Національний університет «Чернігівський колегіум» імені Т.Г.Шевченка
Природничо-математичний факультет
Кафедра інформатики і обчислювальної техніки

Кваліфікаційна робота

освітнього ступеня «бакалавр»

на тему

ДОСЛІДЖЕННЯ ТА РОЗРОБКА КОМПОНЕНТІВ СИСТЕМИ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ СОРТУВАННЯ

Виконала:
Студентка-заочниця
4 курсу, 44-фмт групи
спеціальності
122 Комп’ютерні науки
Кочетова Аміна Миколаївна

Науковий керівник:
к.п.н. Костюченко А.О.

Чернігів, 2025

Роботу подано до розгляду «_____» _____ 2025 року.

Студентка _____ Кочетова А.М.

(підпис) (прізвище та ініціали)

Науковий керівник _____ Костюченко А.О.

(підпис) (прізвище та ініціали)

Рецензент _____ _____

(підпис) (прізвище та ініціали)

Кваліфікаційна робота розглянута на засіданні кафедри
Інформатики і обчислювальної техніки
протокол № _____ від «_____» _____ 2025 року.
Студентка допускається до захисту даної роботи в екзаменаційній комісії.

Завідувач кафедри _____ Горошко Ю. В.

(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

У сучасному світі невпинно зростає обсяг даних, який доводиться опрацьовувати за допомогою обчислювальної техніки. Ефективна обробка таких даних є ключовим показником продуктивності будь-якої системи. Однією з базових важливих операцій, що лежить в основі розв'язування великої кількості прикладних задач є сортування масивів. Воно застосовується в різних галузях – від баз даних і веб-додатків до систем штучного інтелекту, фінансового аналізу та оптимізації виробничих процесів. Тому алгоритми сортування залишаються предметом постійної уваги з боку дослідників, викладачів і розробників.

Не зважаючи на глибоке вивчення теми, сортування продовжує викликати інтерес в різних сферах – від викладання до практичного використання, особливо у контексті розуміння ефективності алгоритмів, їхньої адаптивності до різних типів вхідних даних та поведінки в граничних умовах. Одним з найефективніших способів засвоєння роботи алгоритмів є їх візуалізація. Візуалізація дозволяє не лише побачити послідовність дій алгоритму, але й краще зрозуміти внутрішню логіку його роботи, оцінити продуктивність у реальному часі.

Візуальні інструменти для навчання та аналізу алгоритмів є ефективними в освіті, особливо на технічних та ІТ-спеціальностях закладів вищої освіти. Багато студентів стикаються з труднощами під час вивчення алгоритмів сортування через абстрактність математичних описів. Застосування графічних інтерфейсів і анімації значно підвищує рівень розуміння й засвоєння матеріалу. Також з точки зору практичного програмування, модульні компоненти для візуалізації можуть бути інтегровані у більші системи, які можуть автоматично демонструвати процес виконання алгоритмів. Вказане зумовлює ***актуальність*** теми дослідження.

Об'єктом дослідження в даній роботі є алгоритми сортування та процеси їх візуального представлення.

Предметом дослідження є програмна реалізація та інтерфейсні компоненти, що забезпечують інтерактивну візуалізацію алгоритмів сортування.

Метою дипломної роботи є розробка програмного засобу, який дозволяє демонструвати принцип дії різних алгоритмів сортування з використанням засобів візуалізації.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

- здійснити огляд існуючих алгоритмів сортування та методів їх візуалізації;
- проаналізувати існуючі інструменти і технології розробки графічних інтерфейсів та обрати засіб реалізації програмного продукту;
- реалізувати набір базових алгоритмів сортування з підтримкою анімації процесу;
- провести тестування та дослідити можливості масштабування і розширення системи.

У процесі виконання дипломної роботи були використані такі ***методи*** дослідження: аналіз наукової літератури та існуючих програмних рішень, моделювання алгоритмів сортування, проектування графічного інтерфейсу та реалізація програмного коду з використанням сучасних мов програмування та бібліотек.

Створення системи візуалізації алгоритмів сортування сприятиме глибшому розумінню алгоритмічних процесів і може бути використане як у навчальному процесі, так і в демонстраційних або дослідницьких цілях.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ СОРТУВАННЯ ДАНИХ

1.1. Історія розвитку алгоритмів сортування.

Задача сортування даних має глибокі історичні корені та виникла ще задовго до появи сучасних комп'ютерів. Її витоки пов'язані з необхідністю впорядковувати інформацію для зручності використання, аналізу або прийняття рішень. Спробуємо прослідкувати, як ця задача виникла та еволюціонувала.

Ще в давні часи люди стикалися з необхідністю впорядковувати об'єкти, числа або записи. Наприклад, облік зерна, худоби або податків у стародавньому Вавилоні, Єгипті чи Китаї вимагав ведення впорядкованих записів. У середньовічних монастирях укладалися каталоги книг або списки пожертвувань, які впорядковувалися за алфавітом або датою. У торгівлі сортували товари за вагою, ціною, якістю, що було необхідною умовою для інвентаризації та обліку. Таким чином, сортування завжди виникало там, де потрібно було швидко знаходити інформацію або організовувати її у зручному для споживача вигляді.

У XIX столітті з появою перших державних адміністрацій і компаній, що обробляли великі обсяги даних (наприклад, залізниці, банки, страхові фірми), зросла потреба у швидкому доступі до впорядкованої інформації. Особливо важливу роль тут відіграли картотеки — це були фізичні масиви карток (записів), які сортували вручну за прізвищами (в алфавітному порядку), за датою, за номером рахунку або інвентаризаційним кодом тощо.

Важливе практичне значення проблема сортування даних у великих масивах вперше повстала в США в середині XIX століття. У 1840 році там був створений центральний офіс перепису населення, куди вийшли первинні дані з усіх штатів. В ході перепису було опитано 17 069 453 людини, а кожна анкета складалася з 13 питань. Обсяг одержаних даних був надто великим, а їх обробка традиційним ручним способом потребувала непомірних витрат праці та часу. Негатив ситуації посилювався необхідністю проведення постійних перерахунків та звірки даних, що виникали в результаті помилок, допущених при ручному сортуванні карток з даними. За кожного нового перепису, який

проводився раз на десять років, обсяг інформації, що необхідно опрацьовувати, збільшувався, а разом з ним суттєво збільшувалася вартість і тривалість обробки цих даних. Так, ручна обробка даних перепису населення 1880 року, що складалася з 50 189 209 осіб, вимагала залучення сотен службовців і тривала сім з половиною років.

Зрозуміло, що через надмірний обсяг традиційні методи обробки даних були недостатньо ефективні. Революційним моментом в історії становлення та розвитку алгоритмів сортування став перепис населення у США 1890 року. Перед цим переписом для вирішення проблем сортування даних за ініціативою бюро перепису був проведений конкурс на краще електромеханічне сортувальне обладнання, яке могло б зробити сортування даних більш ефективним – швидким, точним і дешевим. В конкурсі переміг американський інженер та винахідник німецького походження Герман Холлеріт (Herman Hollerith), який розробляв обладнання для роботи з перфокартами – електричну систему табулювання, яка стала відома під назвою Hollerith Electric Tabulating System. Використання цього обладнання при переписах населення США в 1890 і 1900 роках було визнано дуже успішним.

Використання способу табулювання під час сортування даних виявилося настільки ефективним, що попередні підрахунки результатів перепису тривали лише шість тижнів, а повний статистичний аналіз даних в подальшому зайняв два з половиною роки. В результаті обробка даних за допомогою машини Холлеріта зайняло втричі менше часу, ніж вручну. Таким чином, наприкінці XIX століття на зміну ручному сортуванню даних у великих масивах прийшло сортування за допомогою статистичних табуляторів. При цьому використовувався алгоритм ***порозрядного сортування***.

Наступний етап розвитку способів і алгоритмів сортування почався у 1940-х роках з появою перших електронних обчислювальних машин. Алгоритми сортування почали досліджуватись не лише з практичної точки зору, але й з математично, а сам період ознаменувався становленням

інформатики як науки, і сортування відігравало у цьому процесі фундаментальну роль. У цей час:

- зростає кількість інженерних і наукових розрахунків, які потребують попередньої обробки даних;
- виникають системи зчитування інформації з перфокарт, де впорядкування записів грає ключову роль;
- починається теоретичне осмислення програмування, зокрема робота Джона фон Неймана, Алана Т'юрінга, Дональда Кнута та інших.

Саме у 1940–1950-х роках відбувся перехід від ручного та механічного сортування до формального опису алгоритмів сортування в контексті обчислювальних машин. В цей період алгоритми почали описуватись формально псевдокодом або мовами низького рівня. Дослідники прагнули зменшити кількість операцій порівняння і переміщення, побудувати універсальні алгоритми, які працюють ефективно в загальному випадку, класифікувати їх за різними стратегіями (розділяй і володарюй, обмін, вставка тощо).

У ці роки дослідники створюють і вивчають основні алгоритми сортування, які наразі стали класичними:

- ***сортування вставками*** (Insertion Sort) хоч і відоме ще з античності, але активно застосовувалося у перших комп'ютерних програмах як простий та ефективний алгоритм для малих обсягів даних;
- ***сортування злиттям*** (Merge Sort) було запропоновано у 1945 Джоном фон Нейманом (John von Neumann). Історичне значення цього алгоритму полягало в тому, що це був перший відомий рекурсивний алгоритм сортування;
- у 1946 році вийшла перша стаття про алгоритми сортування даних, автором яких був Джон Уільям Мочлі (John William Mauchly) – американський фізик та інженер, один із творців першого в світі

електронного цифрового комп'ютера загального призначення ENIAC. В статті розглядався цілий ряд нових алгоритмів сортування, в тому числі метод ***бінарних вставок***;

- ***сортування бульбашкою*** (Bubble Sort) часто використовувалось у початкових версіях програмного забезпечення. Попри низьку ефективність стало популярним у навчальних курсах через простоту реалізації;
- ***коктейльне*** або ***шейкерне сортування*** (Cocktail Sort) та ***сортування вибором*** (Selection Sort) хоч і мали обмежене практичне застосування, але відгравали важливу роль у навчанні та вивчалися як альтернативи бульбашковому сортуванню.

Ще одним наслідком переходу до сортування даних за допомогою комп’ютерів стало розділення сортування на два типи зовнішнє і внутрішнє, тобто на такі, які використовують чи не використовують дані, що розташовані на пристроях зовнішньої пам’яті.

Різні модифікації методів сортування злиттям і вставками були найбільш поширеними до середини 1950-х років. У середині 1950-х років з розробкою комп’ютерів другого покоління почався наступний крок розвитку алгоритмів сортування. Основними передумовами для цього стали:

- значне підвищення доступності комп’ютерів та їх широке поширення в результаті різкого зменшення їх габаритів та вартості;
- збільшення продуктивності комп'ютерів до десятків тисяч операцій в секунду;
- прискорення написання програм для комп'ютерів в результаті розробки перших мов програмування високого рівня (Фортран, Алгол, Кобол).

У другій половині 1950-х починає формуватися уявлення про обчислювальну складність алгоритмів. Сортування стало однією з перших задач, для яких визначено часову складність у найгіршому, середньому і найкращому випадках та встановлено нижню межу для порівняльного

сортування (через аналіз дерева рішень). Ці поняття стали підґрунтям для розвитку теорії алгоритмів у наступні десятиліття.

В цей час з'явилося багато алгоритмів, які широко використовуються й досі:

- у 1959 році Дональд Левіс Шелл (Donald Lewis Shell) запропонував метод сортування зі зростаючим кроком (shellsort) відомий зараз як ***сортування Шелла***;
- у 1960 році Чарльз Ентоні Річард Хоар (Charles Antony Richard Hoare) розробив метод ***швидкого сортування*** (quicksort);
- у 1964 році Дж. У. Дж. Уильямс (J. V. J. Williams) запропонував метод ***пірамідальної сортування*** (heapsort).

Також інженери почали активно працювати над апаратною реалізацією сортування, зокрема над створенням механічних або електромагнітних пристроїв для сортування перфокарт та розробкою електронних схем, які могли б паралельно обробляти різні об'єкти (ранній аналог сучасних сортувальних мереж).

Чергова хвиля інтересу до алгоритмів сортування відбулася в середині 1970-х років, коли елементною базою комп'ютерів стали великі інтегральні схеми та з'явилася можливість об'єднання обчислювальних потужностей комп'ютерів шляхом створення єдиних обчислювальних центрів.

У 1970–1990-х роках дослідження алгоритмів сортування перейшли на якісно новий рівень. Якщо у 1940–1950-х роках основна увага приділялась базовим методам і апаратному сортуванню, то в подальші десятиліття на перший план вийшли:

- теоретичний аналіз продуктивності алгоритмів;
- алгоритмічна оптимізація;
- спеціалізація під нові архітектури комп'ютерів;
- практичне використання сортування в базах даних, компіляторах, системах управління даними.

У період з середини 1970-х до 1990-х років були досягнуті значні успіхи в збільшенні швидкості сортування за рахунок підвищення ефективності вже відомих до того часу алгоритмів шляхом їх доопрацювання або комбінування. Так нідерландський вчений Едсгер Вібе Дейкстра (Edsger Wybe Dijkstra) у 1981 році запропонував алгоритм ***плавного сортування*** (Smoothsort), який є розвитком пірамідальної сортування (Heapsort). Також зароджується як ідея алгоритм ***Timsort***, який комбінує алгоритми сортування злиттям та сортування включенням (Insertion Sort + Merge Sort). Цей алгоритм в подальшому був втілений у життя Тімом Пітерсом (Tim Peters) у 2002 році та реалізований у Python і Java. В 1997 році Девідом Мюссером (David "Dave" Musser) запропановано алгоритм Introsort (Quicksort + Heapsort), що використовує швидке сортування, але перемикається на пірамідальне сортування, коли глибина рекурсії перевищить деякий наперед визначений рівень. Цей алгоритм застосовується в STL (Standard Template Library) мов C++.

Ще одним напрямом вдосконалення алгоритмів сортування в цей час став пошук оптимальних вхідних послідовностей для різних методів сортування, що дозволило значно скоротити час роботи вже відомих алгоритмів. Такі алгоритми отримали назву адаптивних алгоритмів сортування. Під адаптивним алгоритмом сортування розуміють алгоритм, який використовує переваги вже «відсортованих» елементів у списку вхідних даних для досягнення кращої продуктивності. На відміну від класичних алгоритмів, які завжди проходять через одні й ті самі кроки, незалежно від структури даних, адаптивні алгоритми змінюють свою поведінку, щоб скористатися перевагами часткової впорядкованості, повторюваності елементів або інших специфік.

Ключовими ідеями адаптивних алгоритмів є:

- виявлення послідовних відсортованих підпослідовностей;
- використання локальної впорядкованості, наприклад часткового сортування;
- поєднання простих алгоритмів, які добре працюють в певних умовах;

- обхід даних для попереднього аналізу їх структури.

Перевагами адаптивних алгоритмів є:

- ефективність при частковій впорядкованості;
- висока продуктивність на реальних, а не теоретичних даних;
- ефективність використання у системах з обмеженою пам’яттю.

Однак при цьому адаптивні алгоритми мають і ряд недоліків:

- ускладнена реалізація порівняно з простими алгоритмами;
- знижена ефективність для невпорядкованих даних;
- потреби в додатковій пам’яті в окремих випадках.

Наступним напрямком, який розвивався найбільш інтенсивно в цей період, був пошук розв’язків задачі сортування в класі паралельних алгоритмів, для чого не тільки узагальнювалися раніше відомі парадигми, але і розроблялися принципово нові алгоритми. В цей час створюються паралельні сортувальні мережі, такі як ***Bitonic Sort, Odd-even Mergesort, Sorting Network,*** що застосовуються в апаратних реалізаціях і суперкомп’ютерах.

Наступний етап почався з середини-кінця 1990-х років і, можна вважати, триває до цього часу. Загальними тенденціями розвитку досліджуваного питання в цей період можна вважати:

- переорієнтація на практичну ефективність, а не лише теоретичну складність;
- розвиток адаптивних, гібридних та кеш-ефективних алгоритмів;
- масштабування сортування для розподілених систем (Big Data);
- оптимізація для апаратні особливостей реалізації: CPU cache, SIMD, GPU;
- інтеграція в стандартні бібліотеки мов програмування.
- дослідження задач сортування на частково впорядкованих множинах (задачі розпізнавання частково упорядкованої множини, задачі сортування частково упорядкованого множини з використанням результатів попарних порівнянь елементів, задачі визначення порядку на множині без попередньої інформації).

Актуальність цих задач пояснюється появою і широким використанням комп'ютерів на мікропроцесорах з паралельно-векторною структурою, а також високоефективних мережевих обчислювальних комп'ютерних системах.

В цей час з’являються чи доопрацьовуються такі алгоритми як:

- ***Flashsort*** – алгоритм, що описано в 1998 році Карлом-Дитрихом Нойбертом (Karl-Dietrich Neubert). Ідея алгоритму заснована на спостереженні та загальній концепції, що великі масиви статистичних даних цілком логічно сортувати, у відповідності до теорії ймовірностей, коли кожен елемент займає найбільш ймовірне місце в масиві;
- ***Spreadsort*** – алгоритм сортування, винайдений Стівеном Дж. Россом (Steven J. Ross) у 2002 році. Він поєднує в собі концепції сортувань на основі розподілів, таких як порозрядне сортування (radix sort) та сортування блоками (bucket sort), з концепціями розбиття на частини, що використовується алгоритмами сортувань на основі порівняння, таких як швидке сортування (quicksort) та сортування злиттям (mergesort). Експериментальні результати показали його високу ефективність, часто перевершуючи традиційні алгоритми, особливо при сортуванні текстових рядків;
- ***Burstsort*** – вперше опублікований в 2003 році, з деякими оптимізуючими версіями, опублікованими в пізніші роки. Відноситься до кеш-ефективних алгоритмів сортування рядків, який є варіантами традиційного сортування за основою, але швидші для великих наборів даних поширених рядків.

З розвитком систем штучного інтелекту особливим напрямом досліджень стало вдосконалення та адаптація алгоритмів сортування для машинного навчання та Data Science:

- пбудова ранжувань;
- вибіру топ-K елементів;
- оптимізація алгоритмів кластеризації та пошуку сусідів;
- індексування великих наборів даних.

1.2. Прикладні задачі, що використовують алгоритми сортування.

Інформаційні технології посідають дуже значне, можливо, навіть провідне місце у сучасному світі. Їх використання тісно пов’язане з обробкою великих обсягів даних, що стало звичним завданням як у науці, так і в бізнесі, медицині, транспорті, освіті та інших сферах. Ефективне опрацювання цих даних, управління ними часто починається з їх сортування. А значить, можна стверджувати, що алгоритми сортування даних відіграють ключову роль у забезпеченні швидкого доступу до інформації, її зручного відображення, аналізу та подальшої обробки.

Незважаючи нібито на свою простоту, сортування являє собою основу багатьох складних обчислювальних задач. А отже ефективне вирішення задачі сортування має прямий вплив на продуктивність різного програмного забезпечення.

Розглянемо кілька типових задач, які визначають необхідність у сортування даних.

1. Пошук інформації.

Однією з головних практичних задач, яка обумовлює необхідність сортування масивів, є пошук необхідних даних. Упорядковані дані дозволяють застосовувати ефективні алгоритми пошуку, наприклад бінарний пошук (зокрема, такий пошук може бути застосований для ефективного розв’язку в дитячій грі «більше-менше», де треба вгадати загадане число). У випадку, якщо дані не відсортовані, то єдиний варіант знайти потрібне значення – перебирати їх послідовно, що може займати великий час. Якщо ж масив упорядкований за зростанням чи спаданням, то ми можемо знайти потрібний елемент за набагато менший час. Це критично важливо, коли йдеться про великі обсяги даних, наприклад, у базах даних або пошукових системах.

Стандартною задачею, пов’язаною з таким пошуком, є використання користувачем пошукових інтернет-запитів. Щоб швидко знайти відповідну інформацію серед мільярдів веб-сторінок, для пошукової системи є обов’язковим попереднє сортування та індексація веб-документів. Така ж

логіка простежується і в електронних базах даних клієнтів, реєстрах державних установ, банківських системах тощо.

2. Операційні системи та системне програмування.

Системне програмування охоплює розробку програмного забезпечення, яке безпосередньо взаємодіє з апаратним забезпеченням комп'ютера або забезпечує базову функціональність для роботи прикладних програм. До такого програмного забезпечення належать компілятори, драйвери пристроїв, ядра операційних систем, файлові системи, планувальники задач тощо. В усіх цих компонентах сортування даних відіграє важливу, іноді навіть критичну роль для забезпечення ефективності та стабільності роботи всієї системи:

- ***Планування процесів.*** Операційні системи постійно працюють із чергами задач (процесів чи потоків), які потрібно опрацювати. Ці задачі можуть мати різні пріоритети, час прибуття, тривалість виконання. Для планувальника задач важливо, щоб їх обробка була оптимальною, з метою уникнення затримок, блокувань або надмірного використання ресурсів. У такому контексті сортування процесів за певними критеріями (наприклад, за пріоритетом або за залишковим часом виконання) є необхідним етапом планування. Типовим прикладом мое бути алгоритм планування SJF (Shortest Job First), де процеси сортуються за очікуваним часом виконання. У більш складних алгоритмах, наприклад, багаторівневого планування з витісненням, сортування використовується для динамічного перевпорядкування процесів залежно від зміни умов (наприклад, коли з'являється новий високопріоритетний процес).
- ***Багатозадачність та мультипроцесорні (багатоядерні) системи***. У сучасних багатоядерних процесорах операційна система повинна максимально ефективно розподіляти задачі між різними ядрами. Це передбачає сортування черг задач не лише за пріоритетами, а й за критеріями, пов'язаними з ресурсним навантаженням, залежностями між процесами, використанням кеш-пам'яті та іншими

характеристиками. Наприклад, процеси можуть бути впорядковані за історією використання процесору або пам’яті, щоб уникнути перевантаження одного ядра або кешу.

- ***Управління пам’яттю.*** Алгоритми сортування також використовуються при управлінні оперативною пам’яттю. Наприклад, коли потрібно знайти найменший за розміром блок вільної пам’яті, достатній для розміщення нового процесу або структури даних, система може попередньо відсортувати список вільних блоків за розміром.
- ***Робота з файловою системою.*** Такі файлові системи як NTFS, ext4 та інші, використовують структури даних, в яких файли та каталоги зберігаються в певному порядку. Це суттєво пришвидшує доступ, читання та запис даних на зовнішні носії. Наприклад, при виведенні списку файлів у каталозі сортування може виконуватися за назвою, датою створення, розміром тощо. Крім того, індексовані файлові системи зберігають дані у вигляді дерев, які підтримують порядок елементів. При вставці або видаленні файлів ці структури вимагають часткових операцій сортування, щоб зберігати ефективність пошуку. Також слід згадати процес дефрагментації файлів – упорядкування фізичних блоків файлів на зовнішніх носіях з метою зменшення фрагментації та підвищення швидкості доступу до файлів. Алгоритми, що використовуються при цьому, спираються на сортування блоків за їх розташуванням або часом останнього доступу.
- ***Обробка системних журналів і логів.*** Системні журнали (логи) часто містять великі обсяги інформації про роботу операційної системи, драйверів, процесів і служб. Для аналізу цих даних (зокрема, виявлення помилок, загроз, нестабільної поведінки програм та ін.) необхідно впорядкувати логи за часом, важливістю подій, джерелом тощо. У системах моніторингу та адміністрування автоматизоване сортування логів дозволяє швидко виявити критичні події,

відфільтрувати інформацію та створювати різноманітні звіти. При цьому використовуються алгоритми, здатні працювати в режимі реального часу або алгоритми на потоках даних, де постійно додаються нові записи.

- ***Безпека даних та контроль доступу***. У системах забезпечення безпеки даних сортування часто застосовується для обробки політик доступу (ACL — Access Control List), журналів доступу або запитів до ресурсів. Наприклад, при перевірці прав користувачів у складних корпоративних системах запити можуть сортуватися за рівнем привілеїв або важливістю. Також у системах з великою кількістю користувачів сортування використовується для визначення активності користувачів, частоти входів, спроб несанкціонованого доступу тощо. Це дозволяє побудувати профілі поведінки і виявити потенційні загрози.

3. Веб-технології та розподілені системи.

У веб-програмуванні на стороні сервера часто доводиться працювати з великими наборами даних. Наприклад, стандартною задачею є фільтрація й сортування таблиць, які містять тисячі, а подекуди й сотні тисяч записів. У випадках, коли дані розташовані в різних джерелах або зберігаються розподілено (зокрема, у хмарних базах даних), стає критичною необхідність в ефективних алгоритмах сортування, які можуть працювати в потоковому або розподіленому режимі. Зокрема, можна відмітити наступні сфери використання алгоритмів сортування у веб-технологіях:

- ***Сортування у веб-інтерфейсах.*** Сортування масивів даних є стандартною функцією в багатьох динамічних веб-застосунках: сайти інтернет-магазинів, банківських систем або платформ онлайн-навчання надають можливість сортувати доступну інформацію (товари за ціною, курси за рейтингом, операції за датою тощо). Усі ці дії активують запити до бази даних або веб-серверів, які повинні обробити, відсортувати дані й повернути їх у впорядкованому вигляді.

У більшості випадків сортування реалізується або на стороні клієнта (з використанням JavaScript, якщо обсяг даних невеликий), або на стороні сервера (з використанням PHP, Python, Node.js тощо).

- ***Сортування у хмарних і розподілених системах***. Якщо кількість даних, яка має бути опрацьована, не може бути розміщена в оперативній пам’яті, а зберігається на дисках чи у віддалених сховищах, використовується так зване зовнішнє сортування (external sort), яке працює з блоками інформації. Це є важливим для різноманітних пошукових систем, відеоплатформ або хмарних обчислень, де обробка й упорядкування даних відбувається на рівні кількох фізичних машин. В аналогічних випадках, коли при опрацюванні надвеликих обсягів даних, загальний об’єм всієї сукупності даних значно перевищує можливості одного сервера, на допомогу приходять розподілені системи, у яких дані зберігаються й обробляються паралельно на багатьох вузлах. У таких системах сортування є складним завданням, оскільки воно потребує координації дій між вузлами, ефективної синхронізації та мінімізації обміну даними.
- ***Мережеві аспекти сортування.*** У деяких веб-додатках сортування виконується не лише як частина логіки, але й у рамках передачі даних. Наприклад, у peer-to-peer мережах, CDN-системах або потокових сервісах (як-от YouTube, Netflix) алгоритми впорядковують пакети даних, обирають оптимальні джерела трансляції або визначають порядок доставки частин контенту. Наприклад, у сервісах потокового відео різні потоки можуть бути розміщені у кількох дата-центрах. Система повинна визначити з якого джерела або в якій послідовності користувач має отримувати дані з кожного такого дата-центру. Тут сортування використовується для вибору оптимальних варіантів за часом затримки, навантаженням на сервер, якістю зв’язку тощо.

- ***Сортування в реальному часі та стрімінг-платформи.*** У сучасному веб-сервісах часто виникає необхідність обробки потокових даних у режимі реального часу (наприклад, коментарів до онлайн-трансляцій, біржових котирувань та ін). Тут використовуються спеціалізовані стрімінгові платформи, такі як Kafka, Apache Flink, Spark Streaming. Сортування у таких системах виконується в умовах постійного оновлення даних. Це означає, що алгоритми повинні оновлювати порядок елементів без повного перебору, додаючи нові елементи до вже впорядкованої структури (наприклад, з використанням купи або збалансованого дерева).

4. Аналіз та обробка великих даних.

У сфері Data Science та Big Data сортування є однією з найважливіших операцій попередньої обробки даних. Наприклад, перед виявленням аномалій у даних або обчисленням медіани необхідно упорядкувати масив даних, який буде опрацьовуватись. Деякі алгоритми машинного навчання, зокрема методи, засновані на відстані між об’єктами (наприклад, k-ближчих сусідів), потребують відсортованих даних для прискорення обчислень. Також у фінансовому аналізі часто необхідно ранжувати транзакції, сортувати акції за прибутковістю, виявляти найприбутковіші чи найризикованіші інвестиції. Все це вимагає попереднього сортування великих обсягів інформації.

5. Оптимізація інших алгоритмів

Як вже відзначалося в попередньому пункті, багато складніших алгоритмів залежать від попереднього сортування. Наприклад, у задачі пошуку найближчих пар чисел, для знаходження підмножин з певними властивостями, у жадібних або динамічних алгоритмах відсортовані дані дозволяють значно зменшити час виконання програм. У деяких випадках сортування є невід'ємною частиною стратегій зменшення складності задачі, переходу до рекурсивних або жадібних підходів.

У комбінаторних задачах сортування часто виступає як передумова для ефективного розв’язання, наприклад, при побудові оптимального маршруту, плануванні ресурсів, розподілі навантажень.

У багатьох графових алгоритмах сортування також відіграє ключову роль. Наприклад, у топологічному сортуванні (topological sort) сам процес побудови впорядкованого списку вершин графа — це один з ключових етапів оптимізації побудови порядку виконання задач із залежностями. Алгоритми потоків у мережах (наприклад, Edmonds-Karp) іноді потребують впорядкування шляхів або рівнів для оптимізації пошуку збільшувальних шляхів.

Також попереднього сортування (за координатами, кутами або відстанями) вимагають різні геометричні та просторові алгоритми. Так, побудова опуклої оболонки в алгоритмі Грехема вимагає спочатку відсортувати точки за кутом або координатою X. Побудова діаграм Вороного або тріангуляцій також вимагає упорядкованого представлення точок для ефективного обчислення.

6. Освітнє та професійне значення.

З практичної точки зору, вивчення алгоритмів сортування – це не лише шлях до розуміння роботи комп’ютера з даними, а й основа розвитку алгоритмічного мислення. Кожен алгоритм, від простого сортування бульбашкою до складного швидкого сортування (quicksort), має свої переваги, недоліки, сценарії застосування та вплив на продуктивність. Знання цих особливостей дозволяє обирати оптимальні рішення залежно від умов задачі.

Тому задача сортування може розглядатись як прекрасна платформа для навчання таким важливим поняттям, як масив, складність алгоритму, рекурсія, динамічна пам’ять тощо.

Таким чином, необхідність вивчення та використання алгоритмів сортування випливає з дуже широкого спектра практичних задач, які ми зустрічаємо щодня: від простих дій зручного відображення даних у різних прикладних додатках до обробки гігабайтів даних у промислових системах.

Розуміння механізмів сортування, їх ефективності та сфери застосування є важливою складовою сучасної комп’ютерної грамотності. Вивчення сортування може розглядатись як вміння працювати з інформацією, бачити структуру в хаосі та знаходити оптимальні шляхи обробки даних. Саме тому сортування є фундаментальним елементом у навчанні алгоритмам і програмуванню загалом.

1.3. Огляд основних алгоритмів сортування.

Сортування – одна з фундаментальних задач інформатики, яка лежить в основі багатьох комп'ютерних процесів. Незважаючи на простоту постановки, алгоритми сортування мають критичне значення для ефективної роботи систем зберігання та обробки даних, пошукових механізмів, баз даних, компіляторів, веб-додатків і навіть елементів штучного інтелекту. У реальних обчислювальних системах сотні тисяч операцій базуються на попередньому впорядкуванні масивів, списків або інших структур.

Із розвитком комп'ютерної техніки, збільшенням обсягів даних та появою нових архітектур (багатоядерні процесори, GPU, розподілені системи), задача сортування постала у новому світлі: виникла необхідність у створенні більш швидких, стабільних, адаптивних, багатопотокових та кеш-ефективних алгоритмів. В наслідок цього з'явилася ціла низка нових підходів та гібридних рішень, що поєднують переваги класичних методів з адаптацією до властивостей вхідних даних і технічних обмежень.

Огляд алгоритмів сортування дозволяє зрозуміти їх внутрішні механізми та оцінити складність виконання, виявити оптимальні рішення для конкретних типів задач. У цьому контексті важливим є не тільки аналіз теоретичних характеристик (часова і просторова складність), але й розуміння практичного застосування в сучасному програмному забезпеченні та інфраструктурі даних.

На ефективність алгоритму впливає кілька параметрів, які відображають як теоретичну ефективність, так і практичну придатність. Серед основних параметрів слід відмітити:

1. **Часова складність** (Time Complexity) – визначає, скільки операцій виконує алгоритм у залежності від кількості елементів n і позначається $O(n)$. Це один із найважливіших критеріїв при виборі алгоритму для великих обсягів даних. Окремо розглядають:

- найгірший випадок (Worst-case) – максимальна кількість операцій (наприклад, $O(n^2)$ в алгоритмі сортування бульбашкою).

- найкращий випадок (Best-case) – мінімальна кількість операцій (наприклад, *O(n)* у сортуванні вставками, якщо дані вже відсортовані).
- середній випадок (Average-case) – середня очікувана кількість операцій при випадковому порядку елементів.

2. **Просторова складність** (Space Complexity) – оцінка кількості додаткової пам’яті, яку використовує алгоритм. Просторова ефективність особливо важлива при обмеженій оперативній пам’яті. Розрізняють:

- *in-place алгоритми* – використовують постійну кількість пам’яті, (наприклад, HeapSort, QuickSort);
- не *in-place алгоритми* – потребують додаткової пам’яті (наприклад, MergeSort з окремими масивами).

3. **Кількість порівнянь і перестановок**. У багатьох алгоритмах сортування продуктивність залежить від того, скільки разів елементи порівнюються між собою, а також скільки разів відбуваються перестановк и або переміщення елементів. Це особливо важливо при роботі з великими або складними об’єктами, оскільки у реальних задачах це може більше впливати на швидкість роботи, ніж часова складність алгоритму.

Крім того, для вибору алгоритму та його практичної реалізації в певних прикладних задачах важливу роль можуть відігравати додаткові параметри:

1. **Адаптивність** – наскільки алгоритм враховує часткову впорядкованість вхідних даних. Адаптивні алгоритми можуть працювати значно швидше, якщо масив вже частково впорядкований, а неадаптивні завжди мають однакову поведінку, незалежно від структури вхідних даних.

2. **Складність реалізації** – показує наскільки легко реалізувати алгоритм правильно та ефективно. Деякі алгоритми (наприклад, сортування вставками чи бульбашкою) дуже прості і зрозумілі, тому їх часто використовують у навчальних задачах. Інші (наприклад, Timsort) — складні у реалізації, але набагато ефективніші в реальних задачах.

3. **Типи даних, з якими працює алгоритм**. Для прикладних задач дуже важливим є питання, чи працює вказаний алгоритм тільки з числами, чи може сортувати об’єкти, рядки та інші структури, чи потребує він реалізації операцій порівнювання або хешування? Такі алгоритми мають свої обмеження або переваги на певних класах задач.

4. **Стабільність.** Сортування називається стабільним, якщо елементи з однаковим ключем зберігають свій початковий порядок після сортування. Стабільність важлива при багаторівневому сортуванні або роботі з об’єктами з кількома полями. Наприклад, при сортуванні списку студентів за датою народження, стабільний алгоритм збереже початковий порядок серед працівників з однаковою датою народження.

5. **Паралелізм і масштабованість**. Дана характеристика визначає чи може алгоритм бути ефективно реалізований у багатопоточному, розподіленому або GPU-середовищі? Це актуально для роботи з великими даними, для обчислювальних кластерів або GPU-прискорення.

Всі алгоритми сортування за основним принципом дії, що лежить в основі їх роботи, можна умовно поділити на декілька класів:

1) обміном – поступово обмінюють місцями пари елементів, які стоять у неправильному порядку;
2) вставками – елемент масиву із невпорядкованої частини вставляються у правильне місце;
3) вибором – на кожному кроці знаходиться мінімальний або максимальний елемент і ставиться на своє місце в масиві;
4) розподілом – елементи розподіляються в групи (комірки, відра, корзини, діапазони) за певним критерієм, без порівнянь, після чого сортуються в межах групи та об'єднуються у впорядкований список;
5) злиттям – розбиває масив на підмасиви, сортує їх рекурсивно та зливає у відсортований результат;

6) паралельні – сортування виконується одночасно на кількох ядрах, потоках або вузлах у кластері, із подальшим об’єднанням результатів;
7) гибридні – алгоритми, які поєднують у собі елементи кількох базових алгоритмів для досягнення оптимальної продуктивності в різних умовах.

1.1. **Дурне сортування** (Stupid sort).

Це найпростіший навчальний алгоритм низької ефективності. При мінімальній зміні він трансформується в сортування бульбашкою.

Алгоритм можна описати наступним чином. Переглядаємо масив, починаючи з першого індексу, та порівнюємо сусідні елементи. Якщо знаходимо невідсортовану пару – міняємо елементи місцями та повертаємося на початок масиву. Повторюємо ті ж дії доки під час повного проходу не виявиться жодної невідсортованої пари.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування обмінами	Найгірша $O(n^3)$ Середня $O(n^3)$ Найкраща $O(n^3)$	Загальна $O(n)$ Додаткова $O(1)$

1.2. **Сортування бульбашкою** (Bubble sort).

Один з найпростіших алгоритмів сортування, який не застосовується для вирішення практичних завдань через низьку ефективність, але часто використовується як навчальний алгоритм, оскільки є основою деяких інших, більш ефективних сортувань. Швидко впорядковує тільки масиви дуже невеликого розміру. Алгоритм отримав свою назву від того, що процес сортування за ним нагадує поведінку бульбашок повітря у резервуарі з водою.

Алгоритм можна описати наступним чином. Здійснюється багаторазовий прохід масивом: спочатку від першого до останнього елемента, потім від першого до передостаннього, потім від першого до третього з кінця тощо. Під

час прогону порівнюються сусідні елементи. Якщо вони не впорядковані відносно один одного, то вони міняються місцями. В результаті при кожному проході на кінець поточного підмасиву як бульбашка спливає найбільший (найменший) елемент.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування обмінами	Найгірша $O(n^2)$ Середня $O(n^2)$ Найкраща $O(n)$	Загальна $O(n)$ Додаткова $O(1)$

1.3. **Коктейльне сортування** (Cocktail sort).

Коктейльне сортування є модифікацією сортування бульбашкою. Цей алгоритм носить ще й інші назви: двостороннє сортування бульбашкою (Bidirectional bubble sort), сортування змішуванням (Shuffle sort), шейкерне сортування (Shaker sort), човникове сортування (Shuttle sort), пульсуюче сортування (Ripple sort).

Алгоритм полягає в наступному. Здійснюється багаторазовий прохід масивом, при цьому сусідні елементи порівнюються і, у разі потреби, сони змінюються місцями. При досягненні кінця масиву наприкінці буде максимальний (мінімальний) елемент, після чого напрямок проходу змінюється на протилежний. Таким чином по черзі виштовхуються максимальні та мінімальні елементи масиву в кінець і початок структури відповідно (або навпаки).

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування обмінами	Найгірша $O(n^2)$ Середня $O(n^2)$ Найкраща $O(n)$	Загальна $O(n)$ Додаткова $O(1)$

1.4. **Сортування гребінцем** (Comb sort)

Сортування гребінцем є поліпшенням алгоритму сортування бульбашкою. У сортуванні бульбашкою, коли два елементи порівнюються,

вони завжди порівнюються із сусіднім елементом, тобто мають розрив рівний 1. Основна ідея сортування гребінцем полягає у тому, що цей розрив може бути більший одиниці.

Алгоритм можна описати наступним чином. Відбуваються неодноразові проходи масивом, при яких порівнюються пари елементів, що знаходяться один від одного на деякій відстані. Якщо вони не відсортовані один щодо одного, то проводиться обмін. В результаті максимальні елементи мігрують в один кінець масиву, а мінімальні за значенням – в інший.

При кожному проході відстань між елементами зменшується, доки не досягне мінімального значення. Ця відстань між порівнюваними елементами розраховується за допомогою спеціальної величини, як називається фактором зменшення. Початкова відстань обчислюється як довжина масиву поділена на цей фактор. Після кожного проходу попередня відстань ділиться на фактор зменшення і таким чином виходить нове значення.

В результаті практичних тестів та теоретичних досліджень оптимальне значення для фактора зменшення визначено як $1/\left(1-{}^{1}/_{e^{\varphi}}\right) \approx 1{,}2473309501.$

Слід зазначити, що на масивах до кількох тисяч елементів швидкість цього алгоритму може бути вищою, ніж у швидкого сортування.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування обмінами	Найгірша $O(n^2)$ Середня $\Omega\left(\frac{n^2}{2^p}\right)$, де p – кількість проходів. Найкраща $O(n \cdot \log n)$	Загальна $O(n)$ Додаткова $O(1)$

1.5. **Швидке сортування** (Quicksort)

Вважається одним з найшвидших відомих універсальних алгоритмів сортування масивів, але через наявність низки недоліків на практиці використовується з деякими доопрацюваннями.

Ідея алгоритму полягає в переставлянні елементів масиву таким чином, щоб його можна було розділити на дві частини і кожний елемент з першої частини був не більший за будь-який елемент з другої. Впорядкування кожної з частин відбувається рекурсивно.

Алгоритм можна описати наступним чином. Спочатку обирається із масиву елемент, який називають опорним. В загальному випадку це може бути будь-який з елементів масиву. Проте від вибору опорного елемента в окремих випадках може сильно залежати його ефективність.

Наступним кроком слід порівняти решту елементів масиву з опорним і переставити їх у масиві так, щоб розбити масив на три відрізки, що слідують один за одним: «елементи, що менші за опорний», «рівні опорному» і «більші за опорний».

Для відрізків «менших» і «більших» значень, якщо довжина такого відрізка більша за одиницю, слід рекурсивно виконати ту саму послідовність операцій.

На практиці масив зазвичай ділять не на три, а на дві частини: наприклад, «менші за опорний» і «рівні або більші за опорний». Такий підхід у загальному випадку ефективніший, тому що спрощує алгоритм поділу.

Алгоритм має як переваги, так і певні недоліки, тому існує значна кількість покращень цього алгоритму, що усувають дані недоліки.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування обмінами	Найгірша $O(n^2)$ Середня $O(n \cdot \log n)$ Найкраща $O(n)$	Загальна $O(n)$ Додаткова $O(1)$

2.1. **Сортування вставками** (Insertion sort)

Є одним з базових алгоритмів сортування. В цілому працює досить повільно, проте дуже ефективно сортує майже впорядковані масиви. Також часто використовується як навчальний алгоритм.

Алгоритм можна описати наступним чином. У сортуванні вставками послідовно обробляються відрізки масиву, починаючи з першого елемента і потім послідовно розширюючи підмасив, вставляючи своє місце черговий невідсортований елемент. Для вставки використовується буферна область пам'яті, в якій зберігається елемент, що ще не вставлений на своє місце (так званий ключовий елемент). У підмасиві, починаючи з кінця відрізка, перебираються елементи, які порівнюються із ключовим. Якщо ці елементи більші за ключовий, то вони зсовуються на одну позицію вправо, звільняючи місце для наступних елементів. Якщо в результаті цього перебору трапляється елемент, менший або рівний ключовому, то в поточну вільну комірку можна вставити ключовий елемент.

Найбільш гарно дане сортування показує себе на майже відсортованих масивах. У цьому випадку алгоритм показує найкращу часову складність порівняно з різними більш ефективними алгоритмами сортування, на кшталт швидкого сортування або сортування злиттям. Найгірше алгоритм поводить себе на реверсно-впорядкованих масивах. Усунути проблему можна, використовуючи покращений варіант сортування вставками – сортування Шелла (Shellsort).

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування вставками	Найгірша $O(n^2/2)$ Середня $O(n^2/4)$ Найкраща $O(n)$	Загальна $O(n)$ Додаткова $O(1)$

2.2. **Сортування Шелла** (Shell sort).

Сортування Шелла приблизно так само одержується із сортування вставками, як сортування бульбашкою трансформується в сортування гребінцем.

Алгоритм можна описати наступним чином. Слід відсортувати вставкою підгрупи елементів, але у підгрупі вони йдуть не один за одним, а поступово вибираються з деякою відстанню за індексом. Після початкових проходів,

дельта зменшується, поки відстань між елементами цих незв'язних підмножин не досягне одиниці.

Завдяки початковим проходам з великим кроком більшість малих за значенням елементів перекидаються в одну частину масиву, а більшість великих елементів масиву потрапляють в іншу.

Як відомо, метод вставки дуже ефективно обробляє майже відсортовані масиви. Сортування Шелла при початкових проходах відбувається досить швидко, в результаті чого масив приходить до стану неповної впорядкованості. На завершальному етапі крок дорівнює одиниці, тобто алгоритм трансформується в алгоритм сортування простими вставками.

На відміну від сортування гребінцем, для цього алгоритму невідомий оптимальний набір відстаней. Існує досить багато послідовностей із різними оцінками:

- послідовність Шелла – на першому кроці відстань дорівнює половині довжини масиву, кожен наступний вдвічі менший за попередній. Такий варіант давав швидкість $O(n^2)$.
- послідовність Хіббарда – відстань дорівнює $2^k - 1$, часова швидкість в гіршому випадку – $O(n^{1,5})$,
- послідовність Пратта – відстань розглядається як ступені двійки та трійки, часова швидкість в гіршому випадку – $O(n \cdot log^2 n)$;
- ряд Марсін Сіурою: 701, 301, 132, 57, 23, 10, 4, 1.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування вставками	Найгірша – залежить від кроку Середня – залежить від кроку Найкраща $O(n)$	Загальна $O(n)$ Додаткова $O(1)$

3.1. **Сортування вибором** (Selection sort).

Базовий алгоритм для сортування вибором. Працює повільно, але деякі модифікації (наприклад, пірамідальне сортування) можуть показувати дуже високу ефективність. Часто використовується як навчальний алгоритм.

Алгоритм можна описати наступним чином. Проходимо масив і знаходимо у ньому максимальний елемент. Знайдений максимум міняємо місцями із останнім елементом. Потім проходимо по невідсортованій частині масиву (від першого елемента до передостаннього) і знаходимо в ній максимум, який міняємо місцями з передостаннім елементом масиву. Продовжуємо дії, доки не відсортуємо повністю.

Якщо крім переміщення в кінець масиву максимальних елементів також переміщувати на початок масиву мінімальні елементи, то можна отримати **двостороннє сортування вибором.**

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування вибором	Найгірша $O(n^2/2)$ Середня $O(n^2/2)$ Найкраща $O(n/2)$	Загальна $O(n)$ Додаткова $O(1)$

3.2. **Сортування купою** або **Пірамідальне сортування** (Heap sort)

Сортування пірамідою використовує бінарне сортувальне дерево.

Основна ідея алгоритму – знайти максимальний елемент у невідсортованій частині масиву та поставити його в кінець цього підмасиву. У пошуках максимуму підмасив перебудовується в так зване бінарне сортувальне дерево (воно ж двійкова купа, воно ж піраміда), в результаті чого максимум сам "спливає" на початок масиву. Після цього його слід перемістити на кінець підмасиву. Далі, над частиною масиву, що залишилася, знову здійснюється процедура перебудови в сортувальне дерево з подальшим переміщенням максимуму в кінець підмасиву.

Сортувальне дерево — це таке дерево, у якого виконані умови:

- Кожен лист має глибину або d, або d − 1, де d — максимальна глибина дерева;
- Значення в будь-якій вершині не менші (не більші) за значення їх нащадків.

Алгоритм не має сприятливих і вироджених випадків за складністю у часі. При будь-якому вхідному масиві, навіть якщо він майже відсортований, сортування має одну й ту саму часову складність - *O(n log n)*.

Сортування купою використовується в низці інших алгоритмів сортувань (Тернарне пірамідальне сортування, Сортування декартовим деревом, Плавне сортування), а також в гібридних алгоритмах (J-сортування, Інтроспективне сортування).

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування вибором	Найгірша *O(n · log n)* Середня *O(n · log n)* Найкраща *O(n · log n)*	Загальна *O(n)* Додаткова *O(1)*

4.1. **Блочне сортування** або **Сортування комірками** (Bucket sort).

Основна ідея алгоритму полягає в тому, щоб елементи, що мають бути відсортовані, розподілити між скінченним числом окремих блоків (кошиків, комірок) за певною ознакою так, щоб усі елементи в кожному наступному блоці були завжди більші (або менші), ніж у попередньому. Кожен блок потім сортується окремо за уточненою ознакою або рекурсивно тим же самим методом, або якимось іншим алгоритмом, який може бути більш ефективним, відповідно до уточненої ознаки. Зрештою, всі елементи з блоків переміщуються назад у масив.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування розподілом	Найгірша $O(n^2)$ Середня *O(n +k)*, де k – кількість блоків Найкраща *O(n)*	Найгірша *O(n · k)*

4.2. **Флеш-сортування** (FlashSort).

Даний алгоритм 1998 року представив німецький учений-фізик Карл-Дітріх Нойберт. Аналізуючи результати експериментів, Нойберт дійшов висновку, що великі масиви статистичних даних логічно сортувати, звернувшись до теорії ймовірностей.

Загальна сутність алгоритму полягає в наступному: щоб відсортувати масив можна кожен елемент переставити приблизно на своє місце, яке він має займати у відповідності з найімовірнішим випадком. Сформувавши досить швидко майже відсортований масив, можна потім упорядковати його за допомогою іншого алгоритму, наприклад вставками.

Таким чином, алгоритм флеш-сортування складатиметься з наступних етапів:

1. Знайти в масиві мінімум та максимум. За допомогою них надалі наближено визначається в якій частині масиву повинен знаходитися кожен елемент.
2. Створюється допоміжна таблиця розподілу елементів. Всі елементи, в залежності від їхньої величини, розбиваються на класи і в цій додатковій таблиці вказується скільки елементів належить тому або іншому класу. Якщо розподілити елементи масиву на m класів, то номер класу для елемента A_i визначається за формулою: $K(A_i) = 1 + \left[(m-1)\frac{A_i - A_{min}}{A_{max} - A_{min}}\right]$, де K – місце знаходження елементу в таблиці розподілу.

   Емпіричним шляхом встановлено, що оптимальна кількість класів m для масиву з елементів n розраховується за формулою $m \approx 0.42\ n$.
3. Використовуючи таблицю розподілу, елементи розподіляються приблизно за своїми місцями в масиві.
4. Майже впорядкований на попередньому кроці масив досортовуємо методом простих вставок.

Алгоритм дуже ефективний на великих масивах з рівномірно розподіленими даними. Проте на коротких масивах, на довгих масивах з

даними, що нерівномірно розподілені або на масивах з не до кінця впорядкованим даними алгоритм є дуже неефективним.

Клас алгоритму	Складність за часом	Просторова складність
Сортування розподілом	Найгірша $O(n^2)$ Середня $O(n+m)$ Найкраща $O(n)$	Загальна $O(n+m)$ Додаткова $O(m)$

4.3. **Сортування за розрядами** (Radix Sort)

Алгоритм застосовується для впорядкування елементів, що являють собою ланцюжки будь-якого скінченного алфавіту, як правило цілі числа або рядки.

Порівняння відбувається порозрядно: спочатку порівнюються значення одного крайнього розряду і за результатами цього порівняння групуються всі елементи з однаковим значенням, потім порівнюються значення сусіднього наступного розряду і елементи, всередині утворених на попередньому проході груп, або впорядковуються за результатами порівняння значень цього розряду, або переупорядковуються. Потім ті ж самі дії повторюються для наступного розряду і так доки не будуть переглянуті всі розряди.

На практиці існує два варіанти цього сортування:

- LSD (least significant digit) – від молодшого розряду до старшого. Використовується, як правило, для цілих чисел.

  Наприклад: 1, 3, 8, 10, 31, 100, 300, 301, 303, 310. Тобто, тут значення спочатку сортуються за одиницями, потім сортуються за десятками, зберігаючи відсортованість за одиницями всередині десятків, потім за сотнями, зберігаючи відсортованість за десятками.
- MSD (most significant digit) – від старшого розряду до молодшого. Використовується для рядків, щоб відсортувати їх в алфавітному порядку.

При сортуванні MSD виходить алфавітний порядок, який доречний для сортування рядків тексту. Наприклад, "b, c, d, e, ba, bb, bbc, da" відсортується як "b, ba, bb, bbc, c, d, da, e".

Клас алгоритму	Складність за часом	Просторова складність
Сортування розподілом	$O(n \cdot \omega)$, де ω – довжина елементів	$O(n + \omega)$

4.4. **Сортування підрахунком** (Counting sort)

Алгоритм сортування цілих чисел у діапазоні від 0 до деякої константи *k*, для випадків, коли числа, що мають бути відсортовані, потрапляють у діапазон можливих значень, який достатньо малий в порівнянні з загальною множиною чисел (наприклад, коли необхідно відсортувати кілька мільйонів чисел, які всі знаходяться в діапазоні від 0 до 1000).

Клас алгоритму	Складність за часом	Просторова складність
Сортування розподілом	$O(n + k)$	$O(n + k)$

5.1. **Сортування злиттям** (Merge sort).

Ефективний рекурсивний алгоритм сортування, запропонований Джоном фон Нейманом у 1945 році. Алгоритм був створений під час роботи над «Манхеттенським проектом» як засіб обробки великих масивів статистичних даних. В основі алгоритму лежить принцип «Розділяй та володарюй».

Суть алгоритму полягає в наступному. Необхідно рекурсивно поділити масив на дві половини, відсортувати кожну половину, а потім злити відсортовані половини в один відсортований масив.

Фактично, алгоритм складається з трьох частин:

- ***Поділ***. Масив розбивається на два підмасиви. Для кожного підмасиву слід повторити процес, доки кожен підмасив не стане масивом з одного або двох елементів.
- ***Упорядкування***. Підмасиви сортуються (як варіант, до них рекурсивно застосовується сортування злиттям).

- ***Злиття***. Необхідно послідовно об’єднати підмасиви, щоб створити новий впорядкований масив. Для цього на кожному кроці слід взяти менший (більший) з двох перших елементів підсписків і записати його в результуючий масив. Лічильники номерів елементів результуючого масиву і підмасиву, з якого було взято елемент, збільшити на 1. Коли один з підмасивів вичерпався, додати всі елементи, що залишилися з другого підмасиву в результуючий масив.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Сортування злиттям	Найгірша $O(n \cdot \log n)$ Середня $O(n \cdot \log n)$ Найкраща $O(n \cdot \log n)$	Загальна $O(n)$ Додаткова $O(n)$

6.1. **Спляче сортування** (Sleep sort).

Сортування застосовується лише до натуральних чисел і реалізується мовами програмування, що підтримують багатопоточність.

Для кожного елемента в пам’яті створюється окремий процес, який «спить» кількість секунд (мілісекунд, мікросекунд), що дорівнює значенню відповідного елемента. Як тільки процес «просинається» відповідне значення заноситься у відсортований масив.

При цьому важливо врахувати, що частота системного таймера має бути вище, ніж крок між значеннями, що сортуються. Крім того, якщо максимальне значення буде надто велике, то процес сортування може зайняти дуже великий час (наприклад, якщо елемент вхідного масиву містить лише 2 елементи - 1 та 100000000, то доведеться чекати набагато довше, ніж при будь-якому іншому алгоритмі сортування). Також цей алгоритм не працюватиме для від’ємних чисел, оскільки потік не може спати протягом від’ємного часу.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Паралельне сортування	$O(n)$	$O(n)$

6.2. **Спагетті-сортування** (Spaghetti sort).

Аналоговий паралельний алгоритм, який сортує набір додатніх чисел за лінійний час.

Ідея алгоритму аналогічна до ідеї алгоритму сплячого сортування.

Кожне число може бути представлене у вигляді стрижня сухого спагетті відповідної довжини. Якщо встановити спагетті вертикально на рівній поверхні та зверху на спагетті опускати прес, то щоразу, коли прес буде ламати черговий стрижень, фіксується черговий відсортований елемент.

Недоліки алгоритму майже такі самі, що й у алгоритму сплячого сортування.

Клас алгоритму	**Складність за часом**	**Просторова складність**
Паралельне сортування	*O(n)*	*O(n)*

РОЗДІЛ 2. СТВОРЕННЯ ПРОТОТИПУ СИСТЕМИ ВІЗУАЛІЗАЦІЇ СОРТУВАННЯ ДАНИХ

2.1. Огляд існуючих систем візуалізації алгоритмів сортування

У процесі підготовки та реалізації системи візуалізації алгоритмів сортування доцільно проаналізувати вже існуючі програмні рішення, що реалізують аналогічний функціонал. Це дозволить визначити найкращі практики та підходи, виявити недоліки, які можна врахувати при власній розробці. Нижче наведено огляд найбільш відомих та поширених систем, які використовуються у навчанні, демонстрації та дослідженні алгоритмів сортування.

1. **Visualgo** (visualgo.net) — один із найпопулярніших онлайн-ресурсів для інтерактивної візуалізації алгоритмів і структур даних. Сервіс був розроблений на факультеті комп'ютерних наук Національного університету Сінгапуру. Він підтримує велику кількість алгоритмів: сортування, пошук, обхід графів, роботу з деревами, хеш-таблицями тощо.

Серед переваг слід зазначити наступні:

- ресурс працює у веб-браузері, і не потребує інсталяції на комп'ютер;
- підтримує кілька алгоритмів сортування (Bubble, Insertion, Merge, Quick тощо);
- має функцію покрокової візуалізації з поясненнями;
- доступний багатьма мовами.

Недоліками є:

- відсутність можливості змінювати логіку виконання;
- неможливість переналаштування інтерфейсу.

Інтерфейс програми показаний на рис.2.1.

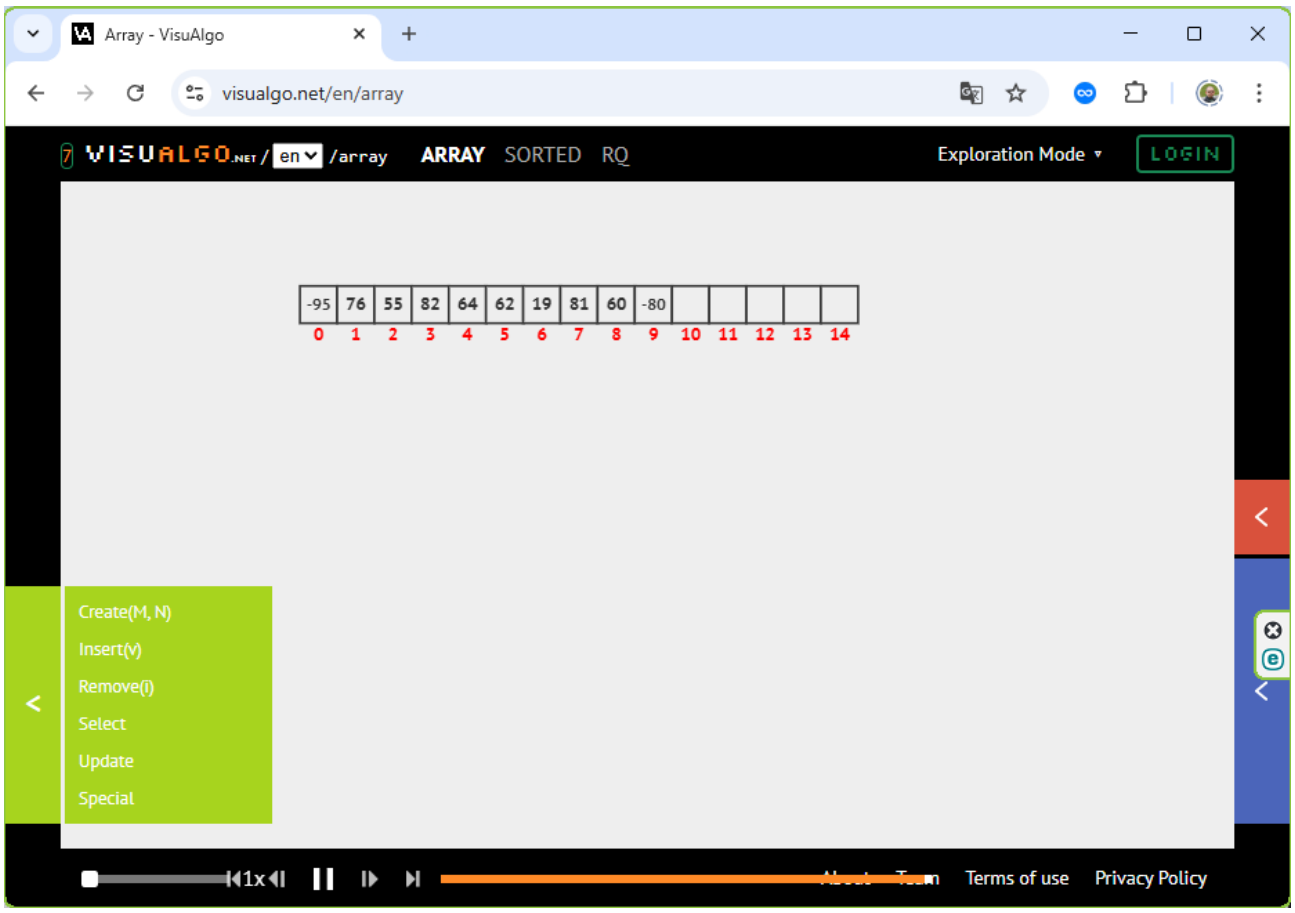

Рис.2.1. Інтерфейс сервісу Visualgo

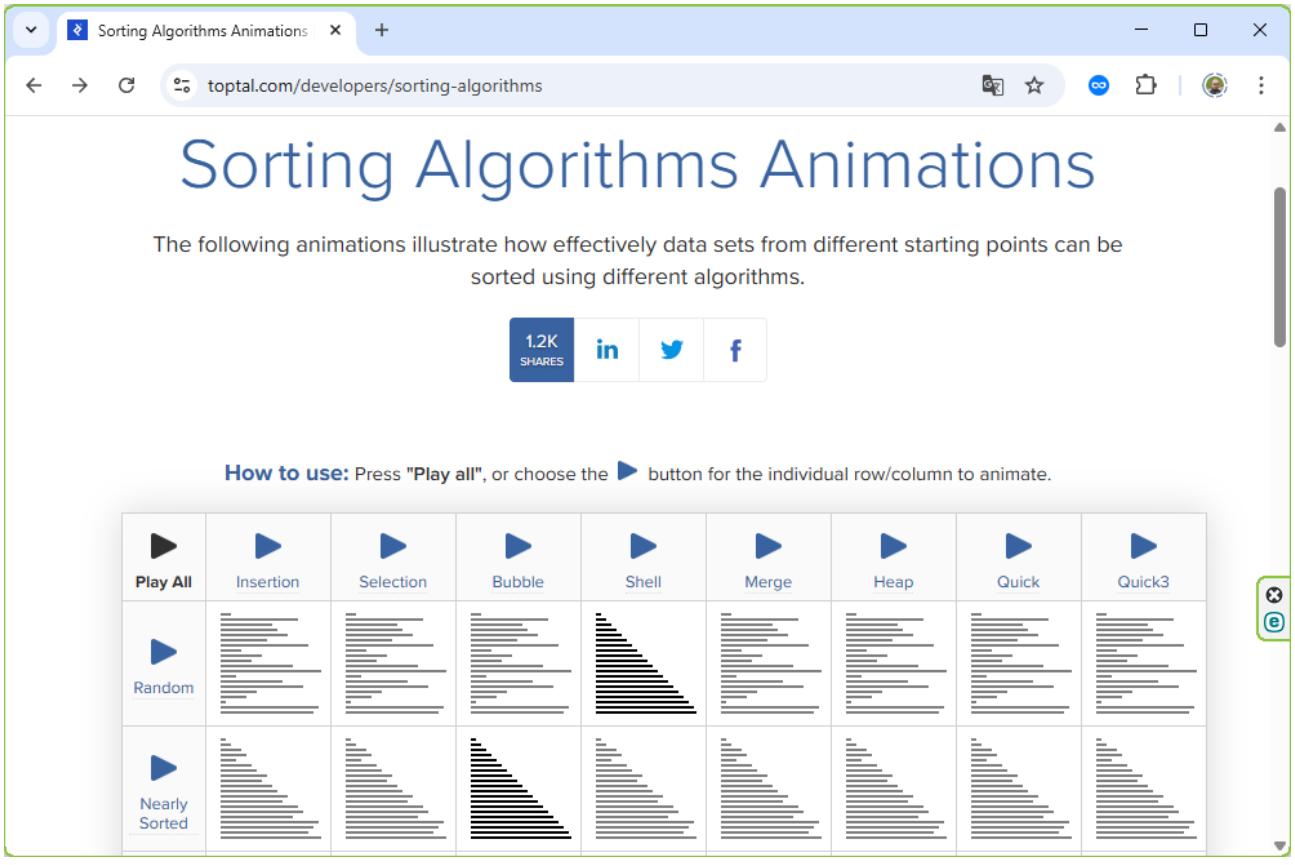

Рис.2.2. Інтерфейс сервісу Sorting Algorithms Animations

2. **Sorting Algorithms Animations** (toptal.com/developers/sorting-algorithms)

Це простий онлайн-сервіс з демонстрацією візуального порівняння швидкодії популярних алгоритмів сортування в реальному часі. Як і попередній сервіс, дозволяє звертатись через веб-браузер і не потребує інсталяції на комп'ютер (рис.2.2).

Переваги сервісу:

- легка, зрозуміла анімація;
- можливість запускати всі алгоритми паралельно для порівняння їхньої швидкості дії;

Недоліки сервісу:

- відсутність інтерфейсу користувача для зміни параметрів.
- мінімальні можливості взаємодії;
- не має можливості модифікації вхідних даних.

3. **Algorithm Visualizer** (algorithm-visualizer.org)

Один із найвідоміших інструментів для візуалізації алгоритмів із відкритим кодом. Сервіс написаний з використанням JavaScript та дозволяє користувачам розробляти на JavaScript власні алгоритми і бачити результати їх виконання.

До беззаперечних переваг сервісу можна віднести:

- анімацію сортування, пошуку, роботи зі структурами даних;
- інтерактивність графіки та синхронізованість коду;
- легко модифікований вихідний код, який доступний на GitHub.

Незначними недоліками можна вважати:

- певну складність для новачків;
- обмежену гнучкість інтерфейсу (відкритий код дозволяє його розширити).

Інтерфейс сервісу показаний на рис.2.3.

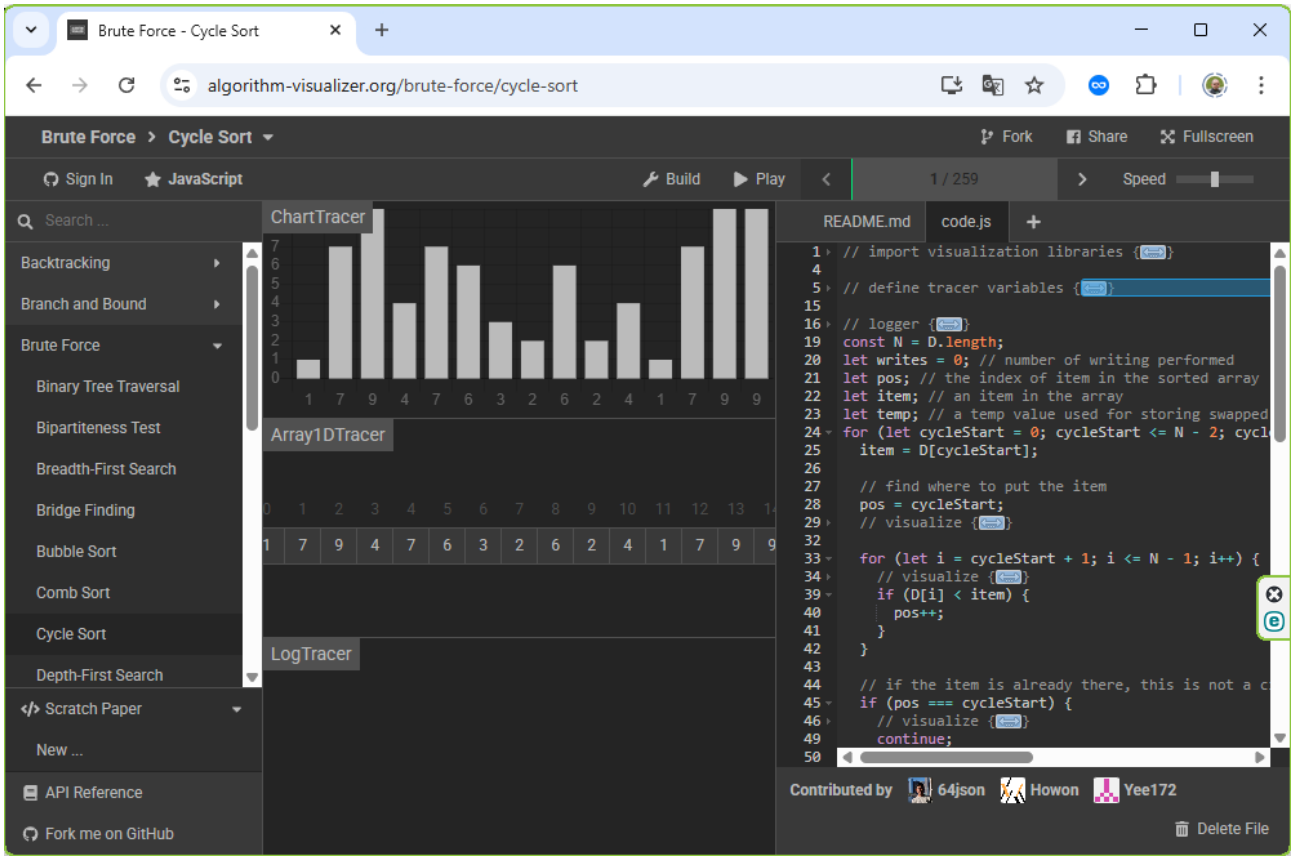

Рис.2.3. Інтерфейс сервісу Algorithm Visualizer

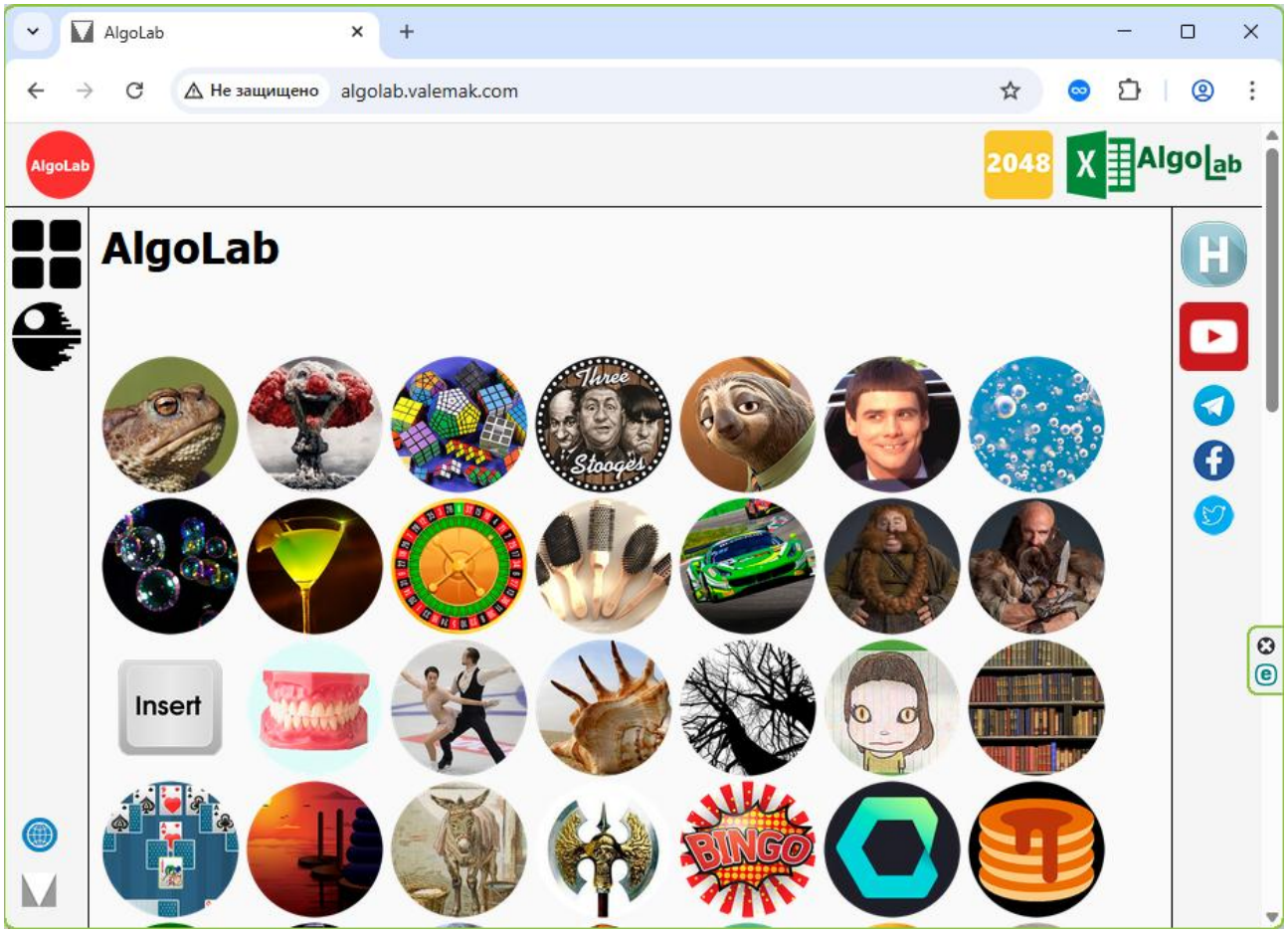

Рис.2.4. Інтерфейс сервісу AlgoLab

4. **AlgoLab** (algolab.valemak.com) – це онлайн-колекція алгоритмів, зокрема методів сортування, створена кіровоградським програмістом Валерієм Макаровим. Сервіс містить пояснення, псевдокоди і короткі реалізації численних класичних алгоритмів серед яких сортування бульбашкою, швидке сортування та інші. Алгоритми згруповані за класом, запропоновано короткий опис механізму роботи, метрики ефективності, приклади або покрокова демонстрація в текстовому форматі.

Переваги:

- доступність через веб-браузер;
- широкий перелік алгоритмів;
- текстові пояснення та псевдокоди алгоритмів;
- наявність анімованих вставок, що ілюструють роботу алгоритмів.

Недоліки AlgoLab:

- відсутність інтерактивності (неможливість змінити вхідні дані);
- сайт не оновлювався останнім часом та має мінімальний розвиток.

Інтерфейс сервісу показаний на рис.2.4.

5. **The Sound of Sorting** (panthema.net/2013/sound-of-sorting) – це відкритий крос‑платформовий додаток, що поєднує візуалізацію та «озвучування» алгоритмів сортування. Програма пропонує багатий набір алгоритмів для візуалізації: Bubble, Insertion, QuickSort, Merge, Heap тощо.

«Фішкою» програми є те, що під час роботи реалізується пооперaційна «озвучка», коли кожне порівняння викликає звук певної висоти, що залежить від значень елементів.

Переваги:

- унікальна аудіовізуалізація;
- широкий вибір алгоритмів;
- крос‑платформенність;
- гнучкість управління.

Недоліки:

- відсутність детальних пояснень для кожного алгоритму
- відсутність підтримки нових алгоритмів.

Інтерфейс програми показаний на рис.2.5.

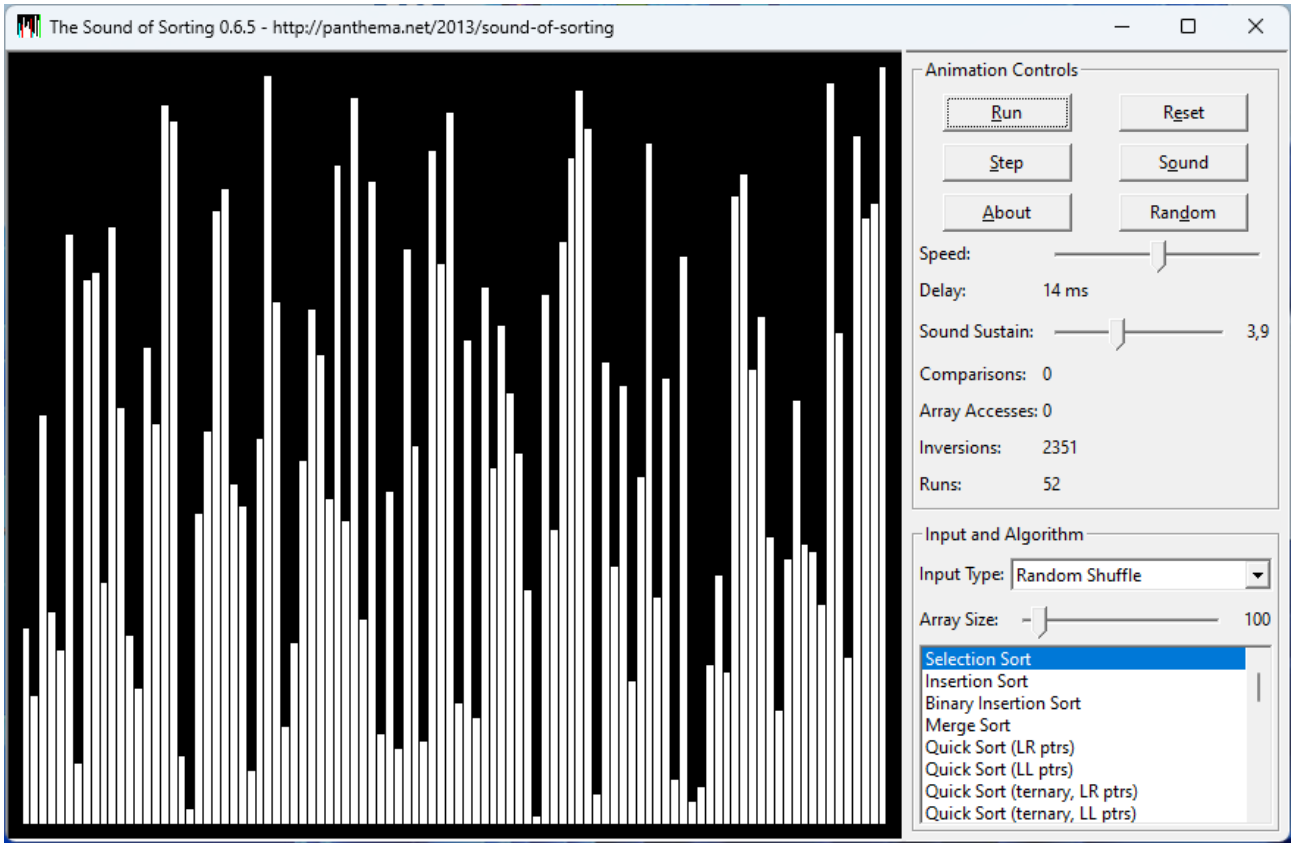

Рис.2.5. Інтерфейс програми The Sound of Sorting

Проаналізувавши деякі з сервісів для візуалізації алгоритмів сортування, можна зробити наступні висновки щодо основних вимог до програмного продукту, призначеного для візуалізації алгоритмів сортування:

- програма повинна відображати процес сортування масиву у вигляді графічної анімації (наприклад, вертикальних стовпчиків або кружечків);
- для кожного кроку повинні бути візуально позначені операції порівняння, обміну, копіювання або вставки;
- система має реалізовувати та демонструвати кілька різних алгоритмів;

- інтерфейс повинен бути інтуїтивно зрозумілим, з підписами до кнопок, поясненнями або підказками;
- користувач повинен мати змогу обрати потрібний алгоритм із випадаючого списку або іншого зручного елемента керування;
- архітектура програми має дозволяти додавати нові алгоритми або варіації сортування.

2.2. Вибір середовища розробки

Розробка програмного забезпечення вимагає не лише глибокого розуміння предметної області, але й вибору інструменту, який забезпечить ефективну реалізацію і візуалізацію алгоритмічних процесів.

Для вибору середовища розробки необхідно було врахувати кілька особливостей:

- просте створення графічного інтерфейсу;
- висока продуктивність;
- гнучкість у роботі з графікою та анімацією;
- зручне налагодження, тестування і підтримка коду;
- сучасний інструментарій і підтримка нових стандартів
- простий і зрозумілий синтаксис мови

З огляду на вказані особливості для створення системи було обрано середовище розробки Delphi 12 Embarcadero (рис.2.6).

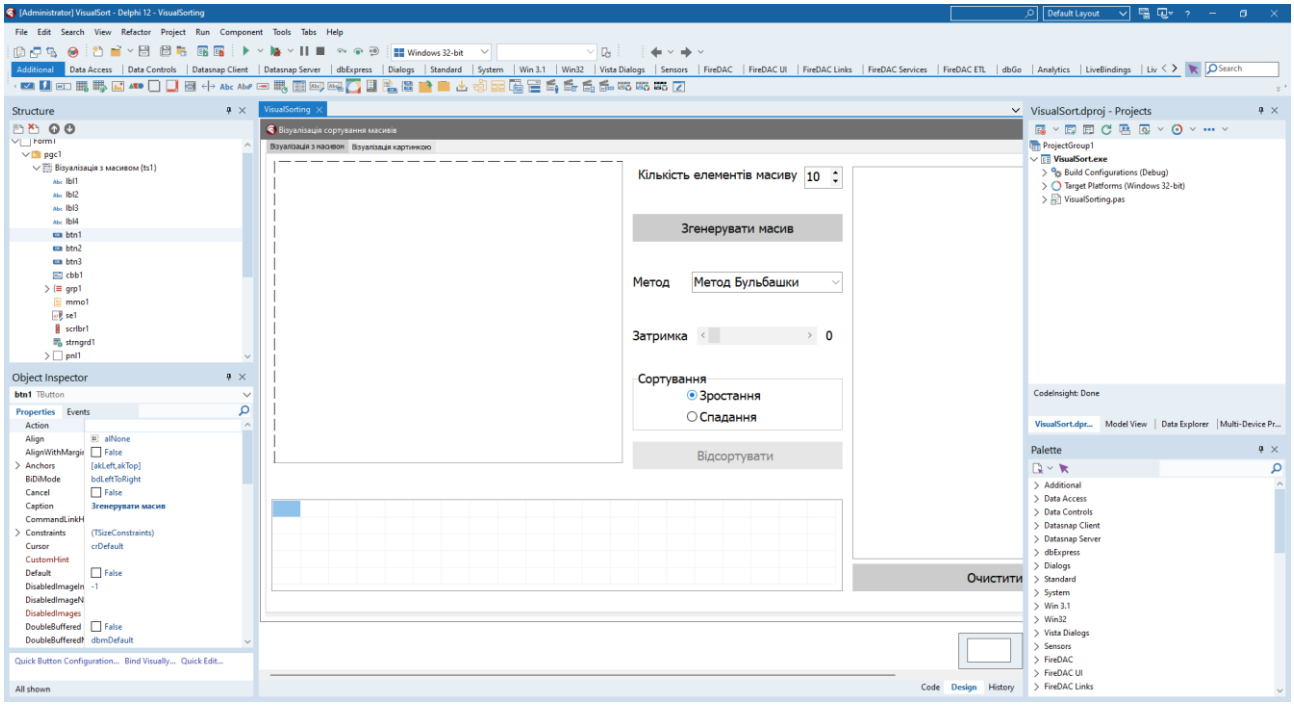

Рис. 2.6. Середовище програмування Delphi 12 Embarcadero

Серед сучасних мов і середовищ програмування посідає важливе місце як потужне та зручне рішення для побудови додатків із графічним інтерфейсом. Delphi 12 надає сучасний зрозумілий, наочний та динамічний інтерфейс візуального проєктування, що дозволяє розробнику без додаткового коду

створювати форми, управляючі елементи (кнопки, панелі, поля, елементи графіки), забезпечуючи швидку реалізацію функціональності. Це особливо важливо на етапах тестування та подальшої модифікації системи.

Основою створення системи є мова програмування Object Pascal, яка поєднує строгість типів і об’єктно-орієнтовану структуру з читабельним, зрозумілим синтаксисом. Завдяки цьому програмний код легко підтримувати, тестувати та розширювати. Це особливо корисно у контексті науково-дослідницької або навчальної розробки, де прозорість реалізації є критично важливою.

Середовище Delphi дає змогу компілювати програми безпосередньо в машинний код для архітектур Win32/Win64. Це забезпечує високу швидкодію та стабільність виконання навіть на комп’ютерах із середніми характеристиками. При реалізації візуалізації алгоритмів сортування швидкість виконання має критичне значення. Продуктивність Delphi дозволяє швидко обробляти великі обсяги даних, перемальовувати графічні компоненти, запускати анімацію та синхронізувати події без затримок.

Також Delphi має розвинуту підтримку графічних операцій через класи TCanvas, TImage, TBitmap, а також бібліотеки для роботи з GIF-анімацією, векторними та растровими форматами. Це дозволяє реалізовувати повноцінну візуалізацію: змінювати колір і розмір елементів масиву, імітувати переміщення, підсвічування, злиття, обмін тощо. Окрім того, у Delphi можна реалізовувати покрокове відображення виконання алгоритмів (через таймери, обробники подій), що є важливим у навчальному контексті.

Крім того, середовище розробки Delphi містить інструменти для поетапного налагодження виконання програми, що дозволяє виявляти та виправляти помилки у логіці алгоритму або його візуалізації. У процесі розробки системи сортування розробник може зручно контролювати значення зміних, логіку перестановок і синхронізувати графіку з алгоритмічними подіями, що позитивно впливає на якість реалізації.

Таким чином, вибір Delphi 12 Embarcadero як середовища реалізації системи візуалізації алгоритмів сортування є обґрунтованим як з точки зору технічної доцільності, так і з позицій зручності розробки, підтримки та подальшого вдосконалення продукту. Стабільність, продуктивність, зручність у створенні інтерфейсу та підтримка графічних можливостей роблять Delphi 12 ефективним інструментом для реалізації програмного забезпечення, призначеного для вивчення та демонстрації алгоритмів.

2.3. Опис роботи розробленої системи

Як уже вказувалося в розділі 2.1, серед вимог до програмного продукту, призначеного для візуалізації алгоритмів сортування, слід особливо виділити наступні:

- програма повинна відображати процес сортування масиву у вигляді графічної анімації (наприклад, вертикальних стовпчиків або кружечків);
- для кожного кроку повинні бути візуально позначені операції порівняння, обміну, копіювання або вставки;
- система має реалізовувати та демонструвати кілька різних алгоритмів;
- інтерфейс повинен бути інтуїтивно зрозумілим, з підписами до кнопок, поясненнями або підказками;
- користувач повинен мати змогу обрати потрібний алгоритм із випадаючого списку або іншого зручного елемента керування;
- архітектура програми має дозволяти додавати нові алгоритми або варіації сортування.

Виходячи з цього, було прийнято рішення при реалізації системи піти двома шляхами, реалізувавши їх у вигляді окремих модулів:

1. Модуль, що містить обмежену кількість алгоритмів, з покроковою візуалізаціїєю їх реалізації, що запрограмовані на рівні програмного коду. Користувач має вплив на елементи візуалізації: зміна кількості елементів масиву, швидкості відображення тощо.
2. Модуль, що дозволяє користувачу самостійно вносити в систему елементи візуалізації у вигляді файлів gif-анімації. Однак при цьому користувач обмежений у впливі на дану візуалізацію.

Обидва модулі реалізовано у вигляді окремих вкладок в програмі: «Візуалізація з масивом» (рис. 2.7) та «Візуалізація картинкою» (рис. 2.8).

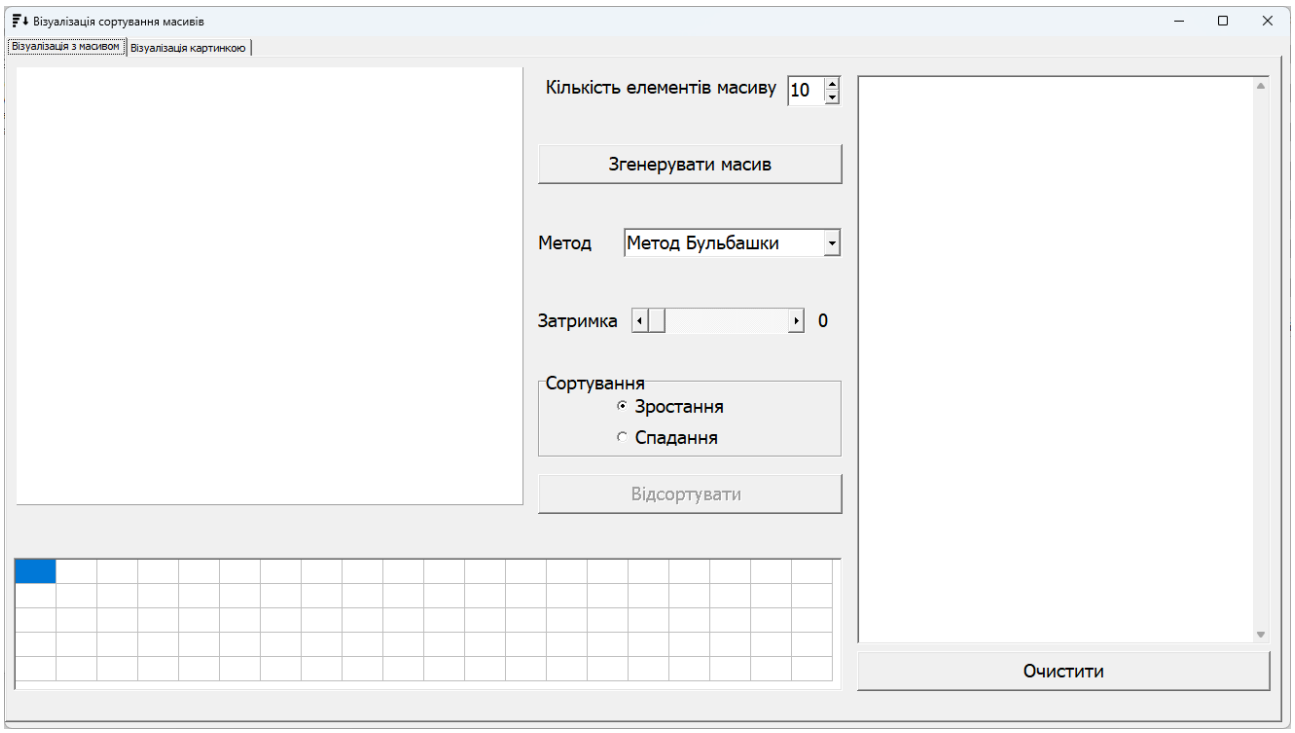

Рис. 2.7. Загальний вигляд програми на вкладці «Візуалізація з масивом»

На екрані закладки «Візуалізація з масивом» наявні наступні елементи управління:

- компонента TImage, в якому відбувається анімація сортування масиву;
- компонента TSpinEdit, за допомогою якого задається кількість елементів масиву від 10 до 100;
- компонента TComboBox, через який відбувається вибір алгоритму для візуалізації;
- компонента TScrollBar, за допомогою якого регулюється швидкість візуалізації;
- дві компоненти TRadioButton, що дозволяєть переключатися між типами сортування (за зростанням чи за спаданням);
- компонента TStringGrid, в якій відображаються елементи масиву;
- компонента TMemo, в якій виводиться послідовність дій при реалізації алгоритму;
- три кнопки – компоненти TButton:

- «Згенерувати масив» – створює масив випадкових чисел заданої кількості;
- «Очистити» – очищає текст послідовності дій при реалізації алгоритму;
- «Відсортувати» – запускає процедуру сортування.

Треба відмітити, що деякі компоненти інтерфейсу можуть бути заблоковані, якщо не виконані умови для їхньої доступності. Так кнопка «Відсортувати» не буде активною, доки не буде згенеровано новий числовий масив.

Користувач системи має виконати наступні дії:

- вказати кількість елементів масиву, який буде згенеровано для подальшої візуалізації алгоритму сортування;
- згенерувати масив випадкових чисел (в діапазоні від 1 до 100), натиснувши кнопку «Згенерувати масив»;
- обрати метод сортування (наразі доступні три методи – Метод бульбашки, Сортування вибором, Швидке сортування);
- встановити рівень затримки часу (в діапазоні від 0 до 10), який означає як швидко буде відбуватися перехід до наступного кроку алгоритму;
- обрати вид сортування – за зростанням чи за спаданням;
- запустити візуалізацію, натиснувши кнопку «Відсортувати».

Під час виконання сортування в програмі (рис 2.8):

- в компоненті TStringGrid відображається поточний стан масиву;
- у компоненті TImage відображається візуалізація масиву, де кожен стовпчик відповідає деякому елемента масиву, а його висота – відповідному числовому значенню;
- у компоненті TMemo виводиться текстовий опис дії на поточному кроці.

Під час виконання сортування користувач може змінювати затримку, прискорюючи або сповільнюючи візуалізацію.

Фрагмент програми, що реалізує візуалізацію алгоритму сортування бульбашкою, наведено в додатку А.

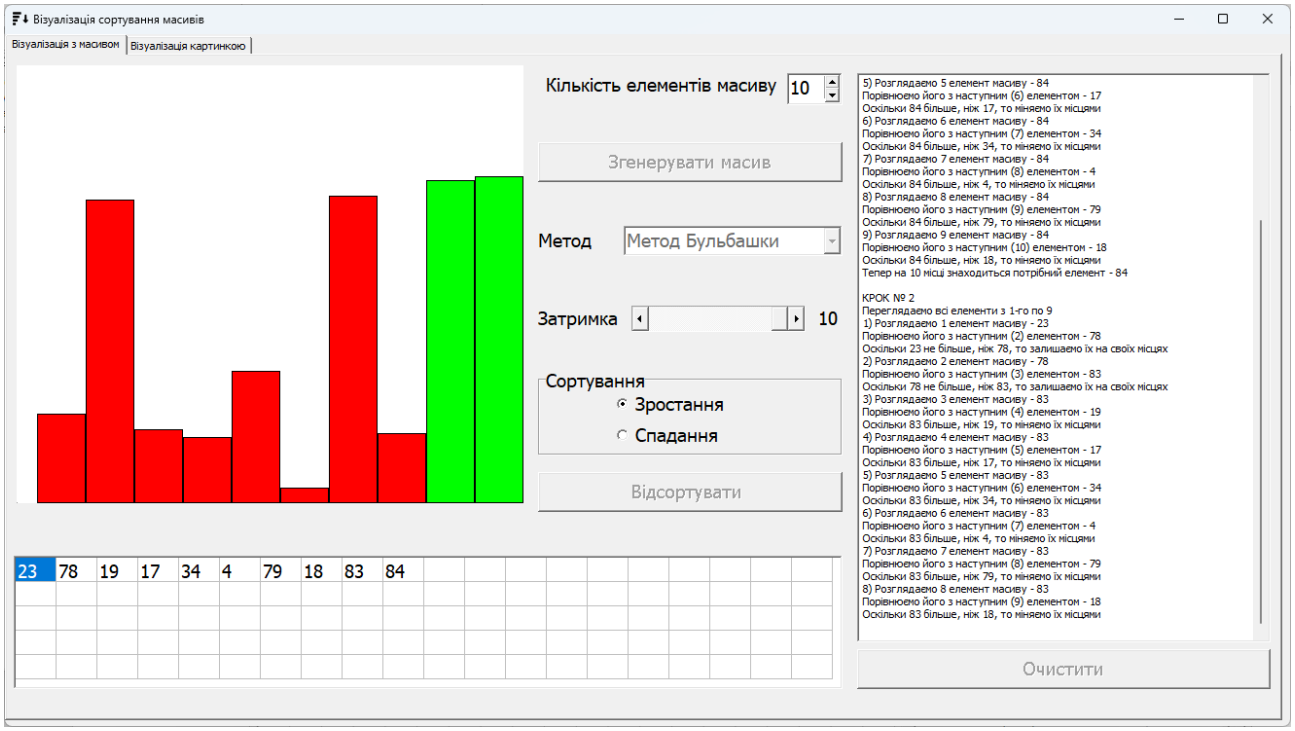

Рис. 2.8. Виконання візуалізації в програмі

На екрані закладки «Візуалізація з картинкою» наявні наступні елементи управління:

- компонента TImage, в якій відбувається відображення анімації сортування масиву;
- компонента TComboBox, через який відбувається вибір алгоритму для візуалізації;
- компонента TScrollBar, за допомогою якого регулюється швидкість візуалізації;
- три кнопки – компоненти TButton:
  - «Наступне», «Попереднє» – які викликають наступну візуалізацію або повертаються до попередньої;
  - «Запустити сортування» – запускає анімацію сортування.

Користувач системи має виконати наступні дії:

- обрати метод сортування із списку, що випадає;

- обрати файл з візуалізацією, натискаючи на кнопки «Наступне», «Попереднє»
- «запустити візуалізацію, натиснувши кнопку «Запустити сортування» (рис. 2.10).

Рис. 2.9. Загальний вигляд програми на вкладці «Візуалізація картинкою»

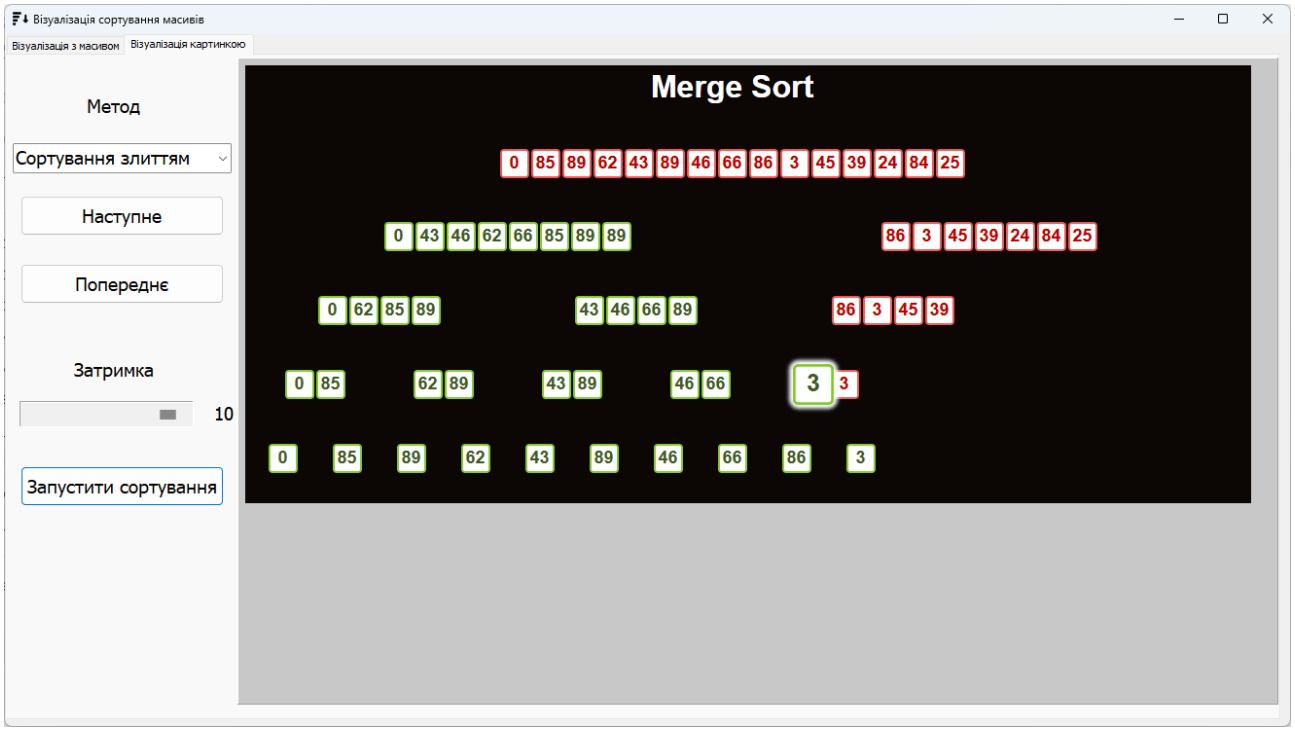

Рис. 2.10. Виконання візуалізації на вкладці «Візуалізація картинкою»

Візуалізація в цьому модулі реалізована з використанням файлів gif-анімації. В основному каталозі з програмою наявний підкаталог GIF, в який користувач має скопіювати gif-файли, що можуть бути використані для візуалізації алгоритмів сортування. Назва файлу має складатись із імені алгоритму (bubble, quick, merge тощо), послідовного номеру, що починається з 0, та розширення **.gif**. В каталозі з програмою також наявний текстовий файл **sort.ini**, в якому визначено кілька блоків властивостей візуалізації в форматі:

- Ім’я алгоритму;
- Шаблон імені файлу;
- кількість файлів візуалізації, що відповідають даному алгоритму.

Текст процедур, що відповідають за завантаження файлів в систему наведено в Додатку Б. Зміст файлу sort.ini, що є складовою дипломної роботи, представлено нижче:

```
Сортування вибором
Selection
4
Швидке сортування
Quick
2
Метод бульбашки
Bubble
4
Шейкерне сортування
Shaker
2
Сортування злиттям
Merge
2
Спагетті-сортування
spagetti
1
```

Відповідно до змісту представленого файлу в програмі наразі реалізовано шість алгоритмів сортування (Сортування вибором, Швидке сортування, Метод бульбашки, Шейкерне сортування, сортування Злиттям, Спагетті-сортування), кожен з яких містить від 1 до 4 візуалізацій у відповідних файлах.

Для розширення кількості візуалізацій користувачу необхідно створити самому або знайти в мережі Інтернет файл з візуалізацією сортування чисел у відповідності із заданим алгоритмом, назвати його іменем, що відповідає такому алгоритму, додавши наступний порядковий номер, якщо такий алгоритм вже описаний в файлі sort.ini, або ж додати відповідні рядки у файл, якщо такого алгоритму ще не було. При наступному запуску програми користувачу буде доступна нова візуалізація.

Висновок

Сортування масивів є однією з базових важливих операцій, що лежить в основі розв’язування великої кількості прикладних задач. Тому алгоритми сортування залишаються предметом постійної уваги з боку дослідників, викладачів і розробників. Зокрема, в освіті ефективним вважається використання візуальних інструментів для навчання та аналізу алгоритмів. Метою дипломної роботи було дослідження та розробка компонентів системи візуалізації алгоритмів сортування.

Під час реалізації теоретичної частини дослідження було проведено ґрунтовний аналіз різних алгоритмів сортування та сфери їх застосування. Розуміння роботи алгоритмів сортування є фундаментальним для формування алгоритмічного мислення, що підтверджує доцільність створення інструментів, які полегшують навчання, демонстрацію й експерименти з цими алгоритмами. Окрему увагу було приділено огляду та класифікації існуючих алгоритмів сортування, було проаналізовано їхню часову та просторову складність, ефективність на практиці та специфіку реалізації. Також було розглянуто історичні аспекти розвитку алгоритмів сортування.

У рамках дослідження також було проаналізовано наявні програмні продукти, призначені для візуалізації алгоритмів сортування такі як VisuAlgo, AlgoLab, The Sound of Sorting та інші. Для кожного інструменту було виявлено їхні переваги та недоліки, узагальнено та сформульовано вимоги до власного інструменту, орієнтованого на автономне використання, відкритість до модифікацій та адаптацію під конкретні навчальні чи дослідницькі завдання.

У якості інструменту реалізації було обрано Delphi 12 Embarcadero, що дало змогу швидко реалізувати зручний графічний інтерфейс та створити відповідний Windows-додаток. Створений прототип системи дозволяє інтерактивно демонструвати роботу таких алгоритмів, як бульбашкове сортування, сортування злиттям, сортування вставками, швидке сортування та інші.

Підсумовуючи вищезазначене, можна стверджувати, що поставлені в дипломній роботі завдання виконано в повному обсязі. Отримані результати можуть бути використані у навчальному процесі для демонстрації принципів роботи алгоритмів сортування, а також як основа подальшого розширення функціональності системи. Надалі можливим є доповнення системи додатковими алгоритмами з відповідною їх візуалізацією, реалізація гнучкого редагування вхідних даних та статистичний аналіз ефективності алгоритмів.

Перелік використаних джерел

1. Алгоритми та структури даних: навчальний посібник [Електронний ресурс] / Коваленко О. О., Ткаченко О. М., Чехместрук Р. Ю. – Вінниця : ВНТУ, 2025. – PDF, 113 с.
2. Алгоритми, дані і структури. [Текст], навч. посіб. / В.М. Ільман, О.П. Іванов, Л.О. Панік. Дніпропет. нац.. ун-т залізн. трансп.ім. акад. В. Лазаряна. – Дніпро, 2019. – 134 с.
3. Безменов М. І. Основи програмування у середовищі Delphi : навч. посібник / М. І. Безменов; Нац. техн. ун-т "Харків. політехн. ін-т". – Харків : НТУ "ХПІ", 2010. – 608 с.
4. Коротєєва Т. О. Алгоритми та структури даних: навч. посібник / Т. О. Коротєєва. – Львів: Львівської політехніки, 2014. – 280 с.
5. Костів О. В., Ярошко С. А. Методи розробки алгоритмів: Тексти лекцій. – Львів: Видавничий центр ЛНУ імені Івана Франка, 2002. – 96 с.
6. Креневич А.П. Алгоритми і структури даних. Підручник. – К.: ВПЦ "Київський Університет", 2021. – 200 с.
7. Кузьменко І.М., Дацюк О.А. Базові алгоритми та структури даних. Навчальний посібник – К: КПІ ім. Ігоря Сікорського, 2022, – 137 с.
8. Яременко, Г. І. Програмування в середовищі Delphi [Електронний ресурс]: навчальний посібник з дисципліни "Програмування та теорія алгоритмів" / Г. І. Яременко, Д. В. Копил; М-во освіти і науки, молоді та спорту України, Черкас. держ. технол. ун-т. – Черкаси: ЧДТУ, 2011. – 187 с.
9. Knuth, D. E. The Art of Computer Programming. Volume 3: Sorting and Searching. — 2nd ed. — Boston: Addison-Wesley, 1998. — 780 p.
10. http://algolab.valemak.com
11. https://algorithm-visualizer.org/
12. https://docwiki.embarcadero.com/RADStudio/Sydney/en/Tutorials
13. https://en.wikipedia.org/wiki/Sorting_algorithm
14. https://learndelphi.org/resources/delphi-programming-for-beginners/

15.https://melander.dk/delphi/gifimage/
16.https://panthema.net/2013/sound-of-sorting/
17.https://uk.wikipedia.org/wiki/Алгоритм_сортування
18.https://visualgo.net/en
19.https://www.toptal.com/developers/sorting-algorithms
20.https://www.youtube.com/channel/UCMmsCQhkz-WlJ-IVBzPhbgA
21.stackoverflow.com/questions/

Додаток А

Процедура візуалізації алгоритму сортування бульбашкою.

```
procedure bubble;
  var i,j,k:Integer;
begin
  ClearPaint;
  PaintMas(a,1,N,$0000FF);
  with Form1 do
    begin
      for i:=1 to n-1 do
       begin
         pb1.Canvas.Brush.Color := $00ff00;
         pb1.Canvas.Rectangle(20,450,h+20,450-a[1]*4);
         mmo1.Lines.Add('КРОК №'+inttostr(i));
         mmo1.Lines.Add('Переглянемо всі елемети з 1-го
по '+inttostr(n-1));
         delay(Form1.scrlbr1.Position*100);
         for j:=1 to n-i do
          begin
            mmo1.Lines.Add(inttostr(j)+')     Розглядаємо
'+inttostr(j)+' елемент масиву - '+inttostr(a[j]));
            mmo1.Lines.Add('Порівнюємо його з наступним
('+inttostr(j+1)+') елементом - '+inttostr(a[j+1]));
            if Form1.rb1.Checked
              then
                begin
                  if a[j]>a[j+1]
                    then
                      begin
                        mmo1.Lines.Add('Оскільки
'+inttostr(a[j])+' більше, ніж '+inttostr(a[j+1])+', то
міняємо їх місцями');
                        Swap(a[j],a[j+1]);
                      end
                    else         mmo1.Lines.Add('Оскільки
'+inttostr(a[j])+' не більше, ніж'+inttostr(a[j+1])+', то
залишаємо їх на своїх місцях')
                end
              else if a[j]<a[j+1]
                     then
                       begin
                         mmo1.Lines.Add('Оскільки
'+inttostr(a[j])+' менше, ніж '+inttostr(a[j+1])+', то
міняємо їх місцями');
```

```
                        Swap(a[j],a[j+1]);
                      end
                     else       mmo1.Lines.Add('Оскільки
'+inttostr(a[j])+' не менше, ніж '+inttostr(a[j+1])+', то
залишаємо їх на своїх місцях');
            PrintMas(a);
            pb1.Canvas.Brush.Color := $FFFFFF;
            pb1.Canvas.fillRect(Rect((j-
1)*h+20,450,(j+1)*h+20,50));
            pb1.Canvas.Brush.Color := $0000FF;
            pb1.Canvas.Rectangle((j-
1)*h+20,450,j*h+20,450-a[j]*4);
            pb1.Canvas.Brush.Color := $00ff00;

pb1.Canvas.Rectangle(j*h+20,450,(j+1)*h+20,450-a[j+1]*4);
            delay(Form1.scrlbr1.Position*100);
          end;
       mmo1.Lines.Add('Тепер  на  '+inttostr(j)+'  місці
знаходиться  потрібний елемент - '+inttostr(a[j]));
       mmo1.Lines.Add('');
       end;
    pb1.Canvas.Brush.Color := $00ff00;
    pb1.Canvas.Rectangle(20,450,h+20,450-a[1]*4);
    mmo1.Lines.Add('Останній     елемент     автоматично
знаходиться на своєму місці');
    mmo1.Lines.Add('Масив відсортовано!');
    end;
end;
```

Додаток Б

Процедури, що відповідають за виклик файлів з gif-анімацією.

```
procedure TForm1.cbb2Change(Sender: TObject);
  var s:string;
begin
  nommet:=cbb2.itemindex;
  s:='gif\'+namefile[nommet]+'0.gif';
  namfil:=s;
  Img1.Picture.LoadFromFile(namfil);
end;

procedure TForm1.Button1Click(Sender: TObject);
  var s:string;
begin
  nomgif:=nomgif-1;
  if nomgif<0 then nomgif:=kilfile[cbb2.itemindex];
  s:='gif\'+namefile[nommet]+inttostr(nomgif)+'.gif';
  namfil:=s;
  Img1.Picture.LoadFromFile(namfil);
end;

procedure TForm1.Button2Click(Sender: TObject);
  var s:string;
begin
  nomgif:=nomgif+1;
  if nomgif>kilfile[cbb2.itemindex] then nomgif:=0;
  s:='gif\'+namefile[nommet]+inttostr(nomgif)+'.gif';
  namfil:=s;
  Img1.Picture.LoadFromFile(namfil);
end;

procedure TForm1.btn4Click(Sender: TObject);
begin
  Img1.Picture.LoadFromFile(namfil);
  (Img1.Picture.Graphic as TGIFImage).AnimationSpeed :=
scrlbr2.Position*1000;
  (Img1.Picture.Graphic as TGIFImage).Animate := True;
end;
```