**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ЛЕСІ УКРАЇНКИ**

**Кафедра комп'ютерних наук та кібербезпеки**

На правах рукопису

**ШЕПЕЛЮК ПЕТРО ВІКТОРОВИЧ**

# ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМИ ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ СОРТУВАННЯ

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Робота на здобуття першого рівня «бакалавр»

Науковий керівник:
**ГЛИНЧУК ЛЮДМИЛА ЯРОСЛАВІВНА,**
кандидат фізико-математичних наук, доцент
кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ

Протокол №_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 2024 р.
Завідувач кафедри

(_____) Гришанович Т. О.

**ЛУЦЬК - 2024**

**ЗМІСТ**

# ВСТУП

**Актуальність теми.** У сучасному світі інформаційних технологій знання алгоритмів сортування є одним із основних компонентів навчання програмування та комп'ютерних наук. Удосконалення цих алгоритмів є важливим для розвитку базових навичок програмістів. Але через абстрактність і складність теоретичного матеріалу вивчення алгоритмів сортування може бути складним для здобувачів освіти. Одним із ефективних способів спрощення даного процесу є використання візуалізації алгоритмів сортування.

Візуалізація алгоритмів сортування дозволяє спостерігати за динамікою їх роботи, що значно полегшує розуміння логіки та механіки їх виконання. Візуальні засоби сприяють кращому запам'ятовуванню та розвитку інтуїтивного розуміння, що робить їх особливо важливими в навчальних процесах. Візуалізація також робить навчання більш інтерактивним і захоплюючим, залучаючи здобувачів освіти до активного вивчення матеріалу.

Сучасні технології пропонують широкий спектр інструментів, які можна використовувати для створення візуалізацій. Тим не менш, деякі існуючі програми не відповідають потребам здобувачів освіти і викладачів. Багато з них мають складний інтерфейс або обмежений набір функцій, що ускладнює їх використання для навчання. Таким чином, надзвичайно важливо створити нову програму для візуалізації алгоритмів сортування, яка буде проста у використанні та має широкий діапазон функцій.

Дослідження в цій галузі сприятиме покращенню якості навчання програмуванню, що дозволить здобувачам освіти краще зрозуміти складні ідеї та швидше отримати навички, необхідні для роботи. Така програма також може бути використана не лише у вищих навчальних закладах, але й у школах і на курсах програмування, розширюючи її застосування.

**Мета роботи –** розробити програму для візуалізації алгоритмів сортування.

Для досягнення мети потрібно виконати такі **завдання**:

- провести огляд та порівняння різних алгоритмів сортування;

- оцінити ефективність кожного алгоритму за критеріями, такими як часова складність, простота реалізації, стабільність та придатність для візуалізації;
- вибрати найбільш підходящі алгоритми для реалізації в програмі;
- здійснити огляд існуючих програм для візуалізації алгоритмів сортування;
- виявити сильні та слабкі сторони існуючих програм, щоб використовувати отриману інформацію для вдосконалення власного продукту;
- реалізувати вибрані алгоритми сортування та забезпечити їхню візуалізацію;
- виконати тестування на різних наборах даних, щоб перевірити коректність роботи алгоритмів та їхню ефективність;
- виявити та виправити всі помилки, що виникли під час тестування, забезпечуючи стабільну та надійну роботу програми.

**Об’єкт дослідження –** алгоритми сортування.

**Предмет дослідження –** процес проєктування та розробки програми для візуалізації алгоритмів сортування за допомогою бібліотеки SFML.

**Матеріал дослідження.**

У ході виконання бакалаврської роботи було опрацьовано широкий спектр джерел фактичного матеріалу. Це включало теоретичні дослідження, наукові статті, підручники, документацію з програмування та практичні приклади. Основні джерела матеріалу, використані у роботі, охоплюють наступні категорії:

- наукові та технічні джерела з теорії алгоритмів і структур даних;
- технічна документація та інструменти для програмування;
- освітні ресурси та інтернет-курси;
- реалізації та приклади коду з відкритих джерел.

**Практичне значення одержаних результатів.**

Розроблений програмний інструмент може стати ефективним навчальним засобом для здобувачів освіти і викладачів, що вивчають курси з алгоритмів та структур даних. Інтерактивна візуалізація алгоритмів сортування надає здобувачам освіти можливість візуально розуміти процеси, що відбуваються під

час виконання різних алгоритмів. Це сприяє кращому засвоєнню матеріалу та полегшує пояснення складних концепцій.

**Апробація результату роботи.**

Результати роботи були представлені та обговоренні на наступних конференціях:

- І Міжнародна науково-практична конференція «Проблеми комп'ютерних наук, програмного моделювання та безпеки цифрових систем». Луцьк-Світязь, 13–16 червня 2024 р.
- XVIII Міжнародна науково-практична конференція студентів і аспірантів «Молода наука Волині: пріоритети та перспективи досліджень» 14 – 15 травня 2024 року

# РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ АЛГОРИТМІВ СОРТУВАННЯ ТА ЇХ ВІЗУАЛІЗАЦІЇ

## 1.1 Поняття алгоритмів сортування

Алгоритм сортування – це конкретний набір інструкцій, який визначає послідовність дій для упорядкування елементів у визначеному порядку [1]. Порядок може бути заснований на будь-якому критерії, наприклад, на числовому значенні, алфавітному порядку або даті. Існує багато різних алгоритмів сортування, кожен з яких має свої переваги та недоліки. Основна мета алгоритму сортування полягає у тому, щоб зробити дані більш організованими та доступними для подальшого використання.

Алгоритми сортування є надзвичайно важливим компонентом для роботи з даними, оскільки вони дозволяють упорядковувати набір елементів у певному структурованому порядку [1]. Алгоритми сортування грають ключову роль в повсякденному житті програмістів і дослідників у галузі обробки даних. Знання та вміння користуватися різними видами алгоритмів сортування є важливою новиною, яка допомагає виконувати завдання розробки програм та оптимізувати процеси обробки даних. У результаті, набір елементів може бути швидко та ефективно упорядкований, що є основою для успішної роботи з даними та досягнення бажаних результатів.

Сортування даних є однією з найважливіших операцій у комп'ютерних науках та інформаційних технологіях. Застосування алгоритмів сортування дозволяє виконувати ефективну обробку великих обсягів інформації, що робить їх незамінними інструментами для програмістів та аналітиків даних. Подальше розширення класифікації алгоритмів сортування сприяє більш точному вибору оптимального підходу до конкретної задачі. У залежності від вимог щодо часу виконання, пам'яті, стабільності та інших параметрів, розглядаються різні види сортування, такі як обмінне сортування, вибіркове сортування, злиття, вставка та багато інших [2].

Сучасні алгоритми сортування використовуються для різних видів даних, включаючи числа, рядки, об'єкти та багатовимірні структури даних [3]. Вони забезпечують не лише правильну організацію даних, але й покращують швидкодію програм та забезпечують ефективну роботу з великими наборами даних. Навички використання алгоритмів сортування є ключовими у професійній діяльності програміста, оскільки вони дозволяють створювати оптимальні рішення та забезпечувати високу швидкодію програмних продуктів.

Важливо також зазначити, що алгоритми сортування мають значний вплив на розробку операційних систем, баз даних, пошукових систем та інших інформаційних технологій [3]. Використання оптимальних алгоритмічних рішень допомагає покращити продуктивність систем, зменшити час виконання операцій і зменшення ресурсів [4]. Знання та розуміння алгоритмів сортування є важливим елементом професійного росту в сфері обробки даних та інформаційних технологій.

### 1.2 Класифікація алгоритмів сортування

Існує декілька способів класифікувати алгоритми сортування. Найпоширеніші класифікації ґрунтуються за місцем виконання і часовою складністю. Алгоритми сортування за місцем виконання є внутрішні і зовнішні [5].

Внутрішні алгоритми сортування сортують дані в оперативній пам'яті. Вони зазвичай швидші, ніж зовнішні алгоритми сортування, але можуть бути обмежені розміром оперативної пам'яті.

Зовнішні алгоритми сортування сортують дані на диску. Вони використовуються, коли набір даних занадто великий, щоб поміститися в оперативній пам'яті [6].

Часова складність алгоритму сортування – це міра того, скільки часу потрібно алгоритму для сортування даних. Вона зазвичай виражається у позначеннях Big O, які описують поведінку алгоритму при збільшенні розміру вхідних даних [7].

Алгоритми сортування поділяються на наступні категорії відповідно до їхньої часової складності:

- алгоритми з часовою складністю O(n^2): ці алгоритми мають квадратичну часову складність, що означає, що час, необхідний для сортування, зростає квадратично з розміром вхідних даних. Такі алгоритми включають сортування вставкою та бульбашкове сортування [7];
- алгоритми з часовою складністю O(n log n): у цих алгоритмах час, необхідний для сортування, зростає логарифмічно з розміром вхідних даних. Такі алгоритми включають швидке сортування та сортування злиттям [7];
- алгоритми з лінійною часовою складністю O(n): ці алгоритми мають лінійну часову складність, що означає, що час, необхідний для сортування, зростає лінійно з розміром вхідних даних [7]. Прикладом такого алгоритму є сортування підрахунком.

Ще одним важливим критерієм є швидкодія алгоритму у різних обставинах. Деякі алгоритми можуть бути більш ефективними для великих масивів даних, тоді як інші можуть показувати кращі результати для невеликих масивів [8]. Важливо враховувати цей фактор при виборі алгоритму сортування для конкретної задачі. Наприклад, алгоритми сортування злиттям можуть бути швидшими для великих масивів, оскільки вони дозволяють розбити задачу на менші підзадачі і об’єднати їхні результати в кінцевий відсортований масив.

Наступним важливим показником, який можна використовувати при класифікації алгоритмів сортування є стабільність алгоритму. Стабільні алгоритми зберігають відносний порядок елементів з однаковим ключем після сортування. Це особливо корисно у випадках, коли необхідно сортувати дані, зберігаючи їх по певному критерію [8]. Наприклад, якщо необхідно сортувати список користувачів за алфавітом та за рейтингом, стабільні алгоритми можуть гарантувати, що користувачі з однаковим рейтингом будуть зберігати своє положення по алфавіту.

Вибір найбільш ефективного алгоритму залежить від різних критеріїв, таких як методи обробки даних, швидкодія, стабільність і інші показники.

Належне дослідження і аналіз цих критеріїв допоможе знайти оптимальний алгоритм сортування для кожного конкретного завдання. Важливо також пам'ятати, що вибір алгоритму сортування може залежати від обмежень, таких як обсяг даних, доступна пам'ять та інші фактори. Враховуючи всі ці аспекти, можна забезпечити оптимальну обробку даних та досягти потрібних результатів.

## 1.3 Основні алгоритми сортування

### 1.3.1 Алгоритми сортування з використанням порівнянь

Алгоритми сортування з використанням порівнянь використовують порівняння елементів масиву для їх відсортування [9]. Серед таких алгоритмів є сортування вибором (Selection Sort), вставками (Insertion Sort), бульбашкою (Bubble Sort), злиттям (Merge Sort), швидке сортування (Quicksort) та сортування Шелла. Кожен з них має свої особливості та ефективність у певних випадках сортування.

Сортування вибором (Selection Sort) – це простий алгоритм сортування, який шукає найменший елемент і ставить його на перше місце, потім шукає наступний найменший елемент і ставить його на друге місце [10]. Цей процес триває до тих пір, поки всі елементи не будуть відсортовані. Хоча алгоритм сортування вибором досить простий, його ефективність залежить від кількості порівнянь, тому для великих масивів його не варто застосовувати. Однак, варто зазначити, що сортування вибором може бути корисним для невеликих масивів або вже частково відсортованих даних, де кількість порівнянь може бути мінімізована. Також, існують різні варіації даного алгоритму, наприклад, сортування вибором з виключенням мінімального елемента, що може підвищити його ефективність. Отже, з урахуванням обмежень алгоритм сортування вибором може бути використаним у відповідних ситуаціях, проте для більших обсягів даних краще використовувати більш ефективні алгоритми сортування.

Алгоритм працює наступним чином:

1. Повторювати кроки 2-4 для кожного елемента масиву, починаючи з першого.
2. Знайти найменший елемент у невідсортованій частині масиву.
3. Поміняти місцями цей найменший елемент з першим елементом невідсортованої частини.
4. Перейти до наступного елемента масиву.

Сортування вставками (Insertion Sort) – алгоритм, який працює шляхом поступового порівняння та вставки елементів у відсортовану послідовність [10]. Алгоритм чудово підходить для сортування невеликих масивів, оскільки його простота і невелика кількість обчислень забезпечують швидке виконання. Завдяки цьому Insertion Sort часто застосовується у випадках, коли потрібно впорядкувати невеликі списки, наприклад, під час сортування допоміжних структур у більш складних алгоритмах. Цей алгоритм дозволяє ефективно впоратися зі сортуванням навіть середніх розмірів масивів. Додатково, він може бути використаний для сортування даних, що містяться в базі даних, та у багатьох інших випадках, коли потрібно впоратися зі впорядкуванням великих об'ємів інформації. Сортування вставками є одним з найпростіших та найбільш ефективних алгоритмів сортування. Він може бути реалізований швидко та з легкістю у багатьох мовах програмування.

Суть роботи алгоритму така:

1. Повторювати кроки 2-4 для кожного елемента масиву, починаючи з другого.
2. Зберегти поточний елемент в змінну.
3. Перемістити елементи у відсортованій частині, які більші за поточний елемент, на одну позицію вправо.
4. Вставити поточний елемент у правильне місце у відсортованій частині.

Приклад його застосування приведений на рисунку 1.1

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

void insertionSort(std::vector<int>& arr) {
    int n = arr.size();
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Рисунок 1.1 – Insertion Sort мовою програмування C++

Сортування бульбашкою (Bubble Sort) – це простий алгоритм сортування, який послідовно порівнює сусідні елементи масиву і, якщо вони розміщені в неправильному порядку, змінює їх місцями [11]. Цей процес повторюється для всіх елементів масиву, поки масив повністю не відсортується. Сортування бульбашкою не є дуже ефективним алгоритмом для сортування великих масивів через його квадратичну складність. Використовувати його на великих масивах може призвести до значної затримки, тому для більш швидкого сортування рекомендується використовувати інші, більш ефективні алгоритми сортування. Однак, з іншого боку, сортування бульбашкою має свої переваги. Наприклад, алгоритм відмінно підходить для відсортування масивів, в яких багато вже відсортованої інформації, оскільки ці елементи не потребують зміни місцями. Крім того, цей алгоритм може бути легко модифікований для сортування за спаданням, заміни даними для порівняння та інших потреб.

Алгоритм працює наступним чином:

1. Повторювати кроки 2-4 до тих пір, поки масив не буде відсортований.
2. Перейти по масиву з початку до передостаннього елемента.
3. Якщо сусідні елементи знаходяться в неправильному порядку, поміняти їх місцями.
4. Зменшити область перевірки на один елемент, оскільки останній елемент вже на своєму місці.

Сортування злиттям (Merge Sort) – це ефективний алгоритм, який розбиває початковий масив на менші частини, і кожну з цих частин сортує окремо [11]. Потім відсортовані частини об'єднуються в один великий сортований масив. Використання даного методу дозволяє ефективно сортувати навіть дуже великі масиви з складністю O(n log n). Сортування злиттям є ефективним методом сортування, проте його використання призводить до необхідності використання додаткового простору пам'яті для зберігання відсортованих частин масиву. Це слід враховувати при реалізації алгоритму. Однак, переваги сортування злиттям полягають у його можливості застосовуватися не тільки для сортування масивів, але й для сортування інших структур даних, наприклад, зв'язних списків або бінарних дерев.

Принцип роботи алгоритму:

1. Рекурсивно розділяти масив на дві частини, поки кожна частина не буде містити один елемент.
2. Зливати дві відсортовані частини в один відсортований масив, повторюючи цей процес рекурсивно до тих пір, поки весь масив не буде злитий.

Швидке сортування (Quicksort) – ефективний алгоритм сортування, який використовує стратегію “розділяй та пануй” [11]. Він швидко сортує масив шляхом розбиття його на менші частини за допомогою опорного елемента та рекурсивно сортує їх. Швидке сортування має середню складність O(n log n), що означає, що для масиву з n елементів його час виконання буде пропорційним до log n, враховуючи розділення масиву на підмасиви під час кожного кроку. Це зробить алгоритм ідеальним для використання при сортуванні великих масивів даних. Швидке сортування є одним з найпоширеніших алгоритмів сортування в практиці, оскільки воно поєднує ефективність та простоту реалізації. Він здатний сортувати масиви будь-якого розміру та у будь-якому порядку за мінімальний час.

Суть роботи алгоритму:

1. Вибрати опорний елемент (зазвичай перший, середній або випадковий).

2. Розділити масив на дві частини: одна частина з елементами меншими за опорний, інша з елементами більшими
3. Рекурсивно сортувати обидві частини.
4. Об'єднати дві відсортовані частини, поміщаючи опорний елемент на його остаточне місце.

Сортування Шелла – це модифікація сортування вставками, яка використовує послідовний набір інтервалів для сортування елементів [12]. Цей алгоритм поєднує в собі властивості сортування вставками та сортування вибором, що дозволяє покращити його ефективність. Сортування Шелла є ефективним для великих масивів та має складність, яка залежить від вибраної послідовності інтервалів. Цей алгоритм був запропонований Дональдом Шеллом у 1959 році і він вважається одним з перших ефективних сортувань, призначених для сортування великих об'ємів даних [12]. У порівнянні із звичайним сортуванням вставками, сортування Шелла пропонує спосіб зменшення кількості перестановок та порівнянь, що призводить до покращення часу виконання алгоритму. Ідея полягає в тому, щоб спочатку відсортувати елементи на віддалених відстанях, а потім поступово зменшувати ці відстані до 1, що дозволяє покращити якість сортування.

Алгоритм працює наступним чином:

1. Вибрати початковий крок (відстань між елементами для порівняння).
2. Повторювати кроки 3-4 для кожного кроку, поки крок не стане 1.
3. Виконати Insertion Sort для елементів, розташованих на поточній відстані.
4. Зменшити крок і повторити.

### 1.3.2 Алгоритми сортування без порівнянь

До алгоритмів сортування, які не використовують операції порівняння відносяться сортування підрахунком (Counting Sort), блочне сортування (Bucket Sort) та сортування за розрядами (Radix Sort). Ці алгоритми базуються на різних підходах до упорядкування даних та можуть бути ефективними для різноманітних видів завдань, де порівняння займає багато часу [10].

Сортування підрахунком (Counting Sort) – метод сортування, який використовує підрахунок кількості елементів з певним значенням у вхідному масиві [12]. Цей алгоритм ефективний для випадків, коли діапазон можливих значень обмежений, а кількість унікальних значень невелика порівняно з загальною кількістю елементів. Сортування підрахунком має лінійну складність за часом, що робить його привабливим варіантом для впорядкування великих масивів даних. Алгоритм сортування підрахунком може бути використаний в різних сферах, таких як наукові дослідження, програмування, статистика й багато інших. Він забезпечує точність та швидкість обробки інформації за допомогою використання різних методів підрахунку, таких як групування, порівняння та індексування. Завдяки своїй ефективності та простоті в реалізації, сортування підрахунком може бути використано в багатьох випадках, де потрібне швидке та точне впорядкування даних. Загалом, сортування підрахунком є потужним засобом, який може значно полегшити роботу з сортуванням даних. Працюючи за лінійний час, він забезпечує ефективну обробку без втрати точності.

Суть роботи алгоритму наступна:

1. Знайти мінімальне і максимальне значення в масиві.
2. Створити масив підрахунків для кожного значення в діапазоні від мінімального до максимального.
3. Підрахувати кількість кожного значення в оригінальному масиві.
4. Змінити масив підрахунків так, щоб кожен елемент показував кількість елементів менших або рівних цьому значенню.
5. Створити відсортований масив, використовуючи масив підрахунків для визначення позицій елементів.

Приклад використання можна спостерігати на рисунку 1.2.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

int findMax(const std::vector<int>& arr) {
    return *std::max_element(arr.begin(), arr.end());
}

void countingSort(std::vector<int>& arr) {
    int maxElement = findMax(arr);
    std::vector<int> count(maxElement + 1, 0);

    for (int num : arr) {
        count[num]++;
    }

    int index = 0;
    for (int i = 0; i <= maxElement; ++i) {
        while (count[i] > 0) {
            arr[index++] = i;
            count[i]--;
        }
    }
}
```

Рисунок 1.2 – Counting Sort мовою програмування C++

Блочне сортування, або Bucket Sort, є алгоритмом сортування, який розділяє елементи вхідного масиву в групи, або блоки, і потім сортує кожну групу окремо [13]. Цей метод є дуже ефективним, коли вхідні дані рівномірно розподілені в заданому діапазоні. Блочне сортування може бути використане для сортування обмеженої кількості унікальних значень, що дозволяє знизити час сортування порівняно з іншими методами, які вимагають порівнянь. За допомогою даного алгоритму можна впоратися з масивом величиною в сотні тисяч або навіть мільйони елементів. Bucket Sort є ефективним і простим для реалізації. Він використовує властивість розподілення даних для ефективного сортування. Застосовуючи цей алгоритм, можна отримати відсортований масив за достатньо короткий час. Bucket Sort є одним із найшвидших алгоритмів сортування для невеликого діапазону цілих чисел.

Алгоритм працює в наступному порядку:

1. Розбити масив на кількість блоків, рівну діапазону значень, розділеному на однакові інтервали.
2. Розподілити елементи списку по блоках відповідно до їх значення.
3. Сортувати елементи в кожному блокові окремо, використовуючи будь-який алгоритм сортування.
4. Об'єднати відсортовані елементи з усіх блоків у послідовність.

На рисунку 1.3 зображено використання блочного сортування.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

void bucketSort(std::vector<float>& arr) {
    int n = arr.size();
    if (n <= 0)
        return;

    std::vector<std::vector<float>> buckets(n);

    for (int i = 0; i < n; ++i) {
        int bucketIndex = n * arr[i];
        buckets[bucketIndex].push_back(arr[i]);
    }

    for (int i = 0; i < n; ++i) {
        std::sort(buckets[i].begin(), buckets[i].end());
    }

    int index = 0;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < buckets[i].size(); ++j) {
            arr[index++] = buckets[i][j];
        }
    }
}
```

Рисунок 1.3 – Bucket Sort мовою програмування C++

Сортування за розрядами, або Radix Sort, є дуже ефективним алгоритмом, що базується на розрядному представленні чисел для їх сортування [14]. Ідея полягає в тому, щоб спочатку впорядкувати всі елементи за молодшим розрядом, потім стабільно впорядкувати за другим розрядом, потім за третім і так далі аж до найстаршого. Оскільки, припускається, що кожен розряд приймає значення з невеликого діапазону, то кожен цикл впорядкування можна виконувати швидко і з малими затратами пам'яті. Це означає, що алгоритм впорядковує елементи спочатку за найменш значущим розрядом, потім за наступним, і так далі, поки всі розряди не будуть враховані. Цей метод дозволяє ефективно впорядкувати числа або рядки, де розряди можуть бути виявлені та зрозумілі, без використання порівнянь. Це робить його особливо привабливим в

певних випадках, коли потрібно впорядкувати великі набори даних або коли час виконання є критичним. Таким чином, сортування за розрядом є потужним і надійним алгоритмом, який знайшов широке застосування в різних галузях, включаючи комп'ютерну науку, фінанси, телекомунікації та багато іншого.

Суть роботи алгоритму:

1. Визначити максимальне число для визначення кількості цифр.
2. Повторювати кроки 3-4 для кожної цифри, починаючи з найменш значущої.
3. Використати алгоритм сортування (наприклад, Counting Sort) для сортування чисел по поточній цифрі.
4. Об'єднати відсортовані числа для переходу до наступної цифри.

Як використати Radix Sort можна побачити на рисунку 1.4.

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

int getMax(const std::vector<int>& arr) {
    return *std::max_element(arr.begin(), arr.end());
}

void countingSort(std::vector<int>& arr, int exp) {
    int n = arr.size();
    std::vector<int> output(n);
    int count[10] = {0};

    for (int i = 0; i < n; i++)
        count[(arr[i] / exp) % 10]++;

    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];

    for (int i = n - 1; i >= 0; i--) {
        output[count[(arr[i] / exp) % 10] - 1] = arr[i];
        count[(arr[i] / exp) % 10]--;
    }

    for (int i = 0; i < n; i++)
        arr[i] = output[i];
}

void radixSort(std::vector<int>& arr) {
    int maxElement = getMax(arr);

    for (int exp = 1; maxElement / exp > 0; exp *= 10)
        countingSort(arr, exp);
}
```

Рисунок 1.4 – Radix Sort мовою програмування C++

Нижче приведена таблиця 1.1, яка показує порівняння алгоритмів сортування описаних вище.

Таблиця 1.1

Порівняння характеристик алгоритмів сортування

| Алгоритм | Складність | Стабільність | Потреба в додатковій пам’яті |
|---|---|---|---|
| Bubble Sort | $O(n^2)$ / $O(n^2)$ | Так | Ні |
| Selection Sort | $O(n^2)$ / $O(n^2)$ | Ні | Ні |
| Insertion Sort | $O(n^2)$ / $O(n^2)$ | Так | Ні |
| Quick Sort | $O(n \log n)$ / $O(n^2)$ | Ні | Ні |
| Merge Sort | $O(n \log n)$ / $O(n \log n)$ | Так | Так |
| Counting Sort | $O(n + k)$ / $O(n + k)$ | Так | Так |
| Radix Sort | $O(nk)$ / $O(nk)$ | Так | Так |
| Bucket Sort | $O(n + k)$ / $O(n^2)$ | Так | Так |
| Shell Sort | $O(n \log^2 n)$ / залежить від вибору кроків | Ні | Ні |

Таблиця 1.1 демонструє різноманітність підходів до впорядкування даних, кожен з яких має свої сильні та слабкі сторони. При виборі алгоритму сортування необхідно враховувати кілька ключових факторів.

По-перше, складність алгоритму. Алгоритми з низькою складністю, такі як Merge Sort та Quick Sort, є ефективними для великих наборів даних, хоча Quick Sort може мати найгірший випадок $O(n^2)$. Алгоритми зі складністю $O(n^2)$, як-от Bubble Sort, Selection Sort і Insertion Sort, краще використовувати для невеликих або майже відсортованих наборів даних.

По-друге, стабільність. Стабільні алгоритми зберігають відносний порядок елементів з однаковими значеннями. Це важливо в завданнях, де цей

порядок має значення. Стабільними алгоритмами є Merge Sort, Counting Sort, Bucket Sort, Insertion Sort. Нестабільні алгоритми можуть змінювати відносний порядок таких елементів. Приклади таких алгоритмів: Quick Sort, Selection Sort, Shell Sort.

По-третє, потреба в додатковій пам'яті. Алгоритми, які не потребують значної додаткової пам'яті, як-от Quick Sort і Heap Sort, можуть бути корисними в умовах обмежених ресурсів. Алгоритми, які потребують додаткової пам'яті, як-от Merge Sort і Counting Sort, можуть забезпечити інші переваги, такі як стабільність або певна швидкість за умови наявності необхідної пам'яті.

Таким чином, вибір алгоритму сортування залежить від конкретних вимог і обмежень задачі, таких як розмір даних, необхідність стабільності, обмеження по пам'яті та характеристика розподілу даних. Кожен алгоритм має свої оптимальні випадки використання, що дозволяє досягти максимальної ефективності при правильному підході.

### 1.4 Візуалізація алгоритмів сортування

Сутність візуалізації алгоритмів сортування відіграє важливу роль у навчальному процесі програмування, адже вона ставить перед учнями ґрунтовне розуміння функціонування та поведінки алгоритмів. Вона забезпечує можливість бачити кожний етап процесу сортування, що полегшує усвідомлення його внутрішньої роботи [15]. Таким чином, здобувачі освіти матимуть змогу не тільки спостерігати за результатом дії алгоритму, а й розуміти, як саме він був досягнутий.

Техніки візуалізації алгоритмів сортування передбачають використання анімацій, діаграм, графів та інших візуальних засобів для представлення процесу сортування [16]. Його інтерактивність дозволяє взаємодіяти з алгоритмами, модифікувати їх параметри та слідкувати за впливом цих змін на результат. Використання різних методів візуалізації робить процес навчання цікавим та ефективним. Ці техніки передбачають використання спеціальних інтерактивних інструментів, що дозволяють імітувати роботу алгоритму крок за

кроком і змінювати його параметри на ходу [17]. Завдяки цьому здобувачі освіти можуть зануритися в процес сортування, а прості й зрозумілі візуалізації дозволяють з'ясувати та запам'ятати правила даних алгоритмів швидше та ефективніше. Використання таких інструментів створює можливість для покращення аналітичних та просторових навичок, розвитку творчого мислення та вирішення проблем. І саме тому візуалізація алгоритмів сортування стає важливою складовою якісної та ефективної освіти.

Візуалізація алгоритмів сортування може бути корисним інструментом для здобувачів освіти, оскільки вона дозволяє краще уявити собі процес сортування. Завдяки візуалізації, учні зможуть побачити всі кроки, необхідні для досягнення бажаного результату [18]. Це дає їм можливість не тільки сприймати кінцевий результат, але і розуміти, як саме цей результат був досягнутий.

З використанням візуалізації алгоритмів сортування, здобувачі освіти матимуть можливість краще усвідомити особливості роботи різних алгоритмів [19]. Вони зможуть побачити, як саме кожен алгоритм переставляє елементи для досягнення правильного порядку сортування. Це дозволить здобувачам освіти отримати глибше розуміння принципу роботи алгоритмів та їх взаємодію.

Візуалізація алгоритмів сортування також може сприяти кращому запам'ятовуванню та усвідомленню різних кроків виконання алгоритмів. Коли здобувачі освіти можуть бачити кожний крок на екрані, це дає їм можливість зосередитися на ньому і запам'ятати його [20]. Таким чином, візуалізація сприяє покращенню пам'яті та засвоєнню інформації про алгоритми сортування.

Узагалі, візуалізація алгоритмів сортування є надзвичайно корисним інструментом для здобувачів освіти, які вивчають програмування. Вона допомагає розширити їх розуміння алгоритмів та полегшує навчання процесу сортування. Завдяки візуалізації, здобувачі освіти зможуть глибше зануритися в світ алгоритмів та краще сприймати їх внутрішню роботу.

## 1.5 Особливості використання бібліотеки SFML для візуалізації алгоритмів сортування

Актуальність використання бібліотеки SFML для візуалізації алгоритмів зумовлена сучасними потребами в ефективному та наочному представленні складних процесів та алгоритмів. Візуалізація допомагає краще розуміти роботу алгоритмів, виявляти їх особливості та покращувати процес навчання для здобувачів освіти. В умовах постійного зростання складності алгоритмів та необхідності їх інтерактивного представлення, вибір інструментів для візуалізації є критично важливим. Застосування таких інструментів дозволяє наочно демонструвати алгоритмічні концепції, покращувати розуміння та сприяти ефективному засвоєнню матеріалу.

SFML (Simple and Fast Multimedia Library) є кросплатформеною бібліотекою, призначеною для роботи з графікою, звуком, введенням та мережею [20]. Вона забезпечує простий і зрозумілий інтерфейс для розробки мультимедійних застосунків та ігор. Основні особливості використання SFML для візуалізації алгоритмів включають її простоту, гнучкість та кросплатформеність.

Інтуїтивно зрозумілий API спрощує процес візуалізації навіть для новачків. Бібліотека надає зручні інструменти для створення графічних об'єктів, роботи з текстурами, спрайтами та текстовими об'єктами [18]. Простота дозволяє швидко почати роботу з бібліотекою та створювати візуалізації без потреби в глибоких знаннях графічного програмування.

SFML підтримує роботу з двовимірною графікою, що робить її ідеальною для демонстрації алгоритмів. Вона також дозволяє легко інтегрувати анімацію та динамічні зміни, що є важливими для створення візуалізацій алгоритмів. За допомогою SFML можна створювати як прості анімації обміну елементів в алгоритмах сортування, так і складніші візуалізації пошуку шляхів на графах або сітках [20].

Можливість запуску програм на різних операційних системах, таких як Windows, macOS та Linux, розширює використання SFML [18]. Це дозволяє

розробникам створювати візуалізації, які будуть працювати на будь-якій платформі без значних змін у коді. Така кросплатформеність робить SFML зручним інструментом для освітніх цілей та розповсюдження програмного забезпечення.

Процес створення візуалізації з SFML включає кілька етапів. Спочатку створюється вікно програми та налаштовуються основні параметри. Потім реалізується логіка алгоритму та готуються дані для візуалізації. Використовуючи об'єкти SFML, відбувається малювання графічних елементів та анімація. Екран постійно оновлюється для відображення змін у реальному часі [20].

На початковому етапі необхідно створити вікно програми та налаштувати його основні параметри, такі як розмір, заголовок та режим відображення. Це робиться за допомогою класу sf::RenderWindow, який відповідає за відображення вікна та обробку подій.

Наступним кроком є реалізація логіки алгоритму, який буде візуалізований. Це може включати обчислення станів, зміни позицій елементів та інші обчислювальні операції. Логіка алгоритму повинна бути інтегрована з графічними об'єктами SFML для забезпечення динамічного відображення процесу.

Після реалізації логіки алгоритму, необхідно створити графічні об'єкти, які будуть відображати стан алгоритму. Використовуючи об'єкти SFML, такі як sf::CircleShape, sf::RectangleShape та sf::Text, можна створювати різноманітні візуальні представлення елементів алгоритму [20].

Для забезпечення анімації та динамічного відображення змін, екран повинен постійно оновлюватися. Це досягається за допомогою циклу рендерингу, який очищає екран, малює всі графічні елементи та відображає їх у вікні програми. Такий підхід дозволяє створювати реалістичні анімації та наочно демонструвати роботу алгоритму.

Переваги SFML можна побачити, порівнюючи його з іншими бібліотеками. Наприклад, SFML має кращий API та кращі графічні можливості, ніж SDL.

SFML надає більш високий рівень абстракції, що значно спрощує процес розробки графічних додатків та візуалізацій. У порівнянні з OpenGL, SFML пропонує вищий рівень абстракції, що дозволяє зосередитися на логіці та візуалізації алгоритму, а не на низькорівневому програмуванні графіки. SFML є ідеальним вибором для тих, хто бажає швидко та ефективно створювати візуалізації без необхідності глибокого вивчення графічного програмування. SFML є більш продуктивним для графічних додатків і має менш навантажений API ніж Qt. Хоча Qt надає багато можливостей для розробки різноманітних додатків, SFML краще підходить для створення графічних візуалізацій завдяки своїй легкості та спеціалізації на мультимедійних задачах [9].

Використання SFML для візуалізації алгоритмів є актуальним та ефективним підходом завдяки її простоті, гнучкості та можливості роботи на різних платформах. SFML надає потужний набір інструментів для створення як базових, так і складних візуалізацій, що дозволяє більш глибоко зрозуміти алгоритми та їх роботу. Враховуючи переваги SFML, його активну спільноту та багатоплатформеність, дана бібліотека є відмінним вибором для тих, хто бажає ефективно візуалізувати алгоритми та покращувати процес навчання.

### 1.6 Огляд та аналіз аналогічних програмних розробок

Для успішного проєктування та розробки програми візуалізації алгоритмів важливо провести огляд і аналіз аналогічних програмних розробок. Цей етап дозволяє ідентифікувати сильні та слабкі сторони аналогічних програмних розробок і визначити можливості для вдосконалення власного продукту.

VisualGO.net – це платформа для візуалізації алгоритмів, яка пропонує широкий спектр функцій для створення анімацій та інтерактивних демонстрацій алгоритмів сортування. На рисунку 1.5 зображено зовнішній вигляд веб платформи VisualGO.

Основні характеристики:

- підтримка різних алгоритмів сортування;

- інтерактивні демонстрації;
- візуалізація даних;

Переваги:

- зручний інтерфейс;
- підтримка різних мов;
- широкий спектр алгоритмів сортування.

Недоліки:

- немає можливості створення візуалізації для великих наборів даних.


Рисунок 1.5 – Зовнішній вигляд VisualGO

ComparisonSort – це онлайн-платформа для візуалізації алгоритмів сортування, розроблена професором Еріком Галлесом з Університету Сан-Франциско. Вона пропонує інтерактивний інтерфейс для візуалізації та порівняння різних алгоритмів сортування, таких як бульбашкове сортування, сортування вибором, вставне сортування, швидке сортування та сортування злиттям. На рисунку 1.6 зображено зовнішній вигляд програми.

Основні переваги:

- інтерактивність;
- порівняння алгоритмів;
- наявність навчальних матеріалів.

Недоліки:

- обмежена кількість алгоритмів;
- немає можливості створювати власні алгоритми;
- повільний для великих наборів даних.

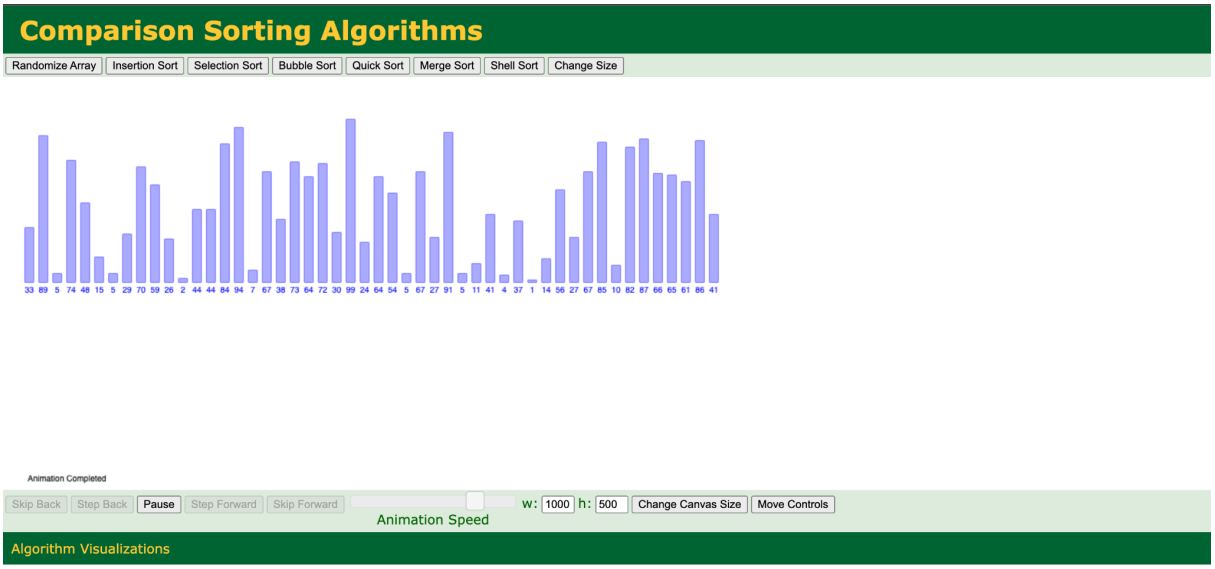


Рисунок 1.6 – Зовнішній вигляд ComparisonSort

AlgoVis – це онлайн-платформа для візуалізації алгоритмів, яка пропонує широкий спектр візуалізацій для різних алгоритмів сортування, включаючи бульбашкове сортування, сортування вибором, вставне сортування, швидке сортування, сортування злиттям та багато інших. AlgoVis також пропонує візуалізації для інших типів алгоритмів, таких як алгоритми пошуку та алгоритми графів. На рисунку 1.7 зображено зовнішній вигляд програми.

Переваги:

- широкий спектр візуалізацій для різних алгоритмів;
- інтерактивний інтерфейс;
- підтримка мобільних пристроїв.

Недоліки:

- деякі візуалізації потребують реєстрації.

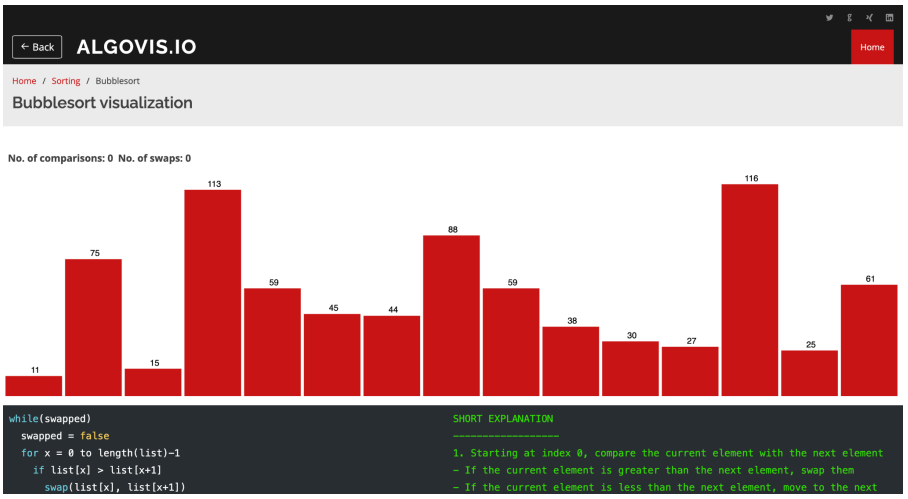


Рисунок 1.7 – Зовнішній вигляд AlgoVis

GeeksforGeeks – це веб-сайт, який пропонує навчальні матеріали з комп'ютерних наук, включаючи алгоритми сортування. GeeksforGeeks пропонує візуалізації для різних алгоритмів сортування, а також код реалізації алгоритмів на різних мовах програмування. На рисунку 1.8 поданий зовнішній вигляд GeeksforGeeks.

Переваги:

- детальні навчальні матеріали;
- візуалізації та код реалізації для багатьох алгоритмів;
- підтримка для різних мов програмування.

Недоліки:

- візуалізації не такі інтерактивні, як на інших платформах;
- не зрозумілий інтерфейс.

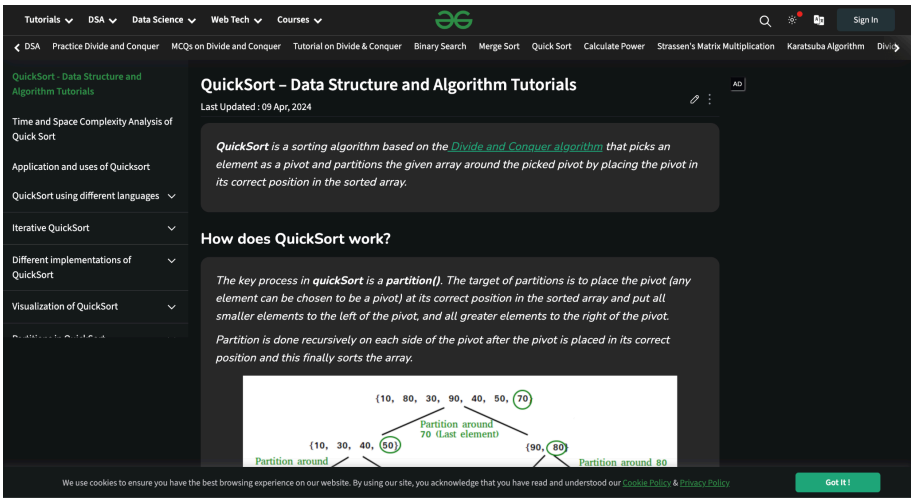


Рисунок 1.8 – Зовнішній вигляд GeeksforGeeks

Переглянувши аналогічні розробки можна сказати, що програма повинна мати наступне:

- широкий спектр алгоритмів сортування, від базових до більш складних, щоб задовольнити потреби користувачів з різним рівнем досвіду;
- інтерактивні та динамічні візуалізації, які дозволять користувачам чітко бачити, як працюють алгоритми сортування;
- щоб програма була максимально простою у використанні і користувачі будь-якого рівня підготовки могли легко з нею працювати;
- потрібно щоб програма була гнучкою та адаптованою до різних потреб користувачів. Це може включати можливість налаштування візуалізацій, вибору алгоритмів сортування.

# РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМИ ДЛЯ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ СОРТУВАННЯ

## 2.1 Постановка задачі, призначення та вимоги до розробки

Основною задачею даної роботи є розробка програми для візуалізації алгоритмів сортування, яка допоможе користувачам зрозуміти роботу різних алгоритмів сортування через їх графічне представлення. Програма має забезпечувати наочне та інтерактивне відображення кожного етапу роботи алгоритмів, що дозволить користувачам детально вивчити принципи їх функціонування.

Програма для візуалізації алгоритмів сортування призначена для використання здобувачами освіти та викладачами. Вона буде корисним інструментом у навчальному процесі, допомагаючи покращити розуміння теоретичних аспектів алгоритмів сортування. Програма також забезпечить візуальне представлення процесу сортування, що сприятиме кращому засвоєнню матеріалу, дозволить користувачам експериментувати з різними алгоритмами та параметрами для глибшого розуміння їхньої роботи.

Основними вимогами до розробки є:

- програма повинна підтримувати візуалізацію основних алгоритмів сортування, таких як бульбашкове сортування, сортування вибором, сортування вставками, швидке сортування, сортування злиттям;
- користувачі повинні мати можливість керувати процесом візуалізації, зупиняти та відновлювати анімацію, змінювати швидкість виконання алгоритмів, а також модифікувати параметри алгоритмів (наприклад, розмір масиву);
- програма має підтримувати режим покрокового виконання алгоритмів, що дозволить користувачам аналізувати кожен окремий крок сортування;
- інтуїтивно зрозумілий і зручний графічний інтерфейс, який забезпечить легкий доступ до всіх функцій програми;

- програма повинна бути достатньо продуктивною для плавної візуалізації навіть при великих наборах даних;
- програма має бути розроблена з урахуванням можливості її запуску на різних операційних системах (Windows, macOS, Linux).

Для забезпечення виконання поставлених задач, програма повинна включати в себе наступні функції:

- підтримка різних алгоритмів сортування;
- можливість вибору алгоритму сортування для візуалізації;
- показ кожного кроку роботи алгоритму у реальному часі;
- опції для запуску, зупинки, відновлення та перезапуску процесу візуалізації;
- можливість уповільнення або прискорення візуалізації для детального розгляду або швидкого перегляду;
- налаштування параметрів алгоритму, таких як розмір масиву або порядок елементів перед початком сортування;
- зручний та інтуїтивно зрозумілий інтерфейс для взаємодії з програмою;
- підтримка роботи на різних операційних системах, таких як Windows, macOS та Linux;
- забезпечення високої продуктивності та оптимальної швидкості виконання алгоритмів, незалежно від платформи.

### 2.2 Опис проєкту

На діаграмі (рис. 2.1) можемо спостерігати, що користувач програми може вибрати алгоритм сортування, візуалізацію якого він хоче побачити. Налаштувати параметри перед запуском візуалізації, наприклад кількість елементів у масиві. Також має доступ до управління елементами керування, такі як кнопки запуску, відновлення та зупинки анімації.

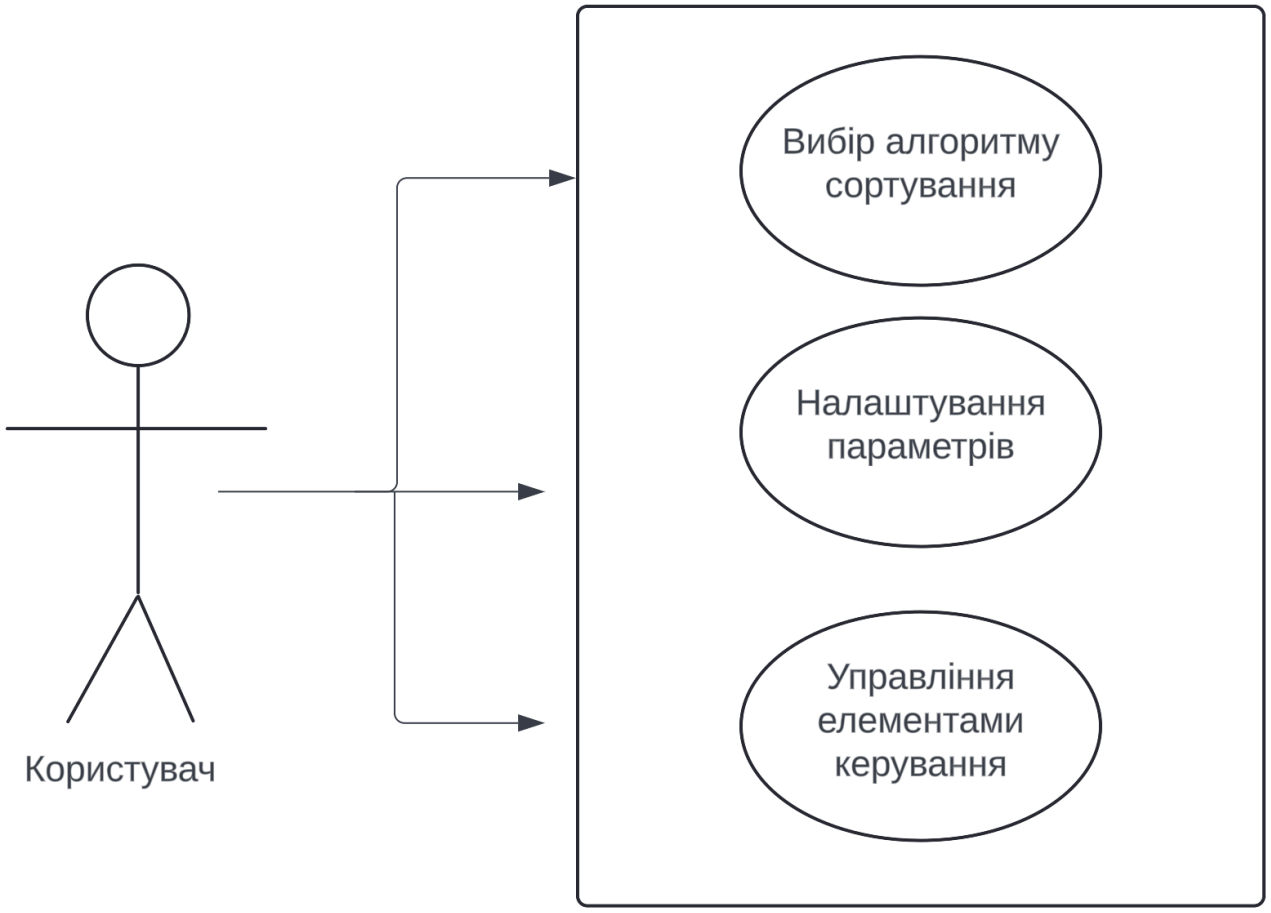


Рисунок 2.1 – Діаграма використання програми

Для програми було розроблено макет попереднього вигляду, який зображений на рисунку 2.2. Головне вікно є центральним елементом інтерфейсу програми, яке містить усі основні елементи керування та області візуалізації. Воно поділяється на кілька функціональних зон: вибір алгоритмів сортування, область візуалізації, панель налаштування.

У зоні вибору алгоритму буду кнопки для вибору одного з підтримуваних алгоритмів сортування (наприклад, бульбашкове сортування, сортування вибором, швидке сортування тощо).

Зона область візуалізації – це основна область, де відображається графічне представлення процесу сортування. Візуалізація здійснюється за допомогою стовпчиків, що змінюють своє положення та колір відповідно до алгоритму.

Панель налаштувань дозволяє користувачам задавати параметри алгоритмів перед їх запуском. Основні елементи панелі налаштувань включають:

- вибір алгоритму: кнопки для вибору одного з підтримуваних алгоритмів сортування;
- параметри алгоритму: опції для встановлення початкових значень елементів масиву, такі як кількість елементів;
- додаткові опції: можливість вмикати або вимикати покрокову анімацію, ставити на паузу, відновлювати візуалізацію алгоритму.

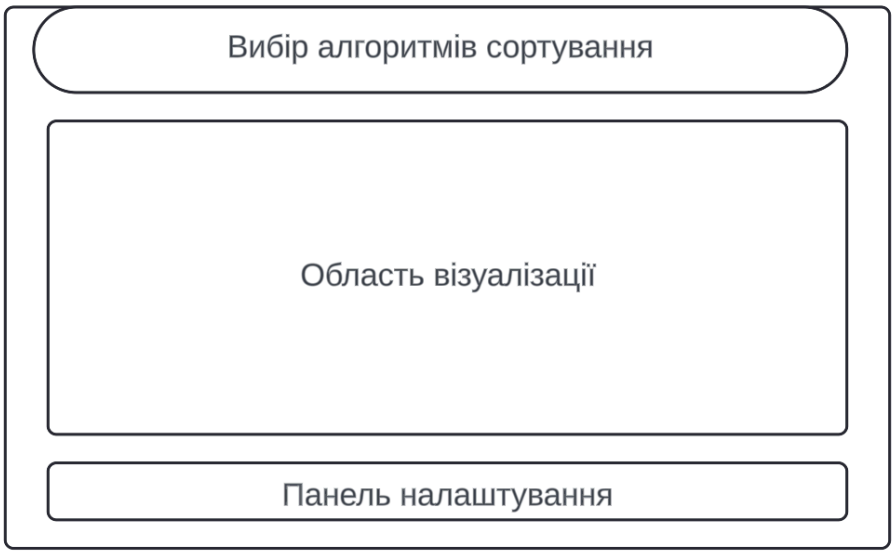


Рисунок 2.2 – Макет програми

### 2.3 Вибір моделі розробки

Вибір моделі розробки програмного забезпечення є важливим етапом, оскільки від нього залежить ефективність процесу створення програми, її якість та відповідність вимогам користувачів. При виборі моделі розробки для даного проєкту враховуються наступні фактори:

- модель повинна дозволяти швидко реагувати на зміни в вимогах та середовищі розробки;

- проєкт повинен мати можливість поступового розширення та покращення функціональності;
- процес розробки повинен бути легко керованим та зрозумілим;
- модель повинна включати етапи тестування та перевірки для забезпечення високої якості кінцевого продукту.

Для даного проєкту було обрано ітеративну модель розробки. Цей вибір обумовлений кількома важливими причинами. По-перше, ітеративна модель забезпечує гнучкість, дозволяючи вносити зміни та вдосконалення на будь-якому етапі розробки, що є критично важливим для проєктів з можливими змінами у вимогах. По-друге, модель підтримує поступове вдосконалення програмного забезпечення через серію ітерацій, кожна з яких додає нову функціональність та вдосконалює попередні версії. По-третє, ітеративна модель дозволяє зменшити ризики, оскільки кожна ітерація включає етапи аналізу, розробки та тестування, що дозволяє виявляти та виправляти помилки на ранніх стадіях.

Процес розробки за ітеративною моделлю буде організовано через кілька основних фаз. Спочатку проводиться ініціалізація проєкту, яка включає аналіз початкових вимог, визначення цілей проєкту та створення загального плану, а також оцінку ресурсів та часу, необхідних для реалізації. Потім відбувається цикл ітерацій, кожна з яких складається з етапів планування, розробки, тестування, оцінки результатів. Планування включає визначення цілей та завдань для поточної ітерації. Розробка полягає у написанні коду та реалізації функціональності, визначеної на етапі планування. Тестування включає перевірку реалізованої функціональності, виявлення та виправлення помилок. Оцінка результатів передбачає аналіз досягнутих результатів, порівняння з початковими вимогами та цільовими показниками.

Після завершення всіх ітерацій проводиться фіналізація проєкту, яка включає інтеграцію всіх компонентів, остаточне тестування, виправлення залишкових помилок, підготовку до впровадження, створення документації.

Вибір ітеративної моделі розробки забезпечує гнучкий підхід до створення програми для візуалізації алгоритмів сортування, дозволяючи поступово вдосконалювати функціональність, швидко реагувати на зміни вимог та забезпечувати високу якість кінцевого продукту.

### 2.4 Обґрунтування вибору інструментальних засобів розробки

Для успішної розробки програми для візуалізації алгоритмів сортування необхідно вибрати оптимальні інструментальні засоби, які забезпечують ефективність розробки, зручність роботи та високу якість кінцевого продукту. Вибір технологій базується на вимогах до продуктивності, гнучкості, сумісності та зручності у використанні.

Visual Studio є потужним інтегрованим середовищем розробки (IDE), яке забезпечує розробників усіма необхідними інструментами для створення високоякісного програмного забезпечення. Це середовище підтримує різні мови програмування, включаючи C++, та надає широкі можливості для налагодження, тестування та управління проєктами. Visual Studio також має зручний інтерфейс та численні плагіни, що дозволяють розширювати його функціональність відповідно до потреб проєкту. Завдяки своїй потужності та гнучкості, Visual Studio є відмінним вибором для розробки складних програмних продуктів.

C++ є високопродуктивною мовою програмування, яка широко використовується для розробки програмного забезпечення, де потрібна висока продуктивність та ефективне управління ресурсами. Мова програмування C++ забезпечує низькорівневий доступ до апаратного забезпечення, що дозволяє створювати ефективні та швидкі програми. C++ також підтримує об'єктно-орієнтоване програмування, що сприяє структурованому підходу до розробки великих проєктів. Використання C++ для розробки програми для візуалізації алгоритмів сортування дозволить досягти високої продуктивності та ефективності.

SFML (Simple and Fast Multimedia Library) є бібліотекою для роботи з мультимедіа, яка забезпечує простий та зручний інтерфейс для створення графічних додатків. Основними перевагами SFML є її легкість у використанні, висока продуктивність та підтримка різних платформ. SFML надає розробникам можливість створювати графічні інтерфейси, працювати зі звуком, обробляти події та взаємодіяти з іншими мультимедійними елементами.

SFML є відмінним вибором для розробки програми для візуалізації алгоритмів сортування з кількох причин. По-перше, її простий інтерфейс дозволяє швидко та легко розробляти графічні елементи, що є важливим для створення візуалізації. По-друге, SFML забезпечує високу продуктивність, що дозволяє ефективно обробляти великі обсяги даних та швидко відображати результати. По-третє, бібліотека підтримує кросплатформеність, що дозволяє розробляти програми, які можуть працювати на різних операційних системах без значних змін у коді.

CMake є інструментом для автоматизації збірки програмного забезпечення, який дозволяє створювати платформи незалежні від конфігураційних файлів. CMake забезпечує зручне управління проєктами, дозволяє легко налаштовувати параметри збірки та інтегрується з різними середовищами розробки. Використання CMake у проєкті забезпечує ефективне управління процесом збірки та зменшить час, необхідний для налаштування середовища розробки на різних платформах.

Вибір інструментальних засобів, таких як Visual Studio, C++, SFML та CMake, є оптимальним для розробки програми для візуалізації алгоритмів сортування. Visual Studio забезпечує потужні можливості для розробки та налагодження коду, C++ надає високу продуктивність та ефективне управління ресурсами, SFML дозволяє легко створювати високопродуктивні графічні додатки, а CMake спрощує процес збірки та управління проєктом. Використання цих технологій у поєднанні забезпечує успішну реалізацію проєкту та високу якість кінцевого продукту.

## 2.5 Особливості програмної реалізації

Програмне забезпечення для візуалізації алгоритмів сортування реалізоване з використанням бібліотеки SFML (Simple and Fast Multimedia Library) для графічного відображення процесу сортування. Основні особливості реалізації включають мультипоточність, використання різних алгоритмів сортування, гнучке керування візуалізацією, та взаємодію з користувачем через GUI.

SFML надає засоби для створення вікон, рендерингу графіки та обробки подій, що робить її ідеальною для розробки графічного інтерфейсу програми.

```
// Ініціалізація вікна
sf::RenderWindow window(sf::VideoMode(1400, 1000), "Sorting Visualization");
```

Рис 2.3 – Ініціалізація вікна

На рисунку 2.3 зображено рядок коду, який створює основне вікно програми розміром 1400x1000 пікселів. Наступні рядки коду (рис. 2.4) використовуються для завантаження шрифту та створення кнопок, які допомагають користувачу обирати різні алгоритми сортування та керувати процесом сортування.

```
// Завантаження шрифтів
sf::Font font;
if (!font.loadFromFile("AbrilFatface-Regular.ttf"))
{
    std::cout << "Font not loaded" << std::endl;
    return -1;
}

// Створення кнопок
sf::Text bubbleSortButton("Bubble Sort", font, 20);
bubbleSortButton.setPosition(10, 10);
```

Рис 2.4 – Створення кнопок і завантаження шрифтів

Для забезпечення плавної роботи інтерфейсу під час виконання алгоритмів сортування використовуються окремі потоки. Це дозволяє програмі залишатися чутливою до дій користувача, таких як зупинка або зміна алгоритму сортування, навіть під час виконання сортування. Використання std::thread

дозволяє здійснювати сортування в окремому потоці, що забезпечує безперебійну роботу графічного інтерфейсу навіть під час виконання інтенсивних обчислень. В даному коді (рис. 2.5) створюється потік sortThread, який виконує вибраний алгоритм сортування. Стан сортування зберігається в атомарній змінній state, що дозволяє уникнути проблем з доступом до ресурсів.

```
std::thread sortThread([&]() {
    while(state != State::EXIT_PROGRAM) {
        if (sortAlgo && state.load() == State::RUN) {

            try {
                sortAlgo(arr);
            } catch (...) {}

            state = State::EXIT;
        }
    }
});
```

Рис 2.5 – Створення потоку

Функція checkState (рис. 2.6) використовується для перевірки стану програми під час виконання сортування. Це дозволяє програмі реагувати на зміну стану (наприклад, зупинку чи паузу) навіть під час виконання довготривалих обчислень.

```
bool checkState()
{
    auto tmp = state.load();

    if (tmp == State::EXIT || tmp == State::EXIT_PROGRAM) {
        throw "Abort";
    }

    bool result = tmp == State::PAUSE;

    std::this_thread::sleep_for(ms);

    tmp = state.load();

    if (tmp == State::EXIT || tmp == State::EXIT_PROGRAM) {
        throw "Abort";
    }

    return result;
}
```

Рис 2.6 – Функція checkState

Функція setSortAlgo (рис. 2.7) дозволяє динамічно змінювати алгоритм сортування під час роботи програми, що надає користувачеві можливість випробувати різні алгоритми без перезапуску програми.

```
void setSortAlgo(void (*function)(std::vector<int>&))
{
    clearSortAlgo();
    sortAlgo = function;
}
```

Рис 2.7 – Функція setSortAlgo

Програма має різні стани, такі як RUN, PAUSE, EXIT, та EXIT_PROGRAM (рис. 2.8), які дозволяють керувати процесом сортування та станом візуалізації. Атомарні змінні та м'ютекси забезпечують безпеку доступу до спільних ресурсів у багатопотоковому середовищі.

```
enum class State
{
    RUN,
    PAUSE,
    EXIT,

    EXIT_PROGRAM
};
```

Рис 2.8 – Стани програми

Програма дозволяє користувачу змінювати швидкість візуалізації процесу сортування за допомогою кнопок (рис. 2.9). Це досягається шляхом регулювання часу затримки між кроками сортування. Кожна кнопка змінює змінну ms, яка визначає тривалість затримки між кроками алгоритму сортування. Це дозволяє користувачу бачити процес сортування у різному темпі. Користувач може керувати сортуванням за допомогою кнопок Pause, Start та End, які дозволяють зупиняти, відновлювати та завершувати сортування відповідно.

```
else if (x2Button.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y))
{
    ms = 50ms;
}
else if (x5Button.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y))
{
    ms = 20ms;
}
else if (x10Button.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y))
{
    ms = 10ms;
}
```

Рис 2.9 – Зміна швидкості візуалізації

Користувач може вибирати різні алгоритми сортування, натискаючи на відповідні кнопки в інтерфейсі. Цей фрагмент коду (рис. 2.10) перевіряє, чи було натиснуто кнопку Bubble Sort, і якщо так, то встановлює алгоритм bubbleSort як поточний, змінюючи колір кнопки, щоб відобразити вибір користувача.

```
if (bubbleSortButtonBg.getGlobalBounds().contains(event.mouseButton.x, event.mouseButton.y))
{
    setSortAlgo(bubbleSort);
    bubbleSortButtonBg.setFillColor(sf::Color::Green);
    // інші кнопки повертаються до початкового кольору
    insertionSortButtonBg.setFillColor(sf::Color::Blue);
    selectionSortButtonBg.setFillColor(sf::Color::Blue);
    mergeSortButtonBg.setFillColor(sf::Color::Blue);
    quickSortButtonBg.setFillColor(sf::Color::Blue);
}
```

Рис 2.10 – Кнопка для вибору алгоритму сортування Bubble Sort

Процес сортування візуалізується у вигляді вертикальних барів, довжина яких відповідає значенню елемента масиву. Кольори барів змінюються, щоб показати порівнювані або обмінювані елементи, що робить процес сортування зрозумілим для користувача. Функція visualizeArray (рис. 2.11) відповідає за відображення поточного стану масиву. Вона використовує SFML для малювання барів, змінюючи їх колір в залежності від їх стану у поточному кроці алгоритму сортування. Візуалізація оновлюється в реальному часі, відображаючи процес сортування поетапно.

```
void visualizeArray(sf::RenderWindow& window, const std::vector<int>& arr)
{
    sf::RectangleShape bar;
    float width = window.getSize().x / static_cast<float>(arr.size());

    for (size_t i = 0; i < arr.size(); ++i)
    {
        if (i == index1 || i == index2)
            bar.setFillColor(sf::Color::Green);
        else
            bar.setFillColor(sf::Color::White);

        bar.setSize(sf::Vector2f(width - 1, arr[i]));
        bar.setPosition(i * width, window.getSize().y - arr[i] - 200);
        window.draw(bar);
    }
}
```

Рис 2.11 – Функція visualizeArray

Події, такі як натискання кнопок миші, обробляються через цикл подій SFML (рис. 2.12). Це дозволяє програмі реагувати на дії користувача в режимі реального часу. Програма також враховує завершення роботи користувача та коректно закриває всі ресурси та потоки. Алгоритм сортування вибирається динамічно під час роботи програми, що реалізовано за допомогою функцій-вказівників.

```
while (window.isOpen())
{
    sf::Event event;
    while (window.pollEvent(event))
    {
        if (event.type == sf::Event::Closed)
        {
            window.close();
        }

        if (event.type == sf::Event::MouseButtonPressed)
        {
            if (bubbleSortButtonBg.getGlobalBounds().contains(event.mouseB
            {
                setSortAlgo(bubbleSort);
                bubbleSortButtonBg.setFillColor(sf::Color::Green);
                // інші кнопки повертаються до початкового кольору
                insertionSortButtonBg.setFillColor(sf::Color::Blue);
                selectionSortButtonBg.setFillColor(sf::Color::Blue);
                mergeSortButtonBg.setFillColor(sf::Color::Blue);
                quickSortButtonBg.setFillColor(sf::Color::Blue);
            }
            else if (insertionSortButtonBg.getGlobalBounds().contains(even
            {
                setSortAlgo(insertionSort);
```

Рис 2.12 – Цикл подій

Графічний інтерфейс забезпечує просте та зрозуміле управління. Користувач може вибирати алгоритм сортування, змінювати розмір масиву, додавати та видаляти елементи з масиву. Кнопки та елементи керування розташовані в зручних місцях, що робить програму інтуїтивно зрозумілою навіть для недосвідчених користувачів. Як виглядає інтерфейс зображено на рисунку 2.13.

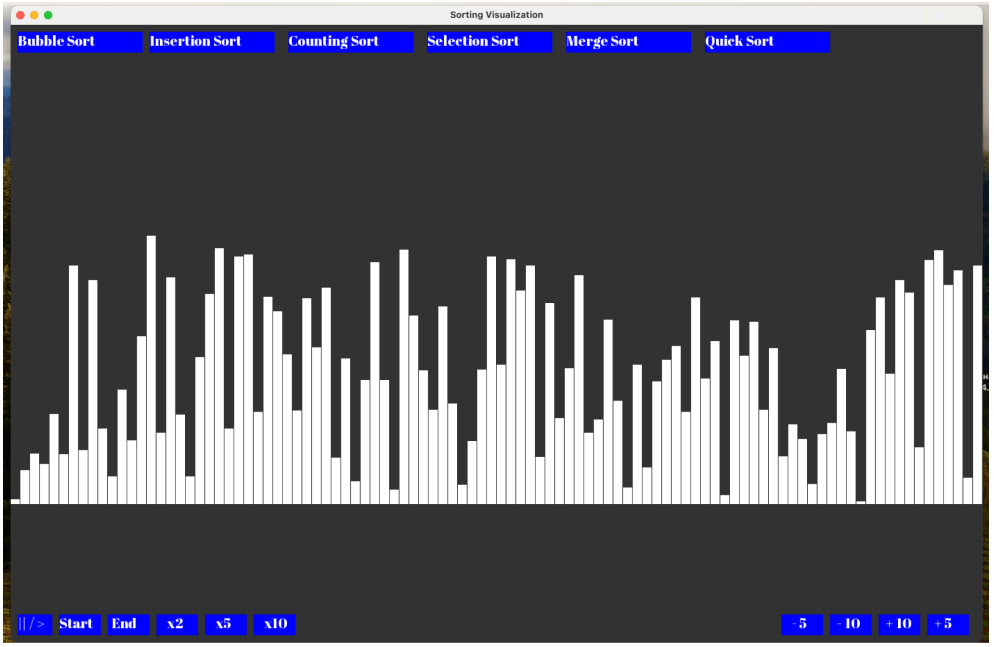


Рис 2.13 – Інтерфейс програми

На рисунку 2.14 зображено програму у роботі.

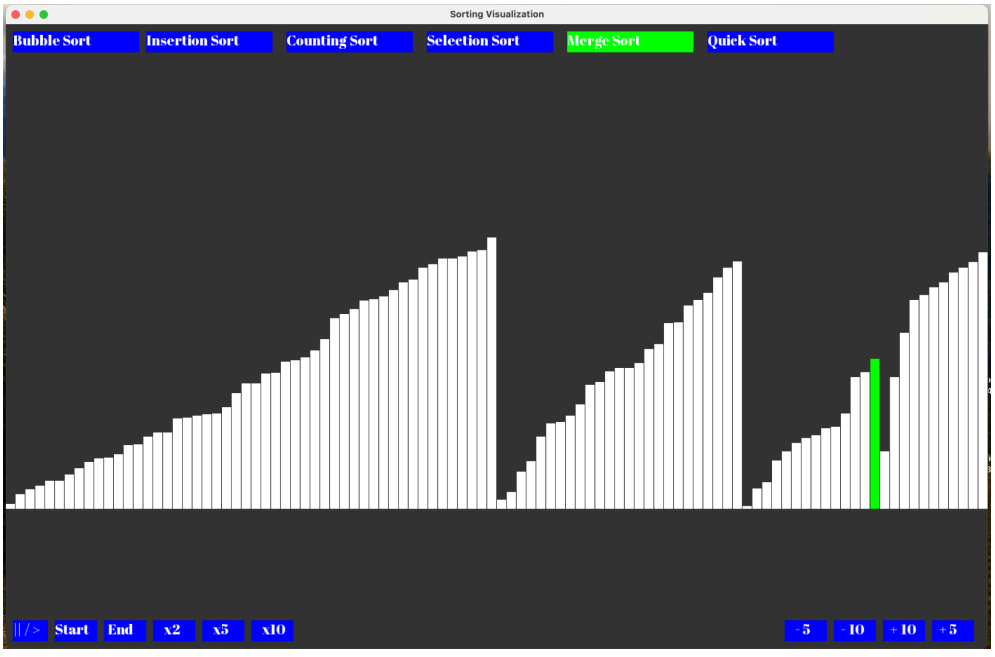


Рис 2.14 – Робота програми

### 2.6 Тестування та налагодження програмної розробки

Для забезпечення якості та надійності розробленого програмного засобу, було проведено кілька видів тестування.

Для кожного алгоритму сортування було створено набір тестових випадків, що включають сортування різних масивів: пусті масиви, масиви з одним елементом, вже відсортовані масиви, масиви з повторюваними елементами та масиви з великим числом елементів. У алгоритмі Quick Sort була знайдена проблема з рекурсивним розділенням масиву, яка приводила до зависання програми при обробці масивів з великим числом однакових елементів. Проблема була вирішена шляхом впровадження тричастинного розділення масиву, що дозволило ефективніше обробляти масиви з великою кількістю однакових значень.

Була виявлена проблема з некоректною передачею параметрів розміру масиву, що призводило до відображення неправильних значень у візуалізації. Була виправлена логіка ініціалізації масиву та його передача до модуля візуалізації, що усунуло проблему.

Програма була перевірена на трьох операційних системах. Тестування включало запуск програми з однаковими даними та параметрами на кожній платформі для перевірки коректності роботи та візуалізації. Програма була протестована з великими наборами даних (до 10,000 елементів) для оцінки її продуктивності та плавності анімації. На macOS було виявлено проблему з відображенням шрифтів у GUI, що ускладнювало читання тексту в інтерфейсі. Було оновлено систему рендерингу шрифтів та інтегровано альтернативний шрифт, який коректно відображається на всіх платформах.

Було створено список функцій програми, включаючи вибір алгоритмів, керування візуалізацією, налаштування параметрів, покрокове виконання. Кожна функція була протестована вручну, щоб переконатися в її правильній роботі. Виявлено, що кнопка "Pause" не завжди коректно зупиняє анімацію в середині кроку сортування. Логіка обробки станів анімації була вдосконалена для забезпечення точної зупинки і відновлення процесу.

## 2.7 Рекомендації по впровадженню та використанню програмної розробки

Впровадження та використання програмної розробки для візуалізації алгоритмів сортування потребує врахування кількох важливих аспектів, які допоможуть максимально ефективно інтегрувати цю програму в навчальний або дослідницький процес.

Перед початком використання програми необхідно встановити всі необхідні компоненти та забезпечити правильне налаштування середовища розробки. Переконатися, що встановлена бібліотека SFML. Слід завантажити бібліотеку з офіційного сайту SFML та інтегрувати її у проект, налаштувавши компілятор для правильної лінковки з SFML. Зазвичай, це включає вказівку на шляхи до заголовочних файлів та бібліотек, а також правильне налаштування параметрів компіляції.

Після завершення налаштування середовища, потрібно конфігурувати початкові параметри програми. Це включає визначення початкового розміру масиву для сортування, вибір алгоритму сортування за замовчуванням та встановлення початкової швидкості візуалізації. Вихідний код надає достатньо гнучкості для модифікацій, тому можна легко адаптувати ці параметри під свої потреби, змінюючи відповідні значення у вихідному коді або додаючи нові параметри через інтерфейс.

Для запуску програми необхідно переконатися, що всі необхідні файли (шрифти, графічні ресурси) знаходяться у вказаних директоріях. У разі виникнення проблем необхідно перевірити налаштування компіляції та переконатися, що всі необхідні бібліотеки правильно підключені.

Інтерфейс програми надає можливість легко взаємодіяти з нею. Основні кнопки для вибору алгоритму сортування та управління швидкістю розташовані у верхній частині вікна. Натискання на кнопки змінює стан програми, дозволяючи миттєво змінювати поточний алгоритм сортування або регулювати швидкість візуалізації. Користувач може зупиняти або відновлювати процес сортування, що дозволяє детально аналізувати кожен етап алгоритму. Такий

підхід особливо корисний у навчальних середовищах, де важливо продемонструвати різні кроки та порівняти їх поведінку.

Програма особливо корисна для навчальних цілей, оскільки дозволяє здобувачам освіти та викладачам спостерігати за роботою різних алгоритмів сортування у реальному часі. Рекомендується використовувати цю програму як інструмент візуалізації під час лекцій або лабораторних робіт з алгоритмів та структур даних. Здобувачі освіти можуть змінювати параметри програми та аналізувати, як ці зміни впливають на роботу алгоритмів. Викладачі можуть використовувати програму для демонстрації теоретичних аспектів алгоритмів сортування, показуючи, як саме ці алгоритми працюють на практиці.

Загалом, для успішного впровадження та використання даної програмної розробки важливо забезпечити ретельну підготовку середовища, активне використання можливостей інтерфейсу, а також забезпечити підтримку та можливості для подальшого розширення функціональності.

# ВИСНОВКИ

Метою бакалаврської роботи стало створення програми для візуалізації алгоритмів сортування з використанням мови програмування C++ та бібліотеки SFML. Для досягнення мети було проведено аналітичний огляд, під час якого визначено завдання, що вирішуватиме готовий продукт, проведено аналіз наявних рішень, а також визначено цільову аудиторію.

Під час проєктування було розроблено архітектуру програми, створено прототип інтерфейсу користувача, а також обрано засоби розробки. Було спроєктовано взаємодію між компонентами програми та визначено методи візуалізації алгоритмів сортування.

Розроблення програмного продукту проходило у кілька етапів. На першому етапі було реалізовано алгоритми сортування та їх візуалізацію. На другому етапі було створено інтерфейс користувача, що дозволяє вибирати алгоритми та контролювати швидкість візуалізації. Тестування програмного продукту включало модульне, інтеграційне та системне тестування, а також перевірку юзабіліті. Тестування програмного продукту було зроблено вручну.

Розроблений програмний продукт успішно виконує поставлені завдання, забезпечуючи наочне та інтерактивне навчання алгоритмам сортування. Програма поєднує в собі функціональність, зручність використання та ефективність, що робить її корисною для здобувачів освіти.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритми сортування. *Велика українська енциклопедія.* URL: https://vue.gov.ua/Алгоритм_сортування (дата звернення: 23.05.2024).

2. Алгоритми сортування: їхня складність і вибір підходящого. *FoxmindEd*. URL: https://foxminded.ua/alhorytmy-sortuvannia/ (дата звернення: 23.05.2024).

3. Алгоритм сортування. *Wikipedia*. URL: https://uk.wikipedia.org/wiki/Алгоритм_сортування (дата звернення: 23.05.2024).

4. Аспекти вивчення розділу «Алгоритми та програми» під час дистанційного навчання. URL: http://eprints.zu.edu.ua/38654/1/яна_compressed.pdf (дата звернення: 23.05.2024).

5. Розробка та оптимізація інтерфейсів баз даних: методи, алгоритми та їх застосування. URL: http://eprints.zu.edu.ua/38654/1/яна_compressed.pdf (дата звернення: 23.05.2024).

6. Система керування проєктами на основі засобів штучного інтелекту. URL: https://krs.chmnu.edu.ua/jspui/bitstream/123456789/3269/1/КРМ_Бондаренко_С_608.pdf (дата звернення: 25.05.2024).

7. Учасники проектів Вікімедіа. Сортування включенням – Вікіпедія. *Вікіпедія.* URL: https://uk.wikipedia.org/wiki/Сортування_включенням (дата звернення: 25.05.2024).

8. 40 algorithms every programmer should know. *Google Books.* URL: https://books.google.com.ua/books?hl=en&lr=&id=4IzrDwAAQBAJ&oi=fnd&pg=PP1&dq=Sorting+algorithms+dictate+the+organization+of+items+within+a+set+of+data.&ots=I6a0Kan6SM&sig=ft1TxBCxtixHbevPI_I4QB2l5cc&redir_esc=y#v=onepage&q&f=false (дата звернення: 25.05.2024).

9. About qt - qt wiki. *Qt Wiki*. URL: https://wiki.qt.io/About_Qt. (дата звернення: 29.05.2024).

10. A modern approach to supporting program visualization. URL: https://link.springer.com/article/10.1007/s11042-020-09611-0 (дата звернення: 25.05.2024).

11. Analysis of comparison-based sorting algorithms. URL: https://ieeexplore.ieee.org/abstract/document/9692969 (дата звернення: 27.05.2024).
12. An evaluation of quick sort and insertion sort algorithms. URL: https://pubs.aip.org/aip/acp/article-abstract/2727/1/040013/2895141/An-evaluation-of-quick-sort-and-insertion-sort?redirectedFrom=fulltext (дата звернення: 25.05.2024).
13. An integrated online pick-to-sort order batching approach for managing frequent arrivals of B2B e-commerce orders under both fixed and variable time-window batching. URL: https://www.sciencedirect.com/science/article/abs/pii/S147403462030094X (дата звернення: 25.05.2024).
14. Asymptotic analysis of the running time performed by various sorting algorithms. *IEEE Xplore.* URL: https://ieeexplore.ieee.org/abstract/document/9498490 (дата звернення: 25.05.2024).
15. A visualization framework to assist interpretation of Learning to Rank algorithms. URL: https://www.sciencedirect.com/science/article/abs/pii/S0097849320301515 (дата звернення: 27.05.2024).
16. A visualization method for functional understanding of high-dimensional pareto-optimal data-sets to aid multi-criteria decision making. URL: https://ieeexplore.ieee.org/abstract/document/9064747 (дата звернення: 27.05.2024).
17. Comparative analysis of sorting and searching algorithms. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4815467 (дата звернення: 27.05.2024).
18. Contributors to Wikimedia projects. Simple and fast multimedia library - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Simple_and_Fast_Multimedia_Library (дата звернення: 29.05.2024).
19. Most common sorting algorithms. *Geeksforgeeks.* URL: https://www.geeksforgeeks.org/sorting-algorithms/ (дата звернення: 27.05.2024).

20. OpenGL - the industry standard for high performance graphics. *OpenGL - The Industry Standard for High Performance Graphics*. URL: https://www.opengl.org/ (дата звернення: 29.05.2024).

21. Performance analysis of various sorting algorithms: comparison and optimization. URL: https://ieeexplore.ieee.org/abstract/document/10444609 (дата звернення: 27.05.2024).

22. Simple directmedia layer - homepage. *Simple DirectMedia Layer - Homepage*. URL: https://www.libsdl.org/ (дата звернення: 29.05.2024).

23. Sorting algorithms and their execution times an empirical evaluation. URL: https://link.springer.com/chapter/10.1007/978-3-030-63665-4_27 (дата звернення: 27.05.2024).

24. Using single-vesicle technologies to unravel the heterogeneity of extracellular vesicles. URL: https://www.nature.com/articles/s41596-021-00551-z (дата звернення: 27.05.2024).

# ДОДАТКИ

## Додаток А

## Технічне завдання на розробку програми для візуалізації алгоритмів сортування

Цей документ описує технічне завдання на розробку програми для візуалізації алгоритмів сортування з використанням мови програмування C++ та бібліотеки SFML. Метою даної розробки є створення навчального інструменту, який дозволяє наочно демонструвати роботу різних алгоритмів сортування.

### ПІДСТАВИ ДЛЯ РОЗРОБКИ

Розробка даної програми здійснюється у рамках виконання бакалаврської роботи. Потреба у розробці даного програмного забезпечення обумовлена необхідністю наочного навчання алгоритмам сортування для здобувачів освіти.

### ПРИЗНАЧЕННЯ РОЗРОБКИ

Програма призначена для демонстрації роботи різних алгоритмів сортування шляхом їхньої візуалізації. Вона дозволить користувачам:

- ознайомитися з принципами роботи різних алгоритмів сортування;
- аналізувати ефективність алгоритмів;
- порівнювати роботу алгоритмів на різних наборах даних.

### ВИМОГИ ДО ПРОГРАМНОГО ЗАСОБУ

Вимоги до функціональних характеристик

1. Підтримка алгоритмів сортування: програма повинна підтримувати візуалізацію наступних алгоритмів сортування:

- сортування бульбашкою (Bubble Sort);
- сортування вставками (Insertion Sort);
- сортування вибором (Selection Sort);
- швидке сортування (Quick Sort);
- сортування злиттям (Merge Sort).

2. Візуалізація: програма повинна візуально відображати процес сортування, включаючи початкове розташування елементів, поточні зміни і кінцевий результат.

3. Інтерфейс користувача: програма повинна мати інтуїтивно зрозумілий графічний інтерфейс, що дозволяє:

- вибирати алгоритм сортування;
- змінювати розмір масиву.

Нефункціональні вимоги

1. Продуктивність: програма повинна працювати без затримок та забезпечувати плавну анімацію візуалізації.

2. Надійність: програма повинна коректно працювати при будь-яких допустимих вхідних даних та не допускати збоїв.

Вимоги до програмної сумісності

Розробка повинна бути сумісною з операційними системами Windows, Linux та MacOs.

## СТАДІЇ І ЕТАПИ РОЗРОБКИ

1. Вивчення алгоритмів сортування та бібліотеки SFML.
2. Розробка архітектури програми.
3. Написання коду для кожного алгоритму сортування.
4. Створення інтерфейсу користувача.
5. Тестування.
6. Виправлення виявлених помилок.

## ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Після розробки було перевірено:

- плавність візуалізації алгоритмів сортування;
- коректне відображення на різних операційних системах;
- зручність інтерфейсу;
- правильність виконання роботи візуалізатора.

**Додаток Б**

## Інструкція користувачу

1. **Загальні відомості**

Програма для візуалізації алгоритмів сортування.

2. **Функціональне призначення**

Основна задача програми полягає у тому, щоб надати користувачам можливість візуально спостерігати за роботою різних алгоритмів сортування. Програма повинна підтримувати інтерактивний інтерфейс, що дозволить користувачам керувати процесом візуалізації, вибирати різні алгоритми та налаштовувати параметри сортування. Важливою функцією буде можливість зупинки, відновлення та уповільнення процесу візуалізації, що дозволить детально розглянути кожний крок алгоритму.

3. **Умови застосування програми**

Програма сумісна з комп'ютерами на базі операційних систем Windows, Linux та MacOs.

4. **Опис роботи програми**

Програма має інтуїтивно зрозумілий веб-інтерфейс, який дозволяє користувачам легко взаємодіяти з інструментом (Рис Б1). Інтерфейс складається з наступних основних компонентів:

- меню вибору алгоритму;
- область візуалізації;
- кнопки керування.

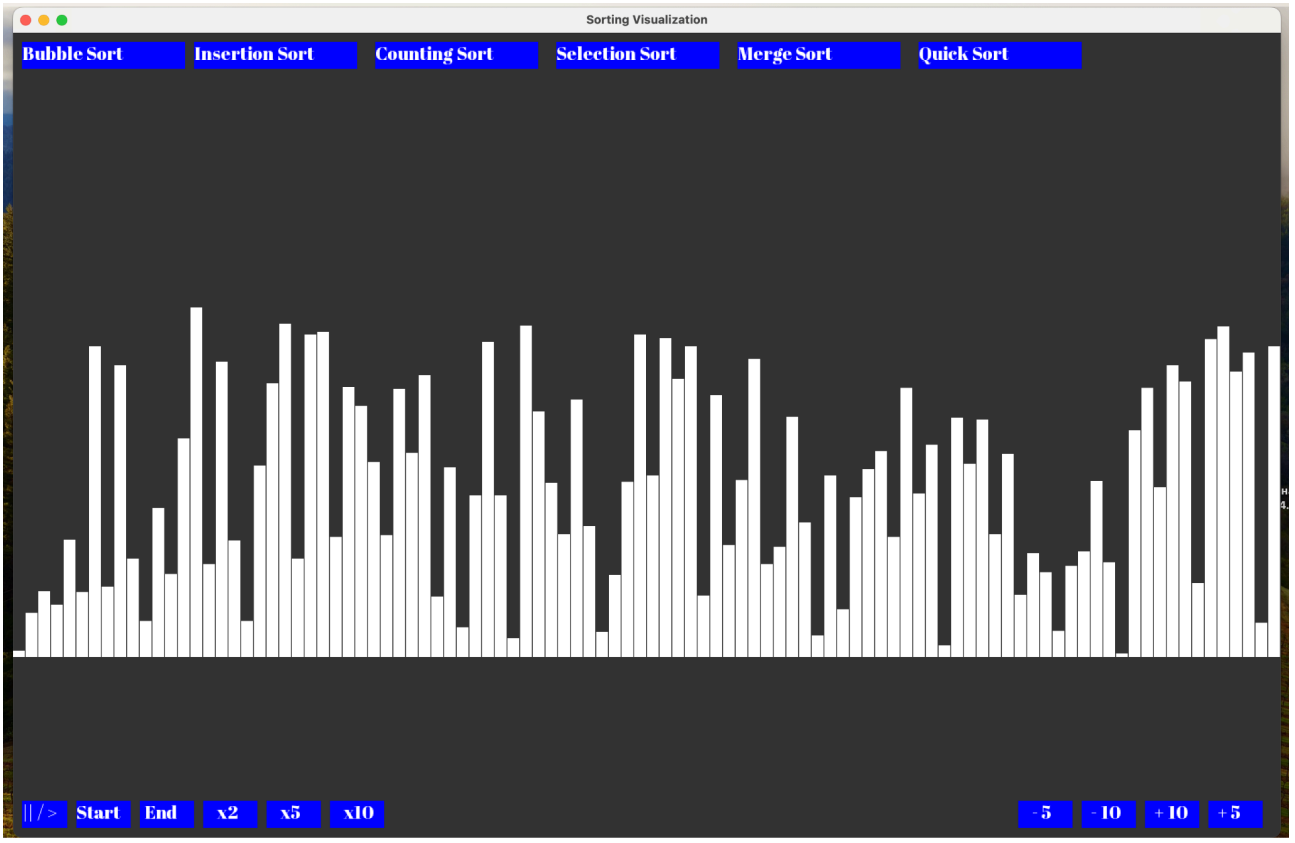


Рис. Б1 – Інтерфейс програми

Меню вибору алгоритму знаходиться у верхній частині інтерфейсу та дозволяє користувачу обирати один з доступних алгоритмів сортування. Список алгоритмів включає:

- Bubble Sort (Бульбашкове сортування);
- Insertion Sort (Сортування вставками);
- Selection Sort (Сортування вибором);
- Quick Sort (Швидке сортування);
- Merge Sort (Сортування злиттям).


Рис. Б2 – Меню вибору алгоритму

Область візуалізації (рис. Б3) розташована в центрі екрану та служить для графічного відображення процесу сортування. Кожен елемент масиву представлений у вигляді вертикальної смуги, висота якої відповідає значенню елемента.

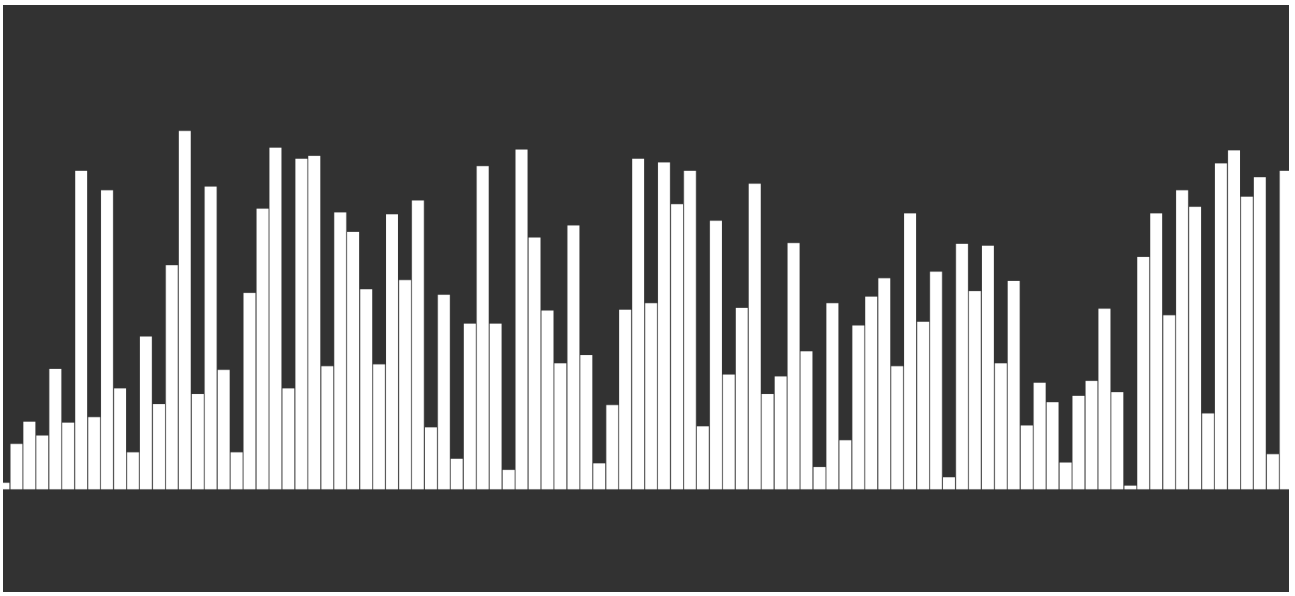

Рис. Б3 – Область візуалізації

Під час роботи алгоритму в реальному часі відображаються зміни в масиві. Анімація показує переміщення елементів, що дає можливість користувачу зрозуміти, як алгоритм обробляє дані на кожному кроці.

Кнопки керування (рис. Б4) розташовані під областю візуалізації і включають наступні функції:

- Start (Запуск): починає візуалізацію обраного алгоритму;
- Pause (Пауза): зупиняє анімацію, дозволяючи користувачу детально розглянути поточний стан масиву;
- End (Завершити): припиняє виконання алгоритму та повертає область візуалізації до початкового стану;


Рис. Б4 – Кнопки керування

Також у програмі присутні додаткові налаштування (рис. Б5), які дозволяють користувачам змінювати параметри візуалізації:

- Розмір масиву: кнопки, що дозволяють встановити кількість елементів у масиві, який буде сортуватись. Це дає можливість спостерігати за поведінкою алгоритмів при різних розмірах даних.

- Швидкість анімації: кнопки для зміни швидкості дозволяють регулювати швидкість виконання анімації, що корисно для кращого розуміння швидкості роботи алгоритму або детального аналізу його кроків.


Рис Б5 – Додаткові налаштування

Процес використання програми:

1. Використовуйте меню вибору алгоритму, щоб обрати потрібний алгоритм сортування.
2. Встановіть розмір масиву та швидкість анімації за допомогою відповідних кнопок.
3. Натисніть кнопку "Start" для початку процесу візуалізації.
4. Використовуйте кнопку "Pause" для зупинки анімації або "End" для завершення процесу.
5. Спостерігайте за анімацією сортування, аналізуючи поведінку алгоритму при різних умовах.

Програмний інструмент для візуалізації алгоритмів сортування є потужним освітнім ресурсом, що поєднує в собі інтерактивність, наочність та гнучкість. Він дозволяє користувачам краще розуміти принципи роботи алгоритмів сортування, аналізувати їх ефективність та застосовувати отримані знання на практиці.

Анотація

Шепелюк П.В. – **Проєктування та розробка програми для візуалізації алгоритмів сортування**.

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп’ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2024 р.

Бакалаврська робота присвячена розробці програмного забезпечення для візуалізації алгоритмів сортування з використанням бібліотеки Simple and Fast Multimedia Library (SFML). Основною метою проекту є створення інструменту, який полегшить навчання та дослідження алгоритмів сортування через їх інтерактивну візуалізацію.

У ході роботи розглянуто шість основних алгоритмів сортування: Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Counting Sort та Merge Sort. Для кожного алгоритму було розроблено і реалізовано інтерфейс, що дозволяє спостерігати за процесом сортування в реальному часі, контролювати швидкість візуалізації, зупиняти чи відновлювати виконання алгоритму, а також здійснювати покроковий аналіз кожного етапу сортування.

Візуалізація реалізована за допомогою бібліотеки SFML, яка забезпечує ефективне та гнучке середовище для графічного відображення процесів сортування. SFML надає інструменти для створення анімацій, що робить процес візуалізації плавним і зрозумілим. Програма побудована на основі об'єктно-орієнтованого підходу, що сприяє її легкій модифікації та розширенню. Архітектура програми передбачає модульність, що дозволяє легко додавати нові алгоритми та функції в майбутньому.

Цей інструмент може бути використаний як навчальний матеріал у курсах з алгоритмів та структур даних, а також для наукових досліджень у галузі обчислювальної інформатики. Він дозволяє здобувачам освіти і дослідникам не лише вивчати теорію алгоритмів сортування, а й бачити їхню роботу на практиці, що сприяє кращому розумінню та аналізу. Користувачі можуть експериментувати з різними наборами даних та параметрами алгоритмів, що

робить програму цінним інструментом для дослідження продуктивності різних методів сортування.

Програма також розроблена з урахуванням крос-платформенності, що дозволяє її використання на різних операційних системах, включаючи Windows, macOS та Linux. Це забезпечує широке охоплення потенційних користувачів та максимальну сумісність.

Ключові слова: алгоритми сортування, візуалізація, SFML, Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort, Counting Sort.