - min_value for num in array]

max_value = max(adjusted_array)

count = [0] * (max_value + 1)

# Calculate count of elements
for i in range(0, size):
    index = adjusted_array[i] // place
    count[index % 10] += 1

# Calculate cumulative count
for i in range(1, 10):
    count[i] += count[i - 1]

# Place the elements in sorted order
i = size - 1
while i >= 0:
    index = adjusted_array[i] // place
    output[count[index % 10] - 1] = adjusted_array[i]
    count[index % 10] -= 1
    i -= 1

# Adjust the sorted elements back to their original values
sorted_array = [num + min_value for num in output]

return sorted_array
```

```
    def radix_sort(self, array):
        # Get maximum and minimum elements
        max_element = max(array)
        min_element = min(array)

        # Create a new array to store the sorted elements
        sorted_array = array

        # Apply counting sort to sort elements based on place value.
        place = 1
        while (max_element - min_element) // place > 0:
            sorted_array = self.countingSort(sorted_array, place)
            place *= 10

        return sorted_array
```

graphic.py

```
import matplotlib.pyplot as plt
import numpy as np
import random
import time

class RadixTime():
    def countingSort(self, array, place):
        size = len(array)
        output = [0] * size
        operations = 0
        max_value = max(array)

        count = [0] * (max_value + 1)

        # Calculate count of elements
```

```
    for i in range(0, size):
        index = array[i] // place
        count[index % 10] += 1
        # Увеличиваем счетчик операций
        operations += 1

    # Calculate cumulative count
    for i in range(1, 10):
        count[i] += count[i - 1]
        # Увеличиваем счетчик операций
        operations += 1

    # Place the elements in sorted order
    i = size - 1
    while i >= 0:
        index = array[i] // place
        output[count[index % 10] - 1] = array[i]
        count[index % 10] -= 1
        i -= 1
        # Увеличиваем счетчик операций
        operations += 4

    # Create a new array to store the sorted elements
    sorted_array = output.copy()

    # Увеличиваем счетчик операций
    operations += size

    return sorted_array, operations

def radix_sort(self, array):
```

```
    # Get maximum element
    max_element = max(array)

    # Create a new array to store the sorted elements
    sorted_array = array.copy()

    # Initialize transition counter
    operations = 0

    # Apply counting sort to sort elements based on place value.
    place = 1
    while max_element // place > 0:
        sorted_array, count = self.countingSort(sorted_array, place)
        operations += count
        place *= 10

    return sorted_array, operations

def measure_time(self, array):
    start_time = time.time()
    sorted_arr = self.radix_sort(array)
    end_time = time.time()

    return end_time - start_time

def plot_complexity(self, array):
    sizes = range(1, len(array) + 1)
    avg_times = []

    # for size in sizes:
    #     times = []
```

```
        #     for _ in range(10):
        #         time_taken = self.measure_time(array)
        #         times.append(time_taken)
        #     avg_time = np.mean(times)
        #     avg_times.append(avg_time)

        #plt.plot(sizes, operat, 'b', label='Average Time')
        plt.plot(sizes, [size for size in sizes], 'r', linestyle='dashdot', label='Best Case')
        plt.plot(sizes, [size for size in sizes], 'g--', label='Worst Case')
        plt.plot(sizes, [size for size in sizes], 'y--', label='Average Case')

        plt.title('Algorithm Complexity: Radix Sort')
        plt.xlabel('Input Size')
        plt.ylabel('Average Time (seconds)')
        plt.legend(loc='upper left')
        plt.show()


class CountingTime():
    def __init__(self):
        super().__init__()
    def counting_sort(self, array):
        operations = 0
        size = len(array)
        output = [0] * size

        # Find the maximum value in the array
        max_value = max(array)
        min_value = min(array)
        range_array = max_value - min_value
```

```
    # Initialize count array
    count = [0] * (range_array + 1)

    # Считаем количество каждого элемента в массиве
    for i in range(size):
        count[array[i] - min_value] += 1
        operations += 1

    # Вычисляем накопленное количество
    for i in range(1, range_array + 1):
        count[i] += count[i - 1]
        operations += 1

    # Восстанавливаем отсортированный массив
    i = size - 1
    while i >= 0:
        output[count[array[i] - min_value] - 1] = array[i]
        count[array[i] - min_value] -= 1
        i -= 1
        operations += 3

    sorted_array = output

    return sorted_array, operations

def measure_time(self, array):
    start_time = time.time()
    sorted_arr = self.counting_sort(array)
    end_time = time.time()

    return end_time - start_time
```

```
    def plot_complexity(self, array):
        sizes = range(1, len(array) + 1)
        avg_times = []
        k = (max(array) - min(array)) // len(array)

        # for size in sizes:
        #     times = []
        #     for _ in range(10):
        #         time_taken = self.measure_time(array)
        #         times.append(time_taken)
        #     avg_time = np.mean(times)
        #     avg_times.append(avg_time)

        #plt.plot(sizes, avg_times, 'b', label='Average case')
        plt.plot(sizes, [size for size in sizes], 'r', linestyle='dashdot', label='Best case')
        plt.plot(sizes, [size for size in sizes], 'g', label='Worst case')
        plt.plot(sizes, [size for size in sizes], 'y--', label='Average case')

        plt.title('Algorithm Complexity: Counting Sort')
        plt.xlabel('Input Size')
        plt.ylabel('Average Time (seconds)')
        plt.legend(loc='upper left')
        plt.show()

class BucketTime():
    def __init__(self):
        super().__init__()
        # self.setWindowTitle("Sorting Visualization")
        # self.resize(800, 600)
```

```
    def measure_time(self, array):
        start_time = time.time()
        sorted_arr = self.bucket_sort(array)
        end_time = time.time()

        return end_time - start_time

    def plot_complexity(self, array):
        sizes = range(1, len(array) + 1)
        avg_times = []
        k = (max(array) - min(array)) // len(array)

        # for size in sizes:
        #     times = []
        #     for _ in range(len(array)):
        #         time_taken = self.measure_time(array)
        #         times.append(time_taken)
        #
        #     avg_time = np.mean(times)
        #     avg_times.append(avg_time)

        #plt.plot(sizes, avg_times, 'b', label='Random case')
        plt.plot(sizes, [size for size in sizes], 'g', label='Best case')
        plt.plot(sizes, [size ** 2 for size in sizes], 'r--', label='Worst case')
        plt.plot(sizes, [size + 10 for size in sizes], 'y', linestyle='dashdot', label='Average
case')

        plt.title('Algorithm Complexity: Bucket Sort')
        plt.xlabel('Input Size')
        plt.ylabel('Average Time (seconds)')
        plt.legend(loc='upper left')
```

```
    plt.show()
def bucket_sort(self, array):
    # Знайдемо максимальне та мінімальне значення у масиві
    max_val = max(array)
    min_val = min(array)
    range_val = max_val - min_val

    # Визначимо кількість блоків та розмір кожного блоку
    size = len(array)
    bucket_size = range_val / size

    # Створимо порожні блоки
    buckets = [[] for _ in range(size)]
    operations = 0
    # Розподілим елементи у масиві по блоках
    for num in array:
        operations += 1
        if min_val < 0:
            index_b = int((num - min_val) // range_val * (size - 1))
        else:
            index_b = int((num - min_val) // range_val * size)
        if index_b == size:
            index_b -= 1
        buckets[index_b].append(num)

    # Зіберемо відсортовані елементи з блоків
    sorted_array = []
    for bucket in buckets:
        # Використовуємо сортування вставкою у кожному блоку
        for i in range(1, len(bucket)):
            key = bucket[i]
```

```
            j = i - 1
            while j >= 0 and bucket[j] > key:
                bucket[j + 1] = bucket[j]
                j -= 1
                operations += 2
            bucket[j + 1] = key
            operations += 1

        sorted_array.extend(bucket)
        #sorted_array += bucket
    return sorted_array, operations
```

dialog.py

```
    class ArrayDialog(QDialog):
    def __init__(self, array):
        super().__init__()
        self.setWindowTitle("Array Dialog")
        self.setFixedSize(800, 600)
        self.setStyleSheet("background-color: #1c007c; color: white; font-size: 20px;
font-family: Arial; font-weight: bold;")
        self.setup_ui(array)

    def setup_ui(self, array):
        self.text_edit = QTextEdit()
        self.text_edit.setReadOnly(True)

        array_text = ', '.join(str(num) for num in array)
        self.text_edit.setPlainText(array_text)

        layout = QVBoxLayout()
        layout.addWidget(self.text_edit)
```

```
self.setLayout(layout)
```