

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»  
(назва дисципліни)  
на тему: Упорядкування масивів

Студента 1-го курсу, групи ІІ-12  
Басараба Олега Андрійовича

Спеціальності 121 «Інженерія програмного забезпечення»

Керівник ст. викладач Головченко М. М.  
(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

Напрямок "ІІЗ"

Курс 1 Група ІІ-12

Семестр 2

## ЗАВДАННЯ

на курсову роботу студента

Басараба Олега Андрійовича

(прізвище, ім'я, по батькові)

1. Тема роботи «Упорядкування масивів»

2. Строк здачі студентом закінченої роботи 12.06.2022

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 10.02.2022

## КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів курсової роботи	Термін виконання етапів роботи	Підписи керівника, студента
1.	Отримання теми курсової роботи	10.02.2022	
2.	Підготовка ТЗ	02.05.2022	
3.	Пошук та вивчення літератури з питань курсової роботи	03.05.2022	
4.	Розробка сценарію роботи програми	04.05.2022	
6.	Узгодження сценарію роботи програми з керівником	04.05.2022	
5.	Розробка (вибір) алгоритму рішення задачі	04.05.2022	
6.	Узгодження алгоритму з керівником	04.05.2022	
7.	Узгодження з керівником інтерфейсу користувача	05.05.2022	
8.	Розробка програмного забезпечення	06.05.2022	
9.	Налагодження розрахункової частини програми	06.05.2022	
10.	Розробка та налагодження інтерфейсної частини програми	07.05.2022	
11.	Узгодження з керівником набору тестів для контрольного прикладу	25.05.2022	
12.	Тестування програми	26.05.2022	
13.	Підготовка пояснювальної записки	05.06.2022	
14.	Здача курсової роботи на перевірку	12.06.2022	
15.	Захист курсової роботи	16.06.2022	

Студент \_\_\_\_\_  
(підпис)

Керівник \_\_\_\_\_  
(підпис)

Головченко Максим Миколайович  
\_\_\_\_\_  
(прізвище, ім'я, по батькові)

"\_\_" \_\_\_\_\_ 20\_\_ р.

## АНОТАЦІЯ

Пояснювальна записка до курсової роботи: 135 сторінок, 42 рисунки, 33 таблиць, 5 посилань.

Об'єкт дослідження: задача впорядкування масивів цілих чисел.

Мета роботи: дослідження алгоритмів сортування масивів, створення програмного забезпечення для сортування масиву введеного користувачем розміру, що складається з випадкових чисел, алгоритмами сортування вибором, пірамідального сортування і плавного сортування.

Вивчено метод розробки програмного забезпечення з використанням об'єктно-орієнтованої парадигми програмування. Приведені змістовні постановки задач, їх індивідуальні математичні моделі, а також описано детальний процес розв'язання кожної з них.

Виконана програмна реалізація алгоритмів сортування вибором, пірамідального сортування, плавного сортування з використанням об'єктно-орієнтованого програмування і створенням графічного інтерфейсу користувача.

АЛГОРИТМИ СОРТУВАННЯ, СОРТУВАННЯ ВИБОРОМ, ПІРАМІДАЛЬНЕ СОРТУВАННЯ, ПЛАВНЕ СОРТУВАННЯ, БІНАРНА КУПА, ЛЕОНАРДОВІ ЧИСЛА, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ

## ЗМІСТ

ВСТУП .....	5
1 ПОСТАНОВКА ЗАДАЧІ .....	7
2 ТЕОРЕТИЧНІ ВІДОМОСТІ .....	8
2.1 Метод сортування вибором .....	8
2.2 Метод пірамідального сортування .....	8
2.3 Метод плавного сортування .....	9
3 ОПИС АЛГОРИТМІВ .....	12
3.1 Загальний алгоритм .....	12
3.2 Алгоритм сортування вибором .....	14
3.3 Алгоритм пірамідального сортування .....	15
3.4 Алгоритм «просійки вниз» для пірамідального сортування .....	16
3.5 Алгоритм плавного сортування .....	17
3.6 Алгоритм пошуку наступного стану послідовності куп на основі поточного та індексу числа Леонардо, що дорівнює розмірності об'єднаних куп .....	18
3.7 Алгоритм «просійки вниз» для плавного сортування .....	19
3.8 Алгоритм знаходження індексу найбільшого кореню серед послідовності куп та індексу відповідного числа Леонардо .....	21
3.9 Алгоритм пошуку попереднього стану купи на основі поточного .....	22
4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .....	23
4.1 Діаграма класів програмного забезпечення .....	23
4.2 Опис методів частин програмного забезпечення .....	23
4.2.1 Стандартні методи .....	23
4.2.2 Користувачські методи .....	25
5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ .....	40
5.1 План тестування .....	40
5.2 Приклади тестування .....	41
6 ІНСТРУКЦІЯ КОРИСТУВАЧА .....	62
6.1 Робота з програмою .....	62
6.2 Формат вхідних та вихідних даних .....	67
6.3 Системні вимоги .....	68
7 АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ .....	69
ВИСНОВКИ .....	86
ПЕРЕЛІК ПОСИЛАНЬ .....	88
ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ .....	89
ДОДАТОК Б ТЕКСТИ ПРОГРАМНОГО КОДУ .....	92

## ВСТУП

Дана курсова робота призначена для вирішення задачі сортування масивів цілих чисел введених користувачем розмірів в порядку неспадання обраним методом.

В розділі «Постановка задачі» буде сформульовано основні вимоги до програмного забезпечення, вхідні дані, необхідні для запуску процесу сортування, вихідні дані, отримані в процесі сортування та необхідні реакції програми на різні дії користувача.

В розділі «Теоретичні відомості» буде описано, що собою являють алгоритми сортування, сформульовано особливості кожного з необхідних для реалізації методів сортування та коротко розібрано основний алгоритм кожного з них.

В розділі «Опис алгоритмів» буде описано загальний алгоритм взаємодії програми з користувачем та буде детально розібрано кожний з необхідних для реалізації алгоритмів сортування.

В розділі «Опис програмного забезпечення» буде наведено UML-діаграму класів ПЗ та описано всі стандартні та користувацькі методи, використані для реалізації програми.

В розділі «Тестування програмного забезпечення» буде передбачено можливі сценарії взаємодії користувача і програми та здійснено тестування кожного зі сценаріїв для доведення роботоспроможності ПЗ в різних ситуаціях.

В розділі «Інструкція користувача» буде описано графічний інтерфейс програми та запропоновано стандартну послідовність дій для взаємодія користувача і програмного забезпечення. Також буде детально описано вхідні і вихідні дані та системні вимоги ПЗ.

В розділі «Аналіз і узагальнення результатів» буде експериментально та аналітично доведено асимптотичну складність кожного з алгоритмів сортування, здійснено перевірку ефективності кожного з алгоритмів під час роботи з масивами, елементи яких впорядковані випадковим чином, масивами, які є близькими до відсортованості, відсортованими масивами та масивами, що відсортовано в зворотному порядку. В кінці розділу буде підсумовано

інформацію про алгоритми, їх ефективність та обрано найбільш працездатний алгоритм.

Ціль даної роботи полягає у розробці програмного забезпечення, яке буде побудовано на об'єктно орієнтованій парадигмі, і ефективно виконуватиме поставлену задачу.

## 1 ПОСТАНОВКА ЗАДАЧІ

Розробити програмне забезпечення, що буде сортувати масив, що складається з цілих чисел, згенерованих випадковим чином, у неспадному порядку наступними методами:

- а) метод сортування вибором;
- б) метод пірамідального сортування;
- в) метод плавного сортування;

Вхідними даними для даної роботи є розмірність масиву та масив відповідного розміру, що складається з цілих чисел, згенерованих випадковим чином після натискання відповідної кнопки. Програмне забезпечення повинне обробляти розмірність масиву в межах від 100 до 50000 елементів. Програмне забезпечення має блокувати для користувача можливість вводити будь-які нецілочисельні значення в поле вводу розмірності масиву. Якщо користувач вводить цілочисельне значення менше 100 або більше 50000, програмне забезпечення повинне змінювати значення поля вводу на 100 або 50000 відповідно. Програмне забезпечення має генерувати масив, елементами якого є випадкові цілі числа в межах від 0 до 50000 після натискання користувачем відповідної кнопки.

Вихідними даними для даної роботи являється масив цілих чисел, відсортованих у неспадному порядку, що записується в текстовий файл після завершення сортування. В текстовий файл також повинні заноситись аналітичні дані, отримані в процесі сортування. Масив до сортування повинен бути записаний в окремий файл. Програмне забезпечення повинне відсортовувати масив за умови, що задана розмірність масиву лежить в межах від 100 до 50000 елементів. Якщо розмірність масиву не перевищує 200 елементів і користувач обрав відповідну опцію, то програмне забезпечення повинне супроводжувати процес сортування візуалізацією у вигляді рухомих стовпців гістограми. Якщо записати масив до сортування чи масив після сортування до текстових файлів не вдалося, програмне забезпечення повинне виводити відповідне повідомлення.



## 2 ТЕОРЕТИЧНІ ВІДОМОСТІ

Алгоритм сортування – це алгоритм, що впорядковує множину однотипних елементів за певною ознакою (ключем сортування). Сортування є типовою задачею обробки даних, що забезпечує розміщення елементів невідсортованого набору значень в порядку монотонного зростання або спадання значення ключа [1].

Опис алгоритмів сортування вибором, пірамідально сортування та плавного сортування буде наведено з врахуванням того, що сортуванню підлягає масив цілих чисел, сортування здійснюється в неспадному порядку та ключем сортування для кожного елементу масиву є його цілочисельне значення.

### 2.1 Метод сортування вибором

Сутність алгоритму сортування вибором (Selection sort) полягає у багаторазовому виконанні операції знаходження елементу з найменшим значенням у невідсортованій частині масиву та перестановці знайденого елементу з першим елементом невідсортованої частини масиву. Після цього перший елемент невідсортованої частини масиву вважається елементом відсортованої частини. На початку сортування увесь масив є невідсортованим. З кожною ітерацією кількість елементів у невідсортованій частині масиву зменшується на одиницю. Процес сортування продовжується доти, поки в невідсортованій частині масиву не залишиться елементів [2].

Часова складність алгоритму –  $O(n^2)$  [3, с. 159].

Ємнісна складність алгоритму –  $O(n)$  всього,  $O(1)$  – допоміжна змінна [3, с. 159].

### 2.2 Метод пірамідального сортування

Пірамідальне сортування (Heap sort, «Сортування купою») – це алгоритм сортування, що базується на порівняннях та використовує структуру даних бінарна купа [4].

Повне бінарне дерево – це бінарне дерево, в якому кожний рівень, за винятком останнього, є повністю заповненим і в якому всі вершини знаходяться якомога лівіше [4].

Бінарна купа – це повне бінарне дерево, елементи якого зберігаються в такому порядку, що ключ батьківської вершини є неменшим (небільшим) за ключі його дітей [4].

Для репрезентації купи в даному алгоритмі буде використовуватися масив `array`, в якому `array[0]` – елемент в корені, а нащадками елементу `array[i]` є `array[2 · i + 1]` та `array[2 · i + 2]` [3, с. 173].

Алгоритм сортування буде складатися з двох основних послідовних етапів [3, с. 173]:

- а) Побудови бінарної купи (дерева сортування) таким чином, щоб для всіх елементів масиву `array` виконувалася умова `array[i] > array[2 · i + 1]` та `array[i] > array[2 · i + 2]` для  $i \in [0; n/2)$ , де  $n$  – розмірність масиву `array`;
- б) Видалення елементів масиву з кореня бінарної купи та перебудови дерева таким чином, щоб воно продовжило залишатися бінарною купою. Тобто для всіх цілих  $i \in [1; n - 2]$  буде здійснюватися обмін `array[0]` з `array[n - i - 1]` та перебудова елементів масиву з індексами із проміжку  $[0; n - i - 2]$  в бінарну купу. Процес триватиме до тих пір, поки в бінарній купі не залишиться всього один елемент.

Часова складність алгоритму –  $O(n \log n)$  [3, с. 174].

Ємнісна складність алгоритму –  $O(n)$  всього,  $O(1)$  – допоміжна змінна [3, с. 174].

### 2.3 Метод плавного сортування

Плавне сортування (Smooth sort) – алгоритм сортування, що базується на порівняннях та є різновидом пірамідального сортування, розроблений Е. Дейкстрою в 1981 році. Як і пірамідальне сортування, має складність в гіршому випадку  $O(n \log n)$ , однак, в плавного сортування є перевага в тому, що при частково відсортованих вхідних даних, його складність наближається до  $O(n)$ , поки в пірамідально сортування складність не залежить від вхідних даних [3, с. 175].

Даний алгоритм використовує так звані числа Леонарда, що обчислюються таким чином [3, с. 175]:

$$\begin{aligned} L(0) &= 1 \\ L(1) &= 1 \\ L(n) &= L(n-1) + L(n-2) + 1 \end{aligned} \quad (2.1)$$

Загальні положення [3, с. 175-176]:

- а) Масив ділиться на групу підмасивів. Кожен підмасив є структурою даних купа. Кожна купа має розмір, що дорівнює одному з чисел Леонардо. При цьому ліві елементи масиву є вузлами найбільш можливої купи. Решта елементів масиву розбиваються на купи за таким же жадібним правилом. Надалі цю групу підмасивів будемо називати послідовність куп. При цьому, розміри куп в цій послідовності будуть знаходитися в невисхідному порядку;
- б) Не існує двох куп, що мають однаковий розмір. Це твердження можна довести так: числа Леонардо ростуть повільніше функції  $2^n$ , тому для будь-якого масиву можна знайти таке число Леонардо, яке буде більше середини. Винятком є купи розмірністю 1;
- в) Ніякі дві купи не матимуть розмір рівним двом послідовним числам Леонардо. Винятком можуть бути тільки дві останні купи. Якщо ми будемо використовувати поспіль дві купи розмірністю  $L(x)$  і  $L(x+1)$ , то їх можна буде замінити однією з розмірністю  $L(x+2)$ .

Наслідки загальних положень [3, с. 176]:

- а) Коренем кожної купи з послідовності куп є елемент масиву з максимальним ключем;
- б) Пошук найбільшого елементу масиву з послідовності куп потрібно здійснювати серед кореневих елементів кожної купи.

Алгоритм плавного сортування буде складатися з двох основних послідовних етапів [3, с. 176-177]:

- а) Формування послідовності куп. На початку сортування послідовність куп є порожньою. Необхідно послідовно додати до неї всі елементи

вихідного масиву зліва на право. Елемент, що додається, може стати або вершиною нової купи, або вершиною купи, що об'єднає найбільш праві купи в послідовності куп.

- б) Для формування відсортованого масиву необхідно на кожній ітерації визначати поточний максимум послідовності куп серед коренів куп і обмінювати знайдений елемент з останнім елементом масиву, виключаючи його з послідовності куп. Цю дію потрібно повторювати, поки послідовність куп не залишиться порожньою. При об'єднанні двох куп або при обміні двох кореневих елементів, щоб гарантувати збереження властивостей бінарної купи, необхідно здійснити «просіювання вниз», якщо новий кореневий елемент не відповідає властивостям бінарної купи.

Для репрезентації розмірностей куп в послідовності куп можна використати цілочисельну беззнакову змінну *state*, двійкове представлення якої слугуватиме для того, щоб позначати, чи наявна купа розміру  $L(x)$  в послідовності куп. Кожен розряд двійкового представлення *state* може набувати значення «1» або «0», тобто купа розмірності  $L(x)$  або представлена в послідовності куп, або ні відповідно. Найбільш ліва купа є найбільшою, а найбільш права – найменшою [5].

### 3 ОПИС АЛГОРИТМІВ

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

Таблиця 3.1 – Основні змінні загального алгоритму та їхні призначення

Змінна	Призначення
arr	Масив цілих чисел, який потрібно відсортувати обраним методом
arrSize	Розмірність масиву arr; вводиться користувачем; повинна знаходитися в межах від 100 до 50000
maxVal	Найбільше значення, якого може набувати елемент масиву arr

#### 3.1 Загальний алгоритм

##### 1. ПОЧАТОК

##### 2. Зчитати розмірність масиву:

2.1. ЯКЩО користувач намагається ввести нецілочисельне значення, ТО:

2.1.1. Заблокувати поле для введення розмірності масиву.

##### 2.2. ІНАКШЕ:

2.2.1. ЯКЩО введенне значення  $\in [100; 50000]$ , ТО:

2.2.1.1. Записати введенне значення в поле для введення розмірності масиву.

2.2.1.2. Записати зчитане значення в arrSize.

2.2.2. ЯКЩО зчитане значення  $< 100$ , ТО:

2.2.2.1. Записати значення 100 в поле для введення розмірності масиву.

2.2.2.2. Записати 100 в arrSize.

2.2.3. ЯКЩО зчитане значення  $> 50000$ , ТО:

2.2.3.1. Записати значення 50000 в поле для введення розмірності масиву.

2.2.3.2. Записати 50000 в arrSize.

3. Згенерувати масив випадкових цілих чисел розміру `arrSize`:
  - 3.1. ЦИКЛ проходу по всіх елементах масиву `arr` (`arri` – поточний елемент):
    - 3.1.1. Записати в `arri` випадкове ціле число  $\in [0; \text{maxVal}]$ .
  - 3.2. Вивести повідомлення про успішну генерацію масиву `arr` розміру `arrSize`.
4. Записати масив `arr` та розмір масиву `arrSize` в текстовий файл:
  - 4.1. ЯКЩО записати масив в текстовий файл не вдалося, ТО:
    - 4.1.1. Вивести відповідне повідомлення.
5. Відсортувати масив `arr` обраним методом:
  - 5.1. ЯКЩО користувач обрав метод сортування вибором, ТО:
    - 5.1.1. ЯКЩО  $\text{arrSize} \leq 200$  та користувач обрав опцію візуалізації процесу сортування, ТО:
      - 5.1.1.1. Візуалізувати процес сортування елементів масиву `arr`.
    - 5.1.2. Відсортувати елементи масиву `arr` згідно алгоритму пірамідального сортування (пункт 3.2).
  - 5.2. ЯКЩО користувач обрав метод пірамідального сортування, ТО:
    - 5.2.1. ЯКЩО  $\text{arrSize} \leq 200$  та користувач обрав опцію візуалізації процесу сортування, ТО:
      - 5.2.1.1. Візуалізувати процес сортування елементів масиву `arr`.
    - 5.2.2. Відсортувати елементи масиву `arr` згідно алгоритму сортування вибором (пункт 3.3).
  - 5.3. ЯКЩО користувач обрав метод плавного сортування, ТО:
    - 5.3.1. ЯКЩО  $\text{arrSize} \leq 200$  та користувач обрав опцію візуалізації процесу сортування, ТО:
      - 5.3.1.1. Візуалізувати процес сортування елементів масиву `arr`.
    - 5.3.2. Відсортувати елементи масиву `arr` згідно алгоритму плавного сортування (пункт 3.5).

5.4. Вивести повідомлення про успішне сортування масиву `arr` розміру `arrSize`, яке міститиме кількість порівнянь та кількість перестановок, здійснених в процесі сортування.

6. Записати відсортований масив `arr`, розмір масиву `arrSize`, обраний метод сортування, кількість порівнянь та кількість перестановок, здійснених в процесі сортування, у текстовий файл:

6.1. ЯКЩО записати масив в текстовий файл не вдалося, ТО:

6.1.1. Вивести відповідне повідомлення.

## 7. КІНЕЦЬ

Перелік всіх основних змінних алгоритму сортування вибором та їхнє призначення наведено в таблиці 3.2.

Таблиця 3.2 – Основні змінні алгоритму сортування вибором та їхні призначення

Змінна	Призначення
<code>arr</code>	Масив цілих чисел, який потрібно відсортувати алгоритмом сортування вибором
<code>arrSize</code>	Розмірність масиву <code>arr</code>
<code>currMinIdx</code>	Поточний індекс мінімального елемента масиву <code>arr</code> з заданого проміжку

## 3.2 Алгоритм сортування вибором

### 1. ПОЧАТОК

2. ЦИКЛ проходу по  $i \in [0; arrSize - 1]$  з кроком 1:

2.1. Знаходження індексу поточного значення `currMinIdx`:

2.1.1. `currMinIdx := i`

2.1.2. ЦИКЛ проходу по  $j \in [i + 1; arrSize - 1]$  з кроком 1:

2.1.2.1. ЯКЩО `arrcurrMinIdx > arrj`, ТО:

2.1.2.1.1. `currMinIdx := j`

2.2. Поміняти місцями `arrcurrMinIdx` та `arri`

### 3. КІНЕЦЬ

Перелік всіх основних змінних алгоритму сортування вибором та їхнє призначення наведено в таблиці 3.3.

Таблиця 3.3 – Основні змінні алгоритму пірамідального сортування та їхні призначення

Змінна	Призначення
arr	Масив цілих чисел, який потрібно відсортувати алгоритмом пірамідального сортування
arrSize	Розмірність масиву arr
currMinIdx	Поточний індекс мінімального елементу масиву arr з заданого проміжку

### 3.3 Алгоритм пірамідального сортування

#### 1. ПОЧАТОК

2. ЦИКЛ проходу по  $i$  від  $\left\lfloor \frac{\text{arrSize}}{2} \right\rfloor - 1$  до 0 з кроком  $-1$ :

2.1. Здійснити «просійку вниз» для алгоритму пірамідального сортування для купи розміром arrSize з кореневим елементом  $i$  (пункт 3.4).

3. ЦИКЛ проходу по  $i$  від arrSize  $- 1$  до 0 з кроком  $-1$ :

3.1. Поміняти місцями  $\text{arr}_i$  та  $\text{arr}_0$

3.2. Здійснити «просійку вниз» для алгоритму пірамідального сортування для купи розміром  $i$  з кореневим елементом 0 (пункт 3.4).

#### 4. КІНЕЦЬ

Перелік всіх основних змінних алгоритму «просійки вниз» для алгоритму пірамідального сортування та їхнє призначення наведено в таблиці 3.4.

Таблиця 3.4 – Основні змінні алгоритму «просійки вниз» для алгоритму пірамідального сортування та їхні призначення

Змінна	Призначення
arr	Масив цілих чисел, який представляє собою купу
rootIdx	Індекс кореневого елементу купи, яку представляє масив arr



## Продовження таблиці 3.4

Змінна	Призначення
maxIdx	Поточний індекс максимального елементу
heapSize	Розмір купи

## 3.4 Алгоритм «просійки вниз» для пірамідального сортування

## 1. ПОЧАТОК

2. maxIdx := rootIdx

3. ПОКИ heapSize &gt; rootIdx:

3.1. **leftChildIdx** :=  $2 \cdot \text{rootIdx} + 1$ 3.2. ЯКЩО leftChildIdx < heapSize та  $\text{arr}_{\text{leftChildIdx}} > \text{arr}_{\text{maxIdx}}$ , ТО:

3.2.1. maxIdx := leftChildIdx

3.3. **rightChildIdx** :=  $2 \cdot \text{rootIdx} + 2$ 3.4. ЯКЩО rightChildIdx < heapSize та  $\text{arr}_{\text{rightChildIdx}} > \text{arr}_{\text{maxIdx}}$ ,  
ТО:

3.4.1. maxIdx := rightChildIdx

3.5. ЯКЩО maxIdx ≠ rootIdx, ТО:

3.5.1. Поміняти місцями  $\text{arr}_{\text{maxIdx}}$  та  $\text{arr}_{\text{rootIdx}}$ 

3.5.2. rootIdx := maxIdx

3.6. ІНАКШЕ:

3.6.1. break

## 4. КІНЕЦЬ

Перелік всіх основних змінних алгоритму плавного сортування та їхнє призначення наведено в таблиці 3.5.

Таблиця 3.5 – Основні змінні алгоритму плавного сортування та їхні призначення

Змінна	Призначення
arr	Масив цілих чисел, який потрібно відсортувати алгоритмом плавного сортування
leoNums	Вектор чисел Леонардо, необхідних для сортування
arrSize	Розмір масиву

## Продовження таблиці 3.5

Змінна	Призначення
currState	Поточний стан послідовності куп

## 3.5 Алгоритм плавного сортування

1. ПОЧАТОК
2. Обчислення достатньої кількості чисел Леонардо:
  - 2.1.  $leo1 := 1$
  - 2.2.  $leo2 := 1$
  - 2.3. ПОКИ  $leo1 \leq arrSize$ :
    - 2.3.1. Додати  $leo1$  в кінець вектору  $leoNums$ .
    - 2.3.2.  $temp := leo1$
    - 2.3.3.  $leo1 := leo2$
    - 2.3.4.  $leo2 := leo2 + temp + 1$
3.  $currState := 0$
4. ЦИКЛ проходу по  $i \in [0; arrSize - 1]$  з кроком 1:
  - 4.1.  $currHeapLeoNumIdx := -1$
  - 4.2. Знаходження наступного стану на основі  $currState$  та знаходження поточного значення  $currHeapLeoNumIdx$  (пункт 3.6).
  - 4.3. ЯКЩО  $currHeapLeoNumIdx \neq -1$ , ТО:
    - 4.3.1. Здійснити «просійку вниз» для алгоритму плавного сортування для значень  $currHeapLeoNumIdx$  та  $i$  (пункт 3.7).
5. ЦИКЛ проходу по  $i$  від  $arrSize - 1$  до 0 з кроком  $-1$ :
  - 5.1.  $heapLeoNumIdx := -1$
  - 5.2.  $maxRootIdx := -1$
  - 5.3. Знайти значення  $maxRootIdx$  та  $heapLeoNumIdx$  за допомогою алгоритму знаходження найбільшого кореню послідовності куп (пункт 3.8).
  - 5.4. ЯКЩО  $maxRootIdx \neq i$ , ТО
    - 5.4.1. Поміняти місцями  $arr_{maxRootIdx}$  та  $arr_i$

5.4.2. Здійснити «просійку вниз» для алгоритму плавного сортування для значень `heapLeoNumIdx` та `maxRootIdx` (пункт 3.7)

5.5. Знаходження попереднього стану на основі `currState` (пункт 3.9).

## 6. КІНЕЦЬ

Перелік всіх основних змінних алгоритму пошуку наступного стану послідовності куп на основі поточного та індексу числа Леонардо, що дорівнює розмірності об'єднаних куп, та їхнє призначення наведено в таблиці 3.6.

Таблиця 3.6 – Основні змінні алгоритму пошуку наступного стану послідовності куп на основі поточного та індексу числа Леонардо, що дорівнює розмірності об'єднаних куп, та їхні призначення

Змінна	Призначення
<code>currState</code>	Поточний стан послідовності куп
<code>mergedHeapsLeoNumIdx</code>	Індекс числа Леонардо, яке дорівнює розміру купи, що була отримана в результаті об'єднання двох куп під час переходу до нового стану
<code>firstTwoSideBySideBits</code>	Змінна, що буде використана для пошуку двох послідовних одиничних біт
<code>idx</code>	Змінна для обчислення положення двох послідовних одиничних біт в двійковому представленні <code>firstTwoSideBySideBits</code>

3.6 Алгоритм пошуку наступного стану послідовності куп на основі поточного та індексу числа Леонардо, що дорівнює розмірності об'єднаних куп

### 1. ПОЧАТОК

2. `mergedHeapsLeoNumIdx` := −1

3. ЯКЩО (`currState` & 7) == 5, ТО:

3.1. `currState` := `currState` + 3

3.2. `mergedHeapsLeoNumIdx` := 3

4. ІНАКШЕ:

4.1. `firstTwoSideBySideBits` := `currState`

4.2.  $\text{idx} := 0$

4.3. ПОКИ

$(\text{firstTwoSideBySideBits} \&\& (\text{firstTwoSideBySideBits} \& 3) \neq 3)$ :

4.3.1.  $\text{firstTwoSideBySideBits} := \text{firstTwoSideBySideBits} \gg 1$

4.3.2.  $\text{idx} := \text{idx} + 1$

4.4. ЯКЩО  $(\text{firstTwoSideBySideBits} \& 3) == 3$ , ТО:

4.4.1.  $\text{currState} := \text{currState} + 1 \ll \text{idx}$

4.4.2.  $\text{mergedHeapsLeoNumIdx} := \text{idx} + 2$

4.5. ІНАКШЕ:

4.5.1. ЯКЩО  $\text{currState} \& 1$ , ТО:

4.5.1.1.  $\text{currState} := \text{currState} | 2$

4.5.2. ІНАКШЕ:

4.5.2.1.  $\text{currState} := \text{currState} | 1$

5. КІНЕЦЬ

Перелік всіх основних змінних алгоритму «просійки вниз» для плавного сортування та їхнє призначення наведено в таблиці 3.7.

Таблиця 3.7 – Основні змінні алгоритму «просійки вниз» для плавного сортування та їхні призначення

Змінна	Призначення
$\text{rootIdx}$	Індекс кореню поточної купи
$\text{heapLeoNumIdx}$	Індекс числа Леонардо, яке дорівнює розміру купи, над якою проводиться «просійка вниз»
$\text{maxIdx}$	Індекс найбільшого поточного вузла
$\text{arr}$	Масив, що представляє послідовність куп
$\text{leoNums}$	Вектор чисел Леонардо, необхідних для сортування

3.7 Алгоритм «просійки вниз» для плавного сортування

1. ПОЧАТОК

2.  $\text{currNodeIdx} := \text{rootIdx}$

3. ПОКИ  $\text{heapLeoNumIdx} \geq 2$ :

3.1.  $\text{maxIdx} := \text{currNodeIdx}$

- 3.2.  $\text{prevHeapLeoNumIdx} := \text{heapLeoNumIdx}$
- 3.3.  $\text{rightChildIdx} := \text{currNodeIdx} - 1$
- 3.4. ЯКЩО  $\text{arr}_{\text{rightChildIdx}} > \text{arr}_{\text{maxIdx}}$ , ТО:
  - 3.4.1.  $\text{prevHeapLeoNumIdx} := \text{heapLeoNumIdx} - 2$
  - 3.4.2.  $\text{maxIdx} := \text{rightChildIdx}$
- 3.5.  $\text{leftChildIdx} := \text{currNodeIdx} - 1$
- 3.6. ЯКЩО  $\text{arr}_{\text{rightChildIdx}} > \text{arr}_{\text{maxIdx}}$ , ТО:
  - 3.6.1.  $\text{prevHeapLeoNumIdx} := \text{heapLeoNumIdx} - 2$
  - 3.6.2.  $\text{maxIdx} := \text{rightChildIdx} - \text{leoNums}_{\text{heapLeoNumIdx}-2}$
- 3.7. ЯКЩО  $\text{currNodeIdx} \neq \text{maxIdx}$ , ТО:
  - 3.7.1. Поміняти місцями  $\text{arr}_{\text{maxIdx}}$  та  $\text{arr}_{\text{currNodeIdx}}$
  - 3.7.2.  $\text{heapLeoNumIdx} := \text{prevHeapLeoNumIdx}$
  - 3.7.3.  $\text{currNodeIdx} := \text{maxIdx}$
- 3.8. ІНАКШЕ:
  - 3.8.1. break

#### 4. КІНЕЦЬ

Перелік всіх основних змінних алгоритму знаходження індексу найбільшого кореню серед послідовності куп та індексу відповідного числа Леонардо та їхнє призначення наведено в таблиці 3.8.

Таблиця 3.8 – Основні змінні алгоритму знаходження індексу найбільшого кореню серед послідовності куп та індексу відповідного числа Леонардо та їхні призначення

Змінна	Призначення
$\text{currState}$	Поточний стан послідовності куп
$\text{lastRootIdx}$	Індекс кореню останньої купи в послідовності куп
$\text{heapLeoNumIdx}$	Індекс числа Леонардо, яке дорівнює розміру купи, якій належить найбільший корінь
$\text{maxRootIdx}$	Індекс найбільшого кореню серед коренів куп з послідовності куп
$\text{currStateCopy}$	Копія поточного стану послідовності куп

## Продовження таблиці 3.8

Змінна	Призначення
leoNums	Вектор чисел Леонардо, необхідних для сортування

3.8 Алгоритм знаходження індексу найбільшого кореню серед послідовності куп та індексу відповідного числа Леонардо

1. ПОЧАТОК
2.  $\text{currHeapLeoNumIdx} := 0$
3.  $\text{currStateCopy} := \text{currState}$
4. ПОКИ  $\neg(\text{currStateCopy} \& 1)$ :
  - 4.1.  $\text{currStateCopy} := \text{currStateCopy} \gg 1$
  - 4.2.  $\text{currHeapLeoNumIdx} := \text{currHeapLeoNumIdx} + 1$
5.  $\text{maxRootIdx} := \text{lastRootIdx}$
6.  $\text{heapLeoNumIdx} := \text{currHeapLeoNumIdx}$
7.  $\text{currRootIdx} := \text{lastRootIdx} - \text{leoNums}_{\text{currHeapLeoNumIdx}}$
8.  $\text{currHeapLeoNumIdx} := \text{currHeapLeoNumIdx} + 1$
9. ПОКИ  $\text{currStateCopy}$ :
  - 9.1. ЯКЩО  $(\text{currStateCopy} \& 1)$ , ТО:
    - 9.1.1. ЯКЩО  $\text{arr}_{\text{currRootIdx}} > \text{arr}_{\text{maxRootIdx}}$ , ТО:
      - 9.1.1.1.  $\text{maxRootIdx} := \text{currRootIdx}$
      - 9.1.1.2.  $\text{heapLeoNumIdx} := \text{currHeapLeoNumIdx}$
    - 9.1.2.  $\text{currRootIdx} := \text{currRootIdx} - \text{leoNums}_{\text{currHeapLeoNumIdx}}$
  - 9.2.  $\text{currStateCopy} := \text{currStateCopy} \gg 1$
  - 9.3.  $\text{currHeapLeoNumIdx} := \text{currHeapLeoNumIdx} + 1$
10. КІНЕЦЬ

Перелік всіх основних змінних алгоритму пошуку попереднього стану послідовності куп на основі поточного та їхнє призначення наведено в таблиці 3.9.

Таблиця 3.9 – Основні алгоритму пошуку попереднього стану послідовності куп на основі поточного та їхні призначення

Змінна	Призначення
currState	Поточний стан послідовності куп
firstSingleBit	Змінна для пошуку першого одиночного біта
idx	Змінна для обчислення положення першого одиночного біта firstSingleBit

### 3.9 Алгоритм пошуку попереднього стану купи на основі поточного

#### 1. ПОЧАТОК

#### 2. ЯКЩО $(currState \& 15) == 8$ , ТО:

2.1.  $currState := currState - 3$

#### 3. ІНАКШЕ:

##### 3.1. ЯКЩО $currState \& 1$ , ТО:

##### 3.1.1. ЯКЩО $(currState \& 3) == 3$ , ТО:

3.1.1.1.  $currState = currState^2$

##### 3.1.2. ІНАКШЕ:

3.1.2.1.  $currState = currState^1$

##### 3.2. ІНАКШЕ:

3.2.1.  $firstSingleBit := currState$

3.2.2.  $idx := 0$

##### 3.2.3. ПОКИ $firstSingleBit \& \& !(firstSingleBit \& 1)$ :

3.2.3.1.  $firstSingleBit := firstSingleBit \gg 1$

3.2.3.2.  $idx := idx + 1$

##### 3.2.4. ЯКЩО firstSingleBit, ТО:

3.2.4.1.  $currState = currState^{(1 \ll idx)}$

3.2.4.2.  $currState = currState | (1 \ll (idx - 1))$

3.2.4.3.  $currState = currState | (1 \ll (idx - 2))$

##### 3.2.5. ІНАКШЕ:

3.2.5.1.  $currState = 0$

#### 4. КІНЕЦЬ

## 4 ОПИС ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 4.1 Діаграма класів програмного забезпечення

Діаграма класів програмного забезпечення наведена на рисунку 4.1.

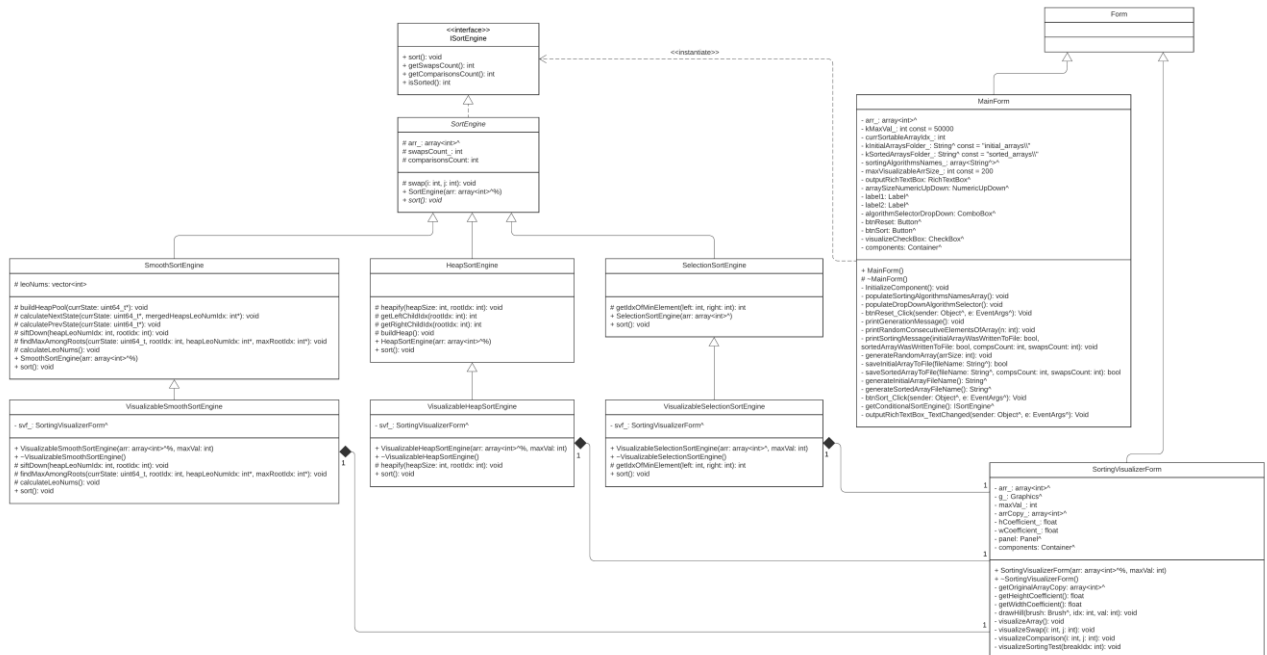


Рисунок 4.1 – Діаграма класів

### 4.2 Опис методів частин програмного забезпечення

#### 4.2.1 Стандартні методи

У таблиці 4.1 наведено стандартні методи, використані при розробці програмного забезпечення.

Таблиця 4.1 – Стандартні методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	vector<int>	push_back	Додає значення вхідного числа в кінець вектору	Число _Val типу int, значення якого потрібно занести в кінець вектору	-	cliext/vector
2	Application	EnableVisualStyles	Дозволяє використання візуальних стилів для застосунку	-	-	-



Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
3	Convert	ToString	Конвертує значення вхідного параметру, який має тип int / long long, в рядок типу String^	Число value типу int / long long	-	-
4	Application	Run				-
5	Application	SetCompatibleTextRenderingDefault	Встановлює для всього застосунку властивість UseCompatibleTextRendering, яка визначена на деяких елементах контролю, за замовчуванням	Булева змінна defaultValue	-	-
6	Form	Show	Відображає форму користувачу	-	-	-
7	Form	Close	Закриває форму	-	-	-
9	Control	CreateGraphics	Створює об'єкт Graphics для елемента керування	-	Об'єкт класу Graphics	-
10	Array	CopyTo	Копіює всі елементи даного масиву в інший визначений масив array, починаючи з індексу index	Масив array типу Array^ та індекс index типу int	-	-
11	-	sleep_for	«Заморожує» процес виконання програми на визначену кількість часу	_Rel_time типу nanoseconds	-	chrono та thread

Продовження таблиці 4.1

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
12	Graphics	FillRectangle	Заповнює вміст прямокутника, який задано парою координат, висотою та шириною, обраним кольором brush	Brush^ brush – колір, який буде заповнено прямокутник; float x та float y – координати верхнього лівого кута прямокутника; float width та float height – ширина і висота прямокутника	Об'єкт класу Graphics	-
13	Graphics	DrawRectangle	Малює прямокутник, який задано парою координат, висотою та шириною, з обраним контуром типу Pen^	Pen^ pen – визначає колір товщину та стиль контуру; float x та float y – координати верхнього лівого кута прямокутника; float width та float height – ширина і висота прямокутника	Об'єкт класу Graphics	-

#### 4.2.2 Користувацькі методи

У таблиці 4.2 наведено користувацькі методи, використані при розробці програмного забезпечення.

Таблиця 4.2 – Користувацькі методи

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
1	SortEngine	sort	Сортує масив	-	-	SortEngine.h
2	SortEngine	getSwapsCount	Повертає поточну кількість перестановок, здійснених в процесу сортування	-	swapsCount типу long long	SortEngine.h
3	SortEngine	getComparisonsCount	Повертає поточну кількість порівнянь між елементами масиву, здійснених в процесі сортування	-	comparisonsCount типу long long	SortEngine.h
4	SortEngine	sortingTest	Здійснює перевірку масиву на відсортованість в порядку неспадання. Повертає індекс першого елемента, що порушує властивість відсортованості або -1, якщо такий елемент відсутній	-	breakIdx типу int	SortEngine.h
5	MainForm	populateSortingAlgorithmsNamesArray	Заповнює рядковий масив назв алгоритмів сортування sortingAlgorithmsNames_	-	-	MainForm.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
6	MainForm	populateDropDownListSelector	Заповнює компонент «випадаючий список» елементами масиву назв алгоритмів сортування sortingAlgorithmsNames	-	-	MainForm.h
7	MainForm	InitializeComponent	Задає початкові значення керуючим компонентам форми	-	-	MainForm.h
8	MainForm	printRandomConsecutiveElementsOfArray	Виводить n послідовних елементів масиву arr_, взятих з випадкового місця масиву; якщо n > розмірності масиву, то виводить увесь масив	int n – кількість послідовних елементів масиву, які потрібно вивести	-	MainForm.h
9	MainForm	printGenerationMessage	Виводить повідомлення про генерацію масиву arr_ в елемент керування outputRichTextBox	-	-	MainForm.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
10	MainForm	btnReset_Click	Метод для обробки події «натиснення на кнопку» кнопки btnReset для генерації масиву arr_ випадкових цілих чисел, виводу відповідного повідомлення в елемент керування outputRichTextBox, розблокування кнопки btnSort, якщо на момент натискання вона була заблокована та керуванням доступності прапорця visualize	EventArgs^ e – поточна «подія»; Object^ sender – об'єкт відправник	-	MainForm.h
11	MainForm	generateRandomArray	Генерує масив arr_ заданого розміру, який складається з випадкових цілих чисел	int arrSize – розмірність масиву	-	MainForm.h
12	MainForm	saveInitialArrayToFile	Зберігає поточний стан масиву arr_ (всі елементи) та його розмір до текстового файлу, який зберігатиметься у відповідній папці	String^ fileName – назва файлу, в який зберігатиметься поточний стан масиву	true, якщо масив вдалося зберегти; інакше - false	MainForm.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
13	MainForm	saveSorted ArrayToFile	Зберігає відсортований масив arr_, його розмір, метод сортування, кількість порівнянь та перестановок до текстового файлу, який зберігатиметься у відповідній папці	String^ fileName – назва файлу, в який зберігатиметься поточний стан масиву; int copmsCount – кількість порівнянь; int swapsCount – кількість перестановок	true, якщо масив вдалося зберегти; інакше - false	MainForm.h
14	MainForm	generateIni tialArrayFile Name	Генерує поточну назву файлу, в який зберігатиметься стан масиву до сортування	-	Назва файлу типу String^	MainForm.h
15	Visualizable SmoothSort Engine	sort	Сортує масив згідно алгоритму плавного сортування; процес сортування супроводжується візуалізацією	-	-	Visualizable SmoothSortE ngine.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
16	MainForm	printSortingMessage	Виводить повідомлення про відсортування масиву arr_ в елемент керування outputRichTextBox	bool initialArrayWasWrittenToFile – визначає, чи був початковий масив збережено до файлу; bool sortedArrayWasWrittenToFile – визначає, чи був відсортований масив збережено у файл; int compmsCount – кількість порівнянь; int swapsCount – кількість перестановок	-	MainForm.h
17	MainForm	btnSort_Click	Метод для обробки події «натискання» на кнопку btnSort; сортує масив arr_ обраним методом, записує масив до сортування і після сортування у текстові файли; виводить відповідне повідомлення в outputRichTextBox	EventArgs^ e – поточна «подія»; Object^ sender – об'єкт відправник	-	MainForm.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
18	MainForm	getConditionalSortEngine	Повертає об'єкт класу, що імплементує інтерфейс ISortEngine та відповідає користувацькому вводу	-	Об'єкт класу, що імплементує інтерфейс ISortEngine	MainForm.h
19	MainForm	updateOutputTextBox	При виклику перевіряє розмірність тексту outputRichTextBox ; якщо вона біла за 2000, то очищує вміст outputRichTextBox	-	-	MainForm.h
20	MainForm	generateSortedArrayFileName	Генерує поточну назву файлу, в який зберігатиметься стан масиву після сортування	-	Назва файлу типу String^	MainForm.h
21	MainForm	outputRichTextBox_TextChanged	Метод для обробки події «зміна тексту» outputRichTextBox ; встановлює смугу прокрутки в нижнє положення	EventArgs^ e – поточна «подія»; Object^ sender – об'єкт відправник	-	MainForm.h



Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
22	SortingVisualizerForm	getHeightCoefficient	Повертає значення hCoefficient_, яке є коефіцієнтом, який буде використано для масштабування висоти стовпця гістограми під час візуалізації процесу сортування	-	Значення hCoefficient_ типу float	SortingVisualizerForm.h
23	SortingVisualizerForm	getWidthCoefficient	Повертає значення wCoefficient_, яке є коефіцієнтом, який буде використано для масштабування ширини стовпця гістограми під час візуалізації процесу сортування	-	Значення wCoefficient_ типу float	SortingVisualizerForm.h
24	SortingVisualizerForm	InitializeComponent	Задає початкові значення керуючим компонентам форми	-	-	SortingVisualizerForm.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
25	SortingVisualizerForm	drawHill	Метод для графічного відображення одного стовпця гістограми	Brush^ brush – колір стовпця; int idx – індекс елементу масиву, який відображається; int val – значення елементу масиву, який відображається	-	SortingVisualizerForm.h
26	SortingVisualizerForm	visualizeArray	Метод для графічного відображення всіх елементів масиву у вигляді стовпців гістограми	-	-	SortingVisualizerForm.h
27	SortingVisualizerForm	visualizeSwap	Метод для графічного відображення процесу перестановки двох елементів масиву	int i та int j – індекси елементів, які переставляються	-	SortingVisualizerForm.h
28	SortingVisualizerForm	visualizeComparison	Метод для графічного відображення процесу порівняння двох елементів масиву	int i та int j – індекси елементів, які порівнюються	-	SortingVisualizerForm.h
29	SortingVisualizerForm	visualizeSortingTest	Метод для графічного відображення процесу перевірки масиву на відсортованість	int breakIdx – індекс першого елементу масиву, який порушує умові відсортованості	-	SortingVisualizerForm.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
30	SelectionSortEngine	getIdxOfMinElement	Знаходить індекс найменшого елементу на проміжку елементів з індексами від left до right	int left та int right – індекси найбільш лівого та найбільш правого елементів з проміжку пошуку	int minIdx – індекс найменшого елементу з проміжку	SelectionSortEngine.h
31	SelectionSortEngine	sort	Сортує масив згідно алгоритму сортування вибором	-	-	SelectionSortEngine.h
32	HeapSortEngine	heapify	«Просійка вниз» для дотримання властивості MaxHeap	int heapSize – розмір бінарної купи; int rootIdx – індекс кореневого елементу поточної купи	-	HeapSortEngine.h
33	HeapSortEngine	getLeftChildIdx	Знаходить індекс лівого нащадку за допомогою індексу батьківського елементу	int rootIdx	Індекс лівого нащадку елементу з індексом rootIdx типу int	HeapSortEngine.h
34	HeapSortEngine	getRightChildIdx	Знаходить індекс правого нащадку за допомогою індексу батьківського елементу	int rootIdx	Індекс правого нащадку елементу з індексом rootIdx типу int	HeapSortEngine.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
35	HeapSortEngine	buildHeap	Метод для побудови бінарної купи з вихідного масиву	-	-	HeapSortEngine.h
36	HeapSortEngine	sort	Сортує масив згідно алгоритму пірамідального сортування	-	-	HeapSortEngine.h
37	SmoothSortEngine	buildHeapPool	Метод для побудови послідовності куп, які мають розміри, рівні числам Леонардо; також оновлює значення поточного стану послідовності куп.	uint64_t* currState – посилання на поточний стан послідовності куп	-	SmoothSortEngine.h
38	SmoothSortEngine	calculateNextState	Метод для розрахунку наступного стану послідовності куп на основі поточного. Оновлює значення currState; оновлює значення mergedHeapsLeonumIdx, якщо перехід до нового стану передбачає об'єднання куп, інакше mergedHeapsLeonumIdx = - 1	uint64_t* currState – посилання на поточний стан послідовності куп; int* mergedHeapsLeonumIdx – посилання на індекс числа Леонардо, яке дорівнює розмірності об'єднаних куп, якщо новий стан передбачає об'єднання куп	-	SmoothSortEngine.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
39	SmoothSort Engine	calculatePrevState	Метод для розрахунку попереднього стану послідовності куп на основі поточного. Оновлює значення currState	uint64_t* currState – посилання на поточний стан послідовності куп	-	SmoothSortEngine.h
40	SmoothSort Engine	siftDown	Метод для «просійки вниз» поточної купи	int heapLeoNumIdx – індекс числа Леонардо, яке дорівнює розміру поточної купи; int rootIdx – індекс кореню поточної купи	-	SmoothSortEngine.h
41	SmoothSort Engine	calculateLeoNums	Розраховую достатню для сортування кількість чисел Леонардо	-	-	SmoothSortEngine.h
42	SmoothSort Engine	sort	Сортує масив згідно алгоритму плавного сортування	-	-	SmoothSortEngine.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
43	SmoothSort Engine	findMaxA mongRoots	Метод для знаходження індексу найбільшого кореню серед коренів куп з послідовності куп. Оновлює значення maxRootIdx та heapLeoNumIdx	uint64_t currState – поточний стан послідовності куп; int rootIdx – індекс кореню останньої купи; int* heapLeoNumIdx – індекс числа Леонард, яке дорівнює розмірності останньої купи; int* maxRootIdx - індекс найбільшого кореню серед коренів куп з послідовності куп	-	SmoothSortE ngine.h
44	Visualizabl eHeapSort Engine	heapify	«Просійка вниз» для дотримання власловості MaxHeap; процес «просійки» супроводжується візуалізацією	int heapSize – розмір бінарної купи; int rootIdx – індекс кореневого елементу поточної купи	-	Visualizable HeapSortEng ine.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
45	VisualizableHeapSortEngine	sort	Сортує масив згідно алгоритму пірамідального сортування; процес сортування супроводжується візуалізацією	-	-	VisualizableHeapSortEngine.h
46	VisualizableSelectionSortEngine	getIdxOfMinElement	Знаходить індекс найменшого елемента на проміжку елементів з індексами від left до right; процес пошуку супроводжується візуалізацією	int left та int right – індекси найбільш лівого та найбільш правого елементів з проміжку пошуку	int minIdx – індекс найменшого елемента з проміжку	VisualizableSelectionSortEngine.h
47	VisualizableSelectionSortEngine	sort	Сортує масив згідно алгоритму сортування вибором; процес сортування супроводжується візуалізацією	-	-	VisualizableSelectionSortEngine.h
48	VisualizableSmoothSortEngine	siftDown	Метод для «просійки вниз» поточної купи; процес «просійки» супроводжується візуалізацією	int heapLeoNumIdx – індекс числа Леонардо, яке дорівнює розміру поточної купи; int rootIdx – індекс кореню поточної купи	-	VisualizableSmoothSortEngine.h

Продовження таблиці 4.2

№ п/п	Назва класу	Назва функції	Призначення функції	Опис вхідних параметрів	Опис вихідних параметрів	Заголовний файл
49	VisualizableSmoothSortEngine	findMaxAmongRoots	Метод для знаходження індексу найбільшого кореню серед коренів куп з послідовності куп. Оновлює значення maxRootIdx та heapLeoNumIdx; процес пошуку супроводжується візуалізацією	uint64_t currState – поточний стан послідовності куп; int rootIdx – індекс кореню останньої купи; int* heapLeoNumIdx – індекс числа Леонард, яке дорівнює розмірності останньої купи; int* maxRootIdx – індекс найбільшого кореню серед коренів куп з послідовності куп	-	VisualizableSmoothSortEngine.h



## 5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

### 5.1 План тестування

Тестування програмного забезпечення, що охоплює тестування основного функціоналу застосунку і тестування реакцій на виключні ситуації, буде проведено за наступним планом:

- а) Тестування правильності введення значень в поле введення розмірності масиву.
  - 1) Тестування при спробі ручного введення нецілочисельних значень.
  - 2) Тестування при ручному введенні замалих та завеликих значень.
  - 3) Тестування при ручному введенні значення, що лежить в допустимих межах.
  - 4) Тестування при спробі введення стрілками керуючого компонента замалих та завеликих значень.
  - 5) Тестування при введенні стрілками керуючого компонента значення, що лежить в допустимих межах.
- б) Тестування правильності встановленню прапорцю, що відповідає за необхідність візуалізації процесу сортування масиву.
  - 1) Тестування при спробі взаємодії з прапорцем, якщо розмірність згенерованого масиву перевищує допустимі для візуалізації процесу сортування межі.
  - 2) Тестування при взаємодії з прапорцем, якщо розмірність згенерованого масиву лежить в допустимих для візуалізації процесу сортування межах.
- в) Тестування можливості вибору алгоритму сортування масиву з використанням спадного списку.
- г) Тестування можливості натискання на кнопку «Sort» без попередньої генерації масиву.
- д) Тестування коректності роботи алгоритмів сортування вибором, пірамідального сортування та плавного сортування.

- 1) Перевірка правильності роботи алгоритму сортування вибором.
  - 2) Перевірка правильності роботи алгоритму пірамідального сортування.
  - 3) Перевірка правильності роботи алгоритму плавного сортування.
- е) Тестування візуалізації процесу сортування масиву обраним методом.
- 1) Перевірка візуалізації процесу сортування алгоритмом сортування вибором.
  - 2) Перевірка візуалізації процесу сортування алгоритмом пірамідального сортування.
  - 3) Перевірка візуалізації процесу сортування алгоритмом плавного сортування.
- ж) Тестування збереження результатів у текстовий файл.
- 1) Тестування збереження початкового стану масиву (до сортування) або масиву у відсортованому стані в текстовий файл, за умови, що папка, до якої зберігатиметься файл або файл з відповідною назвою недоступні.
  - 2) Тестування збереження початкового стану масиву (до сортування) або масиву у відсортованому стані в текстовий файл, за умови, що папка, до якої зберігатиметься файл або файл з відповідною назвою доступні.

## 5.2 Приклади тестування

Проведено тестування програмного забезпечення за визначеним вище планом. Результат кожного тестування наведено в окремій таблиці, в якій фіксується мета тестування, початковий стан програми, вхідні дані, схема проведення тесту, очікуваний результат, стан програми після проведення тестувань.

Результати проведених тестувань наведені у таблицях 5.1-5.17.

Таблиця 5.1 – Приклад роботи програми при спробі ручного введення нецілочисельних значень в поле для вводу розмірності масиву.

Мета тесту	Перевірити можливість введення нецілочисельних символів
Початковий стан програми	Відкрите вікно програми
Вхідні дані	abcd.@
Схема проведення тесту	Ручний ввід розмірності масиву у відповідне поле
Очікуваний результат	Програмне блокування вводу нецілочисельних символів – поле для вводу не змінить свого початкового значення
Стан програми після проведення випробувань	Поле вводу не змінило початкове значення

Таблиця 5.2 – Приклад роботи програми при ручному введенні замалих або завеликих значень в поле для вводу розмірності масиву.

Мета тесту	Перевірити можливість ручного введення завеликих та замалих значень
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тест 1 – 10 Тест 2 – 1000000 Тест 3 – -10
Схема проведення тесту	Ручний ввід розмірності масиву у відповідне поле

Продовження таблиці 5.2

Очікуваний результат	Зміна значення в полі для введення на максимальне (50000, в тесті 2) або мінімальне (100, в тесті 1 і 3) допустиме після натискання користувачем Enter або взаємодії з будь-яким іншим керуючим компонентом застосунку
Стан програми після проведення випробувань	Поле для вводу змінило своє значення на 100 в тестах 1 і 3 та на 50000 в тесті 2

Таблиця 5.3 – Приклад роботи програми при ручному введенні значення, що лежить в допустимих межах, в поле для вводу розмірності масиву.

Мета тесту	Перевірити можливість ручного введення допустимих значень
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тест 1 – 100 Тест 2 – 25000 Тест 3 – 50000
Схема проведення тесту	Ручний ввід розмірності масиву у відповідне поле
Очікуваний результат	Поле для вводу змінює своє значення
Стан програми після проведення випробувань	Поле для вводу змінило своє значення на введене в кожному з тестів

Таблиця 5.4 – Приклад роботи програми при спробі введення стрілками керуючого компоненту замалих та завеликих значень в поле для вводу розмірності масиву.

Мета тесту	Перевірити можливість введення замалих та завеликих значень за допомогою стрілок керуючого компоненту
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тест 1 – спроба ввести значення 50100 (одне натискання на стрілку змінює значення в полі на 100) Тест 2 – спроба ввести значення 0
Схема проведення тесту	Ввід розмірності масиву у відповідне поле за допомогою стрілок керуючого компоненту
Очікуваний результат	Програмне блокування вводу значень за межами допустимих
Стан програми після проведення випробувань	Програма заблокувала введення завеликого і замалого значення. В тесті 1 в полі залишилося значення 50000, а в тесті 2 – 100

Таблиця 5.5 – Приклад роботи програми при введенні стрілками керуючого компоненту значення, що лежить в допустимих межах, в поле для вводу розмірності масиву.

Мета тесту	Перевірити можливість введення допустимих значень за допомогою стрілок керуючого компоненту
Початковий стан програми	Відкрите вікно програми
Вхідні дані	5000

Продовження таблиці 5.5

Схема проведення тесту	Ввід розмірності масиву у відповідне поле за допомогою стрілок керуючого компоненту
Очікуваний результат	Поле для вводу змінює своє значення
Стан програми після проведення випробувань	Поле для вводу змінило своє значення на введене

Таблиця 5.6 – Приклад роботи програми при спробі взаємодії з прапорцем, якщо розмірність згенерованого масиву перевищує допустимі для візуалізації процесу сортування межі.

Мета тесту	Перевірити можливість зміни значення прапорцю, якщо розмірність згенерованого масиву перевищує допустимі для візуалізації сортування межі ( $> 200$ )
Початковий стан програми	Відкрите вікно програми
Вхідні дані	400 в поле для введення розмірності масиву
Схема проведення тесту	Ввести розмірність масиву у відповідне поле, натиснути кнопку «Generate Random Array» та спробувати змінити значення прапорця «Visualize»
Очікуваний результат	Програмне забезпечення заблокувало можливість взаємодії з прапорцем «Visualize», сам керуючий компонент став напівпрозорим
Стан програми після проведення випробувань	Прапорець «Visualize» не змінив початкового значення та став напівпрозорим

Таблиця 5.7 – Приклад роботи програми при спробі взаємодії з прапорцем, якщо розмірність згенерованого масиву лежить в допустимих для візуалізації процесу сортування межах

Мета тесту	Перевірити можливість зміни значення прапорцю, якщо розмірність згенерованого масиву лежить в допустимих для візуалізації процесу сортування межах (не більше 200)
Початковий стан програми	Відкрите вікно програми
Вхідні дані	150 в поле для введення розмірності масиву
Схема проведення тесту	Ввести розмірність масиву у відповідне поле, натиснути кнопку «Generate Random Array» та змінити значення прапорця «Visualize»
Очікуваний результат	Значення прапорця «Visualize» змінилося на протилежне після натискання користувача
Стан програми після проведення випробувань	Значення прапорця «Visualize» змінилося на протилежне після натискання користувача

Таблиця 5.8 – Приклад роботи програми при спробі змінити алгоритм сортування за допомогою спадного списку

Мета тесту	Перевірити можливість зміни значення обраного алгоритму сортування за допомогою спадного списку
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Зміна алгоритму сортування з «Selection Sort» на «Heap Sort»

Продовження таблиці 5.8

Схема проведення тесту	За допомогою спадного списку змінити значення алгоритму сортування (початкове - «Selection Sort») на «Heap Sort»
Очікуваний результат	Значення в керуючому компоненті змінилося на «Heap Sort»
Стан програми після проведення випробувань	Значення в керуючому компоненті змінилося на «Heap Sort»

Таблиця 5.9 – Приклад роботи програми при спробі натискання на кнопку «Sort» без попередньої генерації масиву

Мета тесту	Перевірити можливість почати процес сортування без попередньої генерації масиву
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Спроба натиснути на кнопку «Sort»
Схема проведення тесту	Натиснути на кнопку «Sort»
Очікуваний результат	Програмне блокування можливості взаємодіяти з кнопкою «Sort»; сам керуючий компонент має відрізняється дизайном від стандартного
Стан програми після проведення випробувань	Програма заблокувала можливість взаємодії з кнопкою «Sort»; дизайн кнопки відрізняється від стандартного

Задля тестування коректності роботи кожного з алгоритмів сортування програмний код, що обробляє натискання на кнопку «Sort», було змінено таким чином, що, окрім сортування масиву обраним методом, він створює додатковий масив, тої самої розмірності, що й початковий масив, та копіює всі елементи початкового в себе за допомогою методу стандартного класу `Array::CopyTo()`.



Далі відбувається сортування оригінального масиву обраним методом та сортування масиву-копії стандартним методом `Array::Sort()`. Після цього здійснюється поелементне порівняння двох масивів. Якщо значення всіх елементів масиву-оригіналу збігається зі значеннями елементів масиву-копії, то в елемент керування, що відповідає за вивід базових повідомлень (про генерацію, сортування та запис до файлів), виводиться повідомлення «Sorting is correct», інакше – «Something went wrong».

Таблиця 5.10 – Приклад роботи програми при спробі відсортувати масив методом сортування вибором

Мета тесту	Перевірити коректність роботи власної реалізації алгоритму сортування вибором
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тест 1 – Розмірність масиву: 100 Тест 2 – Розмірність масиву: 25000 Тест 3 – Розмірність масиву: 50000
Схема проведення тесту	Ввести розмірність масиву, натиснути на кнопку «Generate Random Array», обрати метод сортування вибором, натиснути на кнопку «Sort»
Очікуваний результат	Окрім базових повідомлень про сортування в елементі керування, що відповідає за вивід інформації, виведеться повідомлення «Sorting is correct»
Стан програми після проведення випробувань	Повідомлення «Sorting is correct» вивелося разом з іншою довідковою інформацією щодо процесу сортування в усіх трьох тестах

Таблиця 5.11 – Приклад роботи програми при спробі відсортувати масив методом пірамідального сортування

Мета тесту	Перевірити коректність роботи власної реалізації алгоритму пірамідального сортування
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Тест 1 – Розмірність масиву: 100 Тест 2 – Розмірність масиву: 25000 Тест 3 – Розмірність масиву: 50000
Схема проведення тесту	Ввести розмірність масиву, натиснути на кнопку «Generate Random Array», обрати метод пірамідального сортування, натиснути на кнопку «Sort»
Очікуваний результат	Окрім базових повідомлень про сортування в елементі керування, що відповідає за вивід інформації, виведеться повідомлення «Sorting is correct»
Стан програми після проведення випробувань	Повідомлення «Sorting is correct» вивелося разом з іншою довідковою інформацією щодо процесу сортування в усіх трьох тестах

Таблиця 5.12 – Приклад роботи програми при спробі відсортувати масив методом плавного сортування

Мета тесту	Перевірити коректність роботи власної реалізації алгоритму пірамідального сортування
Початковий стан програми	Відкрите вікно програми

Продовження таблиці 5.12

Вхідні дані	Тест 1 – Розмірність масиву: 100 Тест 2 – Розмірність масиву: 25000 Тест 3 – Розмірність масиву: 50000
Схема проведення тесту	Ввести розмірність масиву, натиснути на кнопку «Generate Random Array», обрати метод плавного сортування, натиснути на кнопку «Sort»
Очікуваний результат	Окрім базових повідомлень про сортування в елементі керування, що відповідає за вивід інформації, виведеться повідомлення «Sorting is correct»
Стан програми після проведення випробувань	Повідомлення «Sorting is correct» вивелося разом з іншою довідковою інформацією щодо процесу сортування в усіх трьох тестах

Таблиця 5.13 – Приклад роботи програми при спробі відсортувати масив методом сортування вибором, що супроводжується візуалізацією процесу сортування

Мета тесту	Перевірити коректність візуалізації процесу сортування алгоритмом сортування вибором
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Розмірність масиву: 150 Алгоритм сортування: «Selection Sort» Прапорець «Visualize»: активовано

Продовження таблиці 5.13

Схема проведення тесту	Ввести розмірність масиву, натиснути на кнопку «Generate Random Array», обрати метод сортування вибором, активувати прапорець «Visualize», натиснути кнопку «Sort»
Очікуваний результат	Після натискання кнопки «Sort» відкриється нове вікно, в якому демонструватиметься анімація, що відображає процес сортування вибором; після завершення сортування в тому ж вікні відобразиться анімація перевірки масиву на відсортованість
Стан програми після проведення випробувань	Після натискання кнопки «Sort» відкрилося нове вікно, в якому було візуалізовано процес сортування вибором та процес перевірки масиву на відсортованість

На рисунку 5.1 наведено приклад візуалізації процесу сортування вибором.

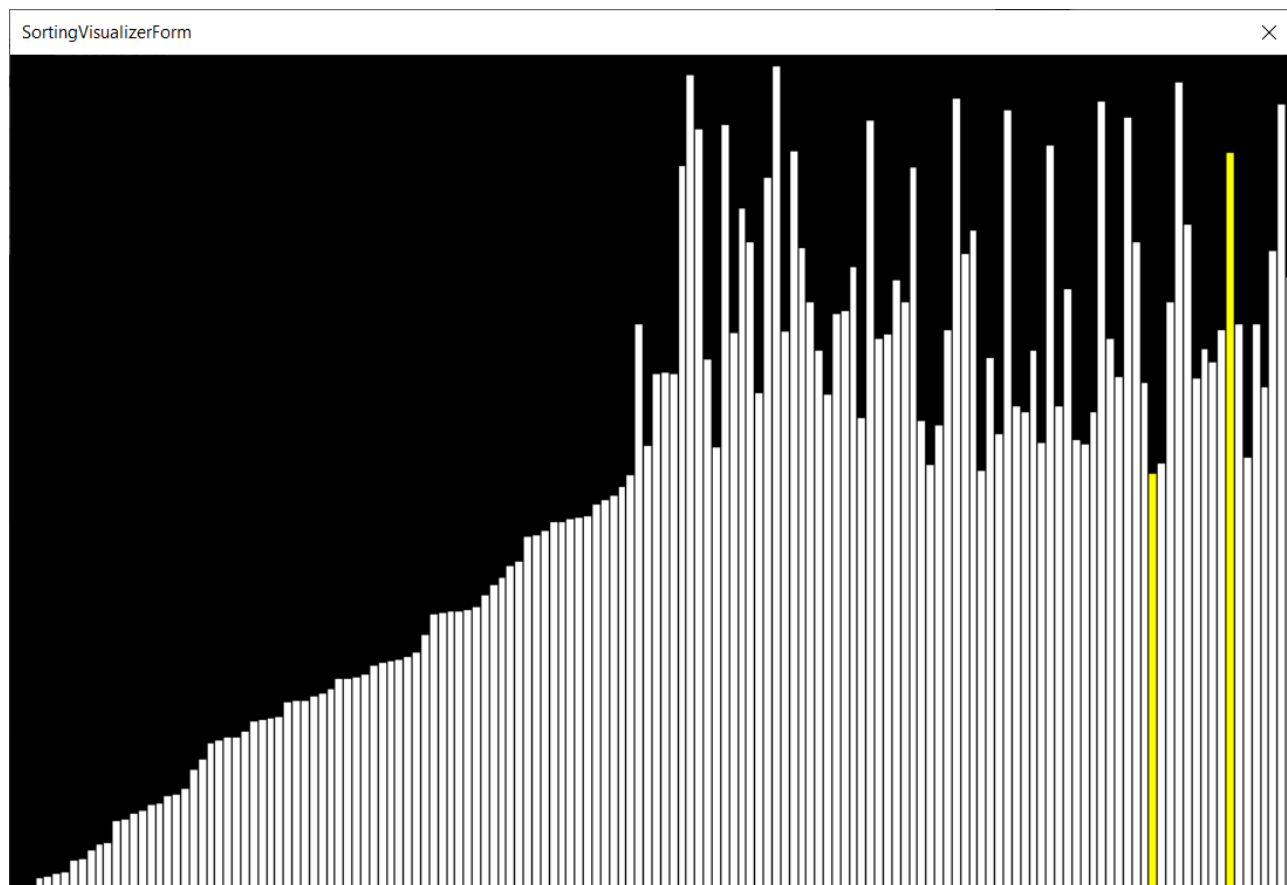


Рисунок 5.1 – Приклад візуалізації процесу сортування вибором

На рисунку 5.2 наведено приклад візуалізації процесу перевірки на відсортованість після сортування вибором.

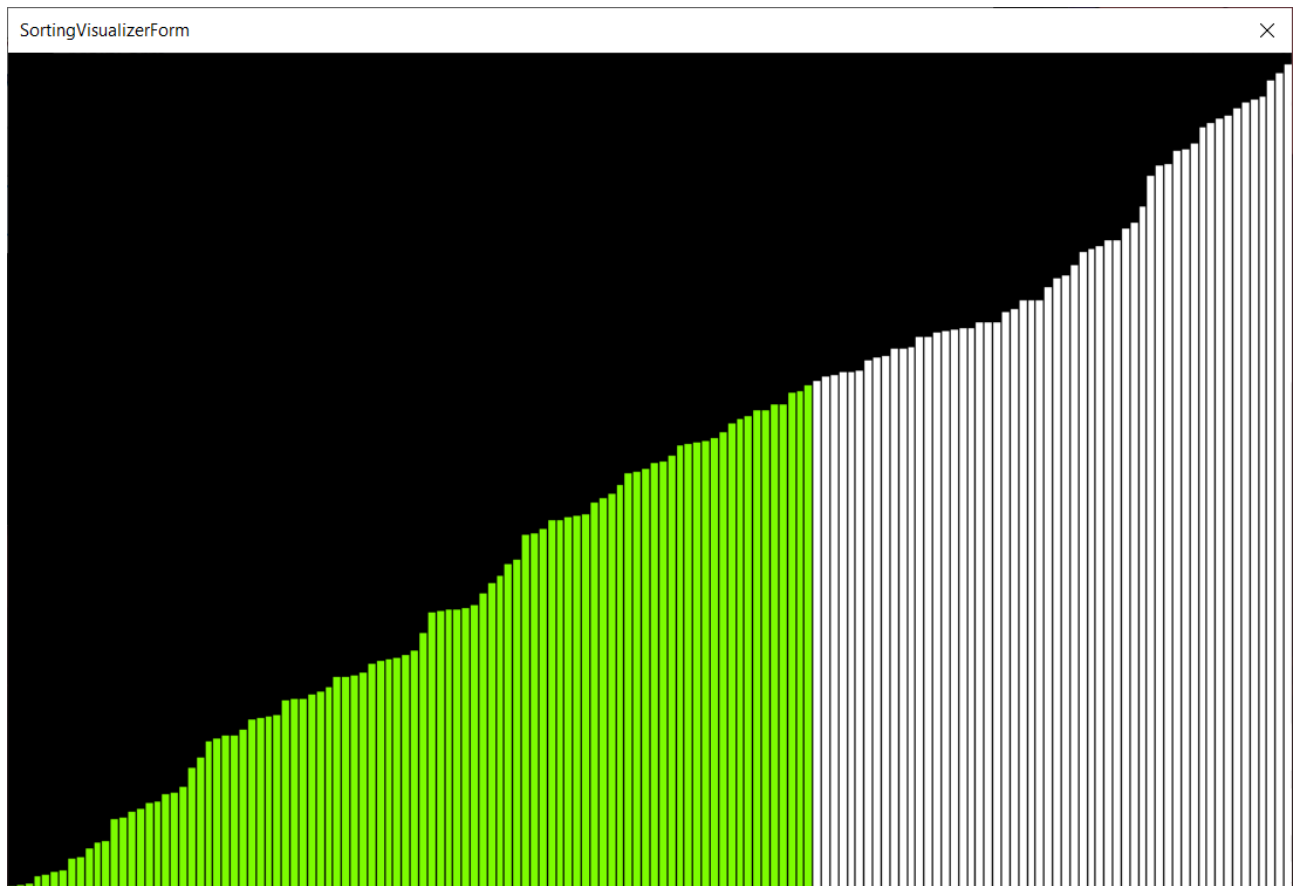


Рисунок 5.2 – Приклад візуалізації процесу перевірки на відсортованість після сортування вибором

Таблиця 5.14 – Приклад роботи програми при спробі відсортувати масив методом пірамідального сортування, що супроводжується візуалізацією процесу сортування

Мета тесту	Перевірити коректність візуалізації процесу сортування алгоритмом пірамідального сортування
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Розмірність масиву: 150 Алгоритм сортування: «Heap Sort» Прапорець «Visualize»: активовано

Продовження таблиці 5.14

Схема проведення тесту	Ввести розмірність масиву, натиснути на кнопку «Generate Random Array», обрати алгоритм пірамідального сортування, активувати прапорець «Visualize», натиснути кнопку «Sort»
Очікуваний результат	Після натискання кнопки «Sort» відкриється нове вікно, в якому демонструватиметься анімація, що відображає процес пірамідального сортування; після завершення сортування в тому ж вікні відобразиться анімація перевірки масиву на відсортованість
Стан програми після проведення випробувань	Після натискання кнопки «Sort» відкрилося нове вікно, в якому було візуалізовано процес пірамідального сортування та процес перевірки масиву на відсортованість

На рисунку 5.3 наведено приклад візуалізації процесу пірамідального сортування.

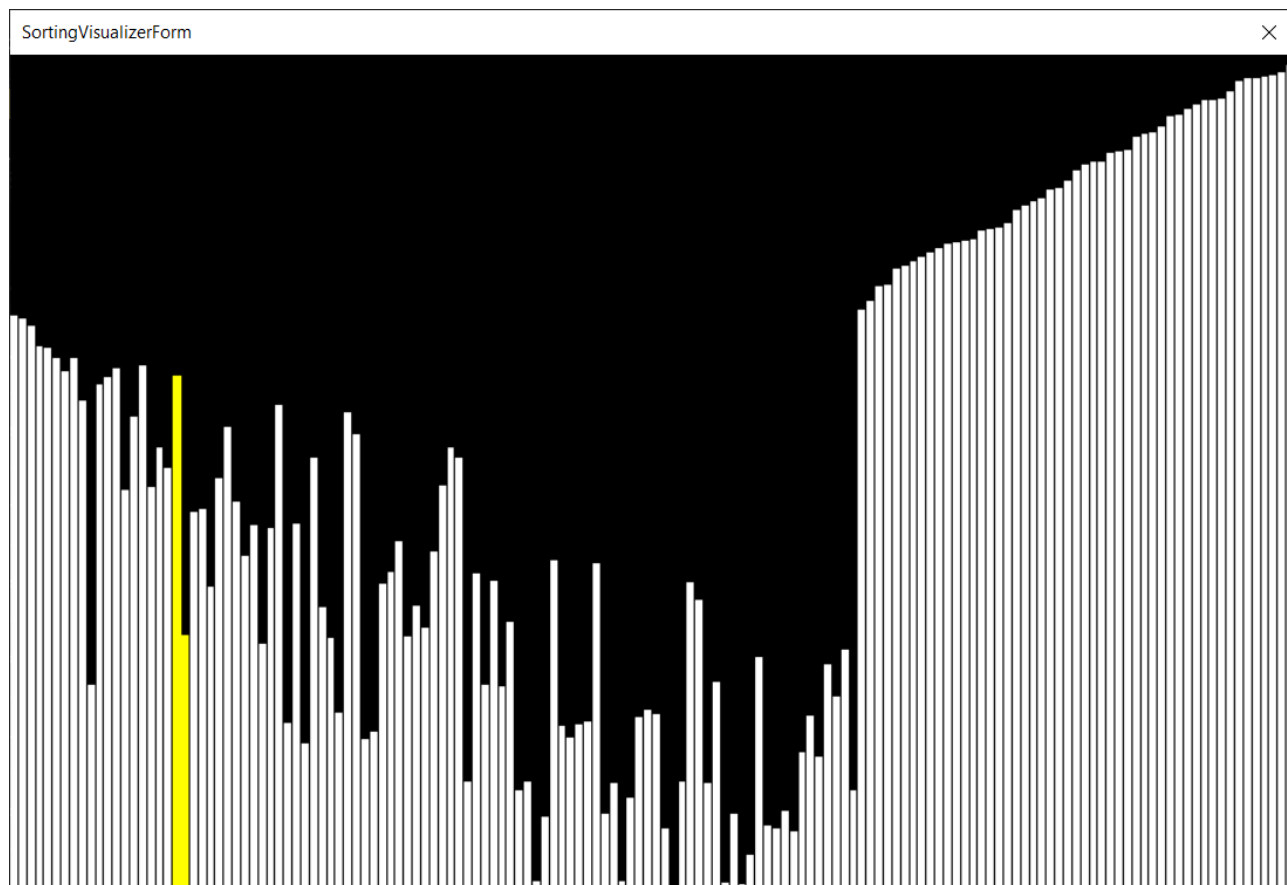


Рисунок 5.3 – Приклад візуалізації процесу пірамідального сортування

На рисунку 5.4 наведено приклад візуалізації процесу перевірки на відсортованість після пірамідального сортування.



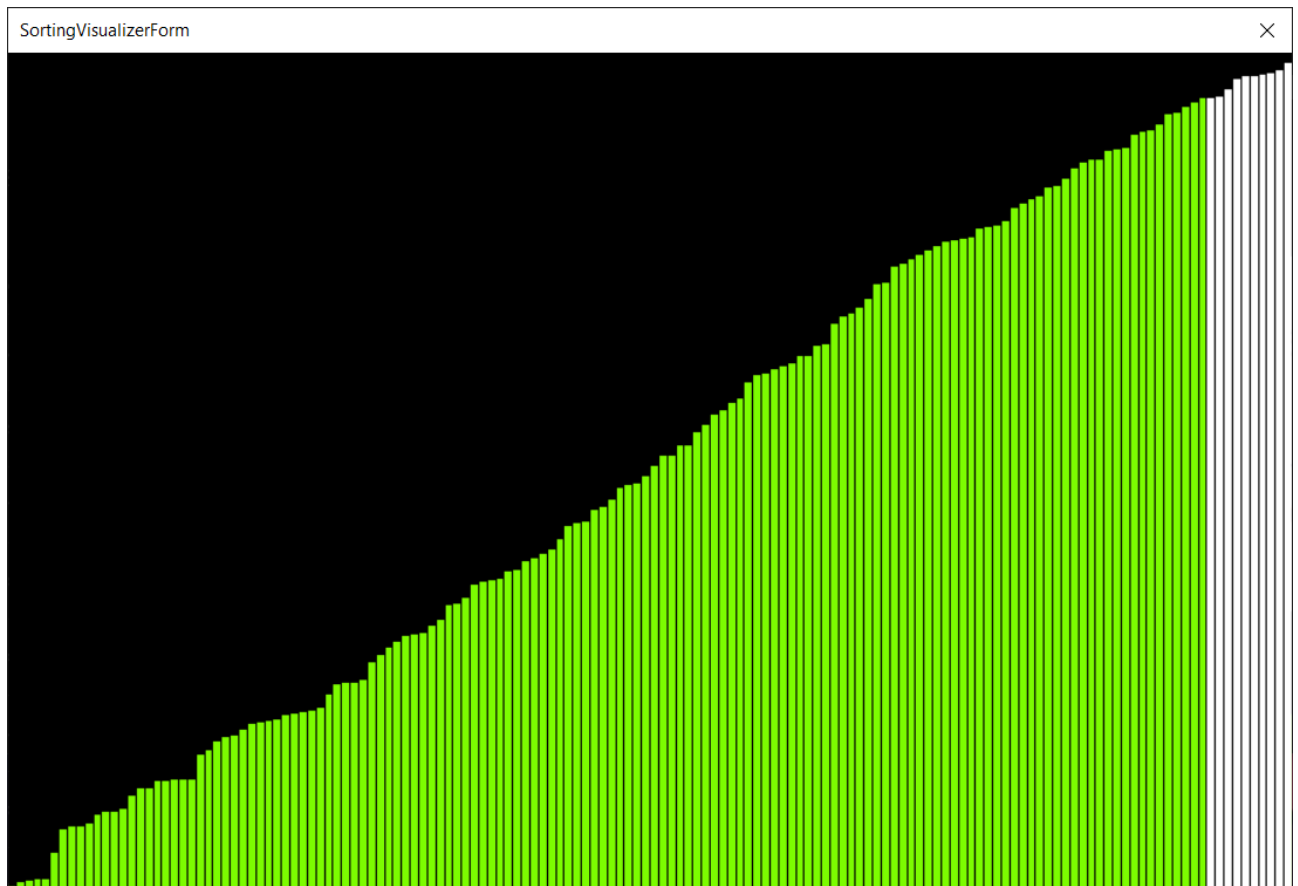


Рисунок 5.4 – Приклад візуалізації процесу перевірки на відсортованість після пірамідального сортування

Таблиця 5.15 – Приклад роботи програми при спробі відсортувати масив методом плавного сортування, що супроводжується візуалізацією процесу сортування

Мета тесту	Перевірити коректність візуалізації процесу сортування алгоритмом плавного сортування
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Розмірність масиву: 150 Алгоритм сортування: «Smooth Sort» Прапорець «Visualize»: активовано

Продовження таблиці 5.15

Схема проведення тесту	Ввести розмірність масиву, натиснути на кнопку «Generate Random Array», обрати алгоритм плавного сортування, активувати прапорець «Visualize», натиснути кнопку «Sort»
Очікуваний результат	Після натискання кнопки «Sort» відкриється нове вікно, в якому демонструватиметься анімація, що відображає процес плавного сортування; після завершення сортування в тому ж вікні відобразиться анімація перевірки масиву на відсортованість
Стан програми після проведення випробувань	Після натискання кнопки «Sort» відкрилося нове вікно, в якому було візуалізовано процес плавного сортування та процес перевірки масиву на відсортованість

На рисунку 5.5 наведено приклад візуалізації процесу плавного сортування.

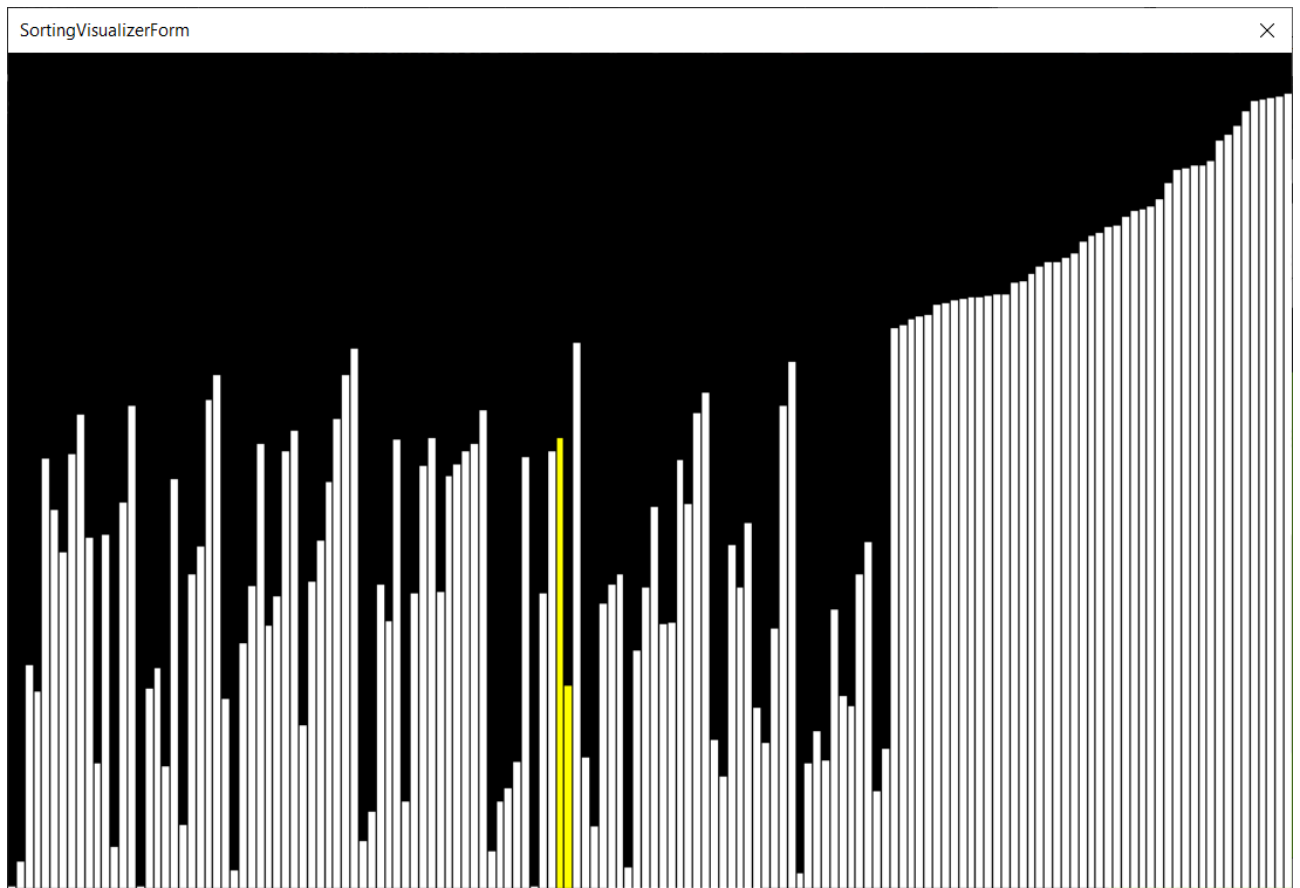


Рисунок 5.5 – Приклад візуалізації процесу плавного сортування

На рисунку 5.6 наведено приклад візуалізації процесу перевірки на відсортованість після плавного сортування.

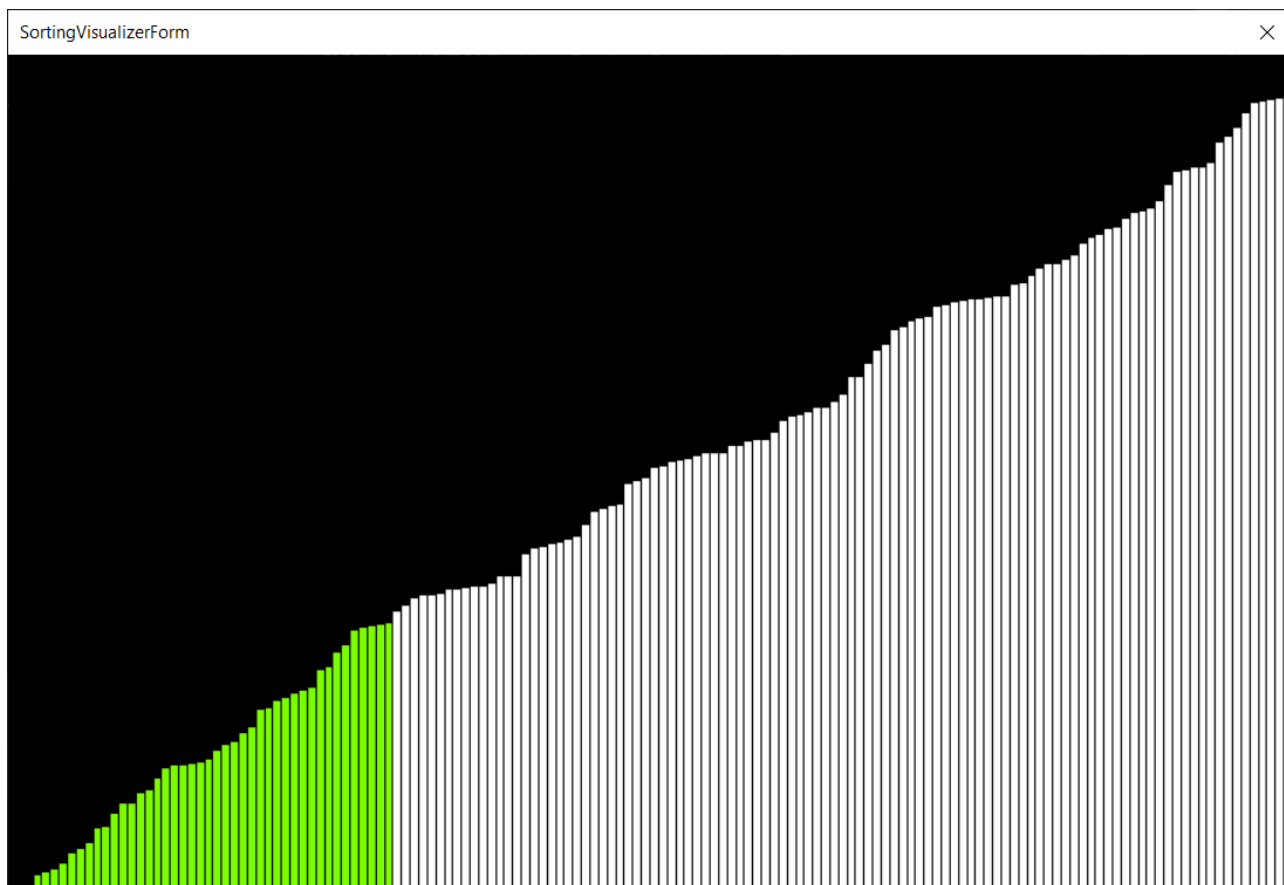


Рисунок 5.6 – Приклад візуалізації процесу перевірки на відсортованість після плавного сортування

Таблиця 5.16 – Приклад роботи програми після натискання кнопки «Sort» при спробі збереження початкового і відсортованого станів масиву в папки або файли, які недоступні

Мета тесту	Перевірити коректність реакції програми на спробу збереження початкового або відсортованого масиву до папки або в файл, які недоступні
Початковий стан програми	Відкрите вікно програми
Вхідні дані	Розмірність масиву: 1000

Продовження таблиці 5.16

Схема проведення тесту	Задати властивість текстового файлу «initial_array_1» в папці «initial_arrays» «Тільки для читання», Ввести розмірність масиву, натиснути на кнопку «Generate Random Array», натиснути на кнопку «Sort»
Очікуваний результат	Вивід повідомлення про помилку при спробі збереження початкового стану масиву до текстового файлу з назвою «initial_array_1»; також в довідковій інформації, яка виводиться у відповідне текстове поле, не буде йтися про збереження початкового масиву до текстового файлу
Стан програми після проведення випробувань	З'явилося повідомлення про помилку збереження початкового масиву до файлу «initial_array_1»; в довідковій інформації, що вивелася у відповідне текстове поле не було повідомлення про збереження початкового масиву до текстового файлу

Таблиця 5.17 – Приклад роботи програми після натискання кнопки «Sort» при спробі збереження початкового і відсортованого станів масиву в папки або файли, які доступні

Мета тесту	Перевірити коректність реакції програми на спробу збереження початкового або відсортованого масиву до папки або в файл, які доступні
------------	--

Продовження таблиці 5.17

Початковий стан програми	Відкрите вікно програми
Вхідні дані	Розмірність масиву: 1000
Схема проведення тесту	Ввести розмірність масиву, натиснути на кнопку «Generate Random Array», натиснути на кнопку «Sort»
Очікуваний результат	Разом з іншою довідковою інформацією у відповідне текстове поле повинні вивестись повідомлення про збереження початкового і відсортованого станів масиву у текстові файли; текстові файли збережуться в папках «initial_arrays» та «sorted_arrays»
Стан програми після проведення випробувань	У відповідне текстове поле вивелася інформація про збереження початкового і відсортованого станів масиву у текстові файли; в папках «initial_arrays» та «sorted_arrays» з'явилися або перезаписалися файли «initial_array_1» та «sorted_array_1» відповідно

## 6 ІНСТРУКЦІЯ КОРИСТУВАЧА

### 6.1 Робота з програмою

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



Рисунок 6.1 – Головне вікно програми

Далі в поле, яке підписане «Array Size: », вводиться розмірність масиву за допомогою натискання на стрілки, що є частиною керуючого компоненту, або шляхом вводу цілочисельного значення з клавіатури (рис. 6.2). При натисканні на верхню стрілку розмірність масиву в полі зростає на 100, при натисканні на нижню стрілку розмірність масиву в полі зменшується на 100. За замовчуванням значення в полі для введення розмірності масиву складає 100. Найбільше значення, яке можливо ввести – 50000. Найменше – 100.



Рисунок 6.2 – Поле для введення розмірності масиву

З використанням випадного списку, що підписаний «Algorithm: », шляхом натискання на один із варіантів списку обирається алгоритм сортування масиву («Selection Sort», «Heap Sort», «Smooth Sort»). Значенням за замовчування для цього компоненту є «Selection Sort». Випадний список для вибору алгоритму сортування можна побачити на рисунку 6.3:

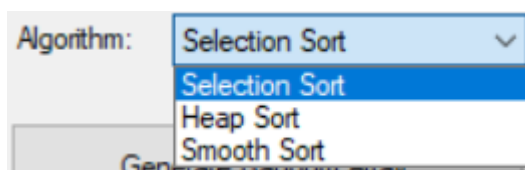


Рисунок 6.3 – Спадний список для вибору алгоритму сортування

Щоб згенерувати масив розміру, що ввів користувач, який складається з випадкових цілих чисел з проміжку від 0 до 50000, потрібно натиснути на кнопку «Generate Random Array» (рис. 6.4).



Рисунок 6.4 – Кнопка для генерації масиву заданого розміру

Після генерації масиву в текстове поле для виводу загальної інформації про роботу застосунку буде записано повідомлення, що повідомлятиме про успішну генерацію масиву заданого розміру і виводитиме 10 послідовних елементів масиву, індекс першого з яких обирається випадковим чином (рис 6.5).

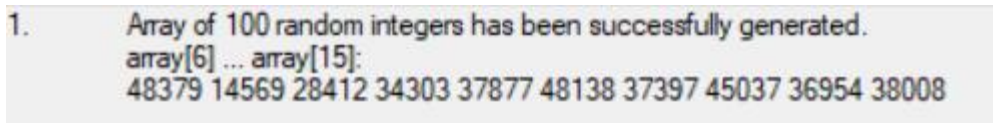


Рисунок 6.5 – Приклад повідомлення про успішну генерацію масиву заданого розміру

Кожне повідомлення в текстовому полі починається з індексу поточного масиву, що буде відсортований.

Для сортування масиву обраним методом необхідно натиснути на кнопку «Sort» (рис. 6.6). Якщо масив попередньо не був згенерований, то кнопка буде недоступною для користувача (рис 6.7).

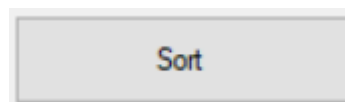


Рисунок 6.6 – Кнопка для сортування масиву обраним методом

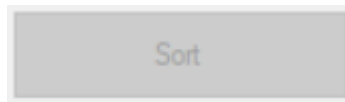


Рисунок 6.7 – Заблоковано кнопка «Sort»

Після завершення процесу сортування в текстове поле виводиться відповідне повідомлення, яке містить розмірність масиву, обраний метод сортування, назву файлу, в який записується масив до сортування, назва файлу, в який записується масив після сортування, кількість перестановок та кількість порівнянь, здійснених в процесі сортування, та 10 послідовних елементів масиву, індекс першого з яких обирається випадковим чином (рис 6.8).



```

1.      Array of 100 random integers has been successfully sorted.
        Initial array has been written to file "initial_array_1.txt".
        Sorted array has been written to file "sorted_array_1.txt".
        array[0] ... array[9]:
        856 1669 1772 2076 2475 2561 3643 4066 4079 4230
        sorting algorithm:  Selection Sort
        comparisons:       4950
        swaps:              100

```

Рисунок 6.8 – Приклад повідомлення про успішне сортування масиву

Індекси, з яких починається кожне повідомлення в текстовому полі, використовуються для найменування текстових файлів, в яких зберігається масив до сортування і масив після сортування.

Текстові файли з масивами до сортування мають назву вигляду «initial\_array\_idx» і зберігаються в папці «initial\_arrays». Приклад вмісту одного з таких файлів (рис. 6.9):

```

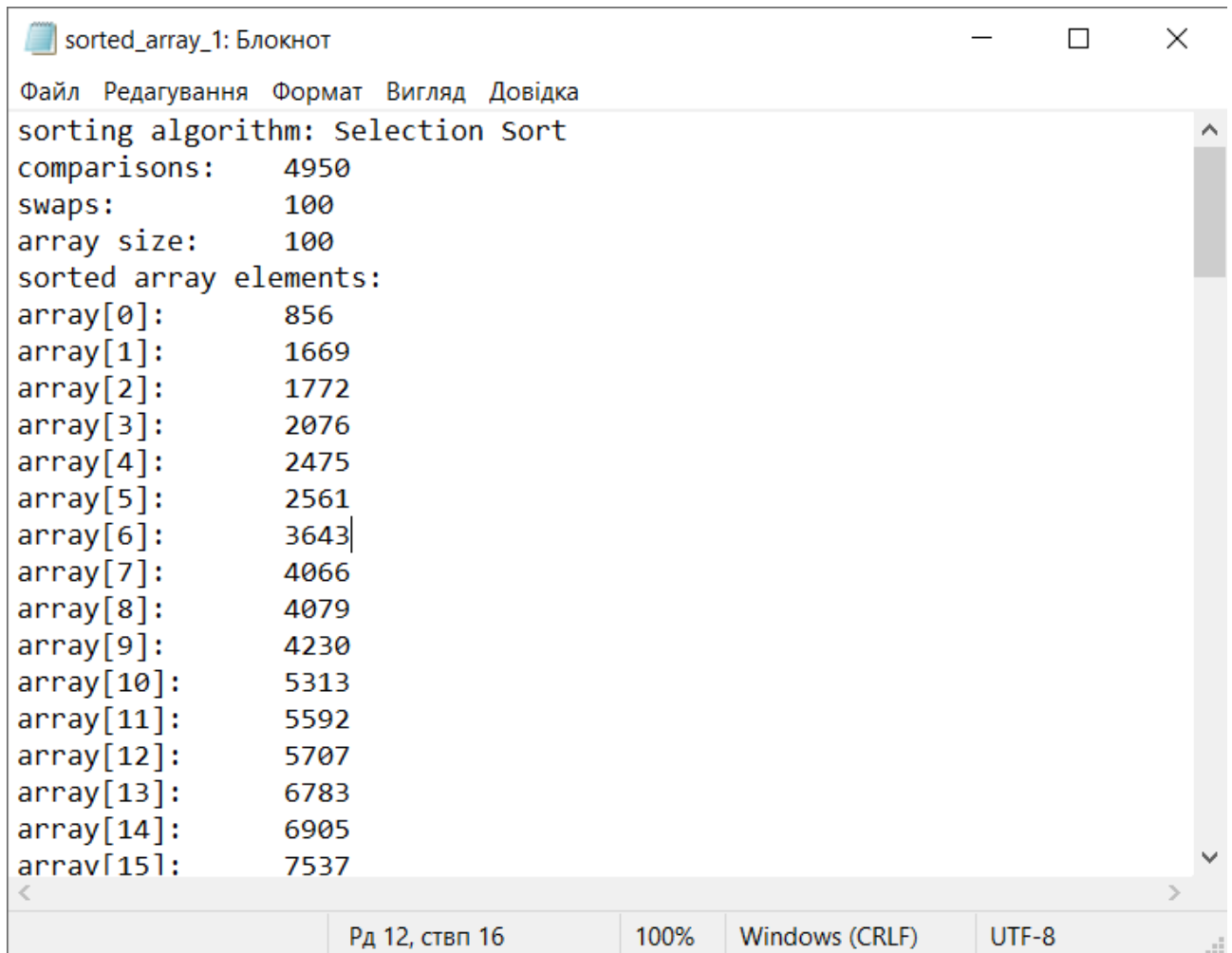
initial_array_1: Блокнот
Файл  Редагування  Формат  Вигляд  Довідка
array size: 100
initial array elements:
array[0]:      17427
array[1]:      11804
array[2]:      41861
array[3]:      45666
array[4]:      22580
array[5]:      34392
array[6]:      29553
array[7]:      30579
array[8]:      45999
array[9]:      22316
array[10]:     35258
array[11]:     23226
array[12]:     22488
array[13]:     47917
array[14]:     39111
array[15]:     8067

```

Рд 1, ствп 1      100%      Windows (CRLF)      UTF-8

Рисунок 6.9 – Приклад вмісту текстового файлу з масивом до сортування

Текстові файли з масивами після сортування мають назву вигляду «sorted\_array\_idx» і зберігаються в папці «sorted\_arrays». Приклад вмісту одного з таких файлів (рис. 6.10):



```

sorted_array_1: Блокнот
Файл Редагування Формат Вигляд Довідка
sorting algorithm: Selection Sort
comparisons:      4950
swaps:            100
array size:       100
sorted array elements:
array[0]:         856
array[1]:         1669
array[2]:         1772
array[3]:         2076
array[4]:         2475
array[5]:         2561
array[6]:         3643
array[7]:         4066
array[8]:         4079
array[9]:         4230
array[10]:        5313
array[11]:        5592
array[12]:        5707
array[13]:        6783
array[14]:        6905
array[15]:        7537
  
```

Рисунок 6.10 – Приклад вмісту текстового файлу з відсортованим масивом

Папки з назвами «initial\_arrays» та «sorted\_arrays» розташовуються в тому ж каталозі, що й виконавчий файл з розширенням \*.exe. Якщо до початку виконання застосунку цих папок не існувало, вони будуть автоматично створені.

Якщо зберегти масив до сортування чи після сортування в текстовий файл не вдалося, то буде виведено відповідне повідомлення (рис. 6.11).

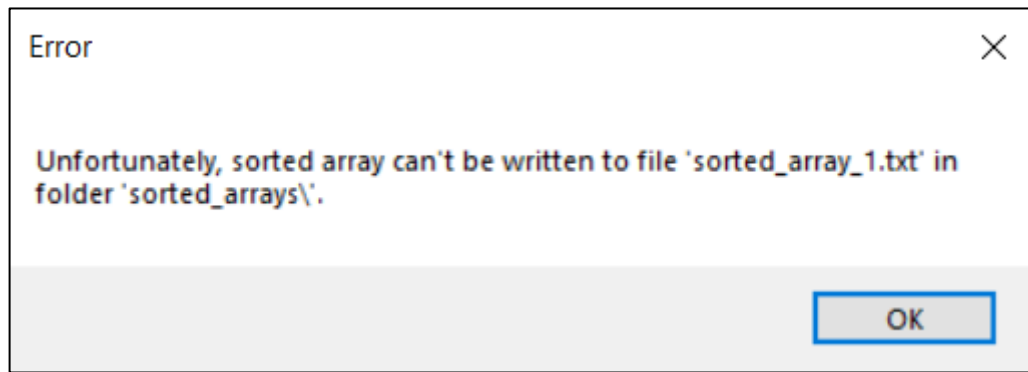


Рисунок 6.11 – Приклад повідомлення про неможливість запису відсортованого масиву до файлу

Щоб сортування масиву супроводжувалося візуалізацією у вигляді рухомих стовпців гістограми, потрібно встановити прапорець, що підписаний «Visualize» (рис. 6.12).

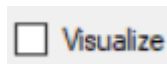


Рисунок 6.12 – Прапорець, який визначає необхідність візуалізації процесу сортування масиву

Якщо розмірність згенерованого масиву більша за 200 елементів, то прапорець «Visualize» стає недоступним (рис. 6.13). Інакше прапорець доступний.

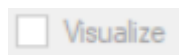


Рисунок 6.13 – Заблокований прапорець «Visualize»

Якщо прапорець «Visualize» встановлений і незаблокований, то при натисканні кнопки «Sort», відкриватиметься нове вікно, в якому буде візуалізовано процес сортування масиву за допомогою рухомих стовпців гістограми (6.14).

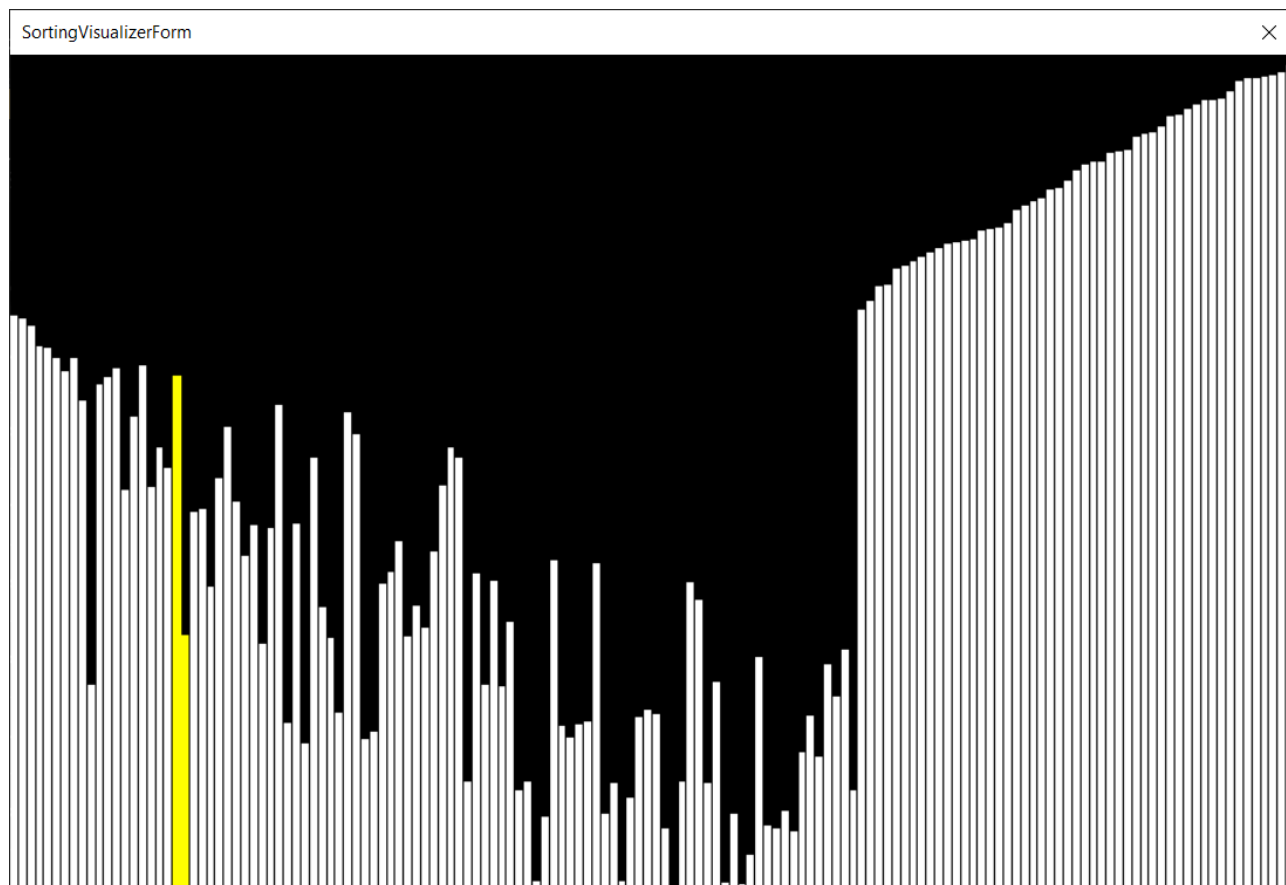


Рисунок 6.14 – Приклад візуалізації процесу сортування в окремому вікні

## 6.2 Формат вхідних та вихідних даних

Користувачем на вхід програми подається розмірність масиву, який потрібно відсортувати, метод сортування, обраний з випадного списку, та встановлений чи не встановлений прапорець «Visualize», що відповідає за необхідність візуалізації процесу сортування масиву. Розмірність масиву є додатним цілим числом в межах від 100 до 50000. Візуалізація процесу сортування обраним методом можлива лише тоді, коли розмірність згенерованого масиву не більша за 200 елементів. Запуск процесу сортування можливий тільки після генерації масиву, яка здійснюється натисканням на кнопку «Generate Random Array».

Результатом виконання програми є відсортований у неспадному порядку масив цілих чисел, який записується до текстового файлу. Окрім відсортованого масиву до файлу також записується його розмірність, алгоритм сортування та кількість порівнянь і перестановок, здійснених в процесі сортування. В окремий файл записується масив до сортування та його розмірність. В текстове поле із

загальною інформацією після завершення сортування виводиться повідомлення про успішне сортування, назва файлу з відсортованим масивом, назва файлу з масивом до сортування, кількість порівнянь та перестановок, здійснених в процесі сортування, та розмірність масиву. Якщо користувач встановив прапорець «Visualize», то процес сортування супроводжується візуалізацією, здійсненою у вигляді рухомих стовпців гістограми.

### 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 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	1 GB RAM	2 GB RAM
Відеоадаптер	ATI Radeon HD з об'ємом VRAM 256 Mb або краще	
Дисплей	1280x720	1920x1080 або краще
Прилади введення	Клавіатура, комп'ютерна миша	
Додаткове програмне забезпечення	Microsoft .NET Framework 4.7.2 або вище	

## 7 АНАЛІЗ І УЗАГАЛЬНЕННЯ РЕЗУЛЬТАТІВ

Головною задачею курсової роботи була реалізація програми для сортування масиву заданого користувачем розміру, що складається з випадкових цілих чисел, наступними методами: сортуванням вибором, пірамідальним сортуванням та плавним сортуванням.

Критичні ситуації у роботі програми під час тестування виявлені не були. Дані, що вводять, користувач ретельно контролюються ще на етапі введення, що дозволяє запобігти виникненню потенційних помилок.

Для перевірки того, чи власні реалізації алгоритмів сортування сортують масив у неспадному порядку, зміню програмний код, що обробляє натискання на кнопку «Sort» таким чином: перед початком сортування обраним методом буде створено додатковий масив, розмір якого дорівнює розміру вихідного масиву; в додатковий масив буде скопійовано всі елементи вихідного стандартним методом `Array::CopyTo()`; далі відбудеться сортування вихідного масиву обраним методом та сортування масиву-копії стандартним методом `Array::Sort()`; після цього в циклі здійснюється поелементне порівняння масиву-оригіналу та масиву-копії; якщо значення всіх відповідних елементів масиву-оригіналу і масиву-копії збігаються, то виводиться повідомлення, що даний алгоритм сортування працює правильно, інакше виводиться повідомлення, що алгоритм сортування працює хибно.

Перевірка коректності роботи кожного алгоритму за допомогою модифікованого застосунку:

а) Алгоритм сортування вибором.

Результат виконання модифікованого застосунку після натискання кнопки «Sort», коли обрано алгоритм сортування вибором (рис. 7.1):

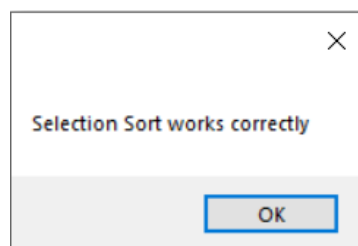


Рисунок 7.1 – Повідомлення про коректність роботи власної реалізації алгоритму сортування вибором

б) Алгоритм пірамідального сортування.

Результат виконання модифікованого застосунку після натискання кнопки «Sort», коли обрано алгоритм пірамідального сортування (рис. 7.2):

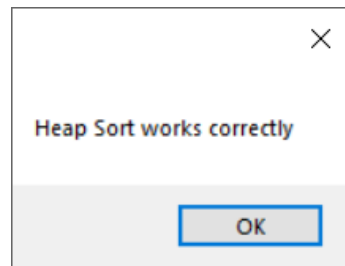


Рисунок 7.2 – Повідомлення про коректність роботи власної реалізації алгоритму пірамідального сортування

в) Алгоритм плавного сортування.

Результат виконання модифікованого застосунку після натискання кнопки «Sort», коли обрано алгоритм плавного сортування (рис. 7.3):

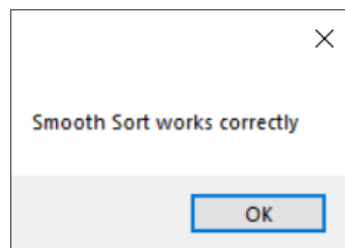


Рисунок 7.3 – Повідомлення про коректність роботи власної реалізації алгоритму плавного сортування

Отже, всі власні реалізації алгоритмів сортування сортують масив в порядку неспадання.

Асимптотична складність алгоритму сортування вибором складає  $O(n^2)$ . Аналітично це доводиться так: операція знаходження індексу найменшого елементу передбачає використання одного циклу, що проходить по всіх елементах масиву, починаючи з елемента з вказаним індексом, отже, складність операції –  $O(n)$ . Ця операція здійснюється для індексу кожного елементу масиву, отже, загальна складність –  $O(n^2)$ .

Експериментально доведу це твердження за допомогою Excel.

На рисунку 7.4 можна побачити таблицю, в якій наведено знаходження коефіцієнту  $k$  – частки від ділення кількості порівнянь, здійснених в процесі

сортування масиву заданого розміру алгоритмом сортування вибором, на значення розмірності масиву в квадраті (заявлена складність алгоритму).

Розмір масиву	Сортування вибором (порівняння)	$n^2$	Коефіцієнт k
5000	12497500	25000000	0,4999
10000	49995000	100000000	0,49995
15000	112492500	225000000	0,499966667
20000	199990000	400000000	0,499975
25000	312487500	625000000	0,49998
30000	449985000	900000000	0,499983333
35000	612482500	1225000000	0,499985714
40000	799980000	1600000000	0,4999875
45000	1012477500	2025000000	0,499988889
50000	1249975000	2500000000	0,49999

Рисунок 7.4 – Процес експериментального доведення асимптотичної складності алгоритму сортування вибором засобами Excel

Оскільки знайдені значення коефіцієнту  $k$  рівні, роблю висновок, що асимптотична складність алгоритму сортування вибором справді складає  $O(n^2)$ .

Асимптотична складність алгоритму пірамідального сортування складає  $O(n \log n)$ . Аналітично це доводиться так: операція «просійки вниз» має складність  $O(\log n)$ , оскільки вона максимум проходить висоту сортувального дерева, яка складає  $\log_2 n$ , де  $n$  – кількість елементів в дереві. Дана операція використовується  $\frac{n}{2}$  рази для побудови бінарної купи і  $n$  раз для остаточного сортування елементів масиву. Отже, складність всього алгоритму –  $O(n \log n)$ .

Експериментально доведу це твердження за допомогою Excel.

На рисунку 7.5 можна побачити таблицю, в якій наведено знаходження коефіцієнту  $k$  – частки від ділення кількості порівнянь, здійснених в процесі сортування масиву заданого розміру алгоритмом пірамідального сортування, на значення розмірності масиву помножене на логарифм за основою два від розмірності масиву (заявлена складність алгоритму).



Розмір масиву	Пірамідальне сортування (порівняння)	$n \cdot \log_2(n)$	Коефіцієнт k
5000	107674	61438,5619	1,752547532
10000	235358	132877,1238	1,771245443
15000	370380	208090,1232	1,779901873
20000	510876	285754,2476	1,787815944
25000	654703	365241,0119	1,792523235
30000	800676	446180,2464	1,794512434
35000	949209	528327,3556	1,796630422
40000	1101636	611508,4952	1,801505635
45000	1254989	695593,6821	1,804198388
50000	1409524	780482,0237	1,805966002
5000000	207318327	111267483,3	1,863242709

Рисунок 7.5 – Процес експериментального доведення асимптотичної складності алгоритму пірамідального сортування засобами Excel

Оскільки знайдені значення коефіцієнту  $k$ , хоч і зростають, але роблять це дуже повільно (значення коефіцієнту для розміру масиву 50000 і 5000000 відрізняються всього на  $\approx 6,3\%$ ) роблю висновок, що  $\lim_{n \rightarrow \infty} k = \text{const}$ . Отже, складність алгоритму пірамідального сортування справді складає  $O(n \log n)$ .

Асимптотична складність алгоритму плавного сортування складає  $O(n \log n)$ . Аналітично це доводиться так: операція «просійки вниз» має складність  $O(\log n)$ , як було сказано вище. Інші операції, що використовуються для розрахунку попереднього і наступного станів послідовності куп та знаходження максимального кореня мають складність  $O(h)$ , де  $h$  – кількість задіяних чисел Леонардо. Оскільки для будь-якого практичного використання кількість чисел Леонардо, які задіяні в процесі сортування, не може перевищувати 64 (максимальна кількість біт цілого беззнакового 64-розрядного числа), то можна вважати, що ці операції мають складність  $O(1)$ . Операція «просійки вниз» використовується  $n$  разів для побудови послідовності куп і  $n$  разів для остаточного сортування елементів масиву. Отже, складність всього алгоритму –  $O(n \log n)$ .

Експериментально доведу це твердження за допомогою Excel.

На рисунку 7.6 можна побачити таблицю, в якій наведено знаходження коефіцієнту  $k$  – частки від ділення кількості порівнянь, здійснених в процесі сортування масиву заданого розміру алгоритмом плавного сортування, на

значення розмірності масиву помножене на логарифм за основою два від розмірності масиву (заявлена складність алгоритму).

Розмір масиву	Плавне сортування (порівняння)	$n \cdot \log_2(n)$	Коефіцієнт k
5000	118874	61438,5619	1,934843465
10000	261882	132877,1238	1,970858433
15000	414384	208090,1232	1,99136794
20000	572512	285754,2476	2,003511776
25000	738114	365241,0119	2,020895726
30000	903360	446180,2464	2,024652609
35000	1072552	528327,3556	2,030089846
40000	1245864	611508,4952	2,037361721
45000	1423180	695593,6821	2,045993281
50000	1599338	780482,0237	2,049166991
5000000	242304958	111267483,3	2,177679864

Рисунок 7.6 – Процес експериментального доведення асимптотичної складності алгоритму плавного сортування засобами Excel

Оскільки знайдені значення коефіцієнту k, хоч і зростають, але роблять це дуже повільно (значення коефіцієнту для розміру масиву 50000 і 5000000 відрізняються всього на  $\approx 6,9\%$ ) роблю висновок, що  $\lim_{n \rightarrow \infty} k = \text{const}$ . Отже, складність алгоритму пірамідального сортування справді складає  $O(n \log n)$ .

Результати тестування ефективності алгоритмів сортування наведено в таблицях 7.1-7.4.

Таблиця 7.1 – Тестування ефективності алгоритмів для масивів, елементами яких є випадкові числа

Розмір масивів	Параметри тестування	Алгоритм		
		Сортування вибором	Пірамідальне сортування	Плавне сортування
5000	Кількість порівнянь	12497500	107674	118874
	Кількість перестановок	5000	57086	50846
10000	Кількість порівнянь	49995000	235358	261882
	Кількість перестановок	10000	124119	111986
15000	Кількість порівнянь	112492500	370380	414384
	Кількість перестановок	15000	194999	177164
20000	Кількість порівнянь	199990000	510876	572512
	Кількість перестановок	20000	268515	244373
25000	Кількість порівнянь	312487500	654703	738114
	Кількість перестановок	25000	343557	315327

## Продовження таблиці 7.1

Розмір масивів	Параметри тестування	Алгоритм		
		Сортування вибором	Пірамідальне сортування	Плавне сортування
30000	Кількість порівнянь	449985000	800676	903360
	Кількість перестановок	30000	420008	385358
35000	Кількість порівнянь	612482500	949209	1072552
	Кількість перестановок	35000	497470	456206
40000	Кількість порівнянь	799980000	1101636	1245864
	Кількість перестановок	40000	576951	531556
45000	Кількість порівнянь	1012477500	1254989	1423180
	Кількість перестановок	45000	656892	607669
50000	Кількість порівнянь	1249975000	1409524	1599338
	Кількість перестановок	50000	737272	681708

Візуалізації результатів таблиці 7.1 наведено на рисунках 7.7-7.9:

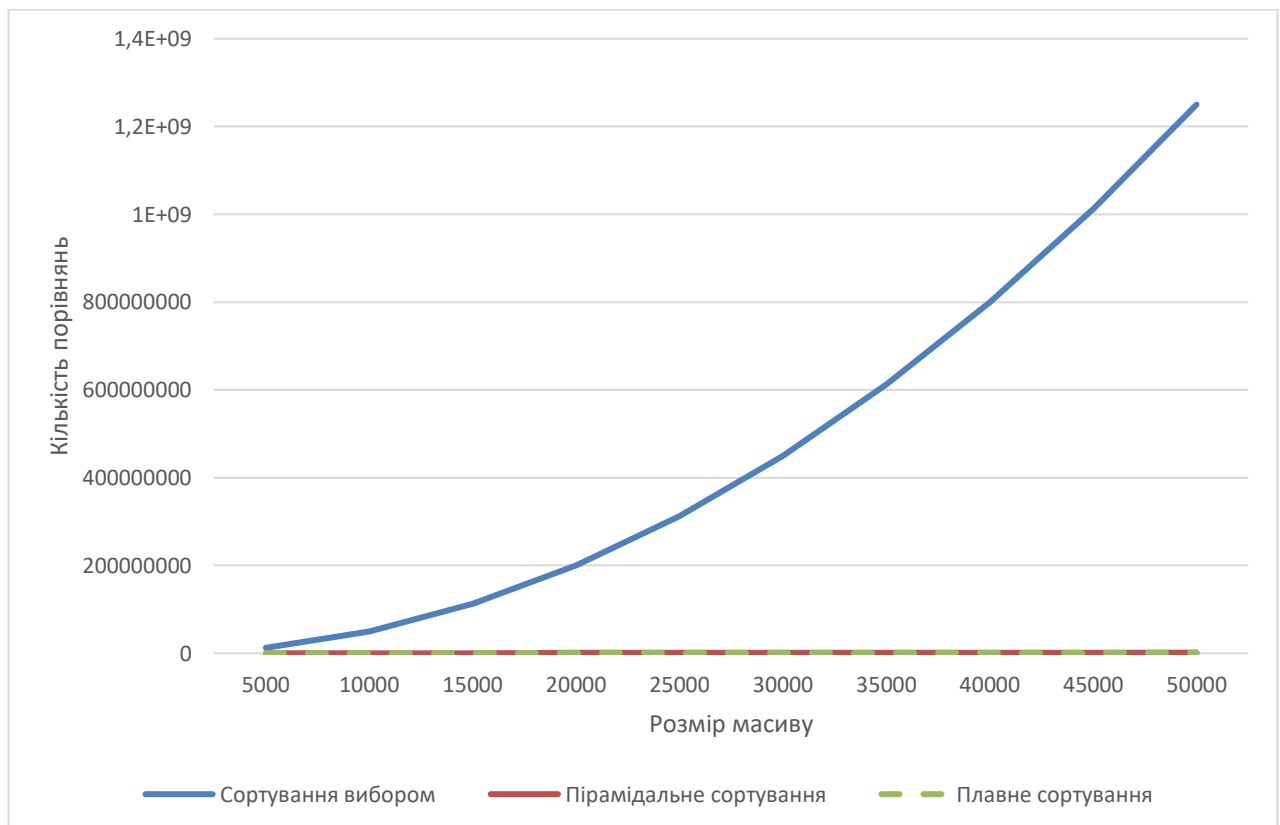


Рисунок 7.7 – Графік залежності кількості порівнянь від розміру вхідного масиву для трьох алгоритмів сортування, якщо масив складається з випадкових чисел

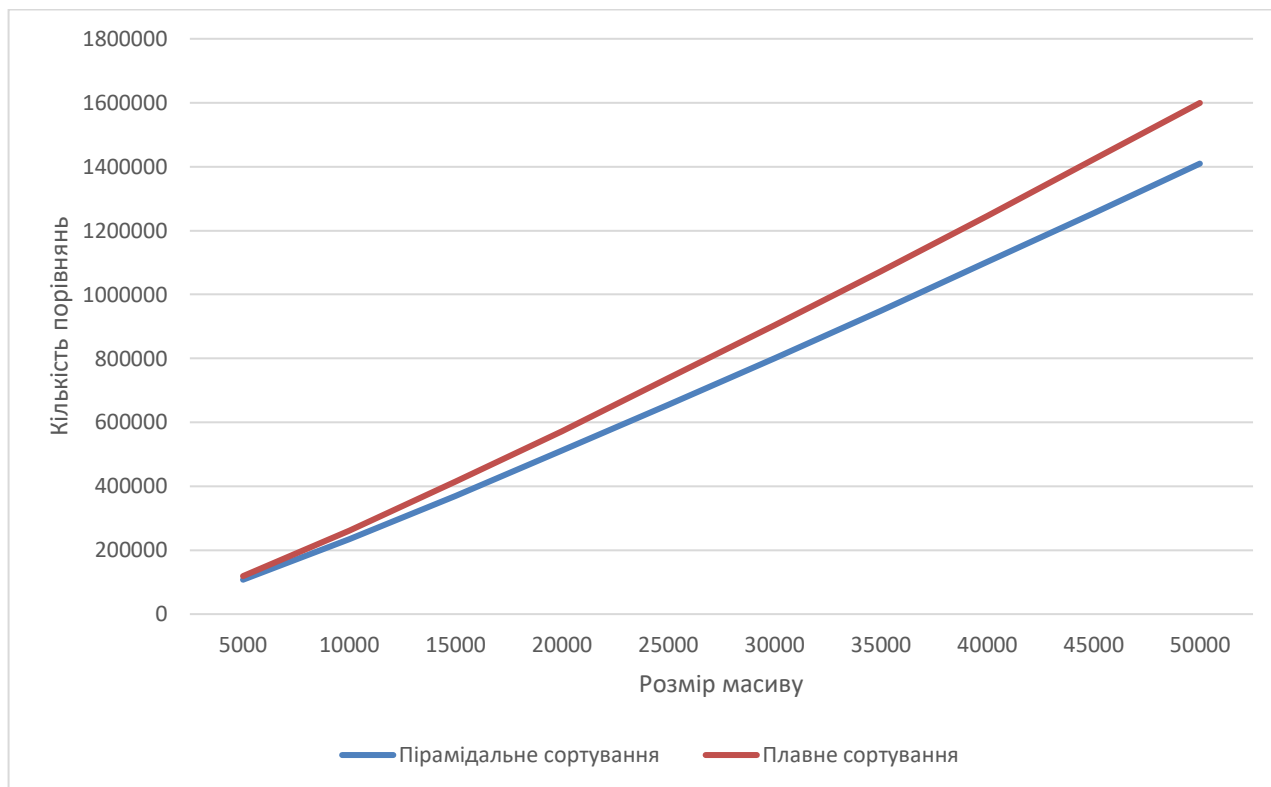


Рисунок 7.8 – Графік залежності кількості порівнянь від розміру вхідного масиву для алгоритмів пірамідального і плавного сортування, якщо масив складається з випадкових чисел

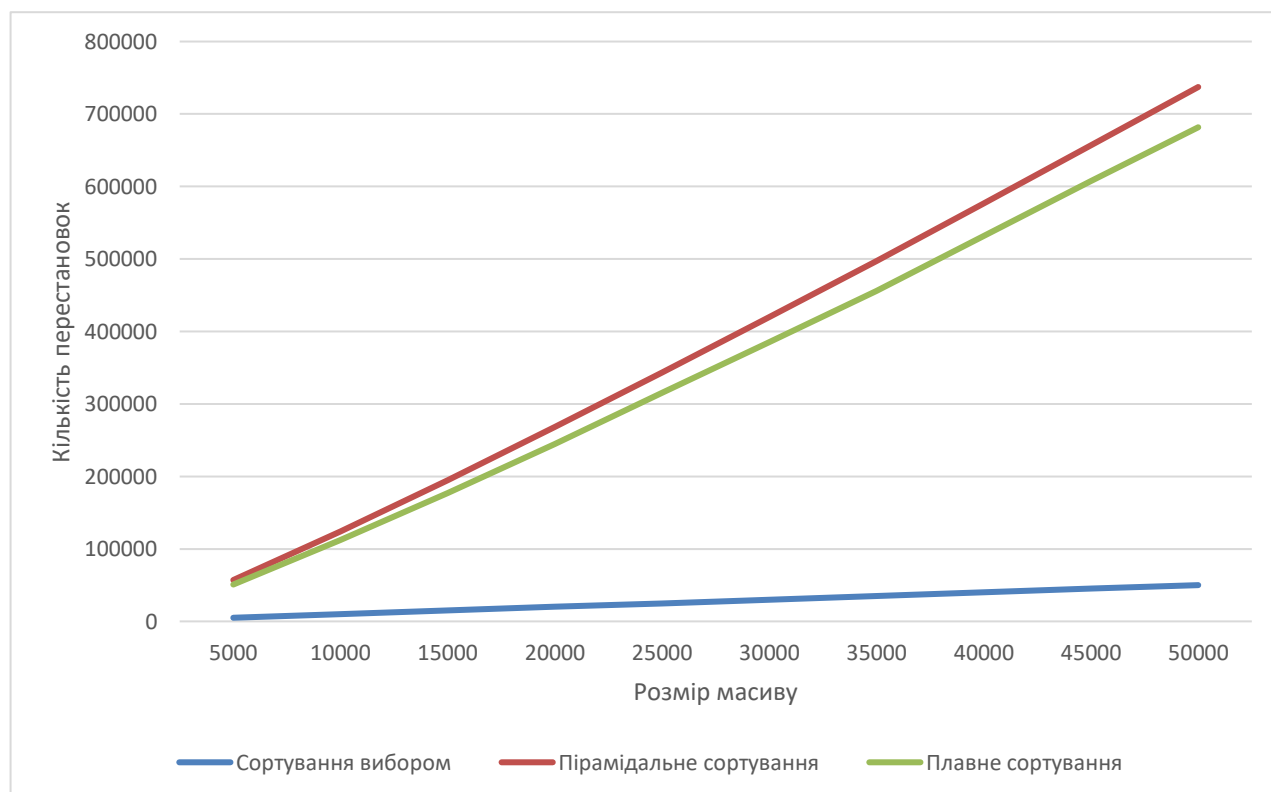


Рисунок 7.9 – Графік залежності кількості перестановок від розміру вхідного масиву для трьох алгоритмів сортування, якщо масив складається з випадкових чисел

Таблиця 7.2 – Тестування ефективності алгоритмів для масивів, що майже відсортовані

Розмір масивів	Параметри тестування	Алгоритм		
		Сортування вибором	Пірамідальне сортування	Плавне сортування
5000	Кількість порівнянь	12497500	110287	47022
	Кількість перестановок	5000	59772	13298
10000	Кількість порівнянь	49995000	241364	98902
	Кількість перестановок	10000	129844	27331
15000	Кількість порівнянь	112492500	375186	177370
	Кількість перестановок	15000	200958	54955
20000	Кількість порівнянь	199990000	523250	204612
	Кількість перестановок	20000	279974	53835
25000	Кількість порівнянь	312487500	661520	323292
	Кількість перестановок	25000	352710	101875
30000	Кількість порівнянь	449985000	809806	380460
	Кількість перестановок	30000	431527	117008
35000	Кількість порівнянь	612482500	955425	563078
	Кількість перестановок	35000	508161	196555
40000	Кількість порівнянь	799980000	1128700	423172
	Кількість перестановок	40000	601049	107177
45000	Кількість порівнянь	1012477500	1283949	490206
	Кількість перестановок	45000	683067	126250
50000	Кількість порівнянь	1249975000	1426118	689828
	Кількість перестановок	50000	757356	215905

Візуалізації результатів таблиці 7.2 наведено на рисунках 7.10-7.11:

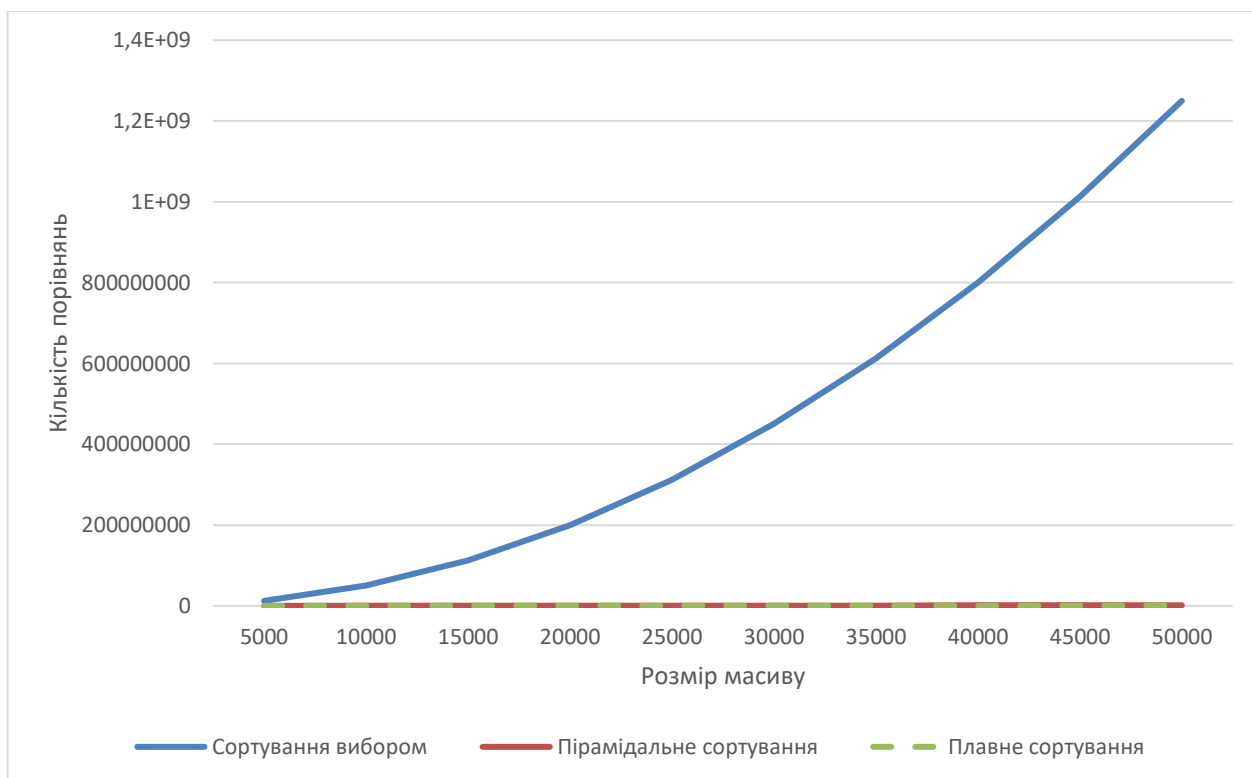


Рисунок 7.10 – Графік залежності кількості порівнянь від розміру вхідного масиву для трьох алгоритмів сортування, якщо масив майже відсортовано

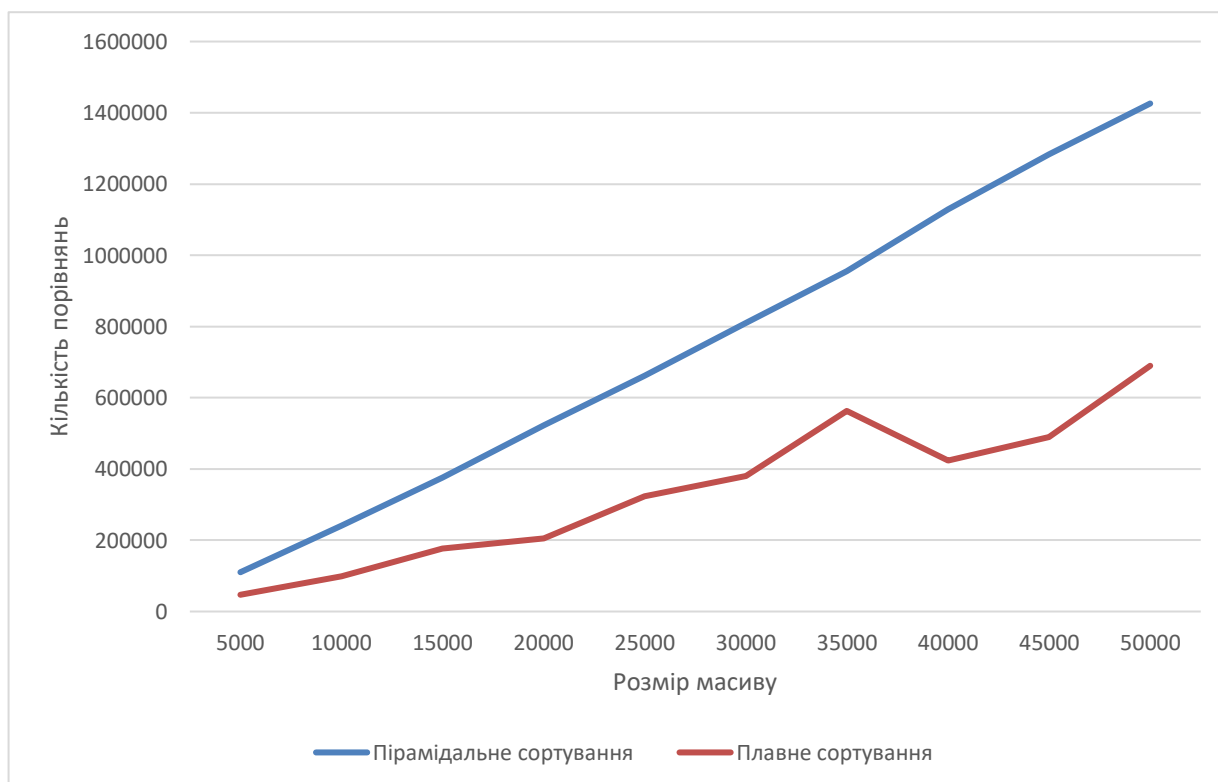


Рисунок 7.11 – Графік залежності кількості порівнянь від розміру вхідного масиву для алгоритмів пірамідального і плавного сортування, якщо масив майже відсортовано

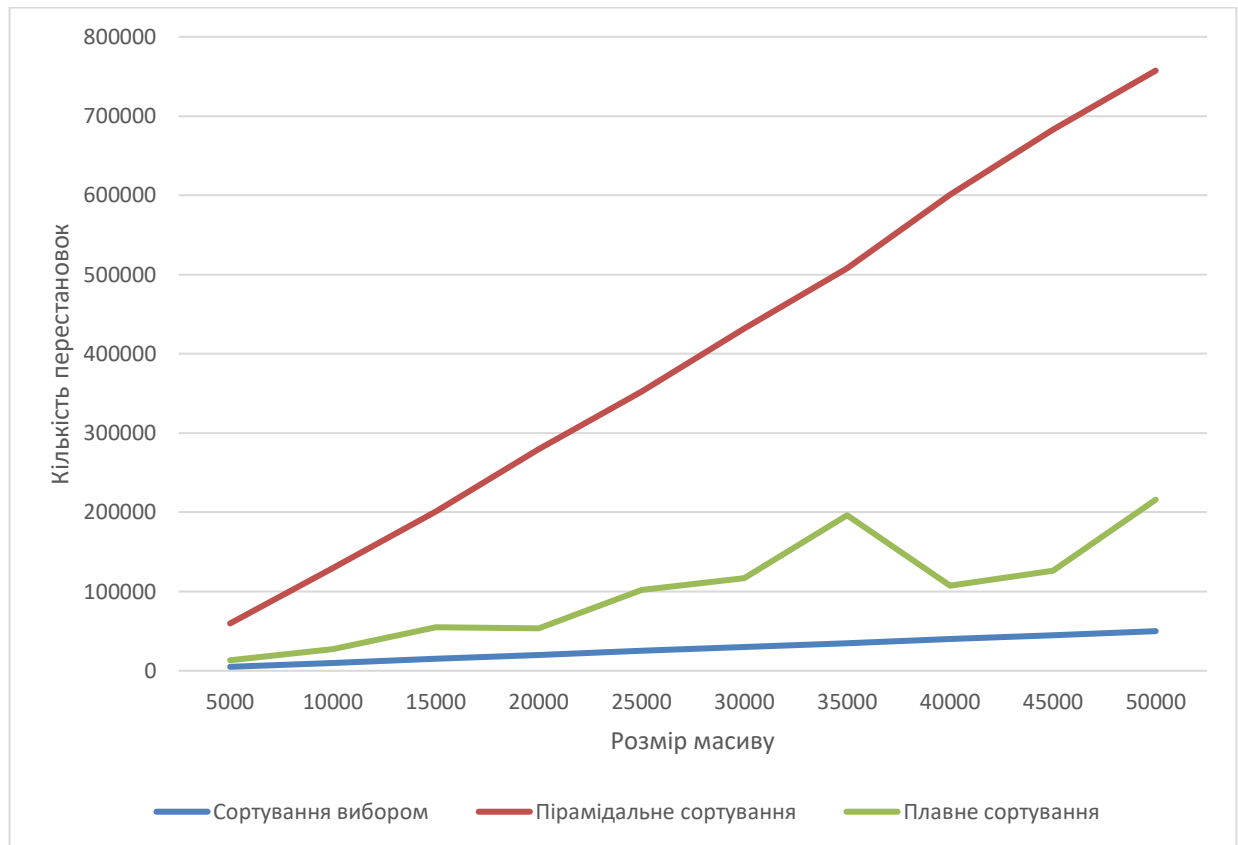


Рисунок 7.12 – Графік залежності кількості перестановок від розміру вхідного масиву для трьох алгоритмів сортування, якщо масив майже відсортовано

Таблиця 7.3 – Тестування ефективності алгоритмів для масивів, що є відсортованими в порядку неспадання

Розмір масивів	Параметри тестування	Алгоритм		
		Сортування вибором	Пірамідальне сортування	Плавне сортування
5000	Кількість порівнянь	12497500	112126	25096
	Кількість перестановок	5000	60932	0
10000	Кількість порівнянь	49995000	244460	53490
	Кількість перестановок	10000	131956	0
15000	Кількість порівнянь	112492500	383177	83874
	Кількість перестановок	15000	205644	0
20000	Кількість порівнянь	199990000	529074	115636
	Кількість перестановок	20000	282878	0
25000	Кількість порівнянь	312487500	677688	147144
	Кількість перестановок	25000	361454	0

Продовження таблиці 7.3

Розмір масивів	Параметри тестування	Алгоритм		
		Сортування вибором	Пірамідальне сортування	Плавне сортування
30000	Кількість порівнянь	449985000	826347	180324
	Кількість перестановок	30000	440100	0
35000	Кількість порівнянь	612482500	979446	215404
	Кількість перестановок	35000	521128	0
40000	Кількість порівнянь	799980000	1138114	246082
	Кількість перестановок	40000	605202	0
45000	Кількість порівнянь	1012477500	1296774	279652
	Кількість перестановок	45000	689270	0
50000	Кількість порівнянь	1249975000	1455438	315000
	Кількість перестановок	50000	773304	0

Візуалізації результатів таблиці 7.2 наведено на рисунках 7.13-7.15:

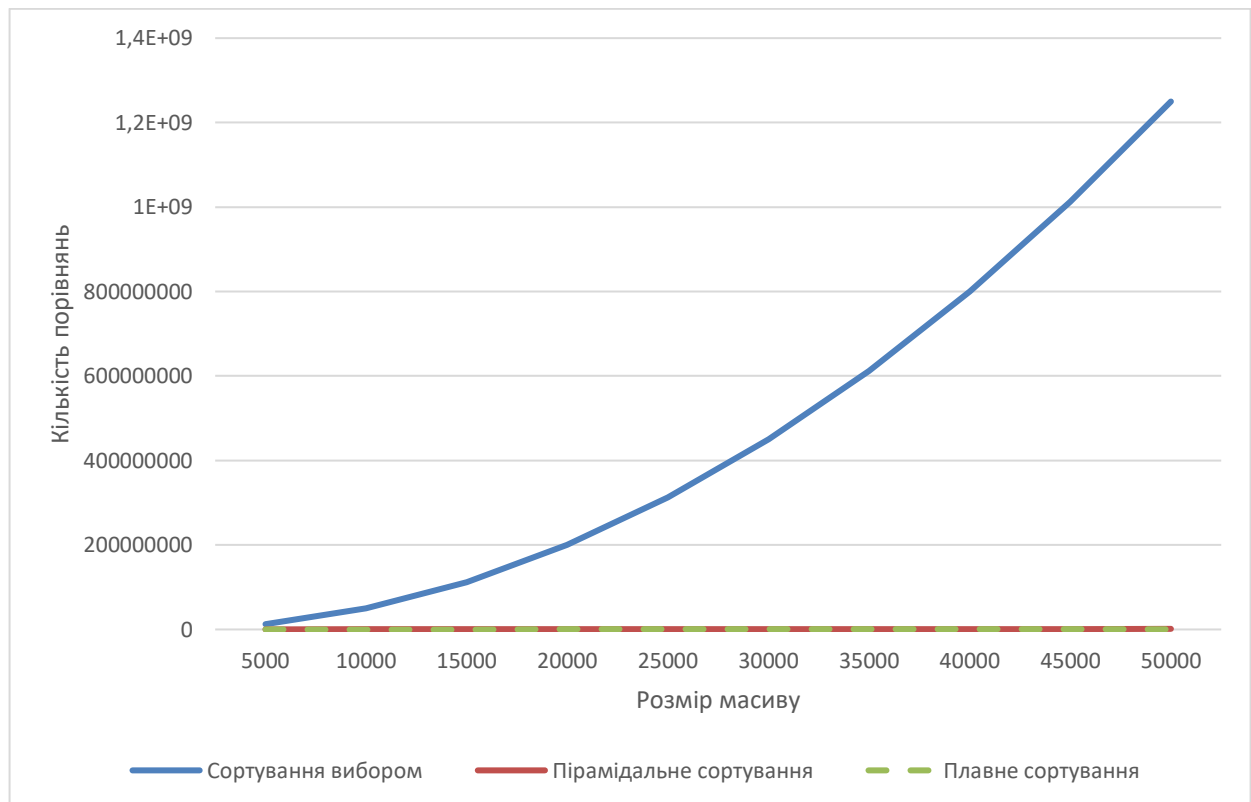


Рисунок 7.13 – Графік залежності кількості порівнянь від розміру вхідного масиву для трьох алгоритмів сортування, якщо масив відсортовано



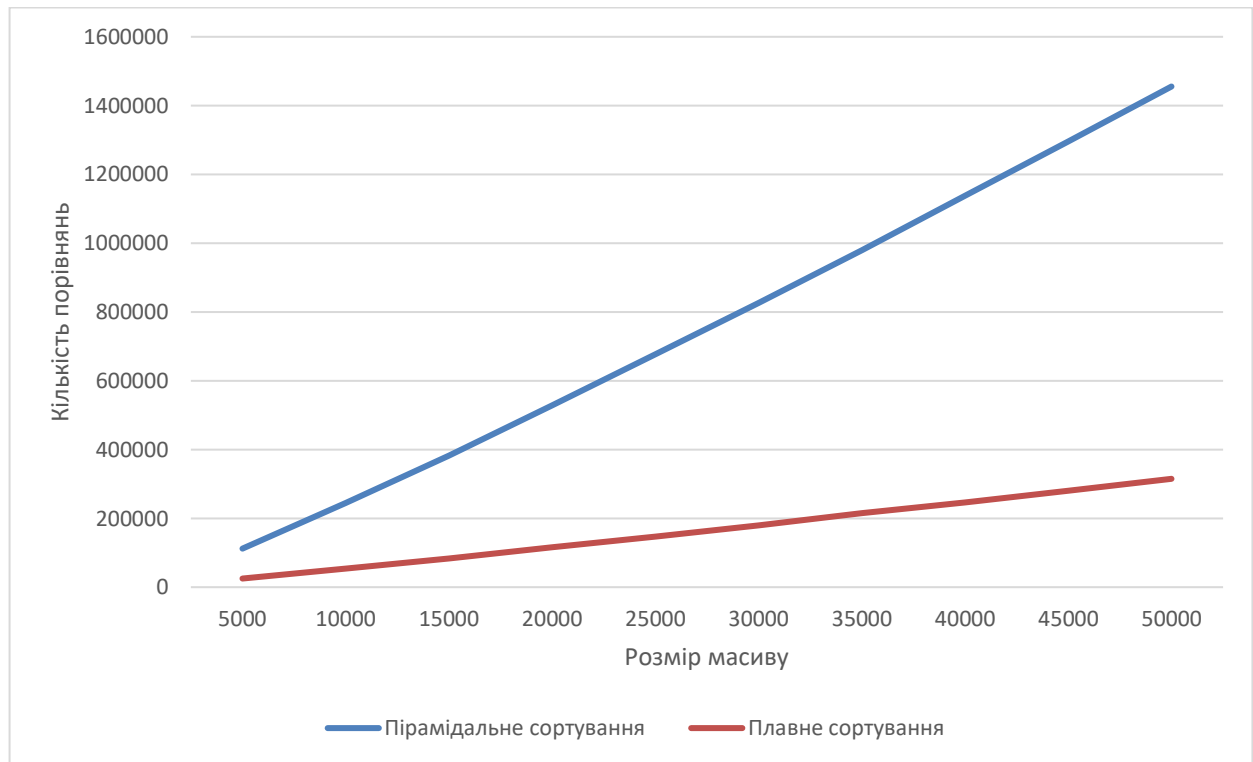


Рисунок 7.14 – Графік залежності кількості порівнянь від розміру вхідного масиву для алгоритмів пірамідального і плавного сортування, якщо масив відсортовано

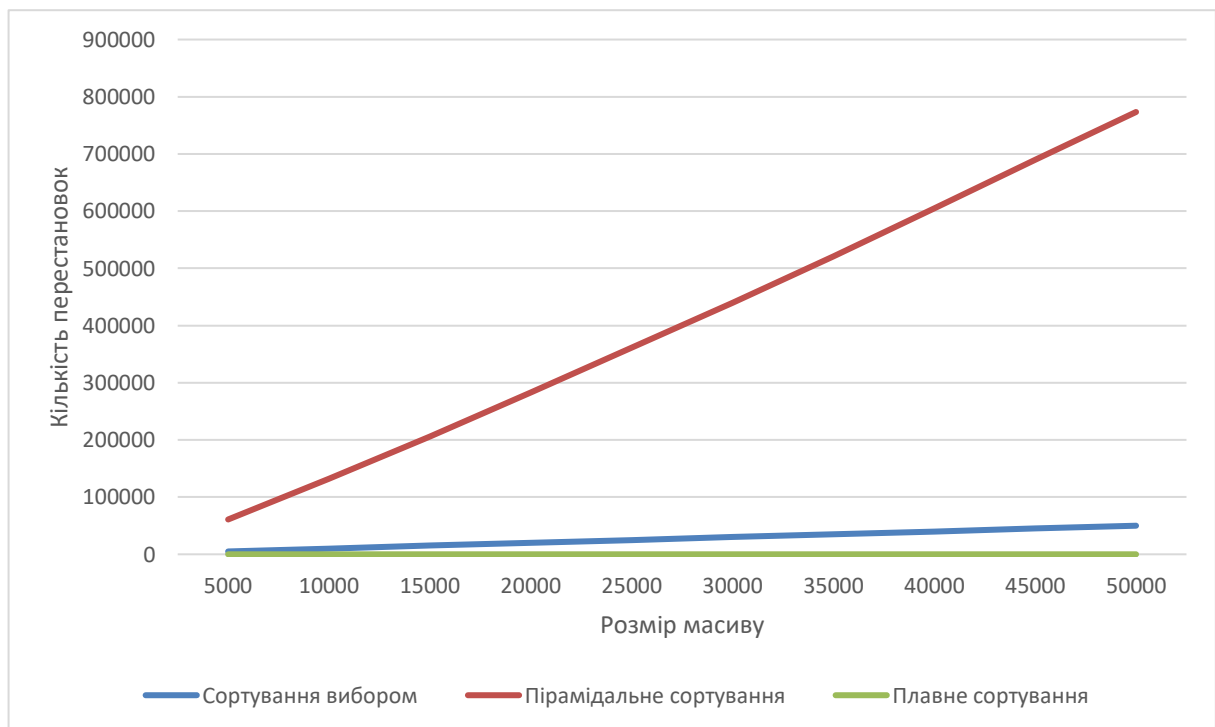


Рисунок 7.15 – Графік залежності кількості перестановок від розміру вхідного масиву для трьох алгоритмів сортування, якщо масив відсортовано  
Візуалізації результатів таблиці 7.4 наведено на рисунках 7.10-7.12:

Таблиця 7.4 – Тестування ефективності алгоритмів для масивів, що є відсортованими в порядку незростання

Розмір масивів	Параметри тестування	Алгоритм		
		Сортування вибором	Пірамідальне сортування	Плавне сортування
5000	Кількість порівнянь	12497500	103227	98288
	Кількість перестановок	5000	53436	41160
10000	Кількість порівнянь	49995000	226682	209852
	Кількість перестановок	10000	116696	87030
15000	Кількість порівнянь	112492500	357789	326842
	Кількість перестановок	15000	184182	134908
20000	Кількість порівнянь	199990000	493307	447804
	Кількість перестановок	20000	254334	184136
25000	Кількість порівнянь	312487500	633719	580204
	Кількість перестановок	25000	326586	239142
30000	Кількість порівнянь	449985000	775687	696670
	Кількість перестановок	30000	399212	284600
35000	Кількість порівнянь	612482500	921588	821900
	Кількість перестановок	35000	473556	333970
40000	Кількість порівнянь	799980000	1067779	959848
	Кількість перестановок	40000	547628	393422
45000	Кількість порівнянь	1012477500	1217920	1099046
	Кількість перестановок	45000	623842	450784
50000	Кількість порівнянь	1249975000	1366047	1226196
	Кількість перестановок	50000	698892	500230

Візуалізації результатів таблиці 7.4 наведено на рисунках 7.16-7.18:

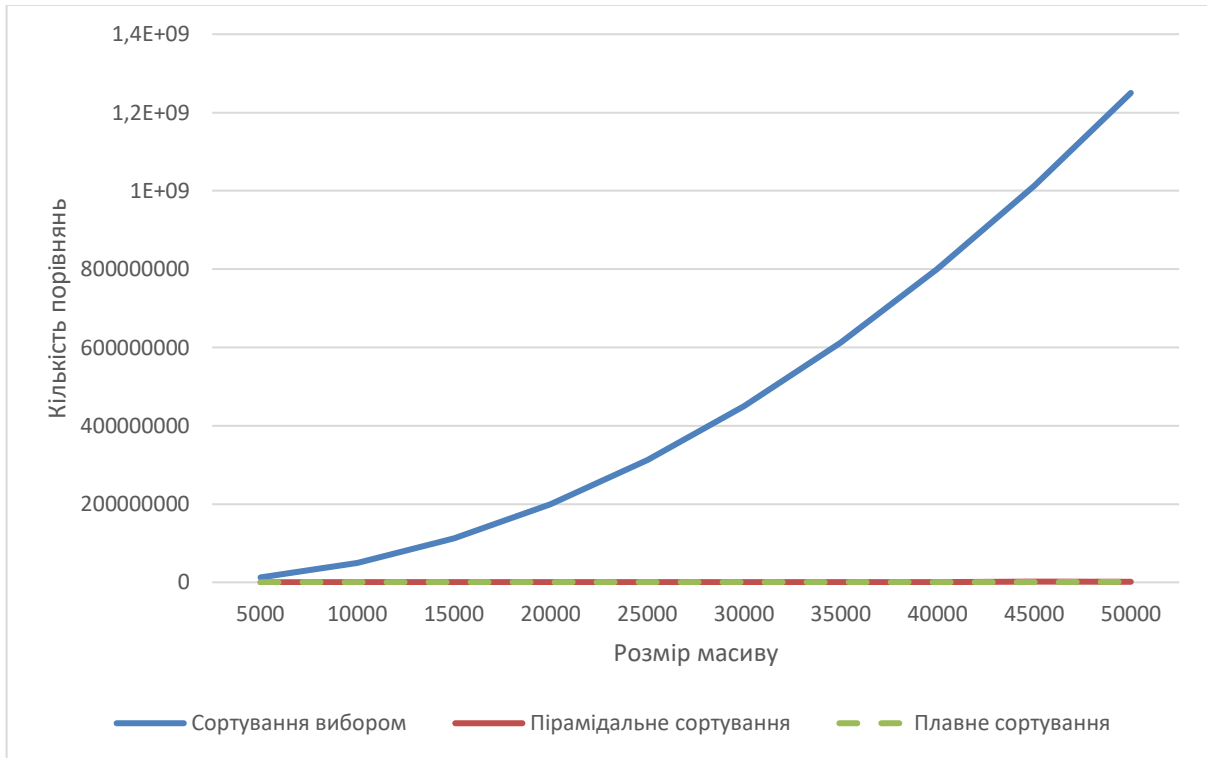


Рисунок 7.16 – Графік залежності кількості порівнянь від розміру вхідного масиву для трьох алгоритмів сортування, якщо масив відсортовано в зворотному порядку

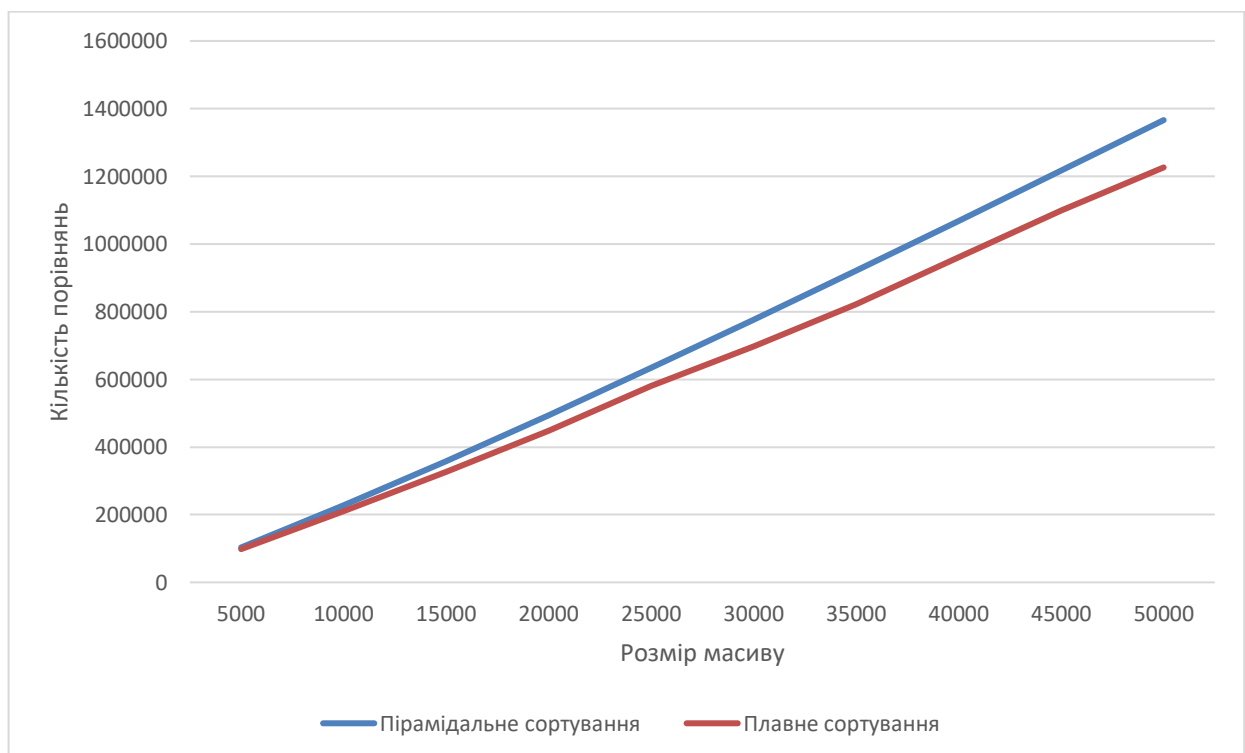


Рисунок 7.17 – Графік залежності кількості порівнянь від розміру вхідного масиву для алгоритмів пірамідального і плавного сортування, якщо масив відсортовано в зворотному порядку

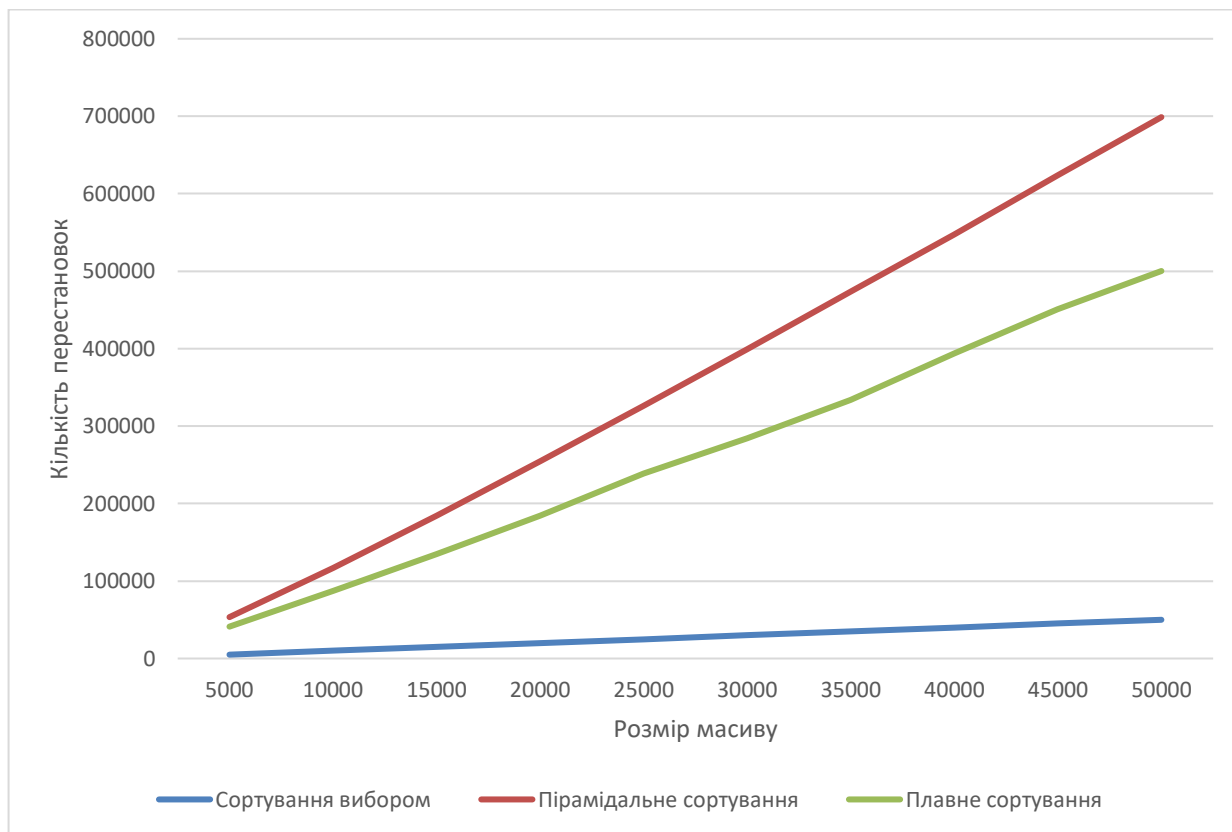


Рисунок 7.18 – Графік залежності кількості перестановок від розміру вхідного масиву для трьох алгоритмів сортування, якщо масив відсортовано в зворотному порядку

Графік залежності кількості порівнянь від розмірності масиву для кожного алгоритму наведено на рисунках 7.19-7.21:



Рисунок 7.19 – Графік залежності кількості порівнянь від розмірності масиву для алгоритму сортування вибором

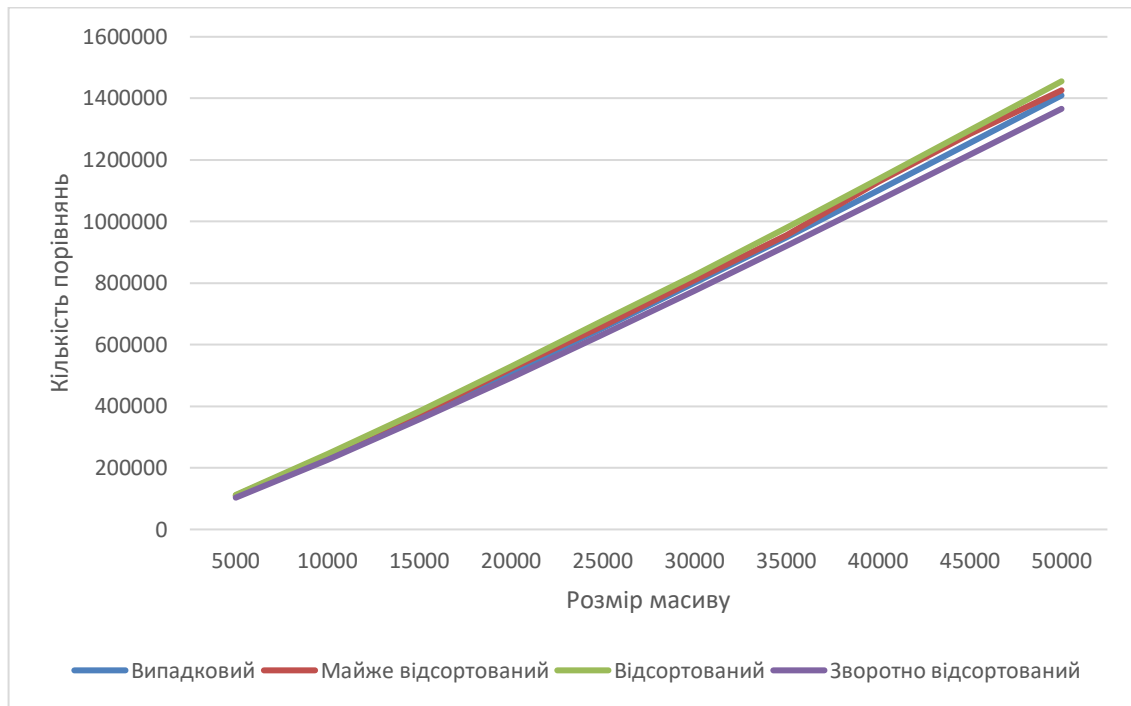


Рисунок 7.20 – Графік залежності кількості порівнянь від розмірності масиву для алгоритму пірамідального сортування

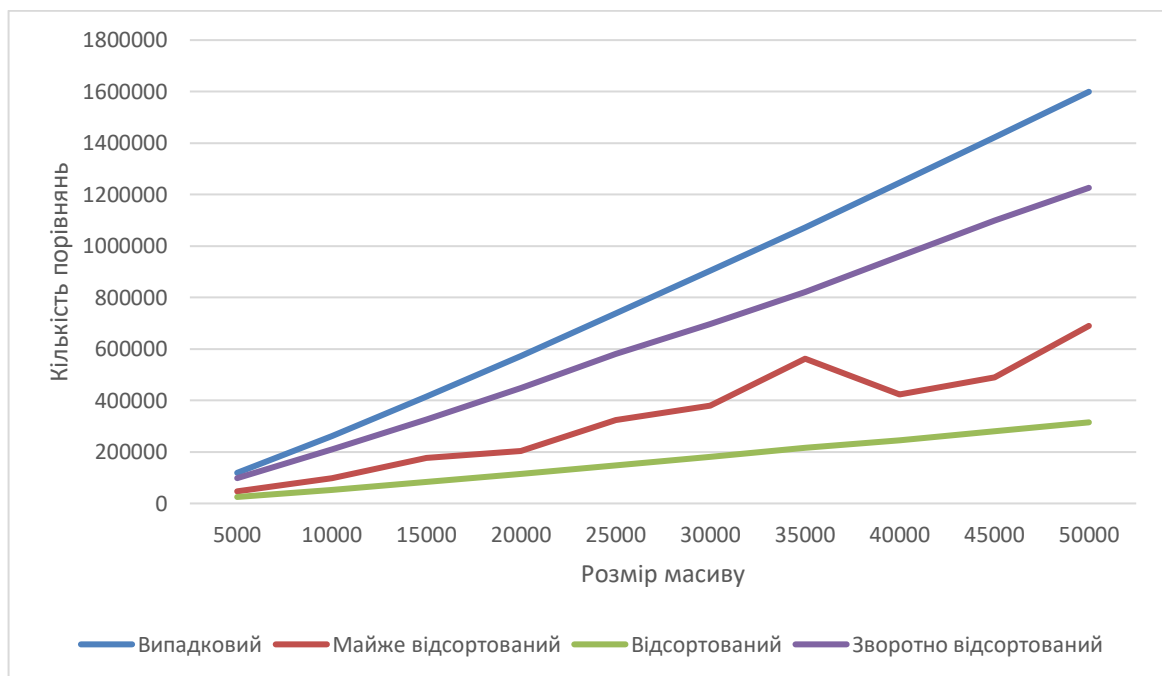


Рисунок 7.21 – Графік залежності кількості порівнянь від розмірності масиву для алгоритму плавного сортування

За результатами тестування можна зробити такі висновки:

- Всі розглянуті алгоритми сортування дозволяють відсортовувати масиви незначних розмірів в порядку неспадання. Алгоритми пірамідального та плавного сортування підходять для сортування

масивів великих та надвеликих розмірів, в той час як сортування вибором є неефективним для таких обсягів даних.

- б) Асимптотична складність алгоритму сортування вибором є квадратичною, тобто –  $O(n^2)$  для будь-яких наборів даних. Асимптотична складність алгоритму пірамідального сортування –  $O(n \log n)$  також для будь-яких наборів даних. Асимптотична складність плавного сортування складає  $O(n \log n)$  для будь-яких наборів даних і близька до  $O(n)$  для таких наборів даних, що близькі до відсортованості.
- в) З розглянутих алгоритмів сортування алгоритм плавного сортування є найоптимальнішим для практичного використання. Його перевагою над сортуванням вибором є значно вища швидкість виконання, а над пірамідальним сортування – те, що його ефективність зростає при роботі з масивами, що близькі до відсортованості.

## ВИСНОВКИ

В даній курсовій роботі було досліджено алгоритми сортування вибором, пірамідального сортування та плавного сортування. Було виконано програмну реалізацію досліджених методів з використанням ООП парадигми засобами мови C++ і елементами C++/CLI та Windows Forms.

В розділі «Постановка задачі» було сформульовано основні вимоги до процесу роботи програми, вхідних та вихідних даних.

В розділі «Теоретичні відомості» буде описано, що собою являють алгоритми сортування, сформульовано особливості кожного з необхідних для реалізації методів сортування (зокрема, асимптотичну складність та необхідність знань про структуру даних «бінарне сортувальне дерево» для реалізації пірамідального та плавного сортування) та коротко розібрано основний алгоритм кожного з них.

В розділі «Опис алгоритмів» було описано загальний алгоритм взаємодії програми з користувачем та було детально розібрано кожний з необхідних для реалізації алгоритмів сортування. Алгоритми пірамідального та плавного сортування було розбито на підзадачі.

В розділі «Опис програмного забезпечення» було наведено UML-діаграму класів ПЗ та описано всі стандартні та користувацькі методи, використані для реалізації програми. UML-діаграма класів продемонструвала, що для реалізації ПЗ було створено та імплементовано інтерфейс «ISortEngine», інтерфейс був імплементований абстрактним класом «SortEngine», який реалізував більшість методів інтерфейсу, окрім методу «sort», та став батьківським для трьох класів («SelectionSortEngine», «HeapSortEngine», «SmoothSortEngine»), що були створені для реалізації кожного з необхідних методів сортування. Кожен клас-нащадок абстрактного класу «SortEngine» інкапсулював власні допоміжні методи та атрибути для здійснення сортування. Кожний клас нащадок по-своєму реалізовував метод «sort» відповідно до того алгоритму, який він представляв. У кожного з трьох класів-нащадків абстрактного класу «SortEngine» є власний клас-нащадок («VisualizableSelectionSortEngine», «VisualizableHeapSortEngine», «VisualizableSmoothSortEngine»), що реалізовував функціонал пов'язаний з

візуалізацією процесу сортування, використовуючи об'єкт класу-форми («SortingVisualizerForm»), на якому виводилась анімація сортування, – між ними є зв'язок композиція. За реалізацію графічного інтерфейсу відповідає клас-форма «MainForm», що також займається інстанціюванням об'єктів класів, що відповідають за сортування масиву.

В розділі «Тестування програмного забезпечення» було передбачено можливі сценарії взаємодії користувача і програми та здійснено тестування кожного зі сценаріїв для доведення роботоспроможності ПЗ в різних ситуаціях. Задля доведення коректності роботи кожного з алгоритмів було тимчасово внесено зміни до програмного коду.

В розділі «Інструкція користувача» було описано графічний інтерфейс програми та запропоновано стандартну послідовність дій для взаємодія користувача і програмного забезпечення. Також було детально описано вхідні і вихідні дані та системні вимоги ПЗ.

В розділі «Аналіз і узагальнення результатів» було експериментально (з використанням Excel) та аналітично доведено асимптотичну складність кожного з алгоритмів сортування, здійснено перевірку ефективності кожного з алгоритмів під час роботи з масивами, елементи яких впорядковані випадковим чином, масивами, які є близькими до відсортованості, відсортованими масивами та масивами, що відсортовано в зворотному порядку. Було побудовано відповідні графіки для візуалізації отриманих даних. В кінці розділу було підсумовано інформацію про алгоритми, їх ефективність та обрано найбільш працездатний алгоритм – алгоритм плавного сортування. Варто зауважити, що алгоритм пірамідального сортування є своєрідною еволюцією алгоритму сортування вибором, яка використовує структуру даних бінарна купа. В свою чергу плавне сортування є вдосконаленням пірамідального, адже дозволяє ефективніше сортувати дані близькі до відсортованості з використанням подібних методів та ідей.



## ПЕРЕЛІК ПОСИЛАНЬ

1. Алгоритми сортування – ВУЕ. URL:  
[https://vue.gov.ua/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\\_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F](https://vue.gov.ua/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D1%81%D0%BE%D1%80%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F) (дата звернення: 02.06.2022).
2. Selection Sort – GeeksforGeeks. URL:  
<https://www.geeksforgeeks.org/selection-sort/> (дата звернення: 02.06.2022).
3. Алгоритми і структури даних: курс лекцій / уклад. М. М. Головченко. Київ : КПІ, 2021. 226 с.
4. HeapSort – GeeksforGeeks. URL: <https://www.geeksforgeeks.org/heap-sort/?ref=gcse> (дата звернення: 02.06.2022).
5. Алгоритмы на C++ (олимпиадный подход): Плавная сортировка (Smooth\_sort). URL: <http://cppalgo.blogspot.com/2010/10/smoothsort.html> (дата звернення: 02.06.2022).

## ДОДАТОК А ТЕХНІЧНЕ ЗАВДАННЯ

КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського

Кафедра  
інформатики та програмної інженерії

Затвердив

Керівник Головченко Максим Миколайович

«19» квітня 2022 р.

Виконавець:

Студент Басараб Олег Андрійович

«19» квітня 2022 р.

## ТЕХНІЧНЕ ЗАВДАННЯ

на виконання курсової роботи

на тему: «Упорядкування масивів»

з дисципліни:

«Основи програмування»

1. *Мета:* Метою курсової роботи є розробка ефективного програмного забезпечення для впорядкування масивів випадково згенерованих чисел обраним методом сортування (сортування вибором, пірамідальне сортування, плавне сортування).
2. *Дата початку роботи:* «19» квітня 2022 р.
3. *Дата закінчення роботи:* «12» червня 2022 р.
4. *Вимоги до програмного забезпечення.*

1) Функціональні вимоги:

- Можливість задавати розмірність масиву для впорядкування (мінімальний розмір – 100 елементів, максимальний – 50000).
- Можливість перевірки введених даних на коректність та виведення відповідних повідомлень у разі некоректних даних.
- Можливість генерувати масив випадкових чисел заданого розміру.
- Можливість обирати методи сортування масиву.
- Можливість сортування масиву обраним методом.
- Можливість графічного відображення процесу сортування для масивів до 200-та елементів.
- Можливість збереження результатів сортування масиву у текстовий файл.
- Можливість відображення статистичних та аналітичних даних для подальшого аналізу ефективності алгоритмів.

2) Нефункціональні вимоги:

- Можливість запуску програмного забезпечення на операційних системах сімейства Windows 10 і вище за наявності Microsoft .NET Framework 4.7.2 і вище.
- 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 19.04.2022 р.)
- 2) Об'єктно-орієнтоване проектування архітектури програмної системи (до 03.05.2022 р.)
- 3) Розробка програмного забезпечення (до 17.05.2022 р.)
- 4) Тестування розробленої програми (до 31.05.2022 р.)
- 5) Розробка пояснювальної записки (до 12.06.2022 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 для вирішення задачі сортування  
масивів*

---

(Найменування програми (документа))

*Електронний носій*

---

(Вид носія даних)

*44 арк, 124 Кб*

---

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

*студента групи ІІІ-12 І курсу*

*Басараб О. А.*

Файл «MainForm.h»

```
#pragma once

#include "SelectionSortEngine.h"
#include "HeapSortEngine.h"
#include "SmoothSortEngine.h"
#include "VisualizableSelectionSortEngine.h"
#include "VisualizableHeapSortEngine.h"
#include "VisualizableSmoothSortEngine.h"
#include "ISortEngine.h"

namespace SortingAlgorithmsApplication {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Клас для відображення графічного користувацького
інтерфейсу
    /// програми SortingAlgorithmsApplication.
    /// Містить компонент NumericUpDown arraySizeNumericUpDown
    /// для введення розміру масиву користувачем.
    /// Містить кнопку btnReset для генерації масиву введеного
розміру,
    /// що складається з випадкових цілих чисел.
    /// Містить кнопку btnSort для сортування масиву обраним
алгоритмом.
    /// Містить компонент ComboBox algorithmSelectorDropDown
    /// для обрання методу сортування (SelectionSort, HeapSort,
SmoothSort).
    /// Містить компонент CheckBox visualizeCheckBox для того,
    /// щоб обрати, чи потрібно візуалізувати процес сортування
масиву.
    /// Містить компонент RichTextBox outputRichTextBox
    /// для виводу інформації про процес генерації масиву та
сортування.
    /// </summary>
    public ref class MainForm : public
System::Windows::Forms::Form
    {
    private:
        /// Масив цілих чисел, що буде згенеровано випадковим
чином,
```

```

        // якщо натиснути кнопку btnReset, і відсортовано обраним
методом.
        array<int>^ arr_;
        // Найбільше можливе значення, якого може набути елемент
масиву arr_.
        literal int kMaxVal_ = 50000;
        // Змінна, що використовується для індексації тих
масивів, що були відсортовані.
        int currSortableArrayIdx_;
        // Назва папки, в якій зберігаються текстові файли,
        // що містять масиви випадкових цілих чисел перед
сортуванням та їхні розміри.
        literal String^ kInitialArraysFolder_ =
"initial_arrays\\";
        // Назва папки, в якій зберігаються текстові файли,
        // що містять відсортовані масиви цілих чисел, їхні
розміри, обраний метод сортування,
        // кількість перестановок та кількість порівнянь,
здійснених в процесі сортування.
        literal String^ kSortedArraysFolder_ = "sorted_arrays\\";
        // Масив рядків-назв методів сортування.
        array<String>^ sortingAlgorithmsNames_;
        // Максимальний розмір масиву, за якого процесу його
сортування може візуалізуватися.
        literal int maxVisualizableArrSize_ = 200;

    public:
        /// <summary> Конструктор класу MainForm. </summary>
        MainForm(void);

    private:
        /// <summary> Метод, що заповнює масив
sortingAlgorithmsNames_ рядками з назвами
        /// алгоритмів сортування. </summary>
        void populateSortingAlgorithmsNamesArray();

    private:
        /// <summary> Метод для заповнення компоненту
algorithmSelectorDropDown
        /// елементами масиву sortingAlgorithms_. </summary>
        void populateDropDownAlgorithmSelector();

    protected:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~MainForm();

```

```

protected:

protected:

    private: System::Windows::Forms::RichTextBox^
outputRichTextBox;
    private: System::Windows::Forms::NumericUpDown^
arraySizeNumericUpDown;
    private: System::Windows::Forms::Label^ label1;
    private: System::Windows::Forms::ComboBox^
algorithmSelectorDropDown;

    private: System::Windows::Forms::Label^ label2;
    private: System::Windows::Forms::Button^ btnReset;
    private: System::Windows::Forms::Button^ btnSort;
    private: System::Windows::Forms::CheckBox^ visualizeCheckBox;

    private:
        /// <summary>
        /// Required designer variable.
        /// </summary>
        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        void InitializeComponent(void);

#pragma endregion
        /// <summary> Метод для обробки натискання на кнопку btnReset.
        /// Генерує масив випадкових цілих чисел arr_.
        /// Виводить відповідне повідомлення в компонент
outputRichTextBox.
        /// Блокує доступ до прапорця visualizeCheckBox, якщо
розмірність масиву > maxVisusalizableArrSize_.
        /// Інакше розблоковує доступ до прапорця visualizeCheckBox.
</summary>
        private: System::Void btnReset_Click(System::Object^ sender,
System::EventArgs^ e);

        /// <summary> Метод виведення повідомлення про генерацію arr_
        /// в outputRichTextBox, що містить розмір масиву та поточне
значення currSortableArrayIdx_. </summary>
        private: void printGenerationMessage();

```



```

    /// <summary> Метод виведення n послідовних елементів масиву
arr_, якщо arr_->Length > n.
    /// Інакше виводиться увесь масив. </summary>
    private: void printRandomConsecutiveElementsOfArray(int n);

    /// <summary> Метод для генерації масиву arr_ розміру arrSize.
    /// Кожному елементу масиву arr_ присвоюється випадкове ціле
значення в межах від 0 до kMaxVal_.
    /// Очищує пам'ять, яку займав arr_ до генерації, якщо arr_ !=
nullptr. </summary>
    /// <param name = "arrSize"> Розмірність масиву для генерації.
</param>
    private: void generateRandomArray(int arrSize);

    /// <summary> Метод для збереження розміру масиву arr_ та
самого масиву arr_ до сортування в текстовий файл.
    /// Текстовий файл зберігається в папку з назвою
initialArraysFolder_. </summary>
    /// <param name = "fileName"> Назва текстового файлу, в який
буде записано розмір масиву та
    /// сам масив до сортування. </param>
    /// <returns> true, якщо файл вдалося зберегти, інакше -
false. </returns>
    private: bool saveInitialArrayToFile(String^ fileName);

    /// <summary> Метод для збереження розміру масиву arr_, самого
масиву arr_
    /// після сортування, назви методу сортування, кількості
перестановок та порівнянь,
    /// здійснених в процесі сортування, в текстовий файл.
    /// Текстовий файл зберігається в папку з назвою
sortedArrayFolder_. </summary>
    /// <param name = "fileName"> Назва текстового файлу, в який
буде записано дані. </param>
    /// <param name = "compsCount"> Кількість порівнянь,
здійснених в процесі сортування. </param>
    /// <param name = "swapsCount"> Кількість перестановок,
здійснених в процесі сортування. </param>
    /// <returns> true, якщо файл вдалося зберегти, інакше -
false. </returns>
    private: bool saveSortedArrayToFile(String^ fileName, long
long compsCount, long long swapsCount);

    /// <summary> Метод для генерації назви поточного файлу, в
якому зберігатиметься
    /// масив до сортування. </summary>
    /// <returns> Назва поточного файлу, в який буде записано
масив ДО сортування. </returns>

```

```

        private: String^ generateInitialArrayFileName();

        /// <summary> Метод для виведення повідомлення про сортування
arr_
        /// та відповідних даних в компонент outputRichTextBox.
</summary>
        /// <param name = "compsCount"> Кількість порівнянь,
здійснених в процесі сортування. </param>
        /// <param name = "swapsCount"> Кількість перестановок,
здійснених в процесі сортування. </param>
        /// <param name = "initialArrayWasWrittenToFile"> Вказує, чи
початковий стан масиву було успішно записано у файл. </param>
        /// <param name = "sortedArrayWasWrittenToFile"> Вказує, чи
відсортований стан масиву було успішно записано у файл. </param>
        private: void printSortingMessage(bool
initialArrayWasWrittenToFile, bool sortedArrayWasWrittenToFile,
long long compsCount, long long swapsCount);

        /// <summary> Метод для обробки натискання на кнопку btnSort.
        /// Записує початковий масив до текстового файлу.
        /// Сортує масив випадкових цілих чисел arr_.
        /// Записує відсортований масив до текстового файлу.
        /// Виводить відповідне повідомлення в компонент
outputRichTextBox. </summary>
        private: System::Void btnSort_Click(System::Object^ sender,
System::EventArgs^ e);

        /// <summary> Метод, який повертає об'єкт класу, що імплементує
інтерфейс
        /// ISortEngine, відповідно до даних, введених користувачем.
</summary>
        /// <returns> Об'єкт класу, що імплементує інтерфейс
ISortEngine, що відповідає
        /// умовам, які введені користувачем. </returns>
        private: ISortEngine^ getConditionalSortEngine();

        /// <summary> Метод, який очищує outputRichTextBox від тексту,
якщо
        /// кількість записаних символів перевищує 2000. </summary>
        private: void updateOutputTextBox();

        /// <summary> Метод для генерації назви поточного файлу, в
якому зберігатиметься
        /// масив після сортування. </summary>
        /// <returns> Назва поточного файлу, в який буде записано
масив після сортування. </returns>
        private: String^ generateSortedArrayFileName();

```

```

    /// <summary> Метод для обробки події "зміна тексту" компоненту
outputRichTextBox.
    /// Встановлює смугу прокрутки outputRichTextBox в нижнє
положення. </summary>
    private: System::Void
outputRichTextBox_TextChanged(System::Object^ sender,
System::EventArgs^ e);
};
}

```

Файл «SortingVisualizerForm.h»

```
#pragma once
```

```
#include <chrono>
#include <thread>
```

```
namespace SortingAlgorithmsApplication {
```

```

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace std::this_thread;
    using namespace std::chrono_literals;

```

```

    /// <summary>
    /// Клас для графічного відображення процесу сортування
масиву.
    /// Містить компонент panel, який використовується як тло для
відображення
    /// гістограм, стовпці якої рухаються і змінюють колір
відповідно до
    /// порівнянь і перестановок, які відбуваються в процесу
сортування масиву.
    /// </summary>
    public ref class SortingVisualizerForm : public
System::Windows::Forms::Form
    {
    private:
        // Атрибут класу, який використовується для відображення
графічної складової процесу
        // сортування масиву.
        Graphics^ g_;
        // Масив, який буде сортуватися.
        array<int>^ arr_;

```

```

        // Максимальне значення, якого може набувати елемент
масиву arr_.
        int maxVal_;
        // Копія оригінального масиву, яка буде використовуватися
для відображення процесу
        // перестановок елементів.
        array<int>^ arrCopy_;
        // Коефіцієнт, який буде використано для масштабування
висоти стовпця гістограми під час
        // візуалізації процесу сортування.
        float hCoefficient_;
        // Коефіцієнт, який буде використано для масштабування
ширини стовпця гістограми під час
        // візуалізації процесу сортування.
        float wCoefficient_;

    /// <summary> Конструктор класу SortingVisualizerForm.
</summary>
    public: SortingVisualizerForm(array<int>^% arr, int maxVal);

    /// <summary> Розрахунок hCoefficient_. </summary>
    /// <returns> Значення hCoefficient_. </returns>
    private: float getHeightCoefficient();

    /// <summary> Розрахунок wCoefficient_. </summary>
    /// <returns> Значення wCoefficient_. </returns>
    private: float getWidthCoefficient();

    public:
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        ~SortingVisualizerForm();

    private: System::Windows::Forms::Panel^ panel;
    protected:

    private:
        /// <summary>
        /// Required designer variable.
        /// </summary>
        System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>

```

```

        void InitializeComponent(void);

#pragma endregion

    /// <summary> Метод для графічного відображення одного стовпця
    гістограми. </summary>
    /// <param name = "brush"> Колір стовпця. </param>
    /// <param name = "idx"> Індекс елементу масиву, який
    відображається. </param>
    /// <param name = "val"> Значення елементу масиву, який
    відображається. </param>
    private: void drawHill(Brush^ brush, int idx, int val);

    /// <summary> Метод для графічного відображення всього масиву
    arr_. </summary>
    public: void visualizeArray();

    /// <summary> Метод для графічного відображення процесу
    перестановки двох елементів масиву. </summary>
    /// <param name = "i"> Індекс елементу який переставляється.
    </param>
    /// <param name = "j"> Індекс елементу який переставляється.
    </param>
    public: void visualizeSwap(int i, int j);

    /// <summary> Метод для графічного відображення процесу
    порівняння двох елементів масиву. </summary>
    /// <param name = "i"> Індекс елементу який порівнюється з
    іншим елементом. </param>
    /// <param name = "j"> Індекс елементу який порівнюється з
    іншим елементом. </param>
    public: void visualizeComparison(int i, int j);

    /// <summary> Метод для графічного відображення процесу
    перевірки масиву на відсортованість. </summary>
    /// <param name = "breakIdx"> Індекс першого елементу, який
    порушує умову відсортованості масиву. </param>
    public: void visualizeSortingTest(int breakIdx);
};
}

```

Файл «ISortEngine.h»

```
#pragma once
```

```

/// <summary> Інтерфейс для класів, призначенням яких є сортування
масивів. </summary>

```

```

interface class ISortEngine {
public:
    /// <summary> Метод для сортування масиву. </summary>
    /// <returns> Відсортований масив. </returns>
    void sort();

    /// <summary> Геттер, що повертає поточну кількість
перестановок. </summary>
    /// <returns> Поточна кількість перестановок. </returns>
    long long getSwapsCount();

    /// <summary> Геттер, що повертає поточну кількість порівнянь.
</summary>
    /// <returns> Поточна кількість порівнянь. </returns>
    long long getComparisonsCount();

    /// <summary> Метод для перевірки масиву на відсортованість.
</summary>
    /// <returns> Індекс першого елемента, який порушує
відсортованість масиву.
    /// Якщо такий елемент відсутній, то повертає -1. </returns>
    int sortingTest();
};

```

Файл «SortEngine.h»

```
#pragma once
```

```
#include "ISortEngine.h"
```

```

/// <summary>
/// Абстрактний клас, що імплементує інтерфейс ISortEngine і
реалізує частину його методів.
/// </summary>
ref class SortEngine abstract : ISortEngine
{
protected:
    /// Масив цілих чисел, який буде відсортовано.
    array<int>^ arr_;
    /// Поточна кількість перестановок.
    long long swapsCount_;
    /// Поточна кількість порівнянь.
    long long comparisonsCount_;

protected:
    /// <summary> Метод для перестановки двох елементів масиву.
</summary>
    /// <param name = "i"> Індекс елемента який переставляється.
</param>

```

```

    /// <param name = "j"> Індекс елементу який переставляється.
</param>
    virtual void swap(int i, int j);

public:
    /// <summary> Конструктор класу SortEngine. </summary>
    /// <param name = "arr"> Посилання на масив, який буде
відсортовано обраним методом. </param>
    SortEngine(array<int>^% arr);

    /// <summary> Геттер, що повертає поточну кількість порівнянь.
    /// Реалізає метод інтерфейсу ISortEngine. </summary>
    /// <returns> Поточна кількість порівнянь. </returns>
    virtual long long getComparisonsCount();

    /// <summary> Геттер, що повертає поточну кількість
перестановок.
    /// Реалізає метод інтерфейсу ISortEngine. </summary>
    /// <returns> Поточна кількість перестановок. </returns>
    virtual long long getSwapsCount();

    /// <summary> Абстрактний метод для сортування масиву.
</summary>
    virtual void sort() abstract;

    /// <summary> Метод для перевірки масиву на відсортованість.
    /// Реалізає метод інтерфейсу ISortEngine. </summary>
    /// <returns> Індекс першого елементу, який порушує умову
відсортованості масиву.
    /// Якщо такий елемент відсутній, то повертає -1. </returns>
    virtual int sortingTest();
};

```

Файл «SelectionEngine.h»

```
#pragma once
```

```
#include "SortEngine.h"
```

```

/// <summary>
/// Клас для сортування масиву алгоритмом сортування вибором.
/// Наслідує абстрактний клас SortEngine.
/// </summary>
ref class SelectionSortEngine : public SortEngine
{

protected:
    /// <summary> Метод для знаходження індексу найменшого
елементу arr_

```

```

    /// серед елементів з індексами від left до right включно.
</summary>
    /// <param name = "left"> Індекс елементу, з якого
починається пошук мінімального. </param>
    /// <param name = "right"> Індекс елементу, на якому
завершується пошук мінімального. </param>
    /// <returns> Індeksu найменшого елементу arr_
    /// серед елементів з індексами від left до right включно.
</returns>
    virtual int getIdxOfMinElement(int left, int right);

public:
    /// <summary> Конструктор класу SelectionSortEngine.
</summary>
    /// <param name = "arr"> Посилання на масив, який буде
відсортовано. </param>
    SelectionSortEngine(array<int>^% arr);

    /// <summary> Метод для сортування масиву алгоритмом
сортування вибором. </summary>
    virtual void sort() override;
};

```

Файл «HeapSortEngine.h»

```
#pragma once
```

```
#include "SortEngine.h"
```

```

/// <summary>
/// Клас для сортування масиву алгоритмом пірамідального
сортування.
/// Наслідує абстрактний клас SortEngine.
/// </summary>
ref class HeapSortEngine : public SortEngine
{
protected:
    /// <summary> "Просійка вниз" для дотримання властивості
MaxHeap. </summary>
    /// <param name="heapSize"> Розмір купи. </param>
    /// <param name="rootIdx"> Корінь купи. </param>
    virtual void heapify(int heapSize, int rootIdx);

    /// <summary> Метод для знаходження індексу лівого нащадка
елементу з індексом rootIdx. </summary>
    /// <param name = "rootIdx"> Індекс елементу, для якого
шукається лівого нащадок. </param>
    /// <returns> Індекс лівого нащадка елементу з індексом
rootIdx. </returns>

```



```

    int getLeftChildIdx(int rootIdx);

    /// <summary> Метод для знаходження індексу правого нащадка
    елементу з індексом rootIdx. </summary>
    /// <param name = "rootIdx"> Індекс елементу, для якого
    шукається правий нащадок. </param>
    /// <returns> Індекс правого нащадка елементу з індексом
    rootIdx. </returns>
    int getRightChildIdx(int rootIdx);

    /// <summary> Метод для побудови бінарної купи (MaxHeap).
</summary>
    void buildHeap();

public:
    /// <summary> Конструктор класу HeapSortEngine. </summary>
    /// <param name = "arr"> Посилання на масив, який буде
    відсортовано. </param>
    HeapSortEngine(array<int>^% arr);

    /// <summary> Метод для сортування масиву алгоритмом
    пірамідального сортування. </summary>
    virtual void sort() override;
};

```

Файл «SmoothSortEngine.h»

```
#pragma once
```

```
#include "SortEngine.h"
#include <cliext/vector>
```

```
using cliext::vector;
```

```

/// <summary>
/// Клас для сортування масиву алгоритмом плавного сортування.
/// Наслідує абстрактний клас SortEngine.
/// </summary>
ref class SmoothSortEngine : public SortEngine
{
protected:
    /// Вектор чисел Леонардо, які будуть використані в процесі
    сортування.
    vector<int> leoNums;

protected:
    /// <summary> Метод для побудови послідовності куп, які мають
    розміри, рівні числам Леонардо.

```

```

    /// Також оновлює поточне значення стану послідовності куп
currState. </summary>
    /// <param name = "currState"> Поточний стан послідовності
куп. </param>
    void buildHeapPool(uint64_t *currState);

    /// <summary> Метод для розрахунку натупного стану на основі
поточного.
    /// Оновлює значення змінної currState. Оновлює значення
    /// mergedHeapsLeoNumIdx, якщо перехід до наступного стану
передбачає злиття куп.
    /// Якщо злиття не відбулося, то mergedHeapsLeoNumIdx = -1.
</summary>
    /// <param name = "currState"> Поточний стан послідовності
куп. </param>
    /// <param name = "mergedHeapsLeoNumIdx"> Індекс числа
Леонардо, яке дорівнює
    /// розмірності об'єднаних куп, якщо новий стан передбачає
об'єднання двох куп. </param>
    void calculateNextState(uint64_t *currState, int*
mergedHeapsLeoNumIdx);

    /// <summary> Метод для розрахунку попереднього стану на
основі поточного.
    /// Оновлює значення змінної currState. </summary>
    /// <param name = "currState"> Поточний стан послідовності
куп. </param>
    void calculatePrevState(uint64_t *currState);

    /// <summary> Метод для "просійки вниз" поточної купи.
</summary>
    /// <param name = "heapLeoNumIdx"> Індекс числа Леонардо, яке
дорівнює розмірності
    /// поточної купи. </param>
    /// <param name = "rootIdx"> Індекс кореню поточної купи.
</param>
    virtual void siftDown(int heapLeoNumIdx, int rootIdx);

    /// <summary> Метод для знаходження індексу найбільшого кореню
серед коренів куп
    /// з послідовності куп. Оновлює значення індексу
максимального кореню maxRootIdx
    /// та значення heapLeoNumIdx - індекс числа Леонардо, яке
дорівнює розмірності купи, якій
    /// належить найбільший корінь. </summary>
    /// <param name = "currState"> Поточний стан послідовності
куп. </param>

```

```

    /// <param name = "rootIdx"> Індекс кореню останньої купи.
</param>
    /// <param name = "heapLeoNumIdx"> Індекс числа Леонардо, яке
дорівнює розмірності
    /// останньої купи. </param>
    /// <param name = "maxRootIdx"> Індекс найбільшого кореню
серед поточної послідовності купи. </param>
    virtual void findMaxAmongRoots(uint64_t currState, int
rootIdx, int *heapLeoNumIdx, int *maxRootIdx);

    /// <summary> Метод для розрахунку достатньої для сортування
кількості Леонардових чисел.
    /// Числа заносяться до вектору leoNums. </summary>
    void calculateLeoNums();

public:
    /// <summary> Конструктор класу SmoothSortEngine. </summary>
    /// <param name = "arr"> Посилання на масив, який буде
відсортовано. </param>
    SmoothSortEngine(array<int>^% arr);

    /// <summary> Метод для сортування масиву алгоритмом
пірамідального сортування. </summary>
    virtual void sort() override;
};

```

Файл «VisualizableSelectionSortEngine.h»

```
#pragma once
```

```
#include "SelectionSortEngine.h"
#include "SortingVisualizerForm.h"
```

```

/// <summary>
/// Клас для сортування масиву алгоритмом сортування вибором.
/// Сортування супроводжується візуалізацією за допомогою
SortingVisualizerForm.
/// Наслідує клас SelectionSortEngine.
/// </summary>
ref class VisualizableSelectionSortEngine : public
SelectionSortEngine
{
private:
    // Атрибут VisualizableSelectionSortEngine для візуалізації
процесу сортування.
    SortingAlgorithmsApplication::SortingVisualizerForm^ svf_;

public:

```

```

    /// <summary> Деструктор класу
VisualizableSelectionSortEngine. </summary>
~VisualizableSelectionSortEngine();

protected:
    /// <summary> Метод для знаходження індексу найменшого
елементу arr_
    /// серед елементів з індексами від left до right включно.
    /// Процес пошуку супроводжуватиметься візуалізацією.
</summary>
    /// <param name = "left"> Індекс елементу, з якого
починається пошук мінімального. </param>
    /// <param name = "right"> Індекс елементу, на якому
завершується пошук мінімального. </param>
    /// <returns> Індeksu найменшого елементу arr_
    /// серед елементів з індексами від left до right включно.
</returns>
    virtual int getIdxOfMinElement(int left, int right) override;

public:
    /// <summary> Конструктор класу
VisualizableSelectionSortEngine. </summary>
    /// <param name = "arr"> Посилання на масив, який буде
відсортовано. </param>
    /// <param name = "maxVal"> Найбільше значення, якого може
набувати елемент масиву arr. </param>
    VisualizableSelectionSortEngine(array<int>^% arr, int maxVal);

    /// <summary> Метод для сортування масиву алгоритмом
сортування вибором.
    /// Процес сортування буде супроводжуватися анімацією.
</summary>
    /// <returns> Відсортований масив arr_. </returns>
    virtual void sort() override;
};

```

Файл «VisualizableHeapSortEngine.h»

```
#pragma once
```

```
#include "HeapSortEngine.h"
```

```
#include "SortingVisualizerForm.h"
```

```

/// <summary>
/// Клас для сортування масиву алгоритмом пірамідального
сортування.
/// Сортування супроводжується візуалізацією за допомогою
SortingVisualizerForm.
/// Наслідує клас HeapSortEngine.

```

```

/// </summary>
ref class VisualizableHeapSortEngine : public HeapSortEngine
{
private:
    /// Атрибут VisualizableHeapSortEngine для візуалізації процесу
    сортування.
    SortingAlgorithmsApplication::SortingVisualizerForm^ svf_;

public:
    /// <summary> Деструктор класу VisualizableHeapSortEngine.
    </summary>
    ~VisualizableHeapSortEngine();

protected:
    /// <summary> "Просійка вниз" для дотримання властивості
    MaxHeap.
    /// Процес "просійки" буде супроводжуватися анімацією.
    </summary>
    /// <param name="heapSize"> Розмір купи. </param>
    /// <param name="rootIdx"> Корінь купи. </param>
    /// <returns> Повне бінарне дерево, в якому елемент, що мав
    індекс rootIdx займе
    /// місце відповідно до властивості MaxHeap. </returns>
    virtual void heapify(int heapSize, int rootIdx) override;

public:
    /// <summary> Конструктор класу VisualizableHeapSortEngine.
    </summary>
    /// <param name = "arr"> Посилання на масив, який буде
    відсортовано. </param>
    /// <param name = "maxVal"> Найбільше значення, якого може
    набувати елемент масиву arr. </param>
    VisualizableHeapSortEngine(array<int>^% arr, int maxVal);

    /// <summary> Метод для сортування масиву алгоритмом
    пірамідального сортування.
    /// Процес сортування буде супроводжуватися анімацією.
    </summary>
    virtual void sort() override;
};

```

Файл «VisualizableSmoothSortEngine.h»

```
#pragma once
```

```

#include "SmoothSortEngine.h"
#include "SortingVisualizerForm.h"

```

```

#include <cliext/vector>

using cliext::vector;

/// <summary>
/// Клас для сортування масиву алгоритмом плавного сортування.
/// Сортування супроводжується візуалізацією за допомогою
SortingVisualizerForm.
/// Наслідує клас SmoothSortEngine.
/// </summary>
ref class VisualizableSmoothSortEngine : public SmoothSortEngine
{
private:
    // Атрибут VisualizableHeapSortEngine для візуалізації процесу
    сортування.
    SortingAlgorithmsApplication::SortingVisualizerForm^ svf_;

public:
    /// <summary> Деструктор класу VisualizableSmoothSortEngine.
    </summary>
    ~VisualizableSmoothSortEngine();

protected:
    /// <summary> Метод для "просійки вниз" поточної купи.
    /// Процес "просійки" супроводжуватиметься візуалізацією.
    </summary>
    /// <param name = "heapLeoNumIdx"> Індекс числа Леонардо, яке
    дорівнює розмірності
    /// поточної купи. </param>
    /// <param name = "rootIdx"> Індекс кореню поточної купи.
    </param>
    virtual void siftDown(int heapLeoNumIdx, int rootIdx)
    override;

    /// <summary> Метод для знаходження індексу найбільшого кореню
    серед коренів куп
    /// з послідовності куп. Оновлює значення індексу
    максимального кореню maxRootIdx
    /// та значення heapLeoNumIdx - індекс числа Леонардо, яке
    дорівнює розмірності купи, якій
    /// належить найбільший корінь.
    /// Процес пошуку супроводжуватиметься візуалізацією.
    </summary>
    /// <param name = "currState"> Поточний стан послідовності
    куп. </param>
    /// <param name = "rootIdx"> Індекс кореню останньої купи.
    </param>

```

```

    /// <param name = "heapLeoNumIdx"> Індекс числа Леонардо, яке
дорівнює розмірності
    /// останньої купи. </param>
    /// <param name = "maxRootIdx"> Індекс найбільшого кореню
серед поточної послідовності купи. </param>
    virtual void findMaxAmongRoots(uint64_t currState, int
rootIdx, int* heapLeoNumIdx, int* maxRootIdx) override;

public:
    /// <summary> Конструктор класу VisualizableSmoothSortEngine.
</summary>
    /// <param name = "arr"> Посилання на масив, який буде
відсортовано. </param>
    /// <param name = "maxVal"> Найбільше значення, якого може
набувати елемент масиву arr. </param>
    VisualizableSmoothSortEngine(array<int>^% arr, int maxVal);

    /// <summary> Метод для сортування масиву алгоритмом плавного
сортування.
    /// Процес сортування буде супроводжуватися анімацією.
</summary>
    virtual void sort() override;
};

```

Файл «MainForm.cpp»

```

#include "MainForm.h"
#include <algorithm>

using namespace SortingAlgorithmsApplication;

MainForm::MainForm(void) : currSortableArrayIdx_{ 1 }
{
    InitializeComponent();
    populateSortingAlgorithmsNamesArray();
    populateDropDownAlgorithmSelector();
}

MainForm::~MainForm()
{
    if (components)
    {
        delete components;
    }

    if (arr_) {
        delete arr_;
    }
}

```

```

        if (sortingAlgorithmsNames_) {
            delete sortingAlgorithmsNames_;
        }
    }

void MainForm::populateSortingAlgorithmsNamesArray() {

    int numberOfAlgorithms{ 3 };
    sortingAlgorithmsNames_ = gcnew
array<String^>(numberOfAlgorithms);

    sortingAlgorithmsNames_[0] = "Selection Sort";
    sortingAlgorithmsNames_[1] = "Heap Sort";
    sortingAlgorithmsNames_[2] = "Smooth Sort";
}

void MainForm::updateOutputTextBox() {
    int maxTextLength{ 2000 };
    if (outputRichTextBox->Text->Length > maxTextLength) {
        outputRichTextBox->Text = "";
    }
}

void MainForm::populateDropDownAlgorithmSelector() {

    for (int i{ 0 }; i < sortingAlgorithmsNames_->Length; ++i) {
        algorithmSelectorDropDown->Items-
>Add(sortingAlgorithmsNames_[i]);
    }

    algorithmSelectorDropDown->SelectedIndex = 0;
}

void MainForm::InitializeComponent(void)
{
    this->outputRichTextBox = (gcnew
System::Windows::Forms::RichTextBox());
    this->arraySizeNumericUpDown = (gcnew
System::Windows::Forms::NumericUpDown());
    this->label1 = (gcnew System::Windows::Forms::Label());
    this->algorithmSelectorDropDown = (gcnew
System::Windows::Forms::ComboBox());
    this->label2 = (gcnew System::Windows::Forms::Label());
    this->btnReset = (gcnew System::Windows::Forms::Button());
    this->btnSort = (gcnew System::Windows::Forms::Button());
    this->visualizeCheckBox = (gcnew
System::Windows::Forms::CheckBox());

```



```

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this->arraySizeNumericUpDown))->BeginInit();
        this->SuspendLayout();
        //
        // outputRichTextBox
        //
        this->outputRichTextBox->Location =
System::Drawing::Point(306, 12);
        this->outputRichTextBox->Name = L"outputRichTextBox";
        this->outputRichTextBox->ReadOnly = true;
        this->outputRichTextBox->Size = System::Drawing::Size(612,
177);
        this->outputRichTextBox->TabIndex = 1;
        this->outputRichTextBox->Text = L"";
        this->outputRichTextBox->TextChanged += gcnew
System::EventHandler(this,
&MainForm::outputRichTextBox_TextChanged);
        //
        // arraySizeNumericUpDown
        //
        this->arraySizeNumericUpDown->Increment =
System::Decimal(gcnew cli::array< System::Int32 >(4) { 100, 0, 0, 0
});
        this->arraySizeNumericUpDown->Location =
System::Drawing::Point(99, 12);
        this->arraySizeNumericUpDown->Maximum = System::Decimal(gcnew
cli::array< System::Int32 >(4) { 50000, 0, 0, 0 });
        this->arraySizeNumericUpDown->Minimum = System::Decimal(gcnew
cli::array< System::Int32 >(4) { 100, 0, 0, 0 });
        this->arraySizeNumericUpDown->Name =
L"arraySizeNumericUpDown";
        this->arraySizeNumericUpDown->Size =
System::Drawing::Size(188, 22);
        this->arraySizeNumericUpDown->TabIndex = 2;
        this->arraySizeNumericUpDown->Value = System::Decimal(gcnew
cli::array< System::Int32 >(4) { 100, 0, 0, 0 });
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->Location = System::Drawing::Point(12, 14);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(81, 17);
        this->label1->TabIndex = 3;
        this->label1->Text = L"Array Size: ";
        //
        // algorithmSelectorDropDown
        //

```

```

        this->algorithmSelectorDropDown->DropDownStyle =
System::Windows::Forms::ComboBoxStyle::DropDownList;
        this->algorithmSelectorDropDown->FormattingEnabled = true;
        this->algorithmSelectorDropDown->Location =
System::Drawing::Point(99, 41);
        this->algorithmSelectorDropDown->Name =
L"algorithmSelectorDropDown";
        this->algorithmSelectorDropDown->Size =
System::Drawing::Size(188, 24);
        this->algorithmSelectorDropDown->TabIndex = 4;
        //
        // label2
        //
        this->label2->AutoSize = true;
        this->label2->Location = System::Drawing::Point(12, 44);
        this->label2->Name = L"label2";
        this->label2->Size = System::Drawing::Size(71, 17);
        this->label2->TabIndex = 5;
        this->label2->Text = L"Algorithm:";
        //
        // btnReset
        //
        this->btnReset->Location = System::Drawing::Point(12, 93);
        this->btnReset->Name = L"btnReset";
        this->btnReset->Size = System::Drawing::Size(275, 44);
        this->btnReset->TabIndex = 6;
        this->btnReset->Text = L"Generate Random Array";
        this->btnReset->UseVisualStyleBackColor = true;
        this->btnReset->Click += gcnew System::EventHandler(this,
&MainForm::btnReset_Click);
        //
        // btnSort
        //
        this->btnSort->Enabled = false;
        this->btnSort->Location = System::Drawing::Point(12, 143);
        this->btnSort->Name = L"btnSort";
        this->btnSort->Size = System::Drawing::Size(180, 44);
        this->btnSort->TabIndex = 7;
        this->btnSort->Text = L"Sort";
        this->btnSort->UseVisualStyleBackColor = true;
        this->btnSort->Click += gcnew System::EventHandler(this,
&MainForm::btnSort_Click);
        //
        // visualizeCheckBox
        //
        this->visualizeCheckBox->AutoSize = true;
        this->visualizeCheckBox->Location =
System::Drawing::Point(201, 156);

```

```

    this->visualizeCheckBox->Name = L"visualizeCheckBox";
    this->visualizeCheckBox->Size = System::Drawing::Size(86, 21);
    this->visualizeCheckBox->TabIndex = 8;
    this->visualizeCheckBox->Text = L"Visualize";
    this->visualizeCheckBox->UseVisualStyleBackColor = true;
    //
    // MainForm
    //
    this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
    this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
    this->ClientSize = System::Drawing::Size(930, 201);
    this->Controls->Add(this->visualizeCheckBox);
    this->Controls->Add(this->btnSort);
    this->Controls->Add(this->btnReset);
    this->Controls->Add(this->label2);
    this->Controls->Add(this->algorithmSelectorDropDown);
    this->Controls->Add(this->label1);
    this->Controls->Add(this->arraySizeNumericUpDown);
    this->Controls->Add(this->outputRichTextBox);
    this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedSingle;
    this->MaximizeBox = false;
    this->Name = L"MainForm";
    this->ShowIcon = false;
    this->Text = L"SortingAlgorithmsApp";
    (cli::safe_cast<System::ComponentModel::ISupportInitialize>(&this->arraySizeNumericUpDown))->EndInit();
    this->ResumeLayout(false);
    this->PerformLayout();

}

void MainForm::generateRandomArray(int arrSize) {

    if (arr_) {
        delete arr_;
    }

    arr_ = gcnew array<int>(arrSize);

    Random^ randVal = gcnew Random();
    for (int i{ 0 }; i < arr_->Length; ++i) {
        arr_[i] = randVal->Next(0, kMaxVal_ + 1);
    }
}

bool MainForm::saveInitialArrayToFile(String^ fileName) {

```

```

try {
    if (!IO::Directory::Exists(kInitialArraysFolder_)) {

        IO::Directory::CreateDirectory(kInitialArraysFolder_);
    }

    array<String>^ files =
IO::Directory::GetFiles(kInitialArraysFolder_);

    IO::StreamWriter^ sw = gcnew
IO::StreamWriter(kInitialArraysFolder_ + fileName);
    sw->WriteLine("array size: " + Convert::ToString(arr_-
>Length));
    sw->WriteLine("initial array elements: ");

    for (int i{ 0 }; i < arr_->Length; ++i) {
        sw->WriteLine("array[" + Convert::ToString(i) + "]:
\t" + Convert::ToString(arr_[i]));
    }
    sw->Close();
}
catch (Exception^ e) {
    MessageBox::Show("Unfortunately, initial array can't be
written to file '" + fileName + "' in folder '" +
kInitialArraysFolder_ + "'.", "Error");

    return false;
}

return true;
}

void MainForm::printGenerationMessage() {

    updateOutputTextBox();

    outputRichTextBox->Text +=
Convert::ToString(currSortableArrayIdx_) + ". \tArray of " +
    Convert::ToString(arr_->Length) + " random integers has
been successfully generated.\n";
    int n{ 10 };
    printRandomConsecutiveElementsOfArray(n);
    outputRichTextBox->Text += "\n";
}

void MainForm::printRandomConsecutiveElementsOfArray(int n) {
    if (arr_->Length > n) {
        Random^ rand = gcnew Random();
    }
}

```

```

        int idx{ rand->Next(0, arr_->Length - n + 1) };
        outputRichTextBox->Text += "\tarray[" +
Convert::ToString(idx) + "]" ... array[" + Convert::ToString(idx + n
- 1) + "]:\n\t";

        for (int i{ 0 }; i < n; ++i) {
            outputRichTextBox->Text +=
Convert::ToString(arr_[idx]) + " ";
            ++idx;
        }
    }
    else {
        outputRichTextBox->Text += "\tall elements of
array:\n\t";
        for (int i{ 0 }; i < arr_->Length; ++i) {
            outputRichTextBox->Text +=
Convert::ToString(arr_[i]) + " ";
        }
    }
    outputRichTextBox->Text += "\n";
}

void MainForm::printSortingMessage(bool
initialArrayWasWrittenToFile, bool sortedArrayWasWrittenToFile,
long long compsCount, long long swapsCount) {

    updateOutputTextBox();

    outputRichTextBox->Text +=
Convert::ToString(currSortableArrayIdx_) + ". \tArray of " +
Convert::ToString(arr_->Length) + " random integers has been
successfully sorted.\n";

    if (initialArrayWasWrittenToFile) {
        outputRichTextBox->Text += "\tInitial array has been
written to file \"" + generateInitialArrayFileName() + "\".\n";
    }
    if (sortedArrayWasWrittenToFile) {
        outputRichTextBox->Text += "\tSorted array has been
written to file \"" + generateSortedArrayFileName() + "\".\n";
    }
    int n{ 10 };
    printRandomConsecutiveElementsOfArray(n);
    outputRichTextBox->Text += "\tsorting algorithm: \t" +
Convert::ToString(sortingAlgorithmsNames_[algorithmSelectorDropDown
->SelectedIndex]) +

```

```

        "\n\tcomparisons: \t" + Convert::ToString(compsCount) +
        "\n\tswaps: \t\t" + Convert::ToString(swapsCount) + "\n\n";
    }

    bool MainForm::saveSortedArrayToFile(String^ fileName, long long
    compsCount, long long swapsCount) {

        try {
            if (!IO::Directory::Exists(kSortedArraysFolder_)) {

                IO::Directory::CreateDirectory(kSortedArraysFolder_);
            }

            IO::StreamWriter^ sw = gcnew
            IO::StreamWriter(kSortedArraysFolder_ + fileName);

            sw->WriteLine("sorting algorithm: " +
            sortingAlgorithmsNames_[algorithmSelectorDropDown->SelectedIndex]);
            sw->WriteLine("comparisons: \t" +
            Convert::ToString(compsCount));
            sw->WriteLine("swaps: \t\t" +
            Convert::ToString(swapsCount));
            sw->WriteLine("array size: \t" +
            System::Convert::ToString(arr_->Length));
            sw->WriteLine("sorted array elements: ");

            for (int i{ 0 }; i < arr_->Length; ++i) {
                sw->WriteLine("array[" + Convert::ToString(i) + "]:
                \t" + Convert::ToString(arr_[i]));
            }
            sw->Close();
        }
        catch (Exception^ e) {
            MessageBox::Show("Unfortunately, sorted array can't be
            written to file '" + fileName + "' in folder '" +
            kSortedArraysFolder_ + "'.", "Error");

            return false;
        }

        return true;
    }

    String^ MainForm::generateInitialArrayFileName() {
        return "initial_array_" +
        Convert::ToString(currSortableArrayIdx_) + ".txt";
    }

```

```

ISortEngine^ MainForm::getConditionalSortEngine() {

    ISortEngine^ se;
    bool visualized = (visualizeCheckBox->Enabled &&
visualizeCheckBox->Checked);

    for (int i{ 0 }; i < sortingAlgorithmsNames_->Length; ++i) {

    }
    switch (algorithmSelectorDropDown->SelectedIndex)
    {
    case 0: {
        if (visualized) {
            se = gcnew VisualizableSelectionSortEngine(arr_,
kMaxVal_);
        }
        else {
            se = gcnew SelectionSortEngine(arr_);
        }
        break;
    }
    case 1: {
        if (visualized) {
            se = gcnew VisualizableHeapSortEngine(arr_,
kMaxVal_);
        }
        else {
            se = gcnew HeapSortEngine(arr_);
        }
        break;
    }
    case 2: {
        if (visualized) {
            se = gcnew VisualizableSmoothSortEngine(arr_,
kMaxVal_);
        }
        else {
            se = gcnew SmoothSortEngine(arr_);
        }
        break;
    }
    default:
        break;
    }

    return se;
}

```

```

String^ MainForm::generateSortedArrayFileName() {
    return "sorted_array_" +
    Convert::ToString(currSortableArrayIdx_) + ".txt";
}

System::Void MainForm::btnSort_Click(System::Object^ sender,
System::EventArgs^ e) {

    bool initialArrayWasWrittenToFile{
    saveInitialArrayToFile(generateInitialArrayFileName()) };

    ISortEngine^ se = getConditionalSortEngine();
    se->sort();

    bool sortedArrayWasWrittenToFile{
    saveSortedArrayToFile(generateSortedArrayFileName(), se-
    >getComparisonsCount(), se->getSwapsCount()) };

    printSortingMessage(initialArrayWasWrittenToFile,
    sortedArrayWasWrittenToFile, se->getComparisonsCount(), se-
    >getSwapsCount());

    if (se) {
        delete se;
    }

    ++currSortableArrayIdx_;
}

System::Void
MainForm::outputRichTextBox_TextChanged(System::Object^ sender,
System::EventArgs^ e) {

    outputRichTextBox->SelectionStart = outputRichTextBox->Text-
    >Length;
    outputRichTextBox->ScrollToCaret();
}

System::Void MainForm::btnReset_Click(System::Object^ sender,
System::EventArgs^ e) {

    int arrSize{ arraySizeNumericUpDown->Value };

    generateRandomArray(arrSize);

    printGenerationMessage();
}

```



```

    if (!btnSort->Enabled) {
        btnSort->Enabled = true;
    }

    if (arr_->Length > maxVisualizableArrSize_) {
        visualizeCheckBox->Enabled = false;
    }
    else {
        visualizeCheckBox->Enabled = true;
    }
}

```

Файл «SortingVisualizerForm.cpp»

```

#include "SortingVisualizerForm.h"

using namespace SortingAlgorithmsApplication;

SortingVisualizerForm::SortingVisualizerForm(array<int>^% arr, int
maxVal) : maxVal_{ maxVal } {

    InitializeComponent();

    arr_ = arr;
    arrCopy_ = gcnew array<int>(arr_->Length);
    arr_->CopyTo(arrCopy_, 0);
    hCoefficient_ = getHeightCoefficient();
    wCoefficient_ = getWidthCoefficient();
    g_ = this->panel->CreateGraphics();
}

SortingVisualizerForm::~~SortingVisualizerForm()
{
    if (components)
    {
        delete components;
    }

    if (arrCopy_) {
        delete arrCopy_;
    }

    if (g_) {
        delete g_;
    }
}

float SortingVisualizerForm::getHeightCoefficient() {

```

```

        return static_cast<float>(panel->Height) /
static_cast<float>(maxVal_);
}

float SortingVisualizerForm::getWidthCoefficient() {
    return static_cast<float>(panel->Width) /
static_cast<float>(arr_->Length);
}

void SortingVisualizerForm::InitializeComponent(void)
{
    this->panel = (gcnew System::Windows::Forms::Panel());
    this->SuspendLayout();
    //
    // panel
    //
    this->panel->Location = System::Drawing::Point(-1, 1);
    this->panel->Name = L"panel";
    this->panel->Size = System::Drawing::Size(1170, 700);
    this->panel->TabIndex = 0;
    //
    // SortingVisualizerForm
    //
    this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);
    this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
    this->BackColor = System::Drawing::SystemColors::Desktop;
    this->ClientSize = System::Drawing::Size(1169, 702);
    this->Controls->Add(this->panel);
    this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedSingle;
    this->MaximizeBox = false;
    this->MinimizeBox = false;
    this->Name = L"SortingVisualizerForm";
    this->ShowIcon = false;
    this->Text = L"SortingVisualizerForm";
    this->ResumeLayout(false);

}

void SortingVisualizerForm::drawHill(Brush^ brush, int idx, int
val) {

    g_->FillRectangle(brush, idx * wCoefficient_, panel->Height -
hCoefficient_ * val, wCoefficient_, static_cast<float>(panel-
>Height));
}

```

```

        g_->DrawRectangle(gcnew Pen(Brushes::Black), idx *
wCoefficient_, panel->Height - hCoefficient_ * val, wCoefficient_,
static_cast<float>(panel->Height));
    }

```

```

void SortingVisualizerForm::visualizeArray() {

```

```

    g_->FillRectangle(Brushes::Black, 0, 0, panel->Width, panel-
>Height);

```

```

    for (int i = 0; i < arr_->Length; ++i) {
        drawHill(Brushes::White, i, arr_[i]);
    }
}

```

```

void SortingVisualizerForm::visualizeSwap(int i, int j) {

```

```

    drawHill(Brushes::Red, i, arrCopy_[i]);
    drawHill(Brushes::Red, j, arrCopy_[j]);

```

```

    int temp = arrCopy_[i];
    arrCopy_[i] = arrCopy_[j];
    arrCopy_[j] = temp;

```

```

    sleep_for(1.0s / arr_->Length);

```

```

    g_->FillRectangle(Brushes::Black, i * wCoefficient_, 0.0,
wCoefficient_, static_cast<float>(panel->Height));
    g_->FillRectangle(Brushes::Black, j * wCoefficient_, 0.0,
wCoefficient_, static_cast<float>(panel->Height));

```

```

    sleep_for(1ns);

```

```

    drawHill(Brushes::White, i, arr_[i]);
    drawHill(Brushes::White, j, arr_[j]);
}

```

```

void SortingVisualizerForm::visualizeComparison(int i, int j) {

```

```

    drawHill(Brushes::Yellow, i, arr_[i]);
    drawHill(Brushes::Yellow, j, arr_[j]);

```

```

    sleep_for(1.0s / arr_->Length);

```

```

    drawHill(Brushes::White, i, arr_[i]);
    drawHill(Brushes::White, j, arr_[j]);
}

```

```

void SortingVisualizerForm::visualizeSortingTest(int breakIdx) {

    sleep_for(0.5s);

    if (breakIdx == -1) {
        for (int i = 0; i < arr_->Length; ++i) {
            drawHill(Brushes::LawnGreen, i, arr_[i]);
            sleep_for(0.001s);
        }
    }
    else {
        int currIdx{ 0 };

        while (currIdx != breakIdx) {
            drawHill(Brushes::LawnGreen, currIdx,
arr_[currIdx]);
            ++currIdx;
            sleep_for(0.001s);
        }

        drawHill(Brushes::Red, currIdx, arr_[currIdx]);
    }

    sleep_for(0.5s);
}

```

Файл «SortEngine.cpp»

```

#include "SortEngine.h"

long long SortEngine::getComparisonsCount() { return
comparisonsCount_; }

long long SortEngine::getSwapsCount() { return swapsCount_; }

SortEngine::SortEngine(array<int>^% arr) : swapsCount_{ 0 },
comparisonsCount_{ 0 } {

    arr_ = arr;
}

int SortEngine::sortingTest() {

    int firstUnsortedIdx{ -1 };
    for (int i{ 1 }; i < arr_->Length; ++i) {
        if (arr_[i - 1] > arr_[i]) {
            return i;
        }
    }
}

```

```

        return firstUnsortedIdx;
    }

    void SortEngine::swap(int i, int j) {
        int temp = arr_[i];
        arr_[i] = arr_[j];
        arr_[j] = temp;
    }

```

Файл «SelectionSortEngine.cpp»

```

#include "SelectionSortEngine.h"

SelectionSortEngine::SelectionSortEngine(array<int>^% arr) :
SortEngine(arr) {}

// Знаходження індексу найменшого елементу масиву в межах [left;
right]
int SelectionSortEngine::getIdxOfMinElement(int left, int right) {
    int minIdx{ left };

    for (int i{ left + 1 }; i <= right; ++i) {
        ++comparisonsCount_;
        if (arr_[i] < arr_[minIdx]) {
            minIdx = i;
        }
    }

    return minIdx;
}

void SelectionSortEngine::sort() {
    int currMinIdx{ 0 };

    for (int i{ 0 }; i < arr_->Length; ++i) {
        currMinIdx = getIdxOfMinElement(i, arr_->Length - 1);
        ++swapsCount_;
        swap(i, currMinIdx);
    }
}

```

Файл «HeapSortEngine.cpp»

```

#include "HeapSortEngine.h"

HeapSortEngine::HeapSortEngine(array<int>^% arr) : SortEngine(arr)
{}

```

```

int HeapSortEngine::getRightChildIdx(int rootIdx) {
    return (2 * rootIdx + 2);
}

int HeapSortEngine::getLeftChildIdx(int rootIdx) {
    return (2 * rootIdx + 1);
}

void HeapSortEngine::heapify(int heapSize, int rootIdx) {
    int maxIdx{ rootIdx };

    while (heapSize > rootIdx) {
        int leftChildIdx{ getLeftChildIdx(rootIdx) };
        if (leftChildIdx < heapSize) {
            ++comparisonsCount_;
            if (arr_[leftChildIdx] > arr_[maxIdx]) {
                maxIdx = leftChildIdx;
            }
        }

        int rightChildIdx{ getRightChildIdx(rootIdx) };
        if (rightChildIdx < heapSize) {
            ++comparisonsCount_;
            if (arr_[rightChildIdx] > arr_[maxIdx]) {
                maxIdx = rightChildIdx;
            }
        }

        if (maxIdx != rootIdx) {
            ++swapsCount_;
            swap(maxIdx, rootIdx);

            rootIdx = maxIdx;
        }
        else {
            break;
        }
    }
}

void HeapSortEngine::buildHeap() {
    for (int i{ arr_->Length / 2 - 1 }; i >= 0; --i) {
        heapify(arr_->Length, i);
    }
}

void HeapSortEngine::sort() {

```

```

    buildHeap();

    for (int i{ arr_->Length - 1 }; i > 0; --i) {
        ++swapsCount_;
        swap(i, 0);

        heapify(i, 0);
    }
}

```

Файл «SmoothSortEngine.cpp»

```

#include "SmoothSortEngine.h"

SmoothSortEngine::SmoothSortEngine(array<int>^% arr) :
SortEngine(arr) {
    calculateLeoNums();
}

void SmoothSortEngine::calculateLeoNums() {
    int leo1{ 1 }, leo2{ 1 };

    while (leo1 <= arr_->Length) {
        leoNums.push_back(leo1);

        int temp = leo1;
        leo1 = leo2;
        leo2 += (temp + 1);
    }
}

void SmoothSortEngine::calculateNextState(uint64_t *currState, int
*mergedHeapsLeoNumIdx) {

    *mergedHeapsLeoNumIdx = -1;

    // виняткова ситуація
    if ((*currState & 7) == 5) { // currState = x0101 (в двійковому
представленні)
        *currState += 3;          // currState = x1000
        *mergedHeapsLeoNumIdx = 3;
    }
    else { // пошук двох послідовних одиничних бітів
        uint64_t firstTwoSideBySideBits = *currState;
        int idx = 0;

        // поки firstTwoSideBySideBits != 0 та
firstTwoSideBySideBits != x011

```

```

        while (firstTwoSideBySideBits && (firstTwoSideBySideBits &
3) != 3) {
            firstTwoSideBySideBits >>= 1;
            idx++;
        }

        if ((firstTwoSideBySideBits & 3) == 3) { //
firstTwoSideBySideBits = x011
            *currState += (uint64_t(1) << idx); // currState =
x100x
            *mergedHeapsLeoNumIdx = idx + 2;
        }
        else {
            if (*currState & 1) { // currState = x001
                *currState |= 2; // currState = x011
            }
            else { // currState = xx00
                *currState |= 1; // currState = xx01
            }
        }
    }
}

```

```

void SmoothSortEngine::calculatePrevState(uint64_t *currState) {

    if ((*currState & 15) == 8) { // currState = x1000
        *currState -= 3; // currState = x0101
    }
    else {
        if (*currState & 1) { // currState = xxx1
            if ((*currState & 3) == 3) { // currState = x011
                *currState ^= 2; // currState = x001
            }
            else { // currState = xx01
                *currState ^= 1; // currState = xx00
            }
        }
        else { // пошук першого одиничного біта
            uint64_t firstSingleBit = *currState;
            int idx = 0;

            // поки firstSingleBit != 0 та firstSingleBit != x001
            while (firstSingleBit && !(firstSingleBit & 1)) {
                firstSingleBit >>= 1;
                idx++;
            }
        }
    }
}

```



```

        if (firstSingleBit) { //
currState = xx1000 (наприклад)
            *currState ^= (uint64_t(1) << idx);
            *currState |= (uint64_t(1) << (idx - 1));
            *currState |= (uint64_t(1) << (idx - 2)); //
currState = xx0110
        }
        else {
            *currState = 0;
        }
    }
}

void SmoothSortEngine::siftDown(int heapLeoNumIdx, int rootIdx) {
    int currNodeIdx = rootIdx;

    while (heapLeoNumIdx >= 2)
    {
        int maxIdx = currNodeIdx;
        int prevHeapLeoNumIdx{ heapLeoNumIdx };

        int rightChildIdx = currNodeIdx - 1;
        ++comparisonsCount_;
        if (arr_[rightChildIdx] > arr_[maxIdx]) {
            prevHeapLeoNumIdx = heapLeoNumIdx - 2;
            maxIdx = rightChildIdx;
        }

        int leftChildIdx = rightChildIdx - leoNums[heapLeoNumIdx -
2];
        ++comparisonsCount_;
        if (arr_[leftChildIdx] > arr_[maxIdx]) {
            prevHeapLeoNumIdx = heapLeoNumIdx - 1;
            maxIdx = leftChildIdx;
        }

        if (currNodeIdx != maxIdx) {
            ++swapsCount_;
            swap(currNodeIdx, maxIdx);
            heapLeoNumIdx = prevHeapLeoNumIdx;
            currNodeIdx = maxIdx;
        }
        else {
            break;
        }
    }
}

```

```

void SmoothSortEngine::buildHeapPool(uint64_t *currState) {
    for (int i{ 0 }; i < arr_->Length; ++i) {

        int currHeapLeoNumIdx{ -1 };
        calculateNextState(currState, &currHeapLeoNumIdx);

        if (currHeapLeoNumIdx != -1) {
            siftDown(currHeapLeoNumIdx, i);
        }
    }
}

void SmoothSortEngine::findMaxAmongRoots(uint64_t currState, int
lastRootIdx, int* heapLeoNumIdx, int* maxRootIdx) {
    int currHeapLeoNumIdx{ 0 };
    while (!(currState & 1)) {
        currState >>= 1;
        currHeapLeoNumIdx++;
    }

    *maxRootIdx = lastRootIdx;
    *heapLeoNumIdx = currHeapLeoNumIdx;
    int currRootIdx{ lastRootIdx - leoNums[currHeapLeoNumIdx] };

    currState >>= 1;
    currHeapLeoNumIdx++;
    while (currState) {
        if (currState & 1) {
            ++comparisonsCount_;
            if (arr_[currRootIdx] > arr_[*maxRootIdx]) {
                *maxRootIdx = currRootIdx;
                *heapLeoNumIdx = currHeapLeoNumIdx;
            }
            currRootIdx -= leoNums[currHeapLeoNumIdx];
        }
        currState >>= 1;
        currHeapLeoNumIdx++;
    }
}

void SmoothSortEngine::sort() {

    uint64_t currState{ 0 };
    buildHeapPool(&currState);

    for (int i{ (int)arr_->Length - 1 }; i >= 0; --i) {
        int heapLeoNumIdx{ -1 };

```

```

        int maxRootIdx{ 0 };
        findMaxAmongRoots(currState, i, &heapLeoNumIdx,
&maxRootIdx);

        if (maxRootIdx != i) {
            ++swapsCount_;
            swap(i, maxRootIdx);
            siftDown(heapLeoNumIdx, maxRootIdx);
        }
        calculatePrevState(&currState);
    }
}

```

Файл «VisualizableSelectionSortEngine.cpp»

```

#include "VisualizableSelectionSortEngine.h"

VisualizableSelectionSortEngine::VisualizableSelectionSortEngine(ar
ray<int>^% arr, int maxVal) : SelectionSortEngine(arr) {

    svf_ = gcnew
SortingAlgorithmsApplication::SortingVisualizerForm(arr, maxVal);
}

VisualizableSelectionSortEngine::~~VisualizableSelectionSortEngine()
{
    delete svf_;
}

int VisualizableSelectionSortEngine::getIdxOfMinElement(int left,
int right) {
    int minIdx{ left };

    for (int i{ left + 1 }; i <= right; ++i) {
        ++comparisonsCount_;
        svf_->visualizeComparison(i, minIdx);
        if (arr_[i] < arr_[minIdx]) {
            minIdx = i;
        }
    }

    return minIdx;
}

void VisualizableSelectionSortEngine::sort() {
    svf_->Show();

    svf_->visualizeArray();
}

```

```

int currMinIdx{ 0 };

for (int i{ 0 }; i < arr_->Length; ++i) {
    currMinIdx = getIdxOfMinElement(i, arr_->Length - 1);

    ++swapsCount_;
    swap(i, currMinIdx);
    svf_->visualizeSwap(i, currMinIdx);
}

svf_->visualizeSortingTest(sortingTest());

svf_->Close();
}

```

Файл «VisualizableHeapSortEngine.cpp»

```

#include "VisualizableHeapSortEngine.h"

VisualizableHeapSortEngine::VisualizableHeapSortEngine(array<int>^%
arr, int maxVal) : HeapSortEngine(arr) {

    svf_ = gcnew
SortingAlgorithmsApplication::SortingVisualizerForm(arr, maxVal);
}

VisualizableHeapSortEngine::~~VisualizableHeapSortEngine() {
    delete svf_;
}

void VisualizableHeapSortEngine::heapify(int heapSize, int rootIdx)
{
    int maxIdx{ rootIdx };

    while (heapSize > rootIdx) {
        int leftChildIdx{ getLeftChildIdx(rootIdx) };
        if (leftChildIdx < heapSize) {
            svf_->visualizeComparison(leftChildIdx, maxIdx);
            ++comparisonsCount_;
            if (arr_[leftChildIdx] > arr_[maxIdx]) {
                maxIdx = leftChildIdx;
            }
        }

        int rightChildIdx{ getRightChildIdx(rootIdx) };
        if (rightChildIdx < heapSize) {
            svf_->visualizeComparison(rightChildIdx, maxIdx);
            ++comparisonsCount_;
            if (arr_[rightChildIdx] > arr_[maxIdx]) {

```

```

        maxIdx = rightChildIdx;
    }
}

    if (maxIdx != rootIdx) {
        ++swapsCount_;
        swap(maxIdx, rootIdx);
        svf_>visualizeSwap(maxIdx, rootIdx);

        rootIdx = maxIdx;
    }
    else {
        break;
    }
}
}

```

```

void VisualizableHeapSortEngine::sort() {
    svf_>Show();

    svf_>visualizeArray();

    buildHeap();

    for (int i{ arr_>Length - 1 }; i > 0; --i) {
        ++swapsCount_;
        swap(i, 0);
        svf_>visualizeSwap(i, 0);

        heapify(i, 0);
    }

    svf_>visualizeSortingTest(sortingTest());

    svf_>Close();
}

```

Файл «VisualizableSmoothSortEngine.cpp»

```

#include "VisualizableSmoothSortEngine.h"

VisualizableSmoothSortEngine::VisualizableSmoothSortEngine(array<int>^% arr, int maxVal) : SmoothSortEngine(arr) {
    svf_ = gcnew
    SortingAlgorithmsApplication::SortingVisualizerForm(arr, maxVal);

    calculateLeoNums();
}

```

```

VisualizableSmoothSortEngine::~VisualizableSmoothSortEngine() {
    delete svf_;
}

void VisualizableSmoothSortEngine::siftDown(int heapLeoNumIdx, int
rootIdx) {
    int currNodeIdx = rootIdx;

    while (heapLeoNumIdx >= 2)
    {
        int maxIdx = currNodeIdx;
        int prevHeapLeoNumIdx{ heapLeoNumIdx };

        int rightChildIdx = currNodeIdx - 1;
        svf_->visualizeComparison(rightChildIdx, maxIdx);
        ++comparisonsCount_;
        if (arr_[rightChildIdx] > arr_[maxIdx]) {
            prevHeapLeoNumIdx = heapLeoNumIdx - 2;
            maxIdx = rightChildIdx;
        }

        int leftChildIdx = rightChildIdx - leoNums[heapLeoNumIdx -
2];
        svf_->visualizeComparison(rightChildIdx, maxIdx);
        ++comparisonsCount_;
        if (arr_[leftChildIdx] > arr_[maxIdx]) {
            prevHeapLeoNumIdx = heapLeoNumIdx - 1;
            maxIdx = leftChildIdx;
        }

        if (currNodeIdx != maxIdx) {
            ++swapsCount_;
            swap(currNodeIdx, maxIdx);
            svf_->visualizeSwap(currNodeIdx, maxIdx);
            heapLeoNumIdx = prevHeapLeoNumIdx;
            currNodeIdx = maxIdx;
        }
        else {
            break;
        }
    }
}

void VisualizableSmoothSortEngine::findMaxAmongRoots(uint64_t
currState, int lastRootIdx, int* heapLeoNumIdx, int* maxRootIdx) {
    int currHeapLeoNumIdx{ 0 };
    while (!(currState & 1)) {
        currState >>= 1;
    }
}

```

```

        currHeapLeoNumIdx++;
    }

    *maxRootIdx = lastRootIdx;
    *heapLeoNumIdx = currHeapLeoNumIdx;
    int currRootIdx{ lastRootIdx - leoNums[currHeapLeoNumIdx] };

    currState >>= 1;
    currHeapLeoNumIdx++;
    while (currState) {
        if (currState & 1) {
            ++comparisonsCount_;
            svf_->visualizeComparison(currRootIdx, *maxRootIdx);
            if (arr_[currRootIdx] > arr_[*maxRootIdx]) {
                *maxRootIdx = currRootIdx;
                *heapLeoNumIdx = currHeapLeoNumIdx;
            }
            currRootIdx -= leoNums[currHeapLeoNumIdx];
        }
        currState >>= 1;
        currHeapLeoNumIdx++;
    }
}

void VisualizableSmoothSortEngine::sort() {
    svf_->Show();

    svf_->visualizeArray();

    uint64_t currState{ 0 };
    buildHeapPool(&currState);

    for (int i{ (int)arr_->Length - 1 }; i >= 0; --i) {
        int heapLeoNumIdx{ -1 };
        int maxRootIdx{ 0 };
        findMaxAmongRoots(currState, i, &heapLeoNumIdx,
&maxRootIdx);

        if (maxRootIdx != i) {
            ++swapsCount_;
            swap(i, maxRootIdx);
            svf_->visualizeSwap(i, maxRootIdx);
            siftDown(heapLeoNumIdx, maxRootIdx);
        }
        calculatePrevState(&currState);
    }

    svf_->visualizeSortingTest(sortingTest());
}

```

```
        svf_->Close();  
    }
```

Файл «main.cpp»

```
#include "MainForm.h"
```

```
using namespace System;  
using namespace System::Windows::Forms;
```

```
int main() {
```

```
    Application::EnableVisualStyles();  
    Application::SetCompatibleTextRenderingDefault(false);  
    Application::Run(gcnew  
SortingAlgorithmsApplication::MainForm);
```

```
    return 0;  
}
```