

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 Основ програмування - 2
(назва дисципліни)
на тему: Упорядкування масивів

Студента 1-го курсу, групи ПІ-12
Васильєва Єгора Костянтиновича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник старший викладач Головченко Максим Миколайович
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Кількість балів: _____
Національна оцінка _____

Члени комісії

(підпис)

(вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

(вчене звання, науковий ступінь, прізвище та ініціали)

Київ- 2022 рік

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

(назва вищого навчального закладу)

Кафедра інформатики та програмної інженерії

Дисципліна Основи програмування

Напрям "ПІЗ"

Курс 1 Група ПІ-12

Семестр 2

ЗАВДАННЯ на курсову роботу студента

(прізвище, ім'я, по батькові)

1. Тема роботи _____

2. Строк здачі студентом закінченої роботи _____

3. Вихідні дані до роботи _____

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

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

6. Дата видачі завдання 24.02.2022

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	24.02.2022	
2.	Підготовка ТЗ	20.04.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	01.05.2022	
4.	Розробка сценарію роботи програми	06.05.2022	
5.	Узгодження сценарію роботи програми з керівником	08.05.2022	
6.	Розробка (вибір) алгоритму рішення задачі	15.05.2022	
7.	Узгодження алгоритму з керівником	17.05.2022	
8.	Узгодження з керівником інтерфейсу користувача	19.05.2022	
9.	Розробка програмного забезпечення	29.05.2022	
10.	Налагодження розрахункової частини програми	31.05.2022	
11.	Розробка та налагодження інтерфейсної частини програми	02.06.2022	
12.	Узгодження з керівником набору тестів для контрольного прикладу	03.06.2022	
13.	Тестування програми	04.06.2022	
14.	Підготовка пояснювальної записки	09.06.2022	
15.	Здача курсової роботи на перевірку	12.06.2022	
16.	Захист курсової роботи	16.06.2022	

Студент _____

(підпис)

Керівник _____

(підпис)

_____ Муха І. П. _____

(прізвище, ім'я, по батькові)

"16" червня 2022 р.

АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 81 сторінка, 15 рисунків, 16 таблиць, 3 посилання.

Об'єкт дослідження: задача упорядкування масивів цілих чисел.

Мета роботи: дослідження методів сортування масивів, створення програмного забезпечення для візуалізації їх роботи.

Вивчено метод розробки програмного забезпечення з використанням принципів ООП. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв'язання кожної з них.

Виконана програмна реалізація алгоритмів упорядкування масивів методами швидкого, інтроспективного та сортування злиттям та їх візуалізація.

УПОРЯДКУВАННЯ МАСИВІВ, МЕТОД ШВИДКОГО СОРТУВАННЯ,
МЕТОД ІНТРОСПЕКТИВНОГО СОРТУВАННЯ, МЕТОД СОРТУВАННЯ
ЗЛИТТЯМ.

ЗМІСТ

Анотація	3
Вступ.....	6
1 Постановка задачі.....	7
2 Теоретичні відомості.....	8
2.1 Метод сортування злиттям (Д. фон Неймана).....	8
2.2 Метод швидкого сортування	8
2.3 Метод інтроспективного сортування.	9
2.4 Майстер-метод.	9
3 Опис алгоритмів	10
3.1 Загальний алгоритм	10
3.2 Алгоритм сортування методом злиття	11
3.3 Алгоритм швидкого сортування	12
3.4 Алгоритм інтроспективного сортування.....	13
3.5 Алгоритм поділу масиву на частини та їх сортування (частина альтернативного алгоритму швидкого сортування).....	14
3.6 Алгоритм пірамідального сортування.....	15
3.7 Алгоритм створення купи.....	15
3.8 Алгоритм створення максимальної купи	16
3.9 Алгоритм створення мінімальної купи	16
4 Опис програмного забезпечення	18
4.1 Діаграма класів програмного забезпечення.....	18
4.2 Опис методів частин програмного забезпечення	19
4.2.1 Стандартні методи	19
4.2.2 Користувацькі методи	21
5 Тестування	28

	5
5.1 План тестування.....	28
5.2 Приклади тестування.....	29
6 Інструкція користувача.....	35
6.1 Робота з програмою.....	35
6.2 Формат вхідних та вихідних даних.....	37
6.3 Системні вимоги	38
7 Аналіз і узагальнення результатів	39
7.1 Перевірка коректності результатів	39
7.2 Тестування ефективності алгоритмів	43
7.3 Аналіз часової складності.....	45
Висновки	48
Перелік Посилань	49
Додаток А Технічне завдання	50
Додаток Б Тексти програмного коду.....	53

ВСТУП

Курсова робота присвячена вивченню методів упорядкування масивів та розробці відповідного програмного забезпечення, яке також має графічно відображувати роботу цих методів. У більшості випадків завдання реалізації сортування масивів покладається на вбудовані методи середовища програмування, однак щоб ефективно обирати один з безлічі алгоритмів в залежності від характеру вхідних даних, необхідно знати які алгоритми взагалі існують та розуміти принципи їх роботи. Саме таке завдання має на меті дана робота, а реалізація візуалізації упорядкування допоможе наочно побачити роботу алгоритмів, та запам'ятати принцип їх дії на майбутнє.

Для виконання курсової роботи необхідно:

- а) коректно поставити задачу;
- б) з'ясувати необхідні теоретичні відомості
- в) описати алгоритми, що будуть використовуватися
- г) описати розроблене програмне забезпечення
- д) розробити план тестування та виконати його
- е) розробити інструкцію користувача
- ж) проаналізувати і узагальнити результати, зробивши певні висновки

1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде сортувати заданий масив наступними методами:

- а) метод сортування злиттям (Д. фон Неймана);
- б) метод швидкого сортування;
- в) метод інтроспективного сортування

Вхідними даними для даної роботи є розмір масиву від 100 до 50000 елементів, межі генерації випадкових чисел для його ініціалізації та метод сортування. Програмне забезпечення повинно генерувати відповідну кількість елементів випадковим чином, сортувати їх обраним методом, виводити повідомлення про успішне сортування та записувати статистичні дані в файл.

Вихідними даними для даної роботи являється сукупність цілих чисел, що є відсортованим масивом, який виводяться на екран у вигляді стовпців, висота яких залежить від відносної величини відповідного елемента, якщо кількість елементів не перевищує 500. Якщо кількість елементів більша, то програмне забезпечення повинно виводити на екран повідомлення про успішне завершення сортування та записувати у файл відповідні статистичні дані і відсортований масив. Програмне забезпечення повинно працювати коректно за умови коректності вхідних даних: межі генерації випадкових чисел є цілими числами, а розмір масиву є цілим числом в межах від 100 до 50000. Якщо це не так, то програма повинна вивести відповідне повідомлення.

2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Сортування масиву означає впорядкування набору фіксованої кількості елементів що зберігаються в послідовно розташованих комірках оперативної пам'яті, за певною ознакою. Більшість алгоритмів сортування засновані на порівняннях, що означає, що впорядкування масиву алгоритмом відбувається лише використовуючи операцію порівняння елементів, не використовуючи їх внутрішню структуру. Відсортувати масив за допомогою порівнянь можна одним з наступних методів.

2.1 Метод сортування злиттям (Д. фон Неймана)

Сутність методу Джона фон Неймана полягає в використанні принципу «Розділяй та володарюй». В основі цього способу сортування лежить злиття двох упорядкованих ділянок масиву в одну впорядковану ділянку іншого масиву. Під час сортування в дві допоміжні черги з основної поміщаються перші дві відсортовані підпоследовності, які потім зливаються в одну і результат записується в тимчасову чергу. Потім з основної черги беруться наступні дві відсортовані підпоследовності і так доти, доки основна черга не стане порожньою. Після цього последовність з тимчасової черги переміщується в основну чергу. І знову продовжується сортування злиттям двох відсортованих підпоследовностей. Сортування триватиме доти, доки довжина відсортованої підпоследовності не стане рівною довжині самої последовності. [1]

2.2 Метод швидкого сортування

Сутність методу швидкого сортування полягає в наступному. Вибрати з масиву елемент, який називають опорним. Це може бути будь-який з елементів масиву. Від вибору опорного елемента не залежить коректність алгоритму, але в окремих випадках може сильно залежати його ефективність. Порівняти всі інші елементи з опорним і переставити їх у масиві так, щоб розбити масив на три безперервних підмасиви, наступні один за одним: «менші опорного», «рівні» і «більші». Для підмасивів «менших» і «більших» значень виконати рекурсивно

ту ж послідовність операцій, якщо довжина відрізка більше одиниці. [2, ст. 168-169]

2.3 Метод інтроспективного сортування.

Він використовує швидке сортування і переключається на пірамідальне сортування, коли глибина рекурсії перевищить деякий заздалегідь встановлений рівень (наприклад, логарифм від числа елементів вхідного масиву). Цей підхід поєднує в собі переваги обох методів і має швидкодію, яку можна порівняти з швидким сортуванням. [2, ст. 174]

2.4 Майстер-метод.

Надає готові розв'язки для рекурентних співвідношень та буде використовуватися при обчисленні часової складності алгоритмів попередніх методів сортування. Майстер-метод розглядає рекурентні співвідношення такого виду $T(n) = a T(\frac{n}{b}) + f(n)$, де $a \geq 1$, $b > 1$

При розгляданні рекурсивних алгоритмів, сталі і функції означають наступне:

n — розмір задачі.

a — кількість підзадач на кожному поступі рекурсії.

$\frac{n}{b}$ — розмір кожної з підпроблем. (Тут мається на увазі, що всі підзадачі однакового розміру.)

$f(n)$ — обсяг роботи поза рекурсивними викликами

а) Якщо $f(n) = O(n^{\log_b(a)-\varepsilon})$ для деякої сталої $\varepsilon > 0$, тоді:

$$T(n) = \Theta(n^{\log_b a}) \quad (2.1)$$

б) Якщо для деякої сталої $k \geq 0$ виконується, що $f(n) = \Theta(n^{\log_b a} \log^k n)$, тоді:

$$T(n) = \Theta(n^{\log_b a} \log^{k+1} n) \quad (2.2)$$

в) Якщо $f(n) = \Omega(n^{\log_b(a)+\varepsilon})$ для деякої сталої $\varepsilon > 0$, а також $af(\frac{n}{b}) \leq cf(n)$ для деякої сталої $c < 1$ і достатньо великих n , тоді:

$$T(n) = \Theta(f(n)) \quad (2.3)$$

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
n	Розмір масиву (за умовчанням 150 елементів)
min_v	Нижня межа генерації випадкових чисел (за умовчанням -100)
max_v	Верхня межа генерації випадкових чисел (за умовчанням 100)
start	Індекс елемента масиву починаючи від якого масив буде сортуватися
end	Індекс елемента масиву до якого масив буде сортуватися
lst	Ім'я масиву
ascending	Тип сортування True – за зростання, False – за спаданням (за умовчанням True)
count_of_swaps	Кількість перестановок
count_of_comparisons	Кількість порівнянь
max_depth_curr	Поточна глибина рекурсії для інтроспективного сортування (за умовчанням $\lfloor \log_2 n \rfloor$)

3.1 Загальний алгоритм

1. ПОЧАТОК

2. Зчитати розмір масиву n

2.1. Якщо n некоректний, вивести повідомлення та залишити значення за замовченням

3. Зчитати діапазон генерації випадкових чисел min_v та max_v

3.1. Якщо min_v або max_v некоректні, вивести повідомлення та залишити значення за замовченням

4. Зчитати метод сортування.

5. ЯКЩО обраний метод сортування злиттям, ТО обробити дані згідно алгоритму методу сортування злиттям (пункт 3.3)
6. ЯКЩО обраний метод швидкого сортування, ТО обробити дані згідно алгоритму методу швидкого сортування (пункт 3.4)
7. ЯКЩО обраний метод інтроспективного сортування, ТО обробити дані згідно алгоритму методу інтроспективного сортування (пункт 3.5).
8. ЯКЩО розмір масиву менше 200 елементів, ТО створити візуалізацію сортування
9. Вивести повідомлення про успішне сортування
10. Записати назву використаного алгоритму, кількість елементів у ньому, кількість порівнянь й перестановок що відбулись при сортуванні та відсортований масив у файл.
11. КІНЕЦЬ

3.2 Алгоритм сортування методом злиття

1. ПОЧАТОК

2. ЯКЩО $end - start > 1$

2.1. $middle = (start + end) // 2$

2.2. обробити дані згідно з методом сортування злиттям (пункт 3.3)
для $start = start$, $end = middle$

2.3. обробити дані згідно з методом сортування злиттям (пункт 3.3)
для $start = middle$, $end = end$

2.4. $left = lst[start, \dots middle]$

2.5. $right = lst[middle, \dots end]$

2.6. $a = 0$

2.7. $b = 0$

2.8. $c = start$

2.9. ПОКИ $a < \text{length}(left)$ та $b < \text{length}(right)$

2.9.1. $\text{count_of_comparisons} += 1$

2.9.2. ЯКЩО $\text{left}[a] < \text{right}[b]$ та $\text{ascending} == \text{True}$ або $\text{left}[a] > \text{right}[b]$ та $\text{ascending} == \text{False}$:

2.9.2.1. $\text{count_of_swaps} += 1$

2.9.2.2. намалювати поточний стан масиву

2.9.2.3. $\text{lst}[c] = \text{lst}[a]$

2.9.2.4. $a += 1$

2.9.3. ІНАКШЕ:

2.9.3.1. намалювати поточний стан масиву

2.9.3.2. $\text{count_of_swaps} += 1$

2.9.3.3. $\text{lst}[c] = \text{lst}[b]$

2.9.3.4. $b += 1$

2.10. ПОКИ $a < \text{length}(\text{left})$:

2.10.1. $\text{count_of_swaps} += 1$

2.10.2. намалювати поточний стан масиву

2.10.3. $\text{lst}[c] = \text{lst}[a]$

2.10.4. $a += 1$

2.10.5. $c += 1$

2.11. ПОКИ $b < \text{length}(\text{left})$:

2.11.1. $\text{count_of_swaps} += 1$

2.11.2. намалювати поточний стан масиву

2.11.3. $\text{lst}[c] = \text{lst}[b]$

2.11.4. $b += 1$

2.11.5. $c += 1$

2.12. повернути lst

3. КІНЕЦЬ

3.3 Алгоритм швидкого сортування

1. ПОЧАТОК

2. ЯКЩО $\text{end} == \text{length}(\text{lst})$:

2.1. $\text{end} -= 1$

3. ЯКЩО $\text{start} \geq \text{end}$:
 - 3.1. повернути
4. $x = \text{lst}[\text{start}]$
5. ДЛЯ i від $\text{start} + 1$ до $\text{end} + 1$ з кроком 1:
 - 5.1. $\text{count_of_comparisons} += 1$
 - 5.2. ЯКЩО $\text{lst}[i] \leq x$ та $\text{ascending} == \text{True}$ або $\text{lst}[i] \geq x$ та $\text{ascending} == \text{False}$:
 - 5.2.1. $j += 1$
 - 5.2.2. ЯКЩО $j \neq i$:
 - 5.2.2.1. $\text{count_of_swaps} += 1$
 - 5.2.2.2. $\text{lst}[i]$ та $\text{lst}[j]$ поміняти місцями
 - 5.2.3. намалювати поточний стан масиву
6. ЯКЩО $\text{start} \neq j$:
 - 6.1. $\text{count_of_swaps} += 1$
 - 6.2. $\text{lst}[\text{start}]$ та $\text{lst}[j]$ поміняти місцями
7. намалювати поточний стан масиву
8. обробити дані згідно з методом швидкого сортування (пункт 3.4) для $\text{start} = \text{start}$, $\text{end} = j - 1$
9. обробити дані згідно з методом швидкого сортування (пункт 3.4) для $\text{start} = j + 1$, $\text{end} = \text{end}$
10. КІНЕЦЬ

3.4 Алгоритм інтроспективного сортування

1. ПОЧАТОК
2. ЯКЩО $\text{end} - \text{start} \leq 1$:
 - 2.1. повернути
3. ІНАКШЕ ЯКЩО $\text{max_depth_curr} == 0$:
 - 3.1. обробити дані згідно з методом пірамідального сортування (пункт 3.7) для $\text{start} = \text{start}$, $\text{end} = \text{end}$
4. ІНАКШЕ $p = \text{partition}(\text{start}, \text{end})$ (пункт 3.6)

5. обробити дані згідно з методом інтроспективного сортування (пункт 3.5) для $\text{start} = \text{start}$, $\text{end} = p + 1$, $\text{max_depth_curr} = \text{max_depth_curr} - 1$
6. обробити дані згідно з методом інтроспективного сортування (пункт 3.5) для $\text{start} = p + 1$, $\text{end} = \text{end}$, $\text{max_depth_curr} = \text{max_depth_curr} - 1$
7. КІНЕЦЬ

3.5 Алгоритм поділу масиву на частини та їх сортування (частина альтернативного алгоритму швидкого сортування)

1. ПОЧАТОК
2. $\text{pivot} = \text{lst}[\text{start}]$
3. $\text{left} = \text{start} - 1$
4. $\text{right} = \text{end}$
5. ПОКИ True:
 - 5.1. $\text{left} += 1$
 - 5.2. ЯКЩО $\text{ascending} == \text{True}$:
 - 5.2.1. ПОКИ $\text{lst}[\text{left}] < \text{pivot}$:
 - 5.2.1.1. $\text{count_of_comparisons} += 1$
 - 5.2.1.2. $\text{left} += 1$
 - 5.3. ІНАКШЕ:
 - 5.3.1. ПОКИ $\text{lst}[\text{left}] > \text{pivot}$:
 - 5.3.2. $\text{count_of_comparisons} += 1$
 - 5.3.3. $\text{left} += 1$
 - 5.4. $\text{right} -= 1$
 - 5.5. ЯКЩО $\text{ascending} == \text{True}$:
 - 5.5.1. ПОКИ $\text{lst}[\text{right}] > \text{pivot}$:
 - 5.5.1.1. $\text{count_of_comparisons} += 1$
 - 5.5.1.2. $\text{right} -= 1$
 - 5.6. ІНАКШЕ:
 - 5.6.1. ПОКИ $\text{lst}[\text{right}] < \text{pivot}$:
 - 5.6.2. $\text{count_of_comparisons} += 1$

5.6.3. right -= 1

5.7. ЯКЩО $\text{left} \geq \text{right}$:

5.7.1. повернути right

5.8. $\text{count_of_swaps} += 1$

5.9. намалювати поточний стан масиву

5.10. $\text{lst}[\text{left}]$ та $\text{lst}[\text{right}]$ поміняти місцями

6. КІНЕЦЬ

3.6 Алгоритм пірамідального сортування

1. ПОЧАТОК

2. обробити дані згідно з алгоритмом створення купи для $\text{start} = \text{start}$, $\text{end} == \text{end}$ (пункт 3.8)

3. ДЛЯ i від $\text{end} - 1$ до start з кроком -1 :

3.1. $\text{count_of_swaps} += 1$

3.2. намалювати поточний стан масиву

3.3. $\text{lst}[\text{start}]$ та $\text{lst}[i]$ поміняти місцями

3.4. ЯКЩО $\text{ascending} == \text{True}$:

3.4.1. обробити дані згідно з алгоритмом створення максимальної купи (пункт 3.9) для $i = 0$, $\text{start} = \text{start}$, $\text{end} = i$

3.5. ІНАКШЕ:

3.5.1. обробити дані згідно з алгоритмом створення мінімальної купи (пункт 3.10) для $i = 0$, $\text{start} = \text{start}$, $\text{end} = i$

4. КІНЕЦЬ

3.7 Алгоритм створення купи

1. ПОЧАТОК

2. $\text{length} = \text{end} - \text{start}$

3. $\text{index} = ((\text{length} - 1) - 1) // 2$

4. ПОКИ $\text{index} \geq 0$

4.1. ЯКЩО $\text{ascending} == \text{True}$:

4.1.1. обробити дані згідно з алгоритмом створення максимальної купи (пункт 3.9) для $i = \text{index}$, $\text{start} = \text{start}$, $\text{end} = \text{end}$

4.2. ІНАКШЕ:

4.2.1. обробити дані згідно з алгоритмом створення мінімальної купи (пункт 3.10) для $i = \text{index}$, $\text{start} = \text{start}$, $\text{end} = \text{end}$

4.3. $\text{index} -= 1$

5. КІНЕЦЬ

3.8 Алгоритм створення максимальної купи

1. ПОЧАТОК

2. $\text{size} = \text{end} - \text{start}$

3. $l = 2 * i + 1$

4. $r = 2 * i + 2$

5. $\text{largest} = i$

6. $\text{count_of_comparisons} += 2$

7. ЯКЩО $l < \text{size}$ і $\text{lst}[\text{start} + l] > \text{lst}[\text{start} + i]$:

7.1. $\text{largest} = l$

8. ЯКЩО $r < \text{size}$ і $\text{lst}[\text{start} + r] > \text{lst}[\text{start} + \text{largest}]$:

8.1. $\text{largest} = r$

9. ЯКЩО $\text{largest} \neq i$:

9.1. $\text{count_of_swaps} += 1$

9.2. $\text{lst}[\text{start} + \text{largest}]$ та $\text{lst}[\text{start} + i]$ поміняти місцями

9.3. намалювати поточний стан масиву

9.4. обробити дані згідно з алгоритмом створення максимальної купи (пункт 3.9) для $i = \text{largest}$, $\text{start} = \text{start}$, $\text{end} = \text{end}$

10.КІНЕЦЬ

3.9 Алгоритм створення мінімальної купи

1. ПОЧАТОК

2. $\text{size} = \text{end} - \text{start}$

3. $l = 2 * i + 1$
4. $r = 2 * i + 2$
5. $smallest = i$
6. $count_of_comparisons += 2$
7. ЯКЩО $l < size$ і $lst[start + l] > lst[start + i]$:
 - 7.1. $smallest = l$
8. ЯКЩО $r < size$ і $lst[start + r] > lst[start + smallest]$:
 - 8.1. $smallest = r$
9. ЯКЩО $smallest \neq i$:
 - 9.1. $count_of_swaps += 1$
 - 9.2. намалювати поточний стан масиву
 - 9.3. $lst[start + largest]$ та $lst[start + i]$ поміняти місцями
 - 9.4. обробити дані згідно з алгоритмом створення мінімальної купи
(пункт 3.9) для $i = smallest$, $start = start$, $end = end$
10. КІНЕЦЬ

4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Діаграма класів програмного забезпечення

У діаграмі класів (рисунок 4.1) наведено UML діаграму, що демонструє класи програмного забезпечення, та їх взаємодію один з одним.

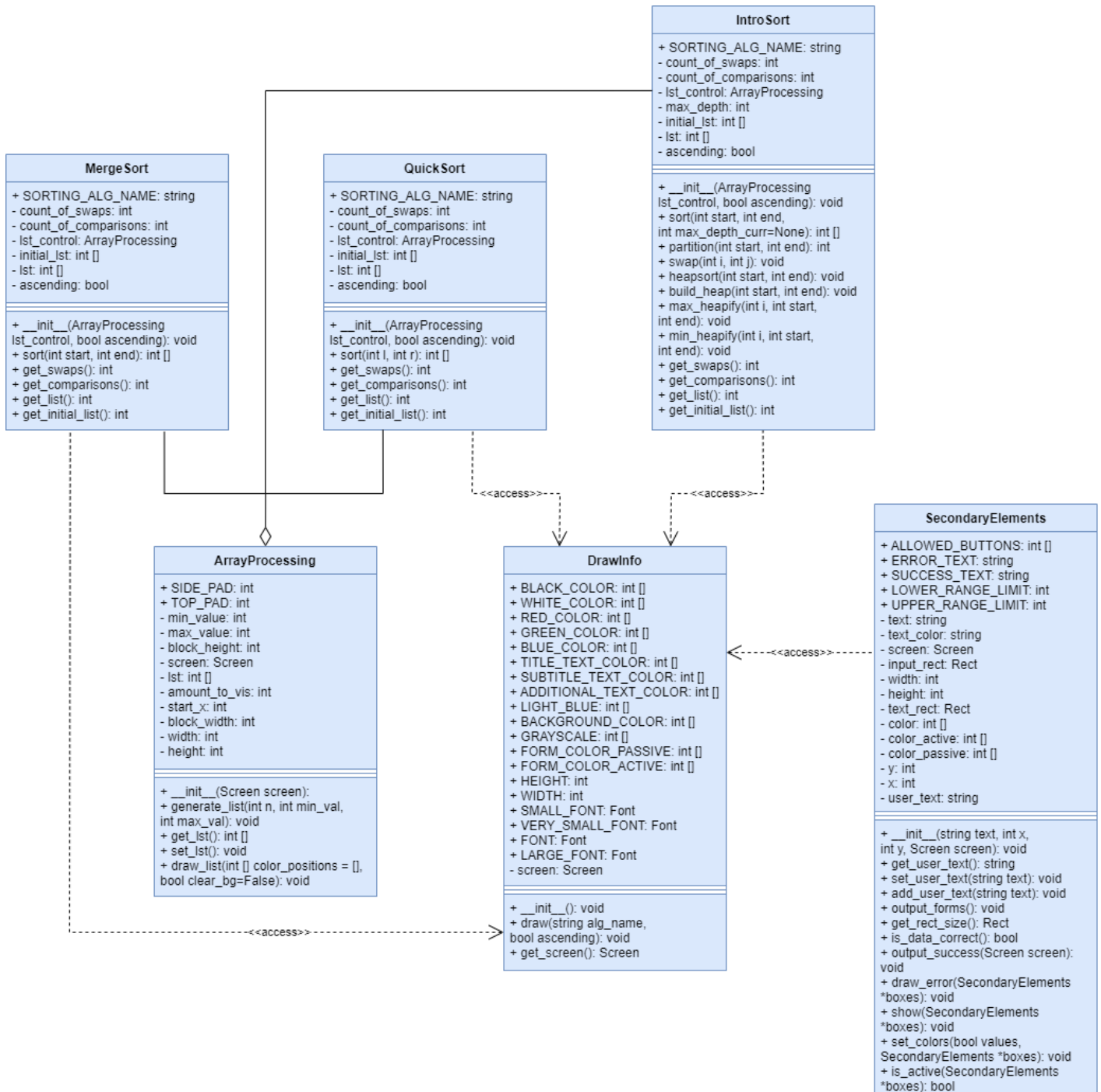


Рисунок 4.1 – Діаграма класів

4.2 Опис методів частин програмного забезпечення

4.2.1 Стандартні методи

У таблиці 4.1 наведені стандартні методи, що були використані при розробці програмного забезпечення, більшість з яких містяться у бібліотеці pygame [3].

Таблиця 4.1– Стандартні методи

№ п/п	Назва класу/модуля	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	random	randint	генерування випадкових чисел для ініціалізації елементів масиву	два цілих числа, що представляють собою діапазон генерації випадкових чисел	ціле випадкове число
2		append	додавання елементу в кінець списку	елемент, що додається до списку	список з доданим елементом
3	pygame.draw	rect	виведення прямокутника на екран	об'єкт типу Surface, колір (у вигляді кортежу або списку трьох цілих чисел), прямокутник (заданий своїми координатами та розмірами)	прямокутник, що обмежує змінені пікселі
4	pygame.display	update	оновлення екрану (або його частини)	прямокутник, що є областю, яка буде оновлюватися	
5	pygame.font	Font	створення шрифту для подальшого виведення тексту	назва шрифту, розмір у вигляді цілого числа	об'єкт типу Font з заданими параметрами

Продовження таблиці 4.1

№ п/п	Назва класу/модуля	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
6	pygame.display	set_mode	ініціалізація вікна для відображення	розмір вікна у вигляді двох цілих чисел	
7	pygame.display	set_ caption	встановлення назви для вікна програми	назва програми	
8	pygame.display	set_icon	встановлення іконки для вікна програми	об'єкт типу Surface	
9	pygame.image	load	створення поверхні з зображення	назва файлу (з шляхом до нього)	об'єкт типу Surface
10	pygame.Surface	fill	заповнення поверхні суцільним кольором	колір	
11	pygame.font. .Font	render	виведення тексту на екран	текст, згладжування (True або False), колір	об'єкт типу Surface з текстом на ній
12	pygame.Surface	blit	виведення поверхні	дві поверхні типу Surface, яку, та на яку необхідно вивести	
13		isdigit	перевіряє, чи всі символи в тексті є цифрами		True якщо перевірка пройдена, інакше False
14	copy	deepcopy	створення глибокої копії об'єкта	об'єкт оригінал	створена копія об'єкту

Продовження таблиці 4.1

№ п/п	Назва класу/модуля	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
15	math	log2	отримання логарифму з основою 2 від числа	число, від якого необхідно взяти логарифм	число, що відповідає логарифму
16	math	floor	округлення числа вниз до найближчого цілого	число, яке необхідно округлити	округлене число
17	os	remove	видалення файлу з каталогу	назва файлу (з шляхом до нього)	
18		open	відкриття файлу	назва файлу (з шляхом до нього), тип відкриття файлу	файловий об'єкт
19		write	запис інформації у файл	текст (або байт), який необхідно записати	
20	time	sleep	додавання затримки у виконанні програми	час у секундах	

4.2.2 Користувацькі методи

У таблиці 4.2 наведені користувацькі методи, що були використані при розробці програмного забезпечення.

Таблиця 4.2– Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
1	ArrayProcessing	__init__	конструктор класу	покажчик на об'єкт класу, екран	

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
2	ArrayProcessing	generate_list	створення масиву	покажчик на об'єкт класу, кількість елементів, мінімальне та максимальне значення діапазону генерації випадкових чисел	
3	ArrayProcessing	get_lst	отримання згенерованого масиву	покажчик на об'єкт класу	створений масив
4	ArrayProcessing	set_lst	розрахунок ширини та висоти умовної одиниці елемента масиву одного стовпчика	покажчик на об'єкт класу	
5	ArrayProcessing	draw_list	виведення поточного стану масиву	покажчик на об'єкт класу, словник з індексами елементів масиву та відповідні їм кольори, прапор необхідності оновлення екрану	
6	DrawInfo	__init__	конструктор класу	покажчик на об'єкт класу	

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
7	DrawInfo	draw	виведення заголовних написів	покажчик на об'єкт класу, назва та тип алгоритму сортування	
8	DrawInfo	get_screen	повернення екрану	покажчик на об'єкт класу	екран
9	SecondaryElements	__init__	конструктор класу	покажчик на об'єкт класу	
10	SecondaryElements	get_user_text	повернення тексту, що ввів користувач	покажчик на об'єкт класу	текст користувача
11	SecondaryElements	set_user_text	встановлення тексту, що ввів користувач	покажчик на об'єкт класу, текст	
12	SecondaryElements	add_user_ text	додавання тексту, до тексту, який ввів користувач	покажчик на об'єкт класу, текст	
13	SecondaryElements	output_forms	виведення тексту разом із формою введення	покажчик на об'єкт класу	
14	SecondaryElements	get_rect_size	отримання об'єкту типу Rect, що є формою для введення	покажчик на об'єкт класу	об'єкт типу Rect який представляє комірку вводу даних
15	SecondaryElements	is_data_ correct	перевірка коректності даних у формі введення	покажчик на об'єкт класу	True в разі коректності даних, інакше False

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
16	SecondaryElements	output_ success	виведення повідомлення про успішне сортування	екран	
17	SecondaryElements	draw_error	виведення повідомлення про некоректні дані під коміркою, яка їх містить	показчик на перший з трьох об'єктів даного класу	
18	SecondaryElements	show	виведення усіх трьох комірок разом з відповідними їм написами	показчик на перший з трьох об'єктів даного класу	
19	SecondaryElements	set_colors	зміна кольору комірок в залежності від їх активності	масив значень логічного типу, що відповідає за активність форми та показчик на перший з трьох об'єктів даного класу	
20	SecondaryElements	is_active	отримання інформації про активність комірки введення даних	показчик на перший з трьох об'єктів даного класу	масив значень логічного типу, що відповідає за активність форми

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
21	IntroSort, QuickSort, MergeSort	__init__	конструктор класу	покажчик на об'єкт класу, об'єкт класу ArrayProcessing, прапор що відповідає за тип сортування (True – за зростанням, False – за спаданням)	
22	IntroSort	partition	швидке сортування частини масиву	покажчик на об'єкт класу, початковий та кінцевий індекс діапазону сортування масиву	індекс опорного елементу на основі якого поділяється масив
23	IntroSort	swap	зміна двох елементів місцями	покажчик на об'єкт класу, два індекси	
24	IntroSort	heapsort	пірамідальне сортування частини масиву	покажчик на об'єкт класу, початковий та кінцевий індекс діапазону сортування масиву	
25	IntroSort	build_heap	створення купи для відповідного сортування	покажчик на об'єкт класу, початковий та кінцевий індекс діапазону сортування масиву	

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
26	IntroSort	max_heapify	створення купи у якій нащадки не більші за предків	покажчик на об'єкт класу, індекс елементу для якого створюється купа, початковий та кінцевий індекс діапазону сортування масиву	
27	IntroSort	min_heapify	створення купи у якій нащадки не менші за предків	покажчик на об'єкт класу, індекс елементу для якого створюється купа, початковий та кінцевий індекс діапазону сортування масиву	
28	IntroSort, QuickSort, MergeSort	sort	сортування масиву відповідним методом	покажчик на об'єкт класу, початковий та кінцевий індекс діапазону сортування масиву	масив у кінцевому стані
29	IntroSort, QuickSort, MergeSort	get_swaps	отримання кількості перестановок, що відбулись при сортуванні	покажчик на об'єкт класу	кількість перестановок
30	IntroSort, QuickSort, MergeSort	get_ comparisons	отримання кількості порівнянь, що відбулись при сортуванні	покажчик на об'єкт класу	кількість порівнянь

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів
31	IntroSort, QuickSort, MergeSort	get_list	отримання поточного масиву	показчик на об'єкт класу	масив у поточному стані
32	IntroSort, QuickSort, MergeSort	get_initial_ list	отримання початкового масиву	показчик на об'єкт класу	масив у початковому стані

5 ТЕСТУВАННЯ

5.1 План тестування

Для подальшого проведення тестування розробимо план, який охоплюватиме основний функціонал програмного забезпечення та матиме на меті виявити можливі помилки при роботі програми.

а) Тестування правильності введених значень.

- 1) Тестування при введенні некоректних символів.
- 2) Тестування при введенні замалих та зовеликих значень.

б) Тестування коректної роботи при незапланованих діях користувача під час візуалізації упорядкування масиву.

- 1) Тестування роботи програми при закритті програми під час візуалізації.
- 2) Тестування роботи програми при спробах змінити метод, тип сортування або вхідні дані під час візуалізації.
- 3) Тестування можливості не аварійно (не закриваючи програму) зупинити візуалізацію.

в) Тестування коректності роботи методів швидкого, інтроспективного та сортування злиттям.

- 1) Перевірка коректності роботи методу швидкого сортування.
- 2) Перевірка коректності роботи методу інтроспективного сортування.
- 3) Перевірка коректності роботи методу сортування злиттям.

г) Тестування запису статистичних даних у файл.

- 1) Тестування запису за відсутності попередньо створеного файлу.
- 2) Тестування запису у файл, який містить у собі якісь дані.
- 3) Тестування запису у файл кількох сесій сортування.

5.2 Приклади тестування

Згідно з планом, проведемо відповідні тести для впевненості у коректності роботи окремих методів та програми в цілому у різних сценаріях шляхом введення у програму відповідних вхідних даних та моніторингу стану програмного забезпечення

- а) Перевіримо роботу програми при введенні некоректних (таблиця 5.1) та замалих і завеликих даних (таблиця 5.2).
- б) Перевіримо реакцію програми на непередбачені дії користувача: раптове закриття програми під час її роботи (таблиця 5.3), зміну вхідних даних (таблиця 5.4), спроби зупинити роботу програми (таблиця 5.5).
- в) Перевіримо правильність роботи методів сортування: швидке (таблиця 5.6), інтроспективне (таблиця 5.7), злиттям (таблиця 5.8).
- г) Перевіримо спроможність запису у файл статистичних даних при його відсутності (таблиця 5.9), наявності у ньому інших даних (таблиця 5.10) та при сортуванні кількох масивів різними методами протягом однієї сесії роботи програми (таблиця 5.11).

Таблиця 5.1 - Приклад роботи програми при введенні некоректних даних

Мета тесту	Перевірити можливість введення некоректних даних
Початковий стан програми	Відкрите вікно програми
Вхідні дані	23b6 %46f f9-17
Схема проведення тесту	Почергове заповнення комірок «Size», «Max value», «Min value»
Очікуваний результат	Повідомлення про помилку формату даних
Стан програми після проведення випробувань	Програма зчитала лише цифри та знак «-», у комірці «Size» опинилося значення 236, у комірці «Max value» 46, у «Min value» 9-17 та напис «Incorrect value» нижче.

Таблиця 5.2 - Приклад роботи програми при введенні замалих і завеликих даних

Мета тесту	Перевірити можливість введення неоптимальних даних
Початковий стан програми	Відкрите вікно програми
Вхідні дані	1) 1000000 1000000 -1000000 2) 50 1 1
Схема проведення тесту	Почергове заповнення комірок «Size», «Max value», «Min value»
Очікуваний результат	Повідомлення про помилку формату даних
Стан програми після проведення випробувань	1) У комірках з'явилися лише перші 5 цифр від введеного числа (більше програма не зчитувала) 2) Під комірками «Max value» і «Min value» з'явилися написи «Incorrect value» (оскільки вони не можуть бути рівними)

Таблиця 5.3 - Приклад роботи програми при закритті її під час візуалізації

Мета тесту	Перевірити реакцію програми на екстремальне закриття
Початковий стан програми	Відкрите вікно програми
Вхідні дані	-
Схема проведення тесту	Запуск візуалізації впорядкування масиву, та закриття програми шляхом натиску «Alt» та «F4»
Очікуваний результат	Миттєве закриття програми
Стан програми після проведення випробувань	Програма успішно завершила роботу (напис у консолі «Process finished with exit code 0»)

Таблиця 5.4 - Приклад роботи програми при спробах змінити метод або тип сортування, вхідні дані під час візуалізації

Мета тесту	Перевірити можливість змінювати початкові дані під час візуалізації
Початковий стан програми	Відкрите вікно програми
Вхідні дані	-
Схема проведення тесту	Запуск візуалізації впорядкування масиву, натискання клавіші «M» для зміни методу сортування, «D» для зміни типу та натиск мишкою на комірку Size для введення нового розміру масиву
Очікуваний результат	Ігнорування дій користувача програмою
Стан програми після проведення випробувань	Програма не сприйняла натискання ЛКМ та клавіш на клавіатурі, тому не змінила свого стану (за винятком тепер вже впорядкованих елементів масиву)

Таблиця 5.5 - Приклад роботи програми при спробі не аварійно (не закриваючи програму) зупинити візуалізацію

Мета тесту	Перевірити можливість зупинки візуалізації без переривання роботи програми
Початковий стан програми	Відкрите вікно програми
Вхідні дані	-
Схема проведення тесту	Запуск візуалізації впорядкування масиву та натискання клавіші «R» для припинення процесу сортування
Очікуваний результат	Зупинка процесу впорядкування елементів
Стан програми після проведення випробувань	Зупинене сортування та створений новий масив відповідно до вхідних даних, що були у комірках

Таблиця 5.6 - Приклад роботи методу швидкого сортування.

Мета тесту	Перевірити роботу методу швидкого сортування
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Метод сортування – швидке сортування
Схема проведення тесту	Вибір вказаного методу, запуск сортування
Очікуваний результат	Коректно відсортований масив
Стан програми після проведення випробувань	На екрані стовпці візуально відсортовані, у файл записан впорядкований масив

Таблиця 5.7 - Приклад роботи методу інтроспективного сортування

Мета тесту	Перевірити роботу методу інтроспективного сортування
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Метод сортування – інтроспективне сортування
Схема проведення тесту	Вибір вказаного методу, запуск сортування
Очікуваний результат	Коректно відсортований масив
Стан програми після проведення випробувань	На екрані стовпці візуально відсортовані, у файл записан впорядкований масив

Таблиця 5.8 - Приклад роботи методу сортування злиттям

Мета тесту	Перевірити роботу методу сортування злиттям
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Метод сортування – сортування злиттям
Схема проведення тесту	Вибір вказаного методу, запуск сортування
Очікуваний результат	Коректно відсортований масив
Стан програми після проведення випробувань	На екрані стовпці візуально відсортовані, у файл записан впорядкований масив

Таблиця 5.9 - Приклад запису статистичних даних за відсутності попередньо створеного файлу

Мета тесту	Перевірити можливість запису у файл, без потреби створювати його власноруч
Початковий стан програми	Відкрите вікно програми
Вхідні дані	-
Схема проведення тесту	Запуск впорядкування масиву, перевірка наявності файлу з вихідними даними
Очікуваний результат	Створення програмним забезпеченням потрібного файлу
Стан програми після проведення випробувань	У каталозі головної програми створено файл та записано у нього вихідні дані

Таблиця 5.10 - Приклад роботи програми при наявності інших даних у вихідному файлі.

Мета тесту	Перевірити можливість запису у файл, який містить в собі іншу інформацію
Початковий стан програми	Відкрите вікно програми
Вхідні дані	-
Схема проведення тесту	Запуск впорядкування масиву, перевірка файлу з вихідними даними
Очікуваний результат	Очищення файлу програмним забезпеченням, з подальшим записом у пустий файл нових даних
Стан програми після проведення випробувань	Вихідний файл містить лише вихідні дані

Таблиця 5.11 - Приклад роботи програми при записі у файл кількох сесій сортування.

Мета тесту	Перевірити можливість записувати у файл статистичні дані про декілька сесій сортування
Початковий стан програми	Відкрите вікно програми
Вхідні дані	1) Розмір – 100, швидке сортування 2) Розмір – 200, сортування злиттям 3) Розмір – 300, інтроспективне сортування
Схема проведення тесту	Запуск впорядкування масиву з трьома наборами різних вхідних даних
Очікуваний результат	Статистичні дані про три сесії сортування
Стан програми після проведення випробувань	Вихідний файл містить інформацію про сортування на всіх наборах вхідних даних

6 ІНСТРУКЦІЯ КОРИСТУВАЧА

6.1 Робота з програмою

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

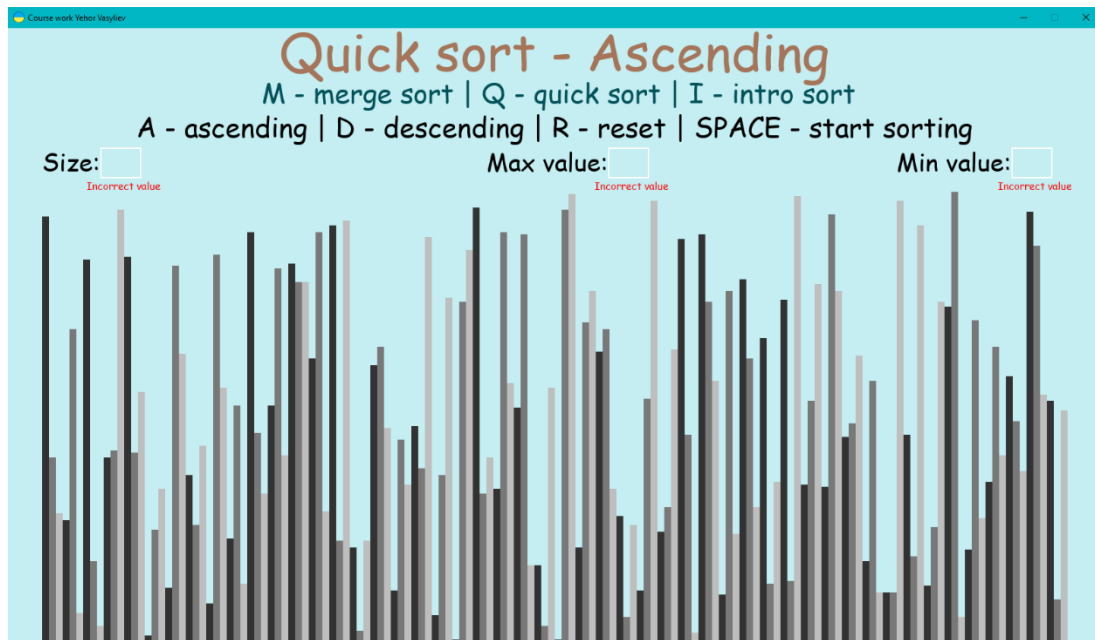


Рисунок 6.1 – Головне вікно програми

Далі за допомогою полів для введення з назвами «Size:», «Max value:», «Min value:» шляхом натиску на відповідні прямокутники поруч необхідно ввести числа з клавіатури (не з блоку NUM PAD), що відповідатимуть розміру масиву, верхній і нижній межі генерування випадкових чисел для його ініціалізації (рисунок 6.2).

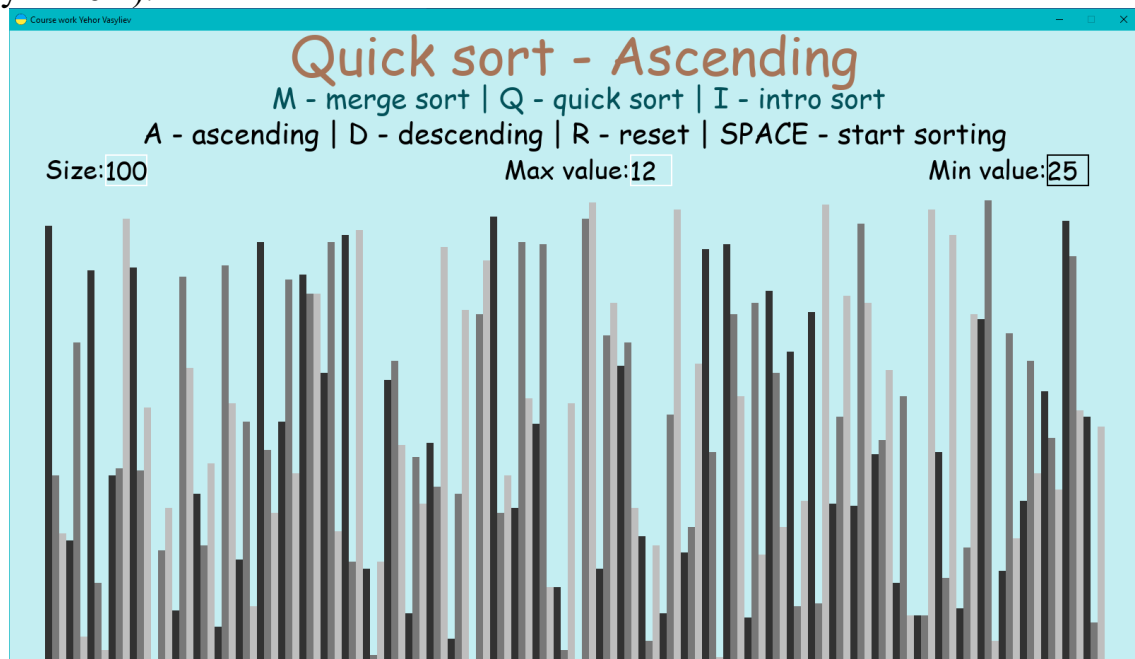
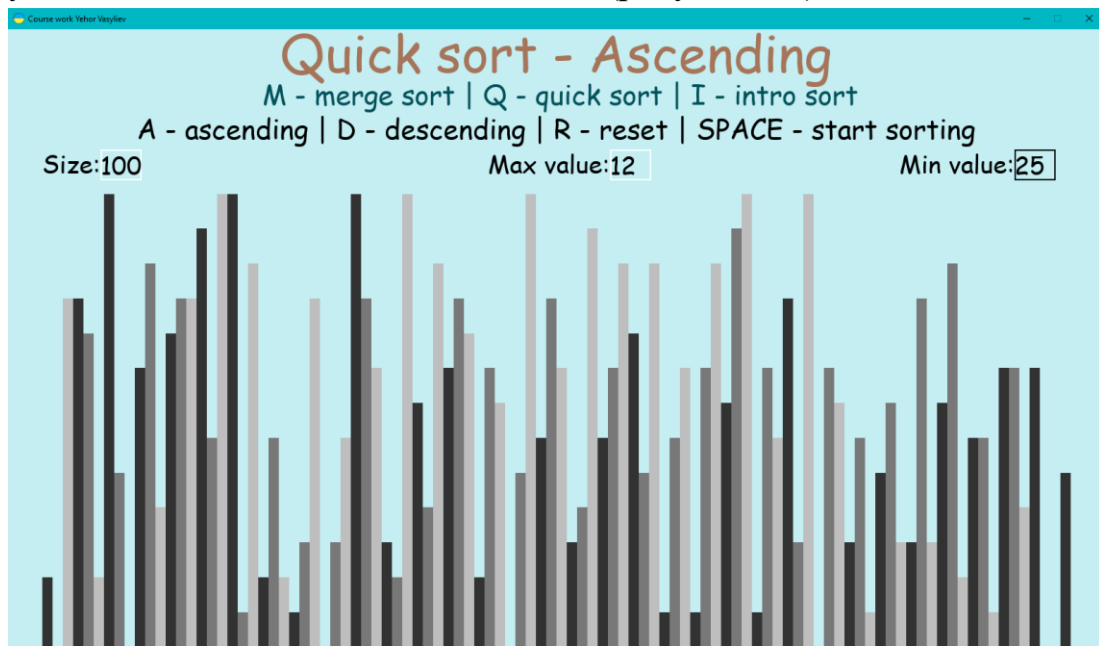


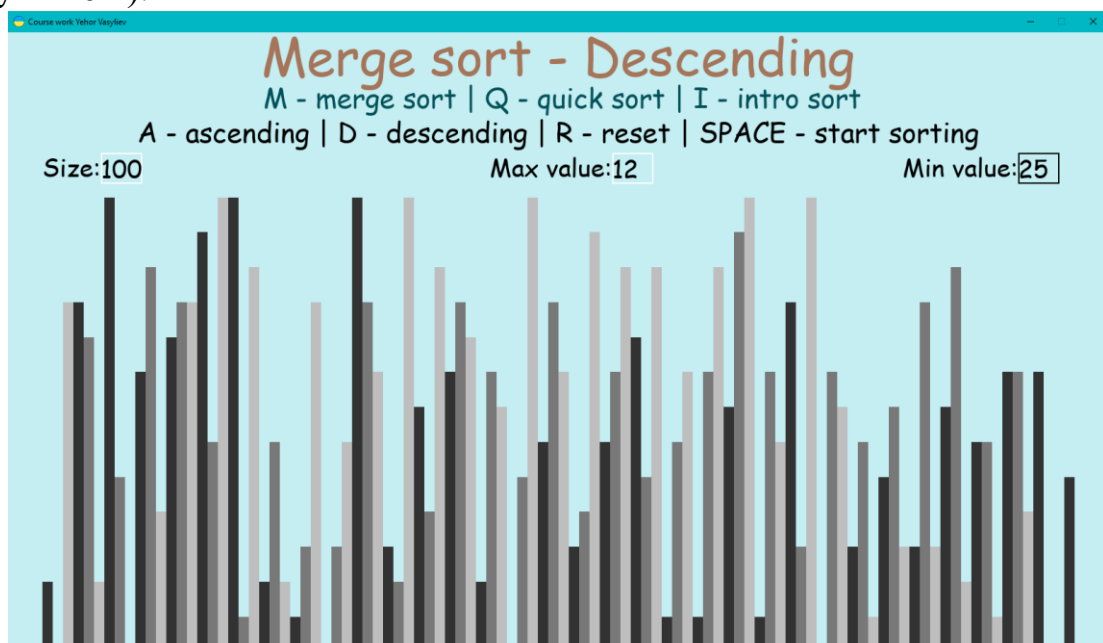
Рисунок 6.2 – Введення вхідних даних

Потім необхідно натиснути кнопку «R» на клавіатурі для генерації нового масиву, що відповідатиме введеним даним (рисуюнок 6.3).



Рисуюнок 6.3 – Підтвердження введених даних

Далі за допомогою клавіш «M», «Q», «I», «A», «D» на клавіатурі обирається метод та тип сортування («A» - за зростанням, «D» - за спаданням), зміна цих параметрів відображатиметься у верхній частині головного вікна (рисуюнок 6.4).



Рисуюнок 6.4 – Вибір методу та типу сортування

Для початку сортування необхідно натиснути клавішу пробіл (space), після чого почнеться візуалізація сортування масиву (у разі якщо введена розмірність масиву ≤ 500) (рисуюнок 6.5).

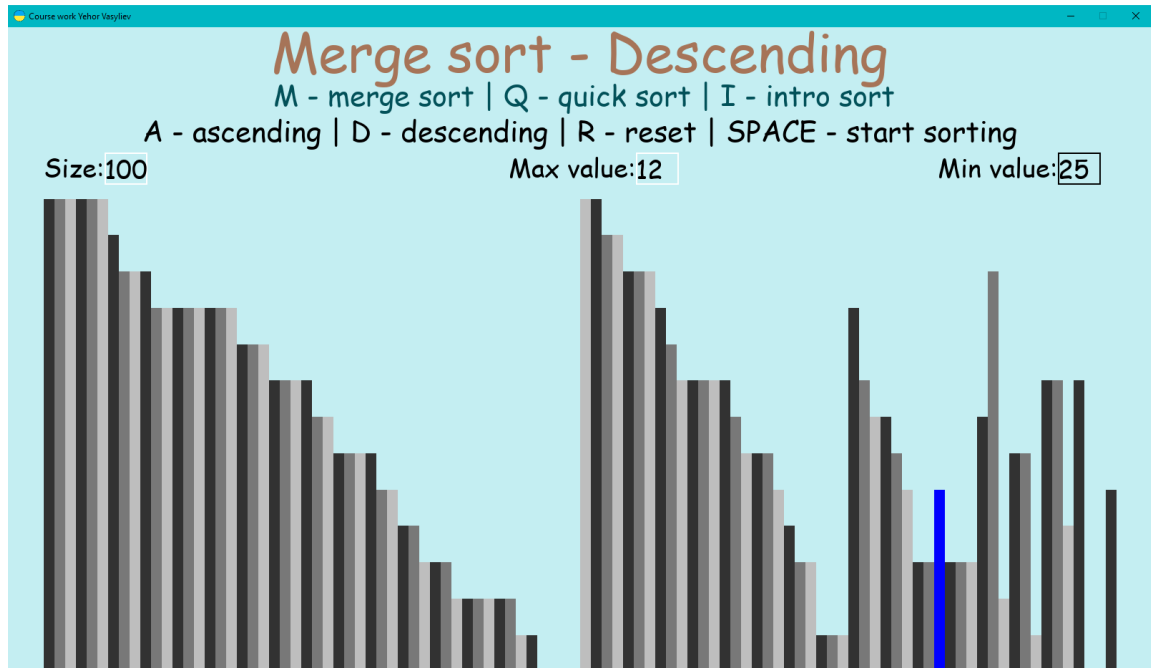


Рисунок 6.5 – Візуалізація сортування

Після завершення сортування виведеться напис **SUCCESS**, а для задання нових вхідних даних необхідно буде натиснути на відповідні поля для вводу та після очищення даних в них шляхом натискання клавіші «BACK SPACE» на клавіатурі повторити усі кроки описані раніше (рисунок 6.6).

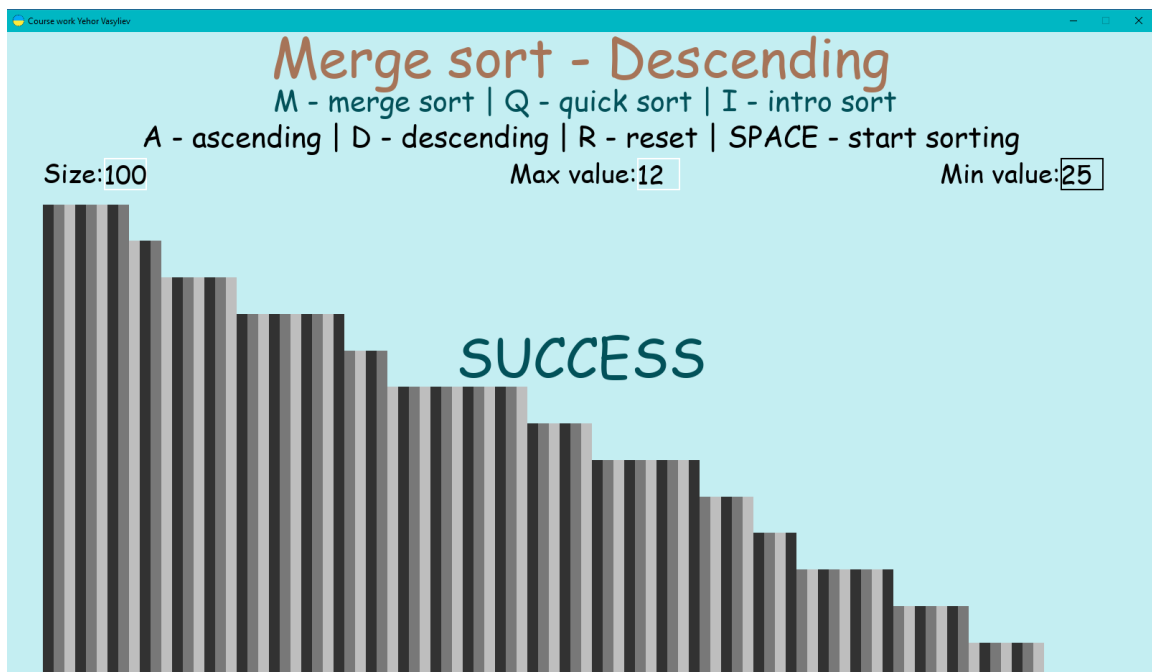


Рисунок 6.6 – успішне завершення сортування

6.2 Формат вхідних та вихідних даних

Користувачем на вхід програми подаються параметри для генерування масиву у вигляді цілих чисел, довжиною до 6 знаків, причому розмір масиву має бути в межах від 100 до 50000 (в іншому випадку буде використано попереднє

значення та виведено повідомлення Incorrect value), а верхня межа має бути більшою за нижню (в іншому випадку програма поміняє їх місцями). Результатом виконання програми є відсортований масив, який записується у файл разом із іншими статистичними даними.

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 7/ Windows 8/Windows 10 (з останніми оновленнями)	Windows 8/Windows 10 (з останніми оновленнями)
Процесор	Intel® Pentium® III 1.0 GHz або AMD Athlon™ 1.0 GHz	Intel® Pentium® D або AMD Athlon™ 64 X2
Оперативна пам'ять	256 MB RAM (для Windows® XP) / 1 GB RAM (для Windows Vista/Windows 7/ Windows 8/Windows 10)	2 GB RAM
Відеоадаптер	Intel GMA 950 з відеопам'яттю об'ємом не менше 64 МБ (або сумісний аналог)	
Дисплей	1600x1900	1080x1920 або краще
Прилади введення	Клавіатура, комп'ютерна миша	
Додаткове програмне забезпечення	Модуль pygame	

7 АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ

7.1 Перевірка коректності результатів

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

Критичні ситуації у роботі програми виявлені не були. Під час тестування було виявлено, що більшість помилок виникало тоді, коли користувачем вводилися не числові вхідні дані. Тому всі дані, які вводить користувач, ретельно перевіряються на валідність і лише потім подаються на обробку програмі.

Для перевірки та доведення достовірності результатів виконання програмного забезпечення було власноруч перевірено елементи вихідного масиву, записаного у файл:

а) Метод швидкого сортування.

Результат виконання методу швидкого сортування наведено на рисунку 7.1:

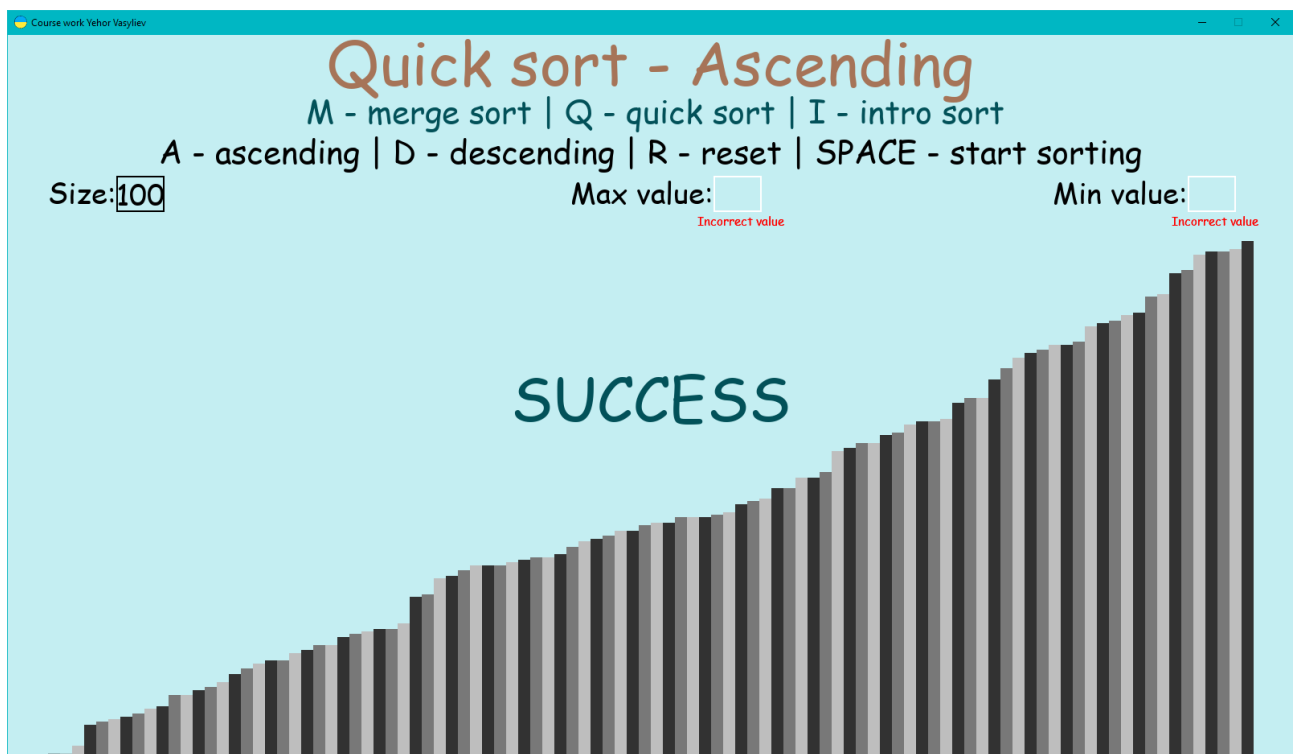


Рисунок 7.1 – Результат виконання методу швидкого сортування

Оскільки результуючий масив дійсно відсортований (рисунок 7.2), то даний метод працює вірно.


```

output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
Sort method: Quick sort
Number of elements: 100
Count of swaps: 308
Count of comparisons: 608
Initial array
-9 31 84 35 48 -88 6 99 -9 -18
-62 55 -39 95 -85 -34 -23 -89 28 -56
98 -64 -41 23 5 -89 -38 -29 57 15
-40 -62 -76 -85 -30 45 32 51 -54 -57
26 9 -63 95 66 -62 42 42 -65 93
43 60 -51 4 -95 35 64 78 -73 35
99 -69 -49 -68 -33 34 19 -39 -15 -66
-25 -12 -72 72 55 -66 39 97 -21 -99
-99 -80 -30 -59 80 -34 -70 -93 -15 57
-20 -60 -26 51 40 15 6 -51 -22 84
Final array
-99 -99 -95 -93 -89 -89 -88 -85 -85 -80
-76 -73 -72 -70 -69 -68 -66 -66 -65 -64
-63 -62 -62 -62 -60 -59 -57 -56 -54 -51
-51 -49 -41 -40 -39 -39 -38 -34 -34 -33
-30 -30 -29 -26 -25 -23 -22 -21 -20 -18
-15 -15 -12 -9 -9 4 5 6 6 9
15 15 19 23 26 28 31 32 34 35
35 35 39 40 42 42 43 45 48 51
51 55 55 57 57 60 64 66 72 78
80 84 84 93 95 95 97 98 99 99
Стр 1, стлб 1  100%  Windows (CRLF)  UTF-8

```

Рисунок 7.2 – Перевірка впорядкованості масиву після його обробки методом швидкого сортування

б) Метод сортування злиттям.

Результат виконання методу сортування злиттям наведено на рисунку 7.3:

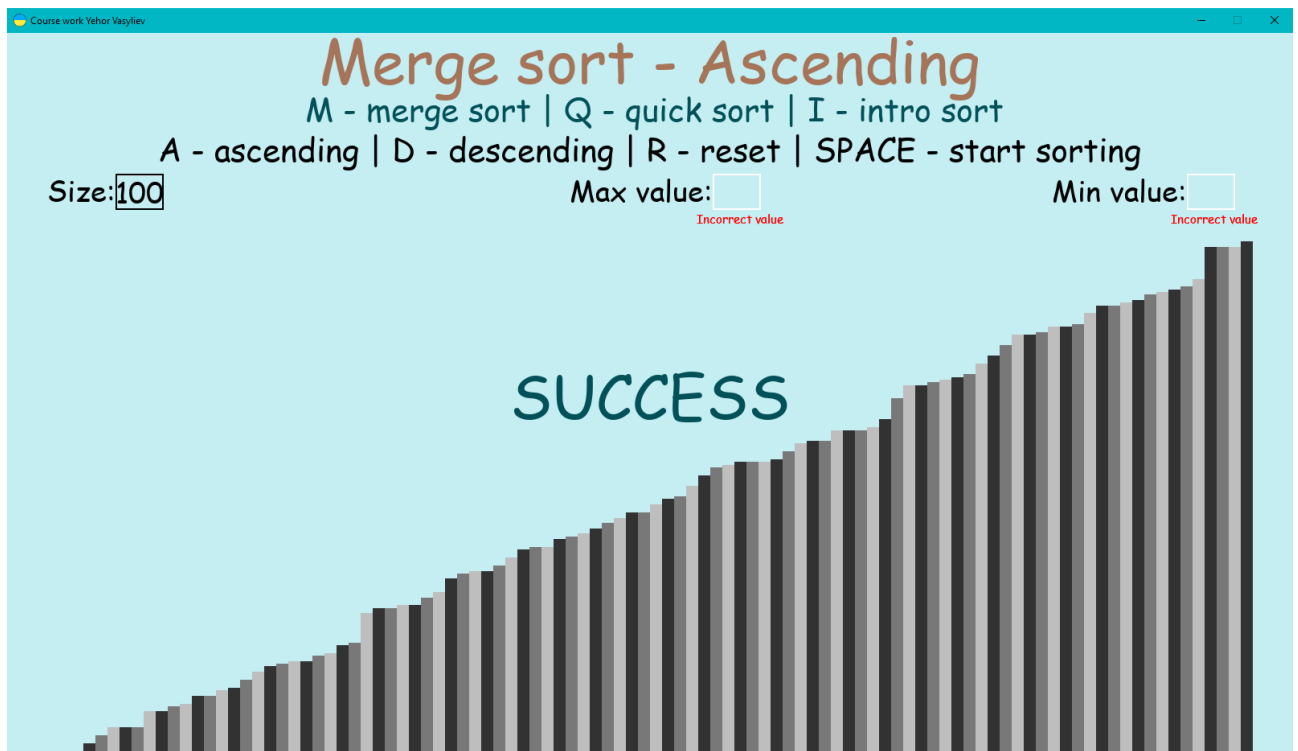
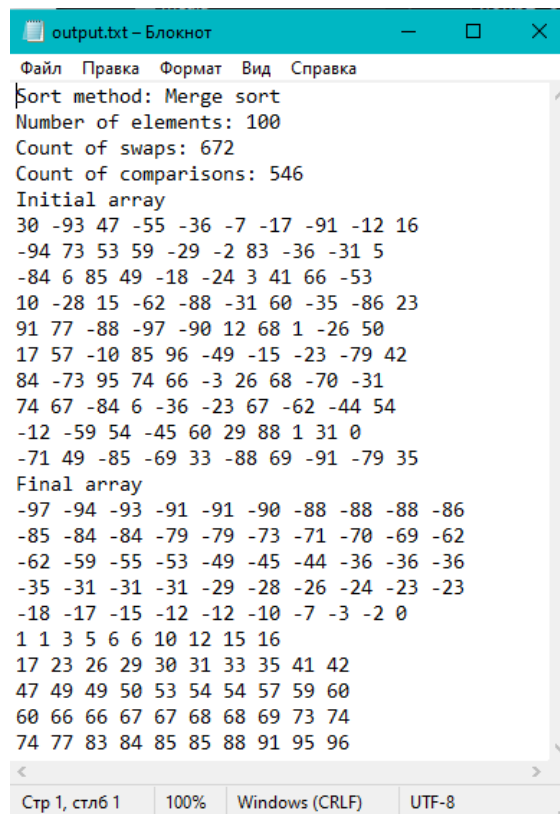


Рисунок 7.3 – Результат виконання методу швидкого сортування

Оскільки результуючий масив дійсно відсортований (рисунок 7.4), то даний метод працює вірно.



```

output.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
Sort method: Merge sort
Number of elements: 100
Count of swaps: 672
Count of comparisons: 546
Initial array
30 -93 47 -55 -36 -7 -17 -91 -12 16
-94 73 53 59 -29 -2 83 -36 -31 5
-84 6 85 49 -18 -24 3 41 66 -53
10 -28 15 -62 -88 -31 60 -35 -86 23
91 77 -88 -97 -90 12 68 1 -26 50
17 57 -10 85 96 -49 -15 -23 -79 42
84 -73 95 74 66 -3 26 68 -70 -31
74 67 -84 6 -36 -23 67 -62 -44 54
-12 -59 54 -45 60 29 88 1 31 0
-71 49 -85 -69 33 -88 69 -91 -79 35
Final array
-97 -94 -93 -91 -91 -90 -88 -88 -88 -86
-85 -84 -84 -79 -79 -73 -71 -70 -69 -62
-62 -59 -55 -53 -49 -45 -44 -36 -36 -36
-35 -31 -31 -31 -29 -28 -26 -24 -23 -23
-18 -17 -15 -12 -12 -10 -7 -3 -2 0
1 1 3 5 6 6 10 12 15 16
17 23 26 29 30 31 33 35 41 42
47 49 49 50 53 54 54 57 59 60
60 66 66 67 67 68 68 69 73 74
74 77 83 84 85 85 88 91 95 96
Стр 1, стлб 1    100%  Windows (CRLF)  UTF-8

```

Рисунок 7.4 – Перевірка впорядкованості масиву після його обробки методом сортування злиттям

в) Метод інтроспективного сортування.

Результат виконання методу інтроспективного сортування наведено на рисунку 7.5:

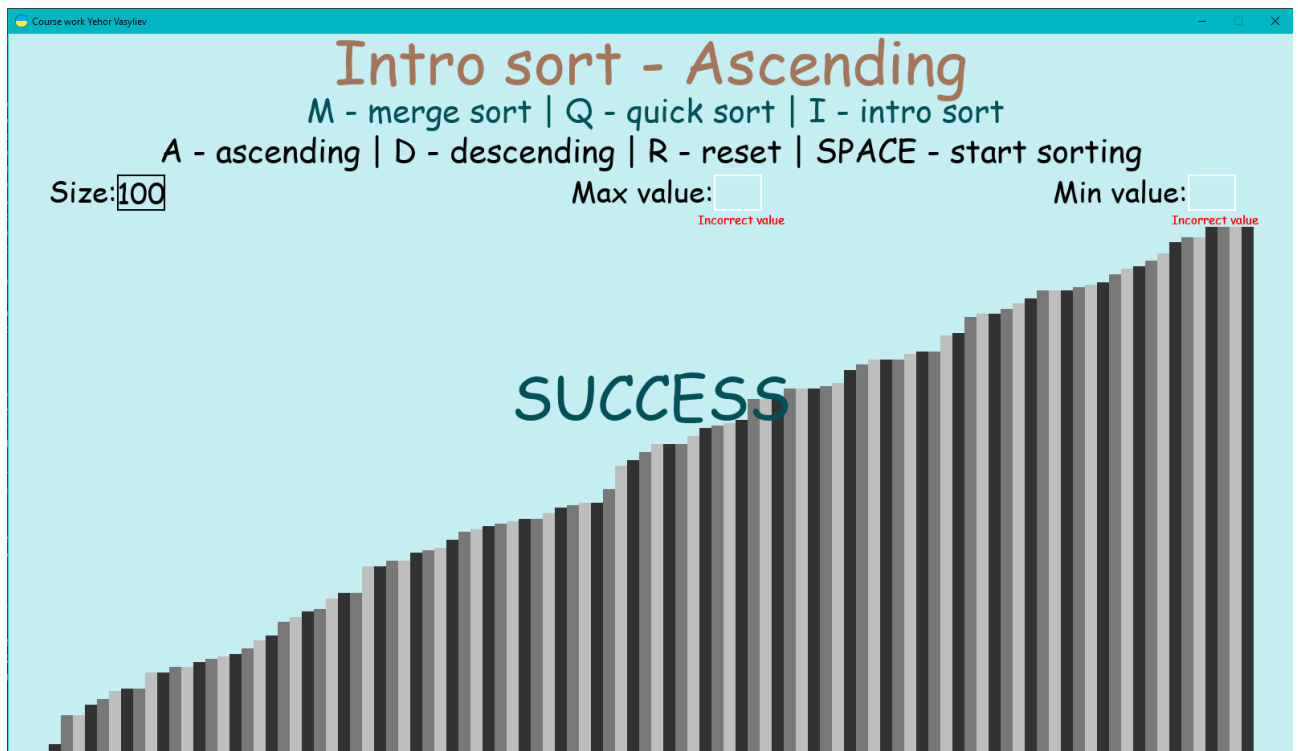
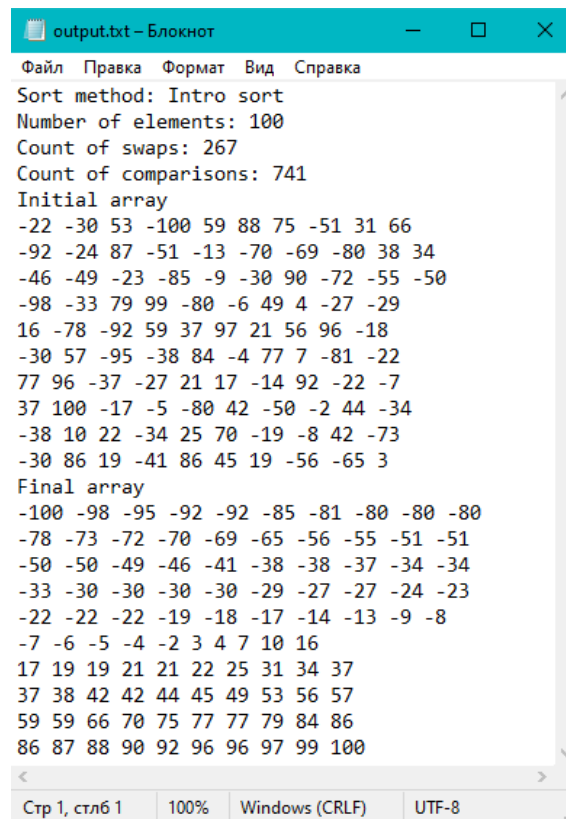


Рисунок 7.5 – Результат виконання методу інтроспективного сортування

Оскільки результуючий масив дійсно відсортований (рисунки 7.6), то даний метод працює вірно.



```

output.txt - Блокнот
Файл  Правка  Формат  Вид  Справка
Sort method: Intro sort
Number of elements: 100
Count of swaps: 267
Count of comparisons: 741
Initial array
-22 -30 53 -100 59 88 75 -51 31 66
-92 -24 87 -51 -13 -70 -69 -80 38 34
-46 -49 -23 -85 -9 -30 90 -72 -55 -50
-98 -33 79 99 -80 -6 49 4 -27 -29
16 -78 -92 59 37 97 21 56 96 -18
-30 57 -95 -38 84 -4 77 7 -81 -22
77 96 -37 -27 21 17 -14 92 -22 -7
37 100 -17 -5 -80 42 -50 -2 44 -34
-38 10 22 -34 25 70 -19 -8 42 -73
-30 86 19 -41 86 45 19 -56 -65 3
Final array
-100 -98 -95 -92 -92 -85 -81 -80 -80 -80
-78 -73 -72 -70 -69 -65 -56 -55 -51 -51
-50 -50 -49 -46 -41 -38 -38 -37 -34 -34
-33 -30 -30 -30 -30 -29 -27 -27 -24 -23
-22 -22 -22 -19 -18 -17 -14 -13 -9 -8
-7 -6 -5 -4 -2 3 4 7 10 16
17 19 19 21 21 22 25 31 34 37
37 38 42 42 44 45 49 53 56 57
59 59 66 70 75 77 77 79 84 86
86 87 88 90 92 96 96 97 99 100
Стр 1, стлб 1  100%  Windows (CRLF)  UTF-8

```

Рисунок 7.6 – Перевірка впорядкованості масиву після його обробки методом інтроспективного сортування

7.2 Тестування ефективності алгоритмів

Для проведення тестування ефективності програми було зроблено 7 сесій сортування масивів різних розмірностей кожним методом з записом необхідних статистичних даних

Результати тестування ефективності алгоритмів сортування масивів наведено в таблиці 7.1:

Таблиця 7.1 – Тестування ефективності методів сортування

Розмірність масиву	Параметри тестування	Метод сортування		
		Швидке	Злиттям	Інтроефективне
100	Кількість порівнянь	582	542	822
	Кількість перестановок	237	-	281
1000	Кількість порівнянь	10949	8719	14530
	Кількість перестановок	5265	-	5052
2500	Кількість порівнянь	38300	25112	33856
	Кількість перестановок	13949	-	12078
5000	Кількість порівнянь	110842	55102	57177
	Кількість перестановок	29747	-	22270
10000	Кількість порівнянь	338066	120341	101136
	Кількість перестановок	49268	-	44197
15000	Кількість порівнянь	696842	189136	171169
	Кількість перестановок	85046	-	68812

Оскільки сортування злиттям фактично ставить на місце елемента масиву елемент з іншого масиву (його лівої чи правої відсортованої частини) і воно не має перестановок як таких, а цифри у вихідному файлі, що відповідають за відповідне поле позначають скільки разів елемент масиву замінювався іншим, то ці дані не відображають справжню ефективність алгоритму і у таблицю занесені не були.

Візуалізація результатів таблиці 7.1 наведено на рисунку 7.7 та 7.8:

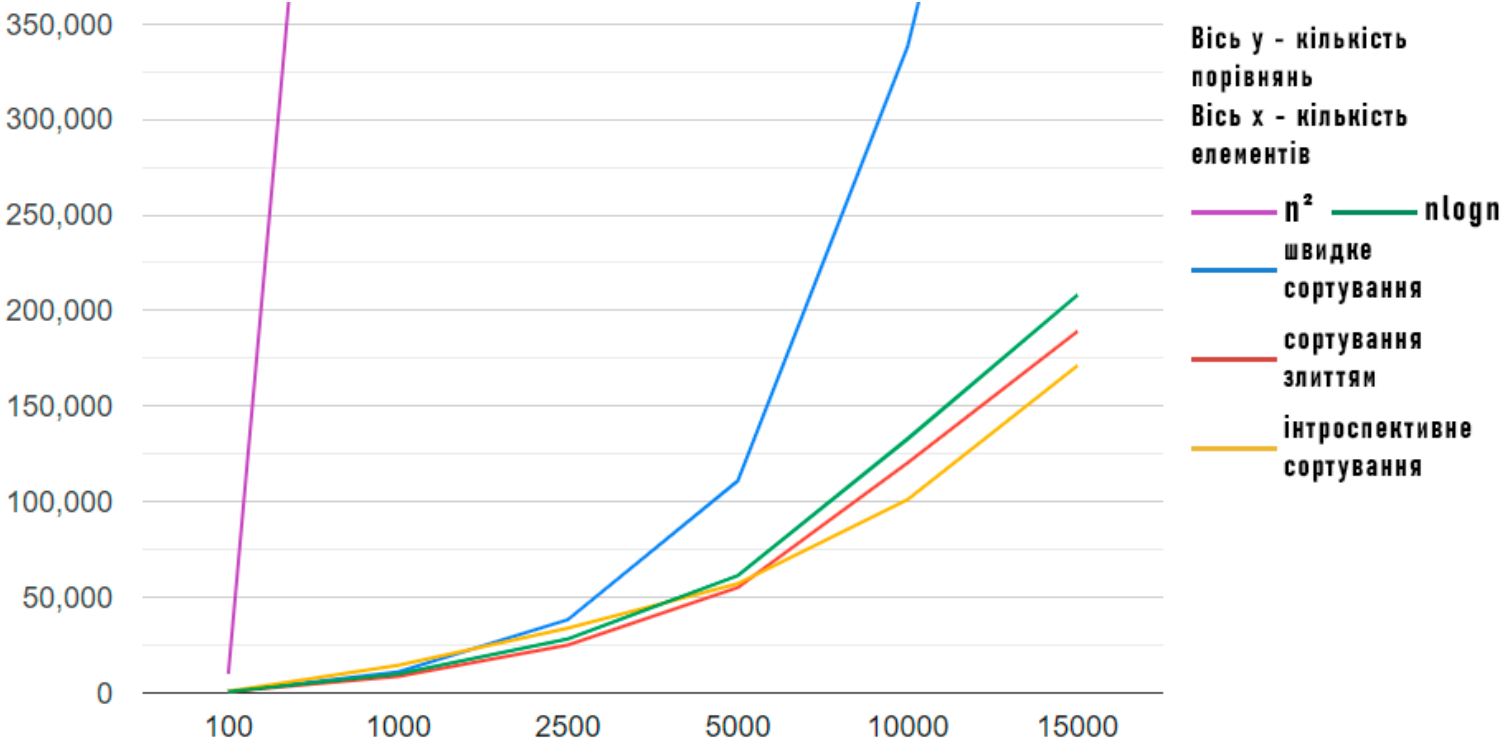


Рисунок 7.7 – Графік залежності кількості порівнянь при сортуванні від розміру

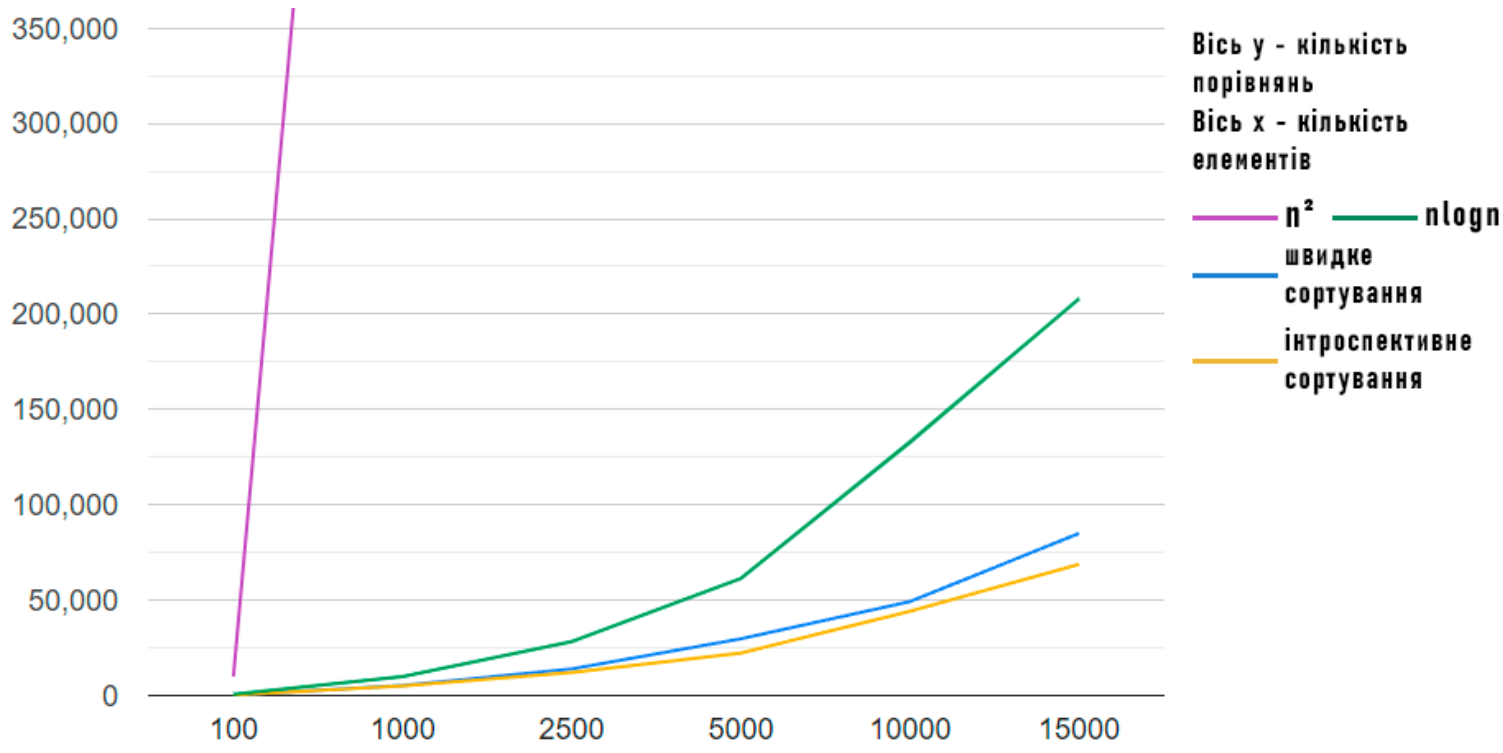


Рисунок 7.8 – Графік залежності кількості перестановок при сортуванні від розміру масиву

7.3 Аналіз часової складності

а) Алгоритм швидкого сортування

Кількість операцій при швидкому сортуванні в загальному випадку можна описати наступним чином: $T(n) = T(k) + T(n-k-1) + \Theta(n)$, де $T(k)$ відповідає за перший рекурсивний виклик з k елементами, менших за опорний, $T(n-k-1)$ – другий рекурсивний виклик з $n-k-1$ елементами, більших за опорний. Розберемо три випадки:

- 1) Найгірший випадок відбувається, коли процес розділення завжди обирає найбільший або найменший елемент як опорний. У моїй стратегії, коли перший елемент завжди вибирається як опорний, найгірша ситуація буде, коли масив вже відсортований у порядку зростання або спадання. В такому випадку кількість операцій можна описати як $T(n) = T(0) + T(n-1) + \Theta(n) = T(n-1) + \Theta(n)$, що еквівалентно часовій складності $O(n^2)$ (для кожного i -го рекурсивного виклику виконується константний час $i + 1$ разів).
- 2) Найкращий випадок відбувається, коли процес розділення завжди обирає середній елемент як опорний. В такому випадку часову кількість операцій можна оцінити як $T(n) = 2T(n/2) + \Theta(n)$, що еквівалентно часовій складності $O(n \log n)$ (за Майстер – методом, формула 2.2).
- 3) Щоб провести аналіз середнього випадку необхідно розглянути всі можливі перестановки масиву та обчислити час, затрачений на кожну перестановку, але уявлення про середній випадок ми можемо отримати розглянувши ситуацію, коли масив кожного разу розбивається на $O(\frac{n}{9})$ та $O(\frac{9n}{10})$ елементів відповідно, тоді кількість операцій можна оцінити як $T(n) = T(\frac{n}{9}) + T(\frac{9n}{10}) + \Theta(n)$, що також еквівалентно часовій складності $O(n \log n)$.

б) Алгоритм сортування злиттям

Алгоритм сортування злиттям завжди ділить масив на дві половини та займає лінійний час на їх об'єднання, тому кількість операцій можна описати наступним чином: $T(n) = 2T(n/2) + \Theta(n)$. З цього випливає, що його часова складність у всіх трьох випадках (найгірший, найкращий, середній) завжди становить $O(n \log n)$ (за Майстер – методом, формула 2.2).

в) Алгоритм інтроспективного сортування

Алгоритм інтроспективного сортування по суті представляє собою покращену версію швидкого сортування, яка перемикається на пірамідальне сортування при занадто великій глибині рекурсії (найгірший випадок швидкого сортування). Враховуючи що пірамідальне сортування має постійну часову складність $O(n \log n)$ ($O(\log n)$ для перебудови купи та $O(n)$ для її створення), а середній та найкращий випадок швидкого сортування також мають часові складності $O(n \log n)$, то і інтроспективне сортування у всіх трьох випадках має часову складність $O(n \log n)$.

За результатами тестування можна зробити такі висновки:

- а) Всі розглянуті методи дозволяють коректно сортувати великі і над великі масиви.
- б) В середньому випадку усі методи мають логарифмічну складність ($O(\log n)$).
- в) З розглянутих методів найоптимальнішим для практичного використання є метод інтроспективного сортування, оскільки він має постійну складність $O(\log n)$, тому кращий за швидке сортування, а на практиці до того ж кращий за сортування злиттям.
- г) Жоден з алгоритмів на практиці (при ініціалізації випадковими числами) не перевищує свою верхню межу часової складності, що видно на графіках (рисунки 7.7, 7.8) однак швидке сортування виявилось найменш ефективним серед розглянутих алгоритмів через неоптимальний вибір опорного елементу.

- д) Незважаючи на найкращу ефективність інтроспективного сортування, питання про доцільність його використання для не надвеликих масивів залишається відкритим, оскільки маючи відносно невелику перевагу над сортуванням методом злиття, його реалізація виявляється набагато складнішою.

ВИСНОВКИ

Під час виконання курсової роботи було закріплено, поглиблено та узагальнено знання, отримані впродовж вивчення ООП та застосовано їх від час розробки програмного забезпечення на тему «Упорядкування масивів». Для розробки даного програмного забезпечення було вивчено необхідні алгоритми сортування масивів, досліджено бібліотеку `pygame` для розробки графічного інтерфейсу та ознайомлено з раніше незаними інструментами мови Python забезпечення візуалізації роботи алгоритмів. Для оформлення пояснювальної записки також було:

- а) розроблено технічне завдання з урахування загальних вимог до курсової роботи;
- б) детально вивчено предметну область поставленого завдання та описано необхідні теоретичні відомості;
- в) розроблено та описано алгоритми заданих методів упорядкування масивів;
- г) створено діаграму класів програмного забезпечення та описано стандартні й користувацькі методи, використані при розробці програмного забезпечення;
- д) розроблено план та виконано тестування програмного забезпечення відповідно до нього в різних умовах та при різних вхідних даних;
- е) розроблено інструкцію використання програмного забезпечення для користувача;
- ж) перевірено правильність роботи розроблених алгоритмів, проаналізовано їх часову складність та зроблено відповідні висновки

ПЕРЕЛІК ПОСИЛАНЬ

1. Вікіпедія: Сортування злиттям. 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%BB%D0%B8%D1%82%D1%82%D1%8F%D0%BC (дата звернення: 15.05.2022).
2. Укладач ст. в. Головченко Максим Миколайович. Алгоритми та структури даних. Київ, 2021.
3. Документація бібліотеки Pygame. URL: <https://www.pygame.org/docs/> (дата звернення: 15.05.2022).

ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра
інформатики та програмної інженерії

Затвердив

Керівник Головченко Максим Миколайович

«08» травня 2022 р.

Виконавець:

Студент Васильєв Єгор Костянтинович

«07» травня 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: Упорядкування масивів

з дисципліни:

«Основи програмування»

Київ 2022

1. *Мета:* Метою курсової роботи є розробка ефективного програмного забезпечення для упорядкування великих масивів різними методами сортування
2. *Дата початку роботи:* «9» березня 2022 р.
3. *Дата закінчення роботи:* «12» травня 2022 р.
4. *Вимоги до програмного забезпечення.*

1) Функціональні вимоги:

- Можливість задавати розмірність масивів в межах від 100 до 50,000 елементів.
- Можливість ініціалізації масиву випадковими значеннями.
- Можливість сортувати масив обраним методом (метод сортування злиттям, метод швидкого сортування, метод інтроспективного сортування).
- Можливість вибору діапазону випадкових значень для ініціалізації.
- Можливість запису результатів сортування у файл.
- Можливість перевірки введених даних на коректність.
- Можливість візуалізації роботи алгоритму для масивів до 500 елементів.
- Можливість збереження статистичних та/або аналітичних даних для подальшого аналізу ефективності алгоритму.

2) Нефункціональні вимоги:

- Можливість запускати програмне забезпечення на операційних системах сімейства Windows 10
- 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 01.05.2022 р.)
- 2) Об'єктно-орієнтоване проектування архітектури програмної системи (до 06.05.2022р.)
- 3) Розробка програмного забезпечення (до 29.05.2022р.)
- 4) Тестування розробленої програми (до 04.06.2022р.)
- 5) Розробка пояснювальної записки (до 09.06.2022 р.).
- 6) Захист курсової роботи (до 16.06.2022 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 завдання упорядкування масивів*

(Найменування програми (документа))

Електронний носій

(Вид носія даних)

27 арк, 30,6 Кб

(Обсяг програми (документа), арк.,

студента групи ПІ-12 І курсу

Васильєва Є.К.

Файл Array_processing.py

```
import pygame
import random
import Config
from Draw_Info import DrawInfo

class ArrayProcessing:
    """Клас для роботи з масивом"""
    SIDE_PAD = 100
    TOP_PAD = 240

    def __init__(self, screen):
        self.__amount_to_vis = Config.MAX_AMOUNT_TO_VIS
        self.__width = DrawInfo.WIDTH
        self.__height = DrawInfo.HEIGHT
        self.__screen = screen

    def generate_list(self, n, min_val, max_val):
        """Генерація масиву, міняє місцями максимальне і
        мінімальне значення в разі необхідності"""
        self.__lst = []
        self.__min_value = min_val
        self.__max_value = max_val
        if self.__max_value < self.__min_value:
            self.__max_value, self.__min_value = self.__min_value,
self.__max_value
        for i in range(n):
            val = random.randint(self.__min_value, self.__max_value)
            self.__lst.append(val)
        self.set_lst()
```

```

def get_lst(self):
    return self.__lst

def set_lst(self):
    """Розрахунок одиничних розмірів комірок"""
    self.__block_width = (self.__width - self.SIDE_PAD) / len(self.__lst)
    self.__block_height = (self.__height - self.TOP_PAD) / (self.__max_value
- self.__min_value)
    self.__start_x = self.SIDE_PAD // 2

def draw_list(self, color_positions={}, clear_bg=False):
    """Виведення стовпців, за умовчанням сірого кольору, якщо функція
викликається із
    функції сортування, то два стовпці що переставляються змінюють
колір на червоний і зелений.
    clear_bg відповідає за необхідність оновлення екрану (при сортуванні
- необхідно, в інших випадках - ні"""
    if clear_bg:
        clear_rect = (self.SIDE_PAD // 2, self.TOP_PAD,
            self.__width - self.SIDE_PAD, self.__height - self.TOP_PAD)
        pygame.draw.rect(self.__screen, DrawInfo.BACKGROUND_COLOR,
clear_rect)
    if len(self.__lst) <= self.__amount_to_vis:
        for i, val in enumerate(self.__lst):
            x = self.__start_x + i * self.__block_width
            y = self.__height - (val - self.__min_value) * self.__block_height
            color = DrawInfo.GRAYSCALE[i % 3]
            if i in color_positions:
                color = color_positions[i]

```



```
pygame.draw.rect(self.__screen, color, (x, y, self.__block_width,  
self.__height - y))
```

```
if clear_bg:
```

```
    pygame.display.update()
```

Файл Config.py

```
from Quick_sort import QuickSort
from Intro_sort import IntroSort
from Merge_sort import MergeSort

n = 150 # початкова кількість елементів
min_v = -100 # межі генерації
max_v = 100 # випадкових чисел
sorting_algorithm = QuickSort # алгоритм сортування за умовчання
path = "output.txt" # шлях до файлу для запису вихідних даних
MAX_AMOUNT_TO_VIS = 500 # максимальна кількість елементів,
# сортування яких буде візуалізуватися
FPS = 75 # кількість оновлень головного циклу в секунду
SOUNDS_ON = True # увімкнення звуків
```

Файл Draw_Info.py

```
import pygame

pygame.init()

class DrawInfo:
    """Клас з константами для малювання та функціями
    виведення на екран головних написів"""
    BLACK_COLOR = 0, 0, 0
    WHITE_COLOR = 255, 255, 255
    RED_COLOR = 255, 0, 0
    GREEN_COLOR = 0, 255, 0
    BLUE_COLOR = 0, 0, 255
    TITLE_TEXT_COLOR = 166, 116, 88
    SUBTITLE_TEXT_COLOR = 2, 81, 89
    ADDITIONAL_TEXT_COLOR = 63, 133, 140
    LIGHT_BLUE = 196, 238, 242
    BACKGROUND_COLOR = LIGHT_BLUE
    GRAYSCALE = [
        (50, 50, 50),
        (120, 120, 120),
        (190, 190, 190)
    ]
    FORM_COLOR_PASSIVE = WHITE_COLOR
    FORM_COLOR_ACTIVE = BLACK_COLOR
    HEIGHT = 900
    WIDTH = 1600
    SMALL_FONT = pygame.font.SysFont('comicsans', 36)
    VERY_SMALL_FONT = pygame.font.SysFont('comicsans', 15)
    FONT = pygame.font.SysFont('comicsans', 40)
```

```
LARGE_FONT = pygame.font.SysFont('comicsans', 75)
```

```
def __init__(self):
```

```
    """Встановлення головного екрану та стовпців"""
```

```
    self.__screen = pygame.display.set_mode((self.WIDTH, self.HEIGHT))
```

```
    pygame.display.set_caption("Course work Yehor Vasyliiev")
```

```
    pygame.display.set_icon(pygame.image.load("media/icon.png"))
```

```
def draw(self, alg_name, ascending):
```

```
    """Виведення головного тексту"""
```

```
    self.__screen.fill(self.BACKGROUND_COLOR)
```

```
    title = self.LARGE_FONT.render(f"{alg_name} - {'Ascending' if  
ascending else 'Descending'}", 1,
```

```
        self.TITLE_TEXT_COLOR)
```

```
    self.__screen.blit(title, ((self.WIDTH - title.get_width()) / 2, -20))
```

```
    sorting = self.FONT.render(" M - merge sort | Q - quick sort | I - intro sort",  
1,
```

```
        self.SUBTITLE_TEXT_COLOR)
```

```
    self.__screen.blit(sorting, ((self.WIDTH - sorting.get_width()) / 2, 65))
```

```
    controls = self.FONT.render("A - ascending | D - descending | R - reset |  
SPACE - start sorting", 1,
```

```
        self.BLACK_COLOR)
```

```
    self.__screen.blit(controls, ((self.WIDTH - controls.get_width()) / 2, 115))
```

```
def get_screen(self):
```

```
    return self.__screen
```

Файл Gen_Secondary.py

```
import pygame
from Draw_Info import DrawInfo

pygame.init()

class SecondaryElements:
    """Клас для обробки другорядних елементів"""
    ALLOWED_BUTTONS = [pygame.K_0, pygame.K_1, pygame.K_2,
                        pygame.K_3, pygame.K_4, pygame.K_5, pygame.K_6, pygame.K_7,
                        pygame.K_8, pygame.K_9, pygame.K_MINUS] # цифри (від 1 до 9
та -) що будуть зчитуватися
    __ERROR_TEXT = "Incorrect value"
    __SUCCESS_TEXT = "SUCCESS"
    __LOWER_RANGE_LIMIT = 50 # допустимі межі розміру
    __UPPER_RANGE_LIMIT = 50000 # масиву згідно з варіантом

    def __init__(self, text, x, y, screen):
        """Оголошення розмірів, координат та кольору другорядного тексту і форм
введення"""
        self.__width = 60
        self.__height = 45
        self.__user_text = ""
        self.__text = text
        self.__text_rect = DrawInfo.SMALL_FONT.render(self.__text, 1,
DrawInfo.BLACK_COLOR)
        self.__color_passive = DrawInfo.FORM_COLOR_PASSIVE
        self.__color_active = DrawInfo.FORM_COLOR_ACTIVE
        self.__text_color = DrawInfo.ADDITIONAL_TEXT_COLOR
        self.__screen = screen
```

```

self.__x = x
self.__y = y
self.__color = self.__color_passive

def get_user_text(self):
    return self.__user_text

def set_user_text(self, text):
    self.__user_text = text

def add_user_text(self, text):
    self.__user_text += text

def output_forms(self):
    """Виведення тексту і форм"""
    users_text_rect = DrawInfo.SMALL_FONT.render(self.__user_text, 1,
DrawInfo.BLACK_COLOR)
    self.__input_rect = pygame.Rect(self.__x + self.__text_rect.get_width(), self.__y
+ 5,
                                max(self.__width, users_text_rect.get_width()),
self.__height)
    pygame.draw.rect(self.__screen, self.__color, self.__input_rect, 2)
    self.__screen.blit(self.__text_rect, (self.__x, self.__y))
    self.__screen.blit(users_text_rect, (self.__input_rect.x, self.__y + 2))

def get_rect_size(self):
    return self.__input_rect

def is_data_correct(self):
    """Перевірка коректності введених даних"""
    if self.__text == "Size:":

```

```

        if not (self.__user_text.isdigit() and int(self.__user_text) in range(
            self.__LOWER_RANGE_LIMIT,
            self.__UPPER_RANGE_LIMIT + 1)):
            return False
        return True
    if self.__text == "Max value:" or self.__text == "Min value:":
        is_digit = lambda x: x.isdigit() if x[:1] != '-' else x[1:].isdigit()
        if not is_digit(self.__user_text):
            return False
        return True
    else:
        return False

    @staticmethod
    def output_success(screen):
        """Виведення повідомлення про успішне сортування"""
        success_text_rect =
        DrawInfo.LARGE_FONT.render(SecondaryElements.__SUCCESS_TEXT, 1,
        DrawInfo.SUBTITLE_TEXT_COLOR)
        screen.blit(success_text_rect, ((DrawInfo.WIDTH -
        success_text_rect.get_width()) / 2,
        (DrawInfo.HEIGHT - success_text_rect.get_height()) / 2))

    @staticmethod
    def draw_error(*boxes):
        """Статичний метод, що оброблює три форми одразу
        і виводить повідомлення у разі їх некоректності"""
        for self in boxes:
            if not self.is_data_correct():

```

```

        error_text_rect =
DrawInfo.VERY_SMALL_FONT.render(self.__ERROR_TEXT,
DrawInfo.RED_COLOR)
        self.__screen.blit(error_text_rect, (self.__input_rect.x - 20, self.__y +
self.__height + 5))
        if boxes[1].__user_text == boxes[2].__user_text:
            error_text_rect =
DrawInfo.VERY_SMALL_FONT.render(boxes[1].__ERROR_TEXT,
DrawInfo.RED_COLOR)
            boxes[1].__screen.blit(error_text_rect, (boxes[1].__input_rect.x - 20,
boxes[1].__y + boxes[1].__height + 5))
            error_text_rect =
DrawInfo.VERY_SMALL_FONT.render(boxes[1].__ERROR_TEXT,
DrawInfo.RED_COLOR)
            boxes[2].__screen.blit(error_text_rect, (boxes[2].__input_rect.x - 20,
boxes[2].__y + boxes[2].__height + 5))

```

```
@staticmethod
```

```
def show(*boxes):
```

```
    """Статичний метод, що виводить одразу три форми"""
```

```
    for box in boxes:
```

```
        box.output_forms()
```

```
@staticmethod
```

```
def set_colors(values, *boxes):
```

```
    """Статичний метод, що приймає список значень які відповідають
за (не) активність форм та відповідно змінює їх кольори"""
```

```
    i = 0
```

```
    for box in boxes:
```

```
        box.__color = box.__color_active if values[i] is True else box.__color_passive
```

```
        i += 1
```



```
@staticmethod
def is_active(*boxes):
    """Статичний метод, що повертає список активності
    форм (завжди активне лише одна)"""
    return [box if box.__color == box.__color_active else False for box in boxes]
```

Файл Intro_sort.py

```

from math import log2, floor
from Draw_Info import DrawInfo
import time
import copy

class IntroSort:
    """Інтроспективне сортування: швидке сортування,
    яке перемикається на пірамідальне при досяжності
    max_depth = floor(log2(len(lst)))"""
    SORTING_ALG_NAME = "Intro sort"

    def __init__(self, lst_control, ascending):
        self.__count_of_comparisons = 0
        self.__count_of_swaps = 0
        self.__lst_control = lst_control
        self.__lst = self.__lst_control.get_lst()
        self.__initial_lst = copy.deepcopy(self.__lst)
        self.__ascending = ascending
        self.__max_depth = floor(log2(len(self.__lst)))

    def sort(self, start, end, max_depth_curr=None):
        if max_depth_curr is None:
            max_depth_curr = self.__max_depth
        if end - start <= 1:
            return
        elif max_depth_curr == 0:
            yield from self.heapsort(start, end)
        else:
            p = self.partition(start, end)

```

```

yield self.__lst
yield from self.sort(start, p + 1, max_depth_curr - 1)
yield self.__lst
yield from self.sort(p + 1, end, max_depth_curr - 1)

```

```
def partition(self, start, end):
```

```

    """Швидке сортування яке повертає індекс елемента,
    який розділяє масив на два: один з елементами
    меншими за опорний, інший з більшими"""
    pivot = self.__lst[start]
    left = start - 1
    right = end

```

```
while True:
```

```

    left += 1
    if self.__ascending:
        while self.__lst[left] < pivot:
            self.__count_of_comparisons += 1
            left += 1

```

```
else:
```

```

    while self.__lst[left] > pivot:
        self.__count_of_comparisons += 1
        left += 1

```

```
right -= 1
```

```
if self.__ascending:
```

```

    while self.__lst[right] > pivot:
        self.__count_of_comparisons += 1
        right -= 1

```

```
else:
```

```

    while self.__lst[right] < pivot:
        self.__count_of_comparisons += 1

```

```
right -= 1
```

```
if left >= right:
```

```
    return right
```

```
self.swap(left, right)
```

```
def swap(self, i, j):
```

```
    """Функція для обміну елементів місцями. Оскільки швидке сортування в
даній реалізації не дозволяє використовувати генератор для повернення
проміжного вигляду масиву, то для забезпечення візуалізації кожного
обміну
```

```
використовується time.sleep(0.01)"""
```

```
self.__count_of_swaps += 1
```

```
self.__lst_control.draw_list({i: DrawInfo.GREEN_COLOR, j:
DrawInfo.RED_COLOR}, True)
```

```
time.sleep(0.01)
```

```
self.__lst[i], self.__lst[j] = self.__lst[j], self.__lst[i]
```

```
def heapsort(self, start, end):
```

```
    """Пірамідальне сортування певної частини масиву"""
```

```
yield from self.build_heap(start, end)
```

```
for i in range(end - 1, start, -1):
```

```
    self.swap(start, i)
```

```
    if self.__ascending:
```

```
        yield from self.max_heapify(i=0, start=start, end=i)
```

```
    else:
```

```
        yield from self.min_heapify(i=0, start=start, end=i)
```

```
def build_heap(self, start, end):
```

```
    """Створення купи"""
```

```
length = end - start
```

```

index = ((length - 1) - 1) // 2 # батько
while index >= 0:
    if self.__ascending:
        yield from self.max_heapify(index, start, end)
    else:
        yield from self.min_heapify(index, start, end)
    index -= 1

def max_heapify(self, i, start, end):
    """Створення купи у якій нащадки не більші за предків"""
    size = end - start
    l = 2 * i + 1
    r = 2 * i + 2
    largest = i
    self.__count_of_comparisons += 2
    if l < size and self.__lst[start + l] > self.__lst[start + i]:
        largest = l
    if r < size and self.__lst[start + r] > self.__lst[start + largest]:
        largest = r
    if largest != i:
        self.swap(start + largest, start + i)
        yield from self.max_heapify(largest, start, end)

def min_heapify(self, i, start, end):
    """Створення купи у якій нащадки не менші за предків"""
    size = end - start
    l = 2 * i + 1
    r = 2 * i + 2
    smallest = i
    self.__count_of_comparisons += 2
    if l < size and self.__lst[start + l] < self.__lst[start + i]:

```

```
        smallest = l
    if r < size and self.__lst[start + r] < self.__lst[start + smallest]:
        smallest = r
    if smallest != i:
        self.swap(start + smallest, start + i)
    yield from self.min_heapify(smallest, start, end)

def get_swaps(self):
    return self.__count_of_swaps

def get_comparisons(self):
    return self.__count_of_comparisons

def get_list(self):
    return self.__lst

def get_initial_list(self):
    return self.__initial_lst
```

Файл Main.py

```
import time
import Config
from Array_processing import ArrayProcessing
from Gen_Secondary import SecondaryElements
from Draw_Info import DrawInfo
from Sound_control import SoundControl
from Intro_sort import IntroSort
from Quick_sort import QuickSort
from Merge_sort import MergeSort
from Write_to_file import write_to_file, clear_file
import pygame

def main():
    pygame.init()
    n = Config.n
    min_v = Config.min_v
    max_v = Config.max_v
    draw_info = DrawInfo()
    running = True
    sorting = False
    ascending = True
    lst_control = ArrayProcessing(draw_info.get_screen())
    lst_control.generate_list(n, min_v, max_v)
    sorting_algorithm = Config.sorting_algorithm
    sorting_alg_name = sorting_algorithm.SORTING_ALG_NAME
    path = Config.path
    clear_file(path)
    sound = SoundControl()
    is_sorted = {'flag': False, 'ascending': True}
```

```

clock = pygame.time.Clock()

box1 = SecondaryElements("Size:", ArrayProcessing.SIDE_PAD / 2, 170,
draw_info.get_screen())

box2 = SecondaryElements("Max value:", DrawInfo.WIDTH / 2 - 100, 170,
draw_info.get_screen())

box3 = SecondaryElements("Min value:", DrawInfo.WIDTH - 3 *
ArrayProcessing.SIDE_PAD, 170, draw_info.get_screen())

while running:
    clock.tick(Config.FPS)
    if sorting:
        try:
            next(sorting_algorithm_generator)
            time.sleep(0.001)
        except StopIteration:
            is_sorted['flag'] = True
            sound.play_sounds("success")
            sorting = False
            sound.stop_sounds("sorting")
            write_to_file(sort_with, path)
        else:
            draw_info.draw(sorting_alg_name, ascending) # обнуляє фон та малює
заголовки
            SecondaryElements.show(box1, box2, box3) # малює прямокутники та
текст до них
            SecondaryElements.draw_error(box1, box2, box3) # залежить від .show
            lst_control.draw_list() # малює стовпці
            if is_sorted['flag'] and is_sorted['ascending'] == ascending:
                SecondaryElements.output_success(draw_info.get_screen()) # напис про
успішне сортування
            pygame.display.update()

```



```

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        running = False
    if event.type == pygame.MOUSEBUTTONDOWN and not sorting:
        if box1.get_rect_size().collidepoint(event.pos):
            SecondaryElements.set_colors([True, False, False], box1, box2,
                                          box3) # встановлює колір на рамку, True- активний
            sound.play_sounds("mouse click")
        elif box2.get_rect_size().collidepoint(event.pos):
            SecondaryElements.set_colors([False, True, False], box1, box2, box3)
            sound.play_sounds("mouse click")
        elif box3.get_rect_size().collidepoint(event.pos):
            SecondaryElements.set_colors([False, False, True], box1, box2, box3)
            sound.play_sounds("mouse click")
        else:
            SecondaryElements.set_colors([False, False, False], box1, box2, box3)
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_r:
            sound.play_sounds("press key")
            is_sorted['flag'] = False
            sound.stop_sounds("sorting")
            sorting = False
            if box1.is_data_correct():
                n = int(box1.get_user_text())
            if box2.is_data_correct():
                max_v = int(box2.get_user_text())
            if box3.is_data_correct() and box3.get_user_text() !=
box2.get_user_text():
                min_v = int(box3.get_user_text())

```

```

lst_control.generate_list(n, min_v, max_v) # аргументи або залишаться
за замовч або ні
elif all([event.key == pygame.K_SPACE, not sorting]):
    if not is_sorted['flag'] or ascending != is_sorted['ascending']:
        sound.play_sounds("press key")
        sorting = True
        is_sorted['flag'] = False # словник для зберігання інформації про
стан масиву
        is_sorted['ascending'] = ascending # для виведення напису про
успішне сортування
        sound.play_sounds("sorting")
        sort_with = sorting_algorithm(lst_control, ascending)
        sorting_algorithm_generator = sort_with.sort(0,
len(lst_control.get_lst()))
elif all([event.key == pygame.K_a, not ascending, not sorting]):
    sound.play_sounds("press key")
    ascending = True
elif all([event.key == pygame.K_d, ascending, not sorting]):
    sound.play_sounds("press key")
    ascending = False
elif all([event.key == pygame.K_q, not sorting]):
    sound.play_sounds("press key")
    sorting_algorithm = QuickSort
    sorting_alg_name = QuickSort.SORTING_ALG_NAME
elif all([event.key == pygame.K_m, not sorting]):
    sound.play_sounds("press key")
    sorting_algorithm = MergeSort
    sorting_alg_name = MergeSort.SORTING_ALG_NAME
elif all([event.key == pygame.K_i, not sorting]):
    sound.play_sounds("press key")
    sorting_algorithm = IntroSort

```

```

        sorting_alg_name = IntroSort.SORTING_ALG_NAME

    elif event.key in SecondaryElements.ALLOWED_BUTTONS and any(
        SecondaryElements.is_active(box1, box2, box3)): # чи є активна
комірка
        self = [x for x in SecondaryElements.is_active(box1, box2, box3) if
            x is not False] # пошук активної комірки
        sound.play_sounds("press key")
        self[0].add_user_text(event.unicode) if len(
            self[0].get_user_text()) < 5 else "" # додавання символу якщо він не
п'ятий і >
        elif event.key == pygame.K_BACKSPACE and
any(SecondaryElements.is_active(box1, box2, box3)):
            self = [x for x in SecondaryElements.is_active(box1, box2, box3) if
                x is not False] # аналогічне видалення символу
            sound.play_sounds("press key")
            self[0].set_user_text(self[0].get_user_text()[:-1])

if __name__ == "__main__":
    main()

```

Файл Merge_sort.py

```
import copy
from Draw_Info import DrawInfo

class MergeSort:
    """Сортування злиттям: формально не має перестановок,
    але можна підраховувати скільки разів поточний елемент
    замінювався на інший з лівої або правої частини
    відсортованого масиву при злитті"""
    SORTING_ALG_NAME = "Merge sort"

    def __init__(self, lst_control, ascending):
        self.__count_of_comparisons = 0
        self.__count_of_swaps = 0
        self.__lst_control = lst_control
        self.__lst = self.__lst_control.get_lst()
        self.__initial_lst = copy.deepcopy(self.__lst)
        self.__ascending = ascending

    def sort(self, start, end):
        """Підсвічується поточний елемент: синім якщо він
        був у правій частині, червоним, якщо у лівій"""
        if end - start > 1:
            middle = (start + end) // 2
            yield from self.sort(start, middle)
            yield from self.sort(middle, end)
            left = self.__lst[start:middle]
            right = self.__lst[middle:end]

            a = 0
```

b = 0

c = start

```

while a < len(left) and b < len(right):
    self.__count_of_comparisons += 1
    if left[a] < right[b] and self.__ascending or left[a] > right[b] and not
self.__ascending:
        self.__count_of_swaps += 1
        self.__lst_control.draw_list({c: DrawInfo.RED_COLOR}, True)
        self.__lst[c] = left[a]
        yield self.__lst
        a += 1
    else:
        self.__lst_control.draw_list({c: DrawInfo.BLUE_COLOR}, True)
        self.__count_of_swaps += 1
        self.__lst[c] = right[b]
        yield self.__lst
        b += 1
    c += 1

while a < len(left):
    self.__count_of_swaps += 1
    self.__lst_control.draw_list({c: DrawInfo.RED_COLOR}, True)
    self.__lst[c] = left[a]
    a += 1
    c += 1

while b < len(right):
    self.__count_of_swaps += 1
    self.__lst_control.draw_list({c: DrawInfo.BLUE_COLOR}, True)
    self.__lst[c] = right[b]

```

```
        b += 1
        c += 1
    yield self.__lst
```

```
def get_swaps(self):
    return self.__count_of_swaps
```

```
def get_comparisons(self):
    return self.__count_of_comparisons
```

```
def get_list(self):
    return self.__lst
```

```
def get_initial_list(self):
    return self.__initial_lst
```

Файл Quick_sort.py

```
import copy
from Draw_Info import DrawInfo

class QuickSort:
    """Швидке сортування, опорним обирається перший елемент масиву"""
    SORTING_ALG_NAME = "Quick sort"

    def __init__(self, lst_control, ascending):
        self.__count_of_comparisons = 0
        self.__count_of_swaps = 0
        self.__lst_control = lst_control
        self.__lst = self.__lst_control.get_lst()
        self.__initial_lst = copy.deepcopy(self.__lst)
        self.__ascending = ascending

    def sort(self, l, r):
        """Червоний - поточний (опорний елемент), синій та зелений - елементи,
        що міняються місцями, причому синій - покажчик що змінюється від
        опорного
        елемента до правого краю розглядувальної ділянки, а зелений - покажчик
        що відповідає кількості елементів менших за опорний"""
        if r == len(self.__lst):
            r -= 1
        if l >= r:
            return
        x = self.__lst[l]
        j = l
        for i in range(l + 1, r + 1):
            self.__count_of_comparisons += 1
```

```

        if self.__lst[i] <= x and self.__ascending or self.__lst[i] >= x and not
self.__ascending:
            j += 1
            if j != i:
                self.__count_of_swaps += 1
                self.__lst[j], self.__lst[i] = self.__lst[i], self.__lst[j]
            self.__lst_control.draw_list(
                {i: DrawInfo.BLUE_COLOR, l: DrawInfo.RED_COLOR, j:
DrawInfo.GREEN_COLOR}, True)
            yield self.__lst
            if l != j:
                self.__count_of_swaps += 1
                self.__lst[l], self.__lst[j] = self.__lst[j], self.__lst[l]
            self.__lst_control.draw_list({l: DrawInfo.RED_COLOR, j:
DrawInfo.GREEN_COLOR}, True)
            yield self.__lst
            yield from self.sort(l, j - 1)
            yield from self.sort(j + 1, r)

def get_swaps(self):
    return self.__count_of_swaps

def get_comparisons(self):
    return self.__count_of_comparisons

def get_list(self):
    return self.__lst

def get_initial_list(self):
    return self.__initial_lst

```


Файл Write_to_file.py

```
import os
```

```
def clear_file(path):
```

```
    try:
```

```
        os.remove(path)
```

```
    except OSError:
```

```
        pass
```

```
def write_to_file(sort_info, path):
```

```
    final_lst = sort_info.get_list()
```

```
    initial_lst = sort_info.get_initial_list()
```

```
    line1 = "Sort method: " + sort_info.SORTING_ALG_NAME + "\n"
```

```
    line2 = "Number of elements: " + str(len(final_lst)) + "\n"
```

```
    line3 = "Count of swaps: " + str(sort_info.get_swaps()) + "\n"
```

```
    line4 = "Count of comparisons: " + str(sort_info.get_comparisons()) + "\n"
```

```
    line5 = "Initial array:\n" + ' '.join(map(str, initial_lst)) + "\n"
```

```
    line6 = "Final array:\n" + ' '.join(map(str, final_lst)) + "\n\n"
```

```
    with open(path, 'at') as text_to_file:
```

```
        text_to_file.write(line1 + line2 + line3 + line4 + line5 + line6)
```