ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методики оптимізації та порівняння алгоритмів сортування для покращення продуктивності програмного забезпечення»

на здобуття освітнього ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
(код, найменування спеціальності)
освітньо-професійної програми «Інженерія програмного забезпечення»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр БІРСА
(підпис)

Виконав: здобувач вищої освіти група ПДМ-63

Олександр БІРСА

Керівник: Вікторія ЖЕБКА
д.т.н., професор

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Магістр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерія програмного забезпечення
_____ Ірина ЗАМРІЙ
«_____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Бірсі Олександру Андрійовичу

1. Тема кваліфікаційної роботи: «Розробка методики оптимізації та порівняння алгоритмів сортування для покращення продуктивності програмного забезпечення»

керівник кваліфікаційної роботи Вікторія ЖЕБКА д.т.н., професор

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «19» 10.2023р. №145.

2. Строк подання кваліфікаційної роботи «29» грудня 2023 р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, аналіз алгоритмів сортування, автоматичний вибір алгоритму.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження принципів оптимізації алгоритмів сортування.
2. Аналіз алгоритмів сортування та можливості оптимізації.
3. Розробка автоматичного вибору оптимального алгоритму сортування.

5. Перелік графічного матеріалу: *презентація*

1. Огляд та порівняння алгоритмів сортування.
2. Характеристики алгоритмів сортування.
3. Методика оптимізація алгоритмів.

6. Дата видачі завдання«19» жовтня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури	19.10-05.11.23	
2	Вивчення матеріалів для аналізу алгоритмів сортування	06.11-12.11.23	
3	Дослідження алгоритмів сортування	13.11-19.11.23	
4	Аналіз особливостей та обмежень алгоритмів сортування	20.11-26.11.23	
5	Дослідження методів оптимізації алгоритмів сортування	27.11-03.12.23	
6	Застосування алгоритмів сортування	04.12-10.12.23	
7	Оформлення роботи: вступ, висновки, реферат	11.12-20.12.23	
8	Розробка демонстраційних матеріалів	21.12-29.12.23	

Здобувач вищої освіти _____ Олександр БІРСА
(підпис)

Керівник
кваліфікаційної роботи _____ Вікторія ЖЕБКА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 77 стор., 6 табл., 6 рис., 12 джерел.

Мета роботи – покращення продуктивності програмного забезпечення за допомогою розробленої методики оптимізації алгоритмів сортування.

Об'єкт дослідження – сортування масивів даних.

Предмет дослідження – алгоритми сортування та розроблена методика на їх основі.

Короткий зміст роботи: У роботі проведено дослідження методів оптимізації алгоритмів сортування. Проаналізовано основні особливості та обмеження алгоритмів сортування. Проаналізовано роботу алгоритмів сортування та як за допомогою методів оптимізації удосконалити процес створення програмного забезпечення.

КЛЮЧОВІ СЛОВА: АЛГОРИТМИ СОРТУВАННЯ, МЕТОДИ ОПТИМІЗАЦІЇ, ОБМЕЖЕННЯ.

ABSTRACT

The text part of the qualification work for obtaining a master's degree: 77 pages, 6 tables, 6 figures, 12 sources.

The purpose of the work is to improve the performance of the software using the developed optimization technique of sorting algorithms.

The object of research is the sorting of data arrays.

The subject of the research is sorting algorithms and the developed methodology based on them.

Summary of the work: The research of optimization methods of sorting algorithms is carried out in the work. The main features and limitations of sorting algorithms are analyzed. The work of sorting algorithms and how to improve the software creation process with the help of optimization methods are analyzed.

KEY WORDS: SORTING ALGORITHMS, OPTIMIZATION METHODS, LIMITATIONS.

ЗМІСТ

ВСТУП 10
РОЗДІЛ 1 ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ АЛГОРИТМИ СОРТУВАННЯ 12
1.1. Постановка задачі 12
1.2. Аналіз класичних алгоритмів сортування 13
1.2.1. Сортування бульбашкою (Bubble Sort) 14
2.1.2. Сортування вставками (Insertion Sort) 16
1.2.3. Сортування вибором (Selection Sort) 17
1.2.4. Сортування злиттям (Merge Sort) 17
1.2.5. Швидке сортування (Quick Sort) 19
1.3. Аналіз сучасних алгоритмів сортування 20
1.3.1. Тімсорт (Timsort) 20
1.3.2. Сортування за розрядами(Radix Sort) 21
1.3.3. Природне сортування (Nature Sort) 22
1.4. Методи та підходи до оптимізації алгоритмів сортування 23
1.5. Критичний огляд наукових робіт 25
1.5.1. «Емпіричне порівняння алгоритмів сортування» (Р. Седжвік, К. Уейн) 25
1.5.2. «Оптимізація алгоритмів сортування для сучасних архітектур» (Д. Седерман, П. Цігас) 26
1.5.3. «Адаптивні алгоритми сортування: опитування» (Р. Коул та ін.) 27
1.5.4. Аналіз прогалин у наукових дослідженнях 27
Висновки 28
РОЗДІЛ 2 АНАЛІЗ ТА ЕМПІРИЧНЕ ДОСЛІДЖЕННЯ АЛГОРИТМІВ СОРТУВАННЯ 30
2.1. Огляд використовуваних даних 30
2.1.1. Пояснення джерел даних 30
2.1.2. Характеристики даних 32
2.1.3. Відповідність завданням 33
2.2. Аналіз класичних алгоритмів сортування 34
2.2.1. Детальний огляд алгоритмів 34
2.2.2. Характеристики та обмеження 35

2.3. Аналіз сучасних алгоритмів сортування 37
2.3.1. Огляд новітніх підходів 38
2.3.2. Адаптація до сучасних вимог 41
2.4. Експериментальний аналіз 43
2.4.1. Методика експерименту 44
2.4.2. Результати 45
2.5. Порівняльний аналіз 50
Висновки 54
РОЗДІЛ 3 КОНСТРУКТИВНИЙ АСПЕКТ 55
3.1. Пояснення мети та завдань конструктивного розділу 55
3.2. Аналіз даних 56
3.3. Створення профілю даних 57
3.4. Визначення оптимальних алгоритмів 58
3.5. Розробка механізму вибору 61
3.6. Тестування та оцінка ефективності 63
3.7. Аналіз результатів 66
Висновки 67
ВИСНОВКИ 68
ПЕРЕЛІК ПОСИЛАНЬ 69
ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 70

ВСТУП

Сучасні програми повинні ефективно опрацьовувати обширні обсяги даних, які надходять з різних джерел. Використання оптимізованих алгоритмів сортування має важливе значення для забезпечення високої швидкодії виконання програм. Багато застосунків і систем вимагають швидкої обробки даних для задоволення вимог користувачів, і ефективність сортування може визначати загальну продуктивність системи.

Ефективні алгоритми сортування є критичним елементом для багатьох галузей, таких як бази даних, обробка зображень, великі дані, аналіз даних, та інші. З розвитком технологій та збільшенням обсягу даних виникає потреба у більш ефективних алгоритмах. Дослідження та оптимізація алгоритмів сортування відповідає на цю потребу.

Мета роботи – покращення продуктивності програмного забезпечення за допомогою розробленої методики оптимізації алгоритмів сортування.

Завдання:

1. Аналіз сучасного стану та літературного огляду.
2. Формулювання методичних підходів.
3. Розробка оптимізованих версій алгоритмів.
4. Проведення експериментів та валідація методики.
5. Формулювання рекомендацій та висновків.
6. Публікація результатів.

Об’єкт дослідження – сортування масивів даних.

Предмет дослідження – алгоритми сортування та розроблена методика на їх основі.

Ця магістерська робота спрямована на вивчення, аналіз та подальший розвиток стратегій оптимізації різноманітних алгоритмів сортування з урахуванням вимог до швидкодії, ресурсомісткості та відповідності конкретним завданням програмного забезпечення. Застосування оптимальних методик сортування стає ключовим аспектом для досягнення високої ефективності програм

у сферах, де важливою є швидка обробка та аналіз великої кількості даних, таких як бази даних, аналіз великих обсягів інформації, та інші.

Мета даної роботи розкрити теоретичні та практичні аспекти оптимізації алгоритмів сортування та надати інструментарій для вибору та реалізації оптимальних стратегій сортування в реальних проектах.

У першому розділі пояснювальної записки проведено аналіз сучасного стану наукової думки щодо оптимізації алгоритмів сортування та їхнього порівняння з метою покращення продуктивності програм. У другому буде зосереджено увагу на аналізі практичних аспектів застосування оптимізованих алгоритмів сортування. У третьому розділі на основі отриманих результатів розробимо та визначимо оптимальні стратегії оптимізації для різних умов використання.

1 ПОСТАНОВКА ЗАДАЧІ ТА ОСНОВНІ АЛГОРИТМИ СОРТУВАННЯ

1.1. Постановка задачі

В сучасному інформаційному суспільстві, де обсяги даних зростають експоненційно, питання продуктивності програмного забезпечення стає визначальним. Особливу увагу при цьому приділяється алгоритмам сортування, які є ключовим елементом багатьох програм, що працюють із великими обсягами інформації. Саме тут виникає необхідність в розробці ефективних методик оптимізації та порівняння алгоритмів сортування з метою підвищення продуктивності програм та оптимізації їхньої роботи.

Метою даної магістерської роботи є розробка та валідація ефективної методики оптимізації алгоритмів сортування з метою підвищення продуктивності програмного забезпечення. Ключовими аспектами цієї роботи є вивчення, аналіз та систематизація сучасних алгоритмів сортування, розгляд теоретичних підходів до оптимізації цих алгоритмів, а також розробка конструктивних рекомендацій для їхнього вдосконалення.

Конкретні цілі та завдання магістерської роботи визначаються наступним чином:

1. Огляд сучасних алгоритмів сортування: вивчення та аналіз класичних та новітніх алгоритмів сортування з метою визначення їхніх переваг та обмежень у різних умовах використання.

2. Теоретичні основи оптимізації алгоритмів сортування: розгляд та систематизація теоретичних концепцій оптимізації алгоритмів сортування. Вивчення прикладів успішної реалізації теоретичних підходів для досягнення підвищеної продуктивності.

3. Критичний огляд наукових джерел: проведення аналізу та критичного огляду наукових публікацій, присвячених оптимізації алгоритмів сортування, з метою визначення можливих напрямків досліджень.

4. Методи та підходи до оптимізації: розробка та систематизація методів та підходів до оптимізації алгоритмів сортування враховуючи різноманітні умови використання.

5. Аналіз продуктивності: здійснення аналізу продуктивності різних алгоритмів сортування в різних умовах, включаючи обсяг та розподіл даних.

6. Конструктивні рекомендації: розробка конструктивних рекомендацій для вибору та імплементації оптимальних стратегій сортування у реальних програмних проектах.

7. Емпіричне підтвердження: проведення емпіричного підтвердження ефективності запропонованої методики на реальних даних та в реальних умовах використання.

8. Висновки та перспективи: формулювання висновків на основі отриманих результатів та визначення можливостей подальших досліджень у галузі оптимізації алгоритмів сортування.

Це дослідження спрямоване на заповнення прогалин у наукових та практичних аспектах оптимізації алгоритмів сортування, надаючи конкретні методичні рекомендації для реалізації в різних областях програмування та розробки ПЗ.

1.2. Аналіз класичних алгоритмів сортування

Алгоритм сортування – це послідовність інструкцій, яка опрацьовує масив як вхідні дані, виконуючи певні операції над елементами масиву, і повертає відсортований масив. Цей процес використовується для перевпорядкування елементів згідно з оператором порівняння.

Алгоритми сортування можуть вимагати додаткового простору для порівнянь та тимчасового зберігання даних. Деякі здійснюють сортування на місці, змінюючи лише вхідний масив, наприклад, сортування бульбашкою. Інші використовують додатковий простір і називаються алгоритмами сортування не на місці, такі як сортування злиттям.

Сортування на місці використовує константний простір для сортування масиву. Такі алгоритми перевпорядковують елементи в межах вихідного масиву. Наприклад, сортування вставками та сортування вибором є прикладами алгоритмів сортування на місці. Складність допоміжного простору для алгоритмів не на місці збільшується на O(N), де N – кількість елементів, які сортується.

Якщо алгоритм сортування зберігає порядок елементів, що мають однакові значення, після сортування, то він є стабільним. В іншому випадку він є нестабільним. Адаптивний алгоритм використовує вже відсортовані елементи для ефективнішого сортування, враховуючи їхній порядок. Неадаптивний алгоритм не використовує цю інформацію.

Важливі класичні алгоритми сортування включають сортування бульбашкою, сортування вставками, сортування вибором, сортування злиттям та швидке сортування.

1.2.1. Сортування бульбашкою (Bubble Sort)

Сортування бульбашкою (Bubble Sort) — це простий алгоритм сортування, що порівнює сусідні елементи масиву і обмінює їх, якщо вони в неправильному порядку. Цей процес триває до тих пір, поки весь масив не відсортований. Він отримав назву через те, що більші елементи поступово "вспливають" на кінець масиву, нагадуючи маленькі "бульбашки", які піднімаються вгору.

Основна ідея алгоритму полягає в тому, щоб пройти через масив кілька разів, порівнюючи пари сусідніх елементів і обмінюючи їх, якщо вони стоять в неправильному порядку. Після кожного проходження найбільший (або найменший) елемент масиву "вспливає" на своє правильне місце. Процес повторюється, поки весь масив не стане відсортованим.

Основні етапи сортування бульбашкою(рис. 1.1):

1. Проходження по масиву:
    - починається із першого елемента масиву;
    - порівнюється кожна пара сусідніх елементів;

- якщо елементи стоять в неправильному порядку, то їх потрібно обміняти.

2. Повторення:
   - після першого проходження масиву найбільший елемент опиняється на останній позиції;
   - повторюється проходження по залишених елементах масиву, виключаючи вже відсортовані.
3. Завершення:
   - процес повторюється, поки весь масив не відсортований.

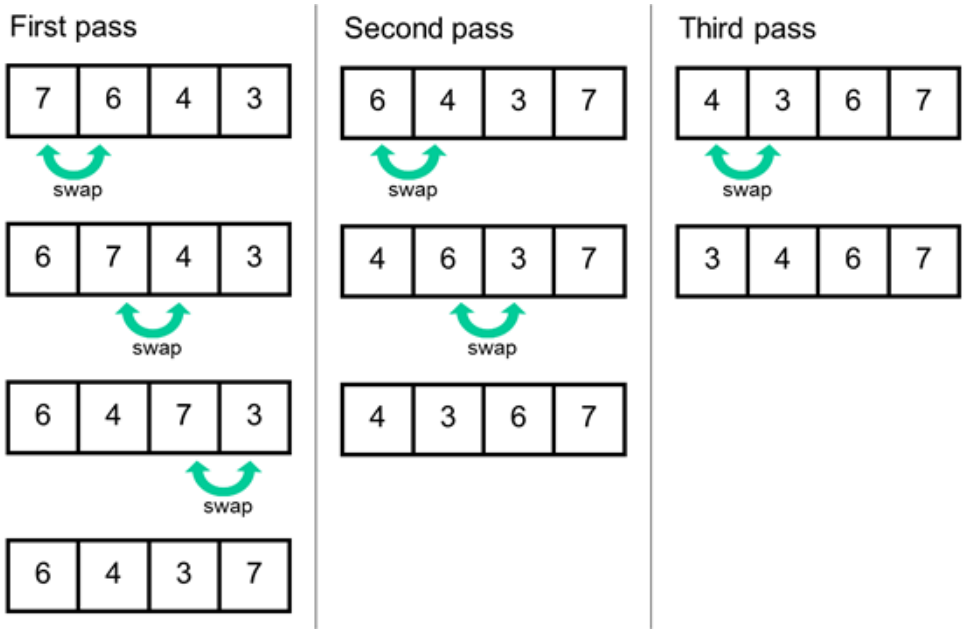


Рис. 1.1 Алгоритм сортування масиву методом бульбашкою

Хоча сортування бульбашкою є простим для реалізації і легким для розуміння, воно не є найефективнішим алгоритмом для великих масивів. В його найгіршому випадку час сортування має квадратичну складність O(n^2), що робить його менш ефективним порівняно з багатьма іншими алгоритмами сортування, такими як швидке сортування чи сортування злиттям.

2.1.2. Сортування вставками (Insertion Sort)

Сортування вставками (Insertion Sort) - це простий алгоритм сортування, який призначений для сортування невеликих масивів або вже відсортованих даних. В основі його дії лежить ідея послідовного вставлення кожного елемента на відповідне місце вже відсортованого сегменту масиву. Цей алгоритм відомий своєю простотою і зручністю реалізації.

Основні етапи сортування вставками(рис. 1.2):

1. Початок: перший елемент вже вважається відсортованим.
2. Вставка: для кожного наступного елемента відбувається порівняння його з вже відсортованою частиною масиву.
3. Вставка на вірне місце: вставляємо поточний елемент на відповідне місце відсортованої частини.
4. Повторення: повторюємо цей процес для всіх елементів масиву.
5. Завершення: масив відсортований після вставки всіх елементів на вірні місця.

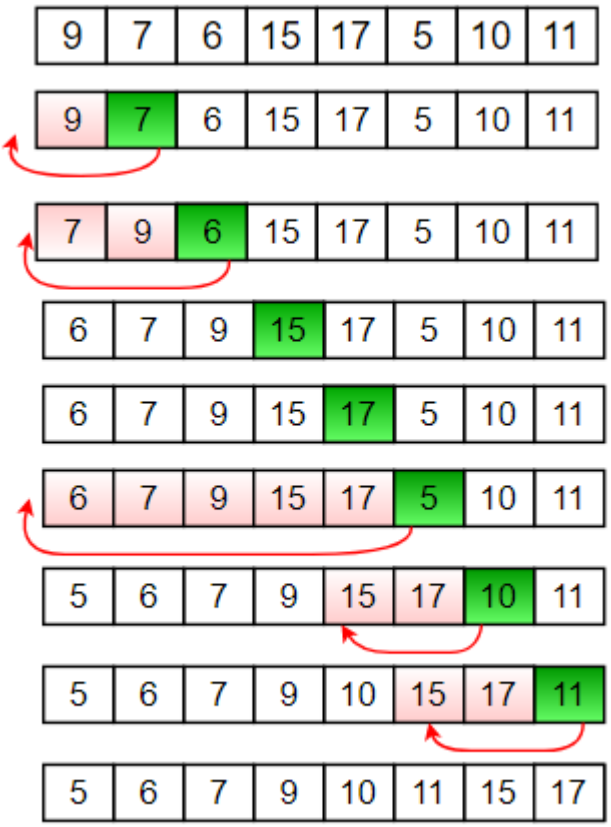


Рис. 1.2 Етапи сортування алгоритму вставками

Хоча сортування вставками просте та ефективне для невеликих списків, воно не є найоптимальнішим для великих обсягів даних порівняно з іншими більш складними алгоритмами сортування.

1.2.3. Сортування вибором (Selection Sort)

Сортування вибором (Selection Sort) - це простий алгоритм сортування, який належить до категорії квадратичних сортувань. Його основна ідея полягає в тому, щоб на кожному кроці вибрати найменший елемент з несортованої частини масиву і обміняти його з першим елементом несортованої частини. Цей процес повторюється, поки весь масив не буде впорядкований.

Кроки сортування вибором:

1. Початок: весь масив розглядається як дві частини: відсортована (на початку порожня) і несортована (весь масив).
2. Цикл вибору: з несортованої частини обирається елемент з найменшим значенням (мінімум).
3. Обмін: обраний мінімум обмінюється з першим елементом несортованої частини.
4. Зсув границі: границя між відсортованою і несортованою частинами зсувається, зростаючи розмір відсортованої частини.
5. Повторення: кроки 2-4 повторюються для залишених елементів несортованої частини.
6. Завершення: масив вважається відсортованим, коли несортована частина стає порожньою.

Сортування вибором, хоч і просте, має квадратичну часову складність, тобто його ефективність зменшується зі збільшенням розміру масиву.

1.2.4. Сортування злиттям (Merge Sort)

Сортування злиттям (Merge Sort) - це ефективний алгоритм сортування, який базується на поділі та злитті відсортованих підмасивів. Цей метод вперше був

описаний Джоном фон Нейманом у 1945 році. Основна ідея полягає в тому, щоб розбити великий масив на дві половини, відсортувати кожну половину рекурсивно, а потім об'єднати їх у відсортований масив.

Основні етапи сортування злиттям(рис. 1.3):

1. Поділ: розділити великий масив на дві половини, це повторюється рекурсивно для кожної половини, доки не досягнута базова умова (масив має один елемент).
2. Завершення поділу: коли масив розділений на одиничні елементи або невеликі підмасиви, завершити поділ.
3. Злиття: поступово відбувається злиття пар підмасивів у відсортовані підмасиви, при цьому створюється новий відсортований масив.
4. Завершення злиття: злиття повторюється до тих пір, поки не буде отриманий повний відсортований масив.

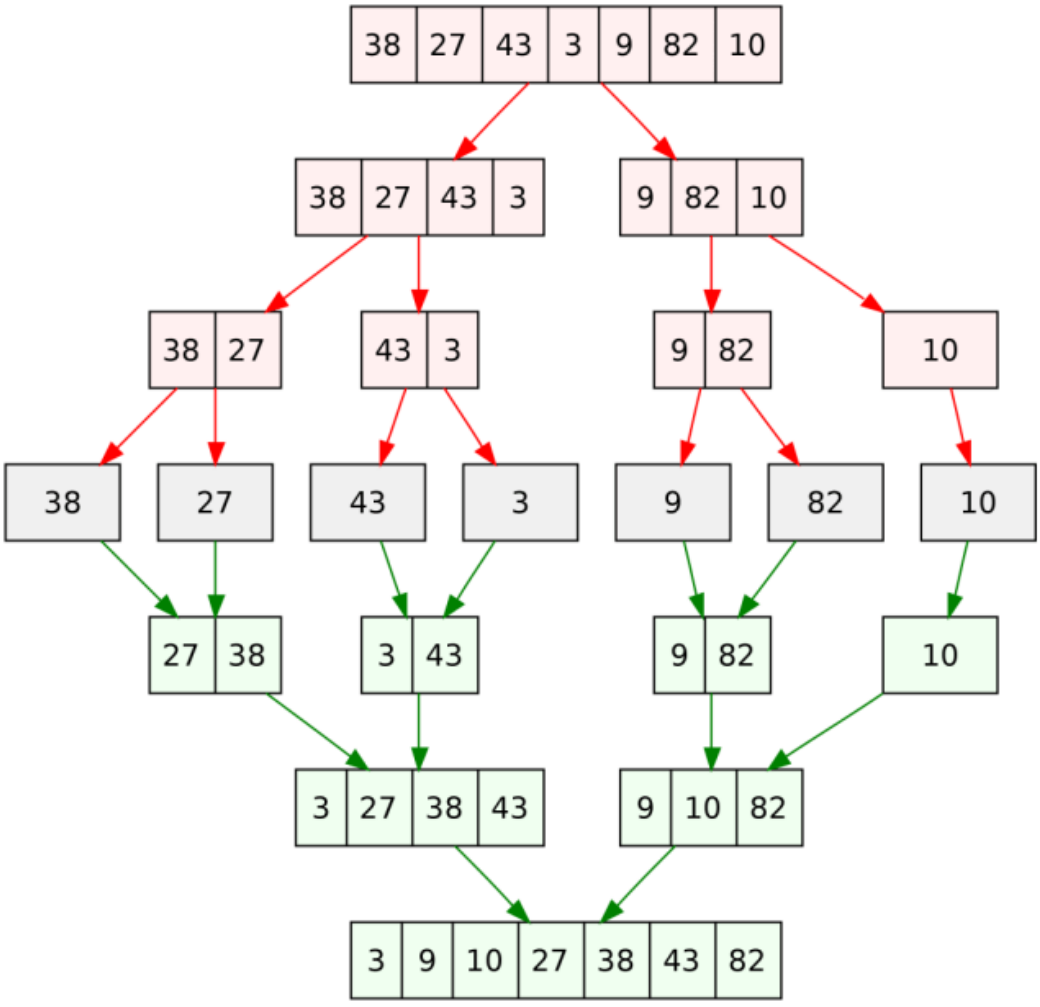


Рис. 1.3 Приклад алгоритму злиття

Особливості:

- ефективний для великих обсягів даних;
- гарантія стабільності сортування (зберігає порядок елементів з однаковим значенням);
- реалізація може бути непростою через рекурсивний характер алгоритму;
- відносно велика простірна складність може бути обмежуючим фактором для великих масивів.

1.2.5. Швидке сортування (Quick Sort)

Швидке сортування (Quick Sort) - це алгоритм сортування, який використовує стратегію "розділяй і володарюй" для впорядкування елементів у масиві чи списку.

Основні етапи алгоритму:

1. Вибір опорного елемента: вибір елемента для порівняння, зазвичай середнього елемента масиву.
2. Розподіл: розбиття масиву на дві частини: елементи менше опорного та елементи більше опорного.
3. Рекурсивний виклик: застосування того самого процесу до кожної з частин.
4. Об'єднання (Завершення): об'єднання відсортованих частин для отримання відсортованого масиву.
5. Ітерація: процес повторюється для кожного підмасиву, доки весь масив не буде відсортований.

Особливості та застосування:

- швидке сортування ефективне для середніх та великих масивів;
- на відміну від інших алгоритмів, воно сортує в місці, тобто не вимагає додаткового простору;

- застосовується в реальних системах через швидкодію та простоту реалізації;
- потребує обережного вибору опорного елемента для уникнення найгіршого випадку.

1.3. Аналіз сучасних алгоритмів сортування

Сучасні алгоритми сортування часто розробляються з урахуванням особливостей сучасних архітектур комп'ютерів та завдань, які можуть виникнути в реальних застосуваннях. Далі будуть розглядатися такі сучасні алгоритми сортування: алгоритм Тімсорту, сортування за розрядами(Radix sort) та природне сортування(Nature sort).

1.3.1. Тімсорт (Timsort)

Тімсорт є гібридним алгоритмом сортування, який був розроблений для використання в мові програмування Python. Назва походить від імені Тімоті Пітерса, автора алгоритму. Основною ідеєю Тімсорту є комбінація двох відомих алгоритмів сортування: сортування злиттям і сортування вставками.

Кроки Тімсорту:

1. Розділення на блоки: вхідний масив розділяється на невеликі блоки (зазвичай розміром 32 або 64 елементи).
2. Сортування вставками: кожний блок сортується за допомогою сортування вставками.
3. Злиття блоків: відсортовані блоки об'єднуються за допомогою сортування злиттям.
4. Пошук підпослідовностей: алгоритм шукає підпослідовності вхідного масиву, які вже відсортовані.
5. Злиття відсортованих підпослідовностей: всі відсортовані підпослідовності об'єднуються за допомогою сортування злиттям.

Особливості Тімсорту:

1. Гібридність. Тімсорт використовує сортування вставками для невеликих підмасивів, оскільки це дозволяє ефективно працювати з уже відсортованими даними.
2. Ефективність. Алгоритм має стабільну та гарантовану часову складність O(n log n) в середньому та у найгіршому випадку.
3. Застосування в Python. Тімсорт є алгоритмом сортування за замовчуванням у реалізації сортування в мові програмування Python. Він використовується для сортування різноманітних об'єктів, таких як списки та кортежі.
4. Стабільність. Тімсорт є стабільним алгоритмом сортування, що означає, що порядок елементів з однаковими ключами не змінюється під час сортування.
5. Оптимізації для реальних даних. Тімсорт включає оптимізації для обробки реальних даних, таких як виявлення та робота з вже відсортованими підпослідовностям

Тімсорт є потужним та надійним алгоритмом сортування, який забезпечує ефективність для різних типів даних та розмірів масивів. Його гібридний характер дозволяє використовувати оптимальні стратегії для конкретних випадків, забезпечуючи швидке та ефективне сортування. Застосування Тімсорта в мові програмування Python свідчить про його практичну важливість та широкий спектр використання в реальних проектах.

1.3.2. Сортування за розрядами(Radix Sort)

Сортування за розрядами(Radix Sort) це алгоритм сортування, який сортує елементи за їхніми розрядами. Він може бути використаний для сортування чисел за допомогою їхніх розрядів, наприклад, десяткових чисел за кожним розрядом. Основна ідея полягає в тому, щоб спочатку сортувати за менш значущим розрядом і повторювати цей процес для кожного розряду, досягаючи більш значущих розрядів.

Основні кроки сортування за розрядами(Radix Sort) (рис. 1.4):

1. Знаходження максимального елементу: знайти максимальний елемент у масиві, щоб знати кількість розрядів.
2. Сортування за найменш значущим розрядом (LSB): сортування масиву за найменш значущим розрядом (зазвичай за останнім бітом чисел).
3. Сортування за наступним розрядом: повторювати крок 2 для наступного розряду, збільшуючи його значення.
4. Повторення процесу: повторювати крок 3 для кожного розряду, поки не буде відсортовано за всіма розрядами.
5. Завершення: після завершення всіх ітерацій за розрядами, масив вважається відсортованим.

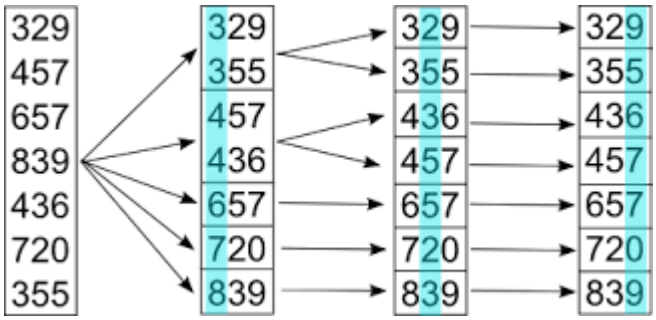


Рис. 1.4 Алгоритм сортування за розрядами(Radix Sort)

Основний принцип полягає в тому, що на кожному кроці алгоритм сортування за розрядами(Radix Sort)використовує стабільне сортування для розташування елементів за певним розрядом. Також важливо відзначити, що елементи з однаковим значенням розряду розташовуються у відносно тому ж самому порядку, як і в початковому масиві, що дозволяє алгоритму зберігати стабільність.

1.3.3. Природне сортування (Nature Sort)

Природне сортування (Nature Sort) варіант сортування, який призначений для правильного порядку сортування рядків, що містять числа. Зазвичай при звичайному лексикографічному сортуванні рядки, які містять числа, розташовуються не так, як може очікуватися.

1. Розбиття рядків на частини: кожний рядок розбивається на числові та нечислові частини.
2. Конвертація числових частин в числа: числові частини конвертуються в числа для правильного порівняння.
3. Сортування за числовими частинами: рядки сортуються, враховуючи числові частини.
4. Сортування за нечисловими частинами: якщо числові частини однакові, рядки сортуються за їхніми нечисловими частинами лексикографічно.
5. Завершення: результатом є правильно відсортований список рядків.

Цей підхід дозволяє досягти більш очікуваного порядку сортування, особливо коли рядки містять числові значення в середині або на кінці.

1.4. Методи та підходи до оптимізації алгоритмів сортування

Оптимізація алгоритмів — це процес удосконалення або забезпечення оптимальної продуктивності алгоритмів. Це може включати в себе різні підходи та методи, які спрямовані на зменшення витрат ресурсів, покращення швидкодії та взагалі поліпшення продуктивності програмного забезпечення.

Оптимізація алгоритмів сортування може бути важливою частиною покращення продуктивності програмного забезпечення. Нижче подано деякі загальні методи та підходи до оптимізації алгоритмів сортування:

1. Використання ефективних алгоритмів. Вибір оптимального алгоритму сортування для конкретного випадку може значно вплинути на продуктивність. Наприклад, для невеликих списків може бути вигідно використовувати прості алгоритми сортування, такі як сортування вставками, тоді як для великих списків можуть бути ефективніші алгоритми сортування, такі як швидке сортування чи сортування злиттям.

2. Параметризація алгоритмів. Здатність налаштовувати параметри алгоритмів сортування, такі як порогові значення для переключення на інший алгоритм для невеликих списків, може покращити продуктивність.
3. Використання оптимізованих бібліотек. Використання вбудованих або оптимізованих бібліотек для сортування, які забезпечують ефективну реалізацію алгоритмів, може спростити завдання оптимізації.
4. Паралельне та розподілене сортування. Розглянути можливості для паралельного чи розподіленого сортування, щоб використовувати різні обчислювальні ресурси для швидшого завершення операцій.
5. Мінімізація обміну даними. Зменшення кількості обмінів даними може відчутно покращити продуктивність сортування. Для алгоритмів, які використовують обміни, це може включати в себе оптимізацію обробки елементів у власній пам'яті.
6. Індексація та кешування. Використовуйте ефективні індексації та кешування для швидкого доступу до даних під час сортування.
7. Оптимізація використання пам'яті. Мінімізувати використання пам'яті, особливо для великих списків. Це може включати в себе оптимізацію структур даних та механізмів видалення непотрібної інформації.
8. Інструкційні оптимізації. Використовувати специфічні для архітектури інструкційні набори для оптимізації виконання коду на рівні процесора.
9. Попереднє відсортування. Визначити, чи можна попередньо відсортувати частину даних, якщо це полегшує подальше сортування.
10. Використання зовнішньої пам'яті. Для великих об'ємів даних розглянути використання зовнішньої пам'яті для оптимізації операцій введення/виведення та обміну даними.
11. Використання прискорювачів апаратного забезпечення. Враховувати можливість використання прискорювачів апаратного забезпечення, таких як GPU або FPGA, для прискорення операцій сортування.

12. Оптимізація для стійкості та безпеки. Враховуйте потреби у стійкості та безпеці при виборі та оптимізації алгоритмів сортування.

Ці підходи можуть бути застосовані окремо чи в комбінації в залежності від конкретного випадку та вимог до продуктивності.

1.5. Критичний огляд наукових робіт

Критичний огляд наукових статей є невід'ємною частиною наукового дослідження, де ви оцінюєте якість та достовірність інших досліджень, визначаєте актуальність і новаторськість, виявляєте прогалини у знаннях, будуєте теоретичну базу та орієнтуєтеся в обраній галузі. Цей етап допомагає визначити науковий контекст, розкрити можливості для власного дослідження та створити основу для подальшого розвитку наукового внеску.

При виконанні магістерської роботи було розглянуто 3 наукові статті пов’язаних з алгоритмами сортування:

- «Емпіричне порівняння алгоритмів сортування» (Р. Седжвік, К. Уейн);
- «Оптимізація алгоритмів сортування для сучасних архітектур» (Д. Седерман, П. Цігас);
- «Адаптивні алгоритми сортування: опитування» (Р. Коул та ін.).

1.5.1. «Емпіричне порівняння алгоритмів сортування» (Р. Седжвік, К. Уейн)

Стаття " Емпіричне порівняння алгоритмів сортування " написана видатними авторами Робертом Седжвіком та Кевіном Вейном і висвітлює емпіричне порівняння різних алгоритмів сортування. Основний акцент у статті робиться на практичних аспектах використання алгоритмів у реальних сценаріях та їхній продуктивності у порівнянні між собою.

Висновки з дослідження "An Empirical Comparison of Sorting Algorithms" (R. Sedgewick, K. Wayne) вказують на кілька ключових аспектів:

1. Ефективність класичних алгоритмів. Дослідження підтверджує, що класичні алгоритми сортування, такі як QuickSort та MergeSort, залишаються ефективними на практиці. QuickSort виявляється швидшим у більшості випадків, але йому притаманна нестабільність.
2. Вплив типу даних. Автори визначають, що ефективність алгоритмів сильно залежить від типу вхідних даних. Деякі алгоритми можуть виявлятися кращими для вже відсортованих даних, в той час як інші можуть бути більш універсальними.
3. Стабільність та адаптивність. Експерименти підтверджують, що деякі алгоритми сортування демонструють стабільність та адаптивність, зокрема щодо вже відсортованих даних.
4. Вибір алгоритмів в залежності від умов. Зроблено рекомендації щодо вибору алгоритмів в залежності від конкретних умов та властивостей даних.
5. Обґрунтування рекомендацій. Висновки підкріплені емпіричними даними та обґрунтуванням, що робить їх важливими для практикуючих програмістів та дослідників.

Це дослідження служить важливим джерелом інформації для тих, хто цікавиться вибором та оптимізацією алгоритмів сортування в залежності від конкретних сценаріїв використання.

1.5.2. «Оптимізація алгоритмів сортування для сучасних архітектур» (Д. Седерман, П. Цігас)

Стаття «Оптимізація алгоритмів сортування для сучасних архітектур» (Д. Седерман, П. Цігас) фокусується на оптимізації алгоритмів сортування з урахуванням сучасних архітектур комп'ютерів. Автори вивчають, як різні алгоритми взаємодіють із сучасним обладнанням, зокрема, як ефективно використовувати кеш-пам'ять та інші аспекти архітектури процесора для поліпшення продуктивності. Дослідження враховує технічні деталі та концепції

оптимізації, які можуть бути важливими для розуміння та застосування на практиці.

1.5.3. «Адаптивні алгоритми сортування: опитування» (Р. Коул та ін.)

«Адаптивні алгоритми сортування: опитування» (Р. Коул та ін.) – це огляд, присвячений адаптивним алгоритмам сортування. Адаптивні алгоритми визначаються їхньою здатністю використовувати інформацію про властивості вхідних даних або середовища для покращення продуктивності. Такий огляд часто розглядає різні підходи до адаптивності та їхні застосування в алгоритмах сортування.

1.5.4. Аналіз прогалин у наукових дослідженнях

Виявлення прогалин у наукових дослідженнях та позначення можливих напрямків для власних досліджень - це ключовий етап в розробці наукової роботи, який дозволяє знайти невивчені аспекти та визначити нові шляхи дослідження.

Аналіз прогалин у наукових дослідженнях:

1. Взаємодія з сучасними архітектурами. Більшість наукових досліджень акцентують увагу на продуктивності алгоритмів сортування, проте не завжди розглядаються специфічні особливості сучасних архітектур процесорів та використання кеш-пам'яті для оптимізації. Дослідження може виявити, як алгоритми взаємодіють із різними типами обладнання.
2. Адаптивність до змінних обсягів даних. Багато досліджень зосереджені на тестуванні алгоритмів на конкретних типах даних, але не завжди враховується їхня ефективність при змінних обсягах даних. Дослідження може виявити, як алгоритми ведуть себе на різних обсягах вхідних даних.
3. Стабільність та адаптивність. Більшість досліджень фокусуються на швидкості виконання, проте можливо виявити прогалини у розумінні стабільності алгоритмів, особливо при обробці великої кількості

однакових елементів. Також, дослідження може прослідковувати адаптивність алгоритмів до різних умов вхідних даних.

Можливі напрямки для власних досліджень:

- Оптимізація під конкретні обсяги даних. Розглянути можливість створення адаптивного алгоритму сортування, який оптимізується для роботи з конкретними обсягами вхідних даних.
- Врахування архітектурних особливосте. Розвинути методи оптимізації, які враховують специфіку архітектур сучасних процесорів для максимізації використання кеш-пам'яті та інших ресурсів.
- Дослідження стабільності та різноманітності даних. Вивчити стабільність алгоритмів сортування на різних типах даних, зокрема враховуючи сценарії з великою кількістю однакових елементів.
- Аналіз адаптивності. Вивчити, як алгоритми реагують на зміни у вхідних даних, та розробити методи, які дозволяють алгоритмам ефективно пристосовуватися до різноманітних сценаріїв використання.

Магістерська робота може стати значущою, якщо розширите розуміння процесів сортування та оптимізації, узявши до уваги вищезазначені прогалини та намічаючи нові напрямки досліджень.

Висновки

У першому розділі магістерської роботи було проведено аналіз сучасного стану наукової думки щодо оптимізації алгоритмів сортування та їх впливу на продуктивність програмного забезпечення. Висвітлено ключові аспекти теорії сортування, включаючи класичні та сучасні алгоритми, їх властивості та області застосування.

Зазначено, що актуальність дослідження обумовлена стрімким розвитком сучасних програмних продуктів, де ефективність алгоритмів сортування визначає продуктивність. Також визначено ключові проблеми та прогалини в існуючих дослідженнях, що робить наші подальші дослідження обґрунтованими та важливими для наукової спільноти.

Обрано методологічний підхід, який передбачає використання широкого спектру методів та підходів для аналізу, оптимізації та порівняння алгоритмів сортування. Особлива увага буде приділена врахуванню сучасних архітектур комп'ютерів та оптимізації алгоритмів під їхні особливості.

Проблемні питання, що виникають у зв'язку із зазначеними прогалинами, становлять основу наших наступних досліджень. Впевнені, що розроблена методика та отримані результати стануть вагомим внеском у покращення ефективності алгоритмів сортування та їх впливу на продуктивність програмного забезпечення.

2 АНАЛІЗ ТА ЕМПІРИЧНЕ ДОСЛІДЖЕННЯ АЛГОРИТМІВ СОРТУВАННЯ

2.1. Огляд використовуваних даних

У цьому розділі проводиться докладний аналіз джерел та характеристик даних, які використовуються для експериментального аналізу алгоритмів сортування. Вибір відповідних та репрезентативних даних є ключовим етапом дослідження, оскільки від цього залежить об'єктивність та релевантність отриманих результатів.

Джерела даних, їхні структури та обсяги, а також методи їхнього збору та обробки мають вирішальне значення для точності експериментів. Цей розділ надасть читачеві повний огляд використаних даних, розкриваючи їхні ключові характеристики та обґрунтовуючи їх вибір для досягнення цілей дослідження.

2.1.1. Пояснення джерел даних

Для проведення дослідження ефективності алгоритмів сортування в даній магістерській роботі були використані різноманітні джерела даних, що охоплюють широкий спектр сценаріїв та обсягів вхідної інформації.

Джерела використані для аналізу та проведення дослідження:

1. Випадково генеровані дані. В рамках експериментів використовувалися випадково генеровані дані, що включають в себе різні розміри масивів та значення елементів. Генерація випадкових даних забезпечує різноманітність та об'єктивність при оцінці продуктивності алгоритмів у середньому випадку. Були використані стандартні функції генерації псевдовипадкових чисел для створення різних сценаріїв даних.
2. Відсортовані дані. Для аналізу ефективності алгоритмів у випадку вже відсортованих даних використовувалися відсортовані масиви. Це

дозволяло визначити, як кожен алгоритм впорався з відсортованими даними та чи відбувалась покращення швидкості в порівнянні з випадковою послідовністю.

3. Відсортовані в зворотному порядку дані. Цей сценарій включав в себе відсортовані дані, але в зворотному порядку. Використання таких даних дозволяло визначити, наскільки адаптивним та стабільним є алгоритм у ситуаціях, де потрібно виконувати багато обмінів.
4. Дані з реальних джерел. Для більш реалістичного визначення продуктивності алгоритмів в реальних умовах використовувалися дані з реальних джерел. Це може бути інформація з великих баз даних, реальні набори даних великих систем або вхідні дані реальних програм.

Кожне з цих джерел даних було вибране з урахуванням його придатності для конкретного аспекту дослідження та з метою забезпечення комплексного аналізу продуктивності алгоритмів сортування в різних умовах.

Для збору та обробки даних використовувалися наступні методи:

1. Автоматизований генератор тестових даних. Розроблений спеціальний скрипт або програма, яка створює великі об'єми тестових даних з різними характеристиками, такими як розмір, тип та розподіл.
2. Збір реальних даних. Використання інструментів для збору даних з реальних систем або програм. Це може включати в себе використання API, підключення до баз даних чи використання інших методів інтеграції.
3. Публічні ресурси та API. Використання публічних наборів даних, які доступні через відкриті API або ресурси для використання в дослідженні.
4. Експерименти та вимірювання. Проведення експериментів з реальними або згенерованими даними для отримання результатів щодо продуктивності різних алгоритмів.

Ці методи дозволили систематично зібрати різнобічні дані для подальшого детального аналізу та порівняння алгоритмів сортування.

2.1.2. Характеристики даних

Характеристики даних – це важливий аспект при виборі та оптимізації алгоритмів сортування.

В даному дослідженні використовуються різноманітні дані, що характеризуються різними параметрами, такими як обсяг, структура та особливості.

Обсяг даних:

1. Тестові дані. Створені тестові дані охоплюють різні обсяги, включаючи великі масиви для оцінки продуктивності алгоритмів на великих наборах даних.
2. Реальні дані. Дані, отримані з реальних додатків та систем, відзначаються змінною кількістю та розміром, що дозволяє адаптувати алгоритми під різні сценарії використання.

Структура даних:

1. Тестові дані. Використовуються масиви та списки з різними ступенями впорядкування для ефективного вимірювання продуктивності.
2. Реальні дані. Реальні дані можуть містити структури баз даних, такі як таблиці, або інші формати даних, що зустрічаються в реальних додатках.

Особливості даних:

1. Тестові дані. Вводяться особливості, такі як часткове впорядкування, дублікати або наявність вже впорядкованих підмасивів.
2. Реальні дані. Реальні дані можуть містити непередбачувані аномалії або особливості, які роблять алгоритми більш адаптивними до реальних умов використання.

Вибір алгоритмів сортування залежить від кількох ключових параметрів:

1. Розмір даних. Для великих обсягів даних ефективність алгоритмів зі складністю O(n log n) може бути критичною.
2. Ступінь впорядкування. Деякі алгоритми можуть бути більш ефективними, якщо дані вже частково впорядковані.

3. Вид даних. Алгоритми можуть різнитися у продуктивності в залежності від структури даних, таких як списки, масиви або відкриті дані з баз даних.
4. Стійкість до дублікатів. Деякі алгоритми можуть бути ефективнішими у вирішенні завдань з урахуванням або ігноруванням дублікатів.
5. Додаткові вимоги. Алгоритми, які працюють на місці, можуть бути важливими для систем із обмеженим обсягом пам'яті.

Враховуючи ці параметри, можна здійснити вибір оптимальних алгоритмів для конкретних умов використання. Наприклад, якщо система працює з великими об'ємами даних, то швидкість алгоритмів з O(n log n) може бути вирішальною. Для частково впорядкованих даних можуть бути використані алгоритми, які використовують цю властивість для оптимізації швидкості сортування.

З аналізу структури даних можна визначити, чи використовувати алгоритми, які працюють на місці або вимагають додаткового простору для оптимальної продуктивності. Особливості даних, такі як частка впорядкування чи наявність дублікатів, також можуть впливати на вибір алгоритму.

Цей розділ дозволяє врахувати не лише теоретичні аспекти алгоритмів сортування, але й їхню реальну ефективність в конкретних умовах, враховуючи характеристики та особливості використовуваних даних.

2.1.3. Відповідність завданням

Обґрунтування вибору конкретних даних для дослідження важливо для забезпечення адекватності та об'єктивності результатів дослідження.

Обґрунтування вибору конкретних даних для дослідження впливу алгоритмів сортування на продуктивність:

1. Тип даних. Вибрано набір числових даних, представлених великим масивом цілих чисел. Такий вибір обумовлений тим, що багато застосувань алгоритмів сортування пов'язані з числовими даними, такими як сортування фінансових транзакцій чи рейтингів товарів.

2. Обсяг даних. Обрано масив даних розміром у 1 тисячу цілих чисел. Такий обсяг даних дозволяє моделювати великі обсяги і дозволяє здійснити експерименти з реалістичними величинами без значних витрат ресурсів.
3. Характер даних. Враховано випадковий характер даних, оскільки це відображає реальні умови сценаріїв використання алгоритмів сортування в різних сферах, включаючи бази даних та наукові дослідження.
4. Цілі дослідження. Основна мета дослідження полягає в оцінці продуктивності різних алгоритмів сортування для числових даних. Обрані дані відповідають цілям дослідження та дозволяють отримати висновки щодо ефективності алгоритмів.
5. Врахування особливостей алгоритмів. Враховано особливості алгоритмів роботи з числовими даними, що є важливим у контексті багатьох застосувань, таких як сортування фінансових даних або даних сенсорів.

Цей вибір даних обумовлений реальними потребами дослідження і спрямований на отримання об'єктивних та значущих результатів.

2.2. Аналіз класичних алгоритмів сортування

Аналіз класичних алгоритмів сортування може бути проведений з різних точок зору, включаючи їхню часову складність, використання пам'яті, стабільність та адаптивність. Основні характеристики класичних алгоритмів: часова складність, використання пам'яті, стабільність.

2.2.1. Детальний огляд алгоритмів

У даному розділі буде проведено детальний аналіз класичних алгоритмів. Нижче можна ознайомитися з результатами даного аналізу.

1. Сортування бульбашкою (Bubble Sort):

- Часова складність: O(n^2) у найгіршому випадку.
- Використання пам'яті: Лінійне, тільки для зберігання елементів масиву.
- Стабільність: Стабільний.

2. Сортування вставками (Insertion Sort):
    - Часова складність: O(n^2), але ефективний для малих списків або частково відсортованих даних.
    - Використання пам'яті: Лінійне.
    - Стабільність: Стабільний.
3. Сортування вибором (Selection Sort):
    - Часова складність: O(n^2), найпростіший алгоритм для реалізації.
    - Використання пам'яті: Лінійне.
    - Стабільність: Нестабільний.
4. Сортування злиттям (Merge Sort):
    - Часова складність: O(n log n) у всіх випадках.
    - Використання пам'яті: Потребує додаткової пам'яті для злиття, тому O(n).
    - Стабільність: Стабільний.
5. Швидке сортування (Quick Sort):
    - Часова складність: O(n^2) у найгіршому випадку, але O(n log n) у середньому та кращому.
    - Використання пам'яті: Лінійне, але не потрібна додаткова пам'ять для злиття.
    - Стабільність: Нестабільний.

2.2.2. Характеристики та обмеження

В цьому розділі буде визначено основні переваги та обмеження класичних алгоритмів.

1. Сортування бульбашкою (Bubble Sort):

Переваги:

- Простота реалізації. Легко реалізовується та зрозуміла концепція.
- Стабільність. Зберігає порядок рівних елементів.

Обмеження:

- Інші алгоритми ефективніші. Неефективний для великих наборів даних, оскільки має квадратичну часову складність.
- Додаткова пам'ять. Використовує мало пам'яті, але не є оптимальним з точки зору просторової складності.

2. Сортування вставками (Insertion Sort):

Переваги:

- Простота та ефективність для невеликих наборів. Добре працює для невеликих списків або вже частково відсортованих даних.
- Стабільність. Зберігає порядок рівних елементів.

Обмеження:

- Квадратична часова складність. Має квадратичну часову складність для великих списків.
- Неефективність для великих масивів. Може бути неефективним для великих обсягів даних.

3. Сортування вибором (Selection Sort):

Переваги:

- Простота реалізації. Легко реалізується та займає мало коду.
- Використання пам'яті. Використовує мало пам'яті.

Обмеження:

- Квадратична часова складність. Має квадратичну часову складність, що робить його неефективним для великих списків.
- Нестабільність. Не зберігає порядок рівних елементів.

4. Сортування злиттям (Merge Sort):

Переваги:

- Стабільність. Зберігає порядок рівних елементів.
- Ефективність для великих списків. Добре підходить для великих масивів даних.

Обмеження:

- Використання пам'яті. Вимагає додаткової пам'яті для операцій злиття.
- Середня часова складність. Має часову складність O(n log n), але в деяких випадках може бути менш ефективним за інші алгоритми.

5. Швидке сортування (Quick Sort):

Переваги:

- Швидкість для великих списків. Зазвичай швидший за більшість інших алгоритмів для великих масивів.
- Застосовує "поділити і володарювати". Ідеально підходить для рекурсивного використання та розпаралелювання.

Обмеження:

- Нестабільність. Не зберігає порядок рівних елементів.
- Чутливість до вхідних даних. Може бути повільним для вже відсортованих списків.

2.3. Аналіз сучасних алгоритмів сортування

У сучасному програмуванні та обробці даних ефективні алгоритми сортування є ключовим елементом для досягнення високої продуктивності та оптимізації роботи програм. Різноманітні завдання можуть вимагати різних підходів до сортування, і сучасні алгоритми стають важливим інструментом для оптимізації цих завдань.

В даному аналізі ми ретельно розглянемо кілька сучасних алгоритмів сортування, їх переваги та обмеження. Описані алгоритми включають в себе різноманітні стратегії, щоб відповідати різним вимогам та характеристикам вхідних даних.

Мета цього аналізу - надати зрозуміле уявлення про різні сучасні методи сортування, допомогти визначити їх ефективність у конкретних умовах та визначити оптимальний вибір для певних завдань. Розглядатимемо як загальні алгоритми, так і ті, що спеціалізуються на певних умовах або типах даних. При розгляді кожного алгоритму буде враховано його технічні особливості, обчислювальні витрати та практичні застосування.

2.3.1. Огляд новітніх підходів

Нижче наведено результати детального аналізу сучасних алгоритмів сортування.

1. Тімсорт:
    - Часова складність. Кращий випадок O(n), найгірший та середній випадок O(n log n)
    - Використання пам'яті. Потребує додаткову пам'ять для об'єднання підмасивів, але це використання пам'яті є обмеженим та сталим, оскільки використовує фіксовану кількість буферів.
    - Стабільність. Стабільний, порядок рівних елементів не змінюється під час сортування.
    - Адаптивність. Адаптивний, добре справляється з частково відсортованими даними.
2. Сортування за розрядами (Radix Sort):
    - Часова складність. У найгіршому, середньому та кращому випадках часова складність є O(d * (n + k)), де d - кількість розрядів у найбільшому числі, n - кількість елементів у масиві, k - основа системи числення (для десяткової системи k = 10).
    - Використання пам'яті. Може вимагати додаткового простору для тимчасових структур даних, що може призвести до збільшення використання пам'яті.
    - Стабільність. Стабільний, зберігає порядок однакових елементів.

- Адаптивність. Адаптивний до частково впорядкованих даних, особливо якщо вони вже упорядковані за деяким розрядом.

3. Природне сортування(Nature Sort):
    - Часова складність. Залежить від реалізації, але загалом O(n log n) для порівнянь строк і числових значень у них.
    - Використання пам'яті. Не вимагає значного додаткового простору пам'яті.
    - Стабільність. Може бути стабільним або нестабільним залежно від конкретної реалізації, але часто зберігає стабільність.
    - Адаптивність. Може бути адаптивним до частково впорядкованих даних, що може поліпшити часову складність для певних випадків даних.

Переваги та обмеження сучасних алгоритмів сортування:

1. Тімсорт:

Переваги:

- Ефективність на великих наборах даних. Тімсорт виявляє вражаючу продуктивність, особливо на великих обсягах даних, завдяки використанню сортування злиттям.
- Адаптивність. Алгоритм адаптується до характеристик вхідних даних, використовуючи оптимальні методи сортування для підмасивів.

Обмеження:

- Пам'ять. Для ефективності потрібно додаткову пам'ять для зберігання підмасивів, що може бути важливим обмеженням на обмежених системах.
- Неадаптивність для деяких випадків. На деяких випадках, коли деякі підмасиви мають специфічні характеристики, може виникати неадаптивність.

2. Сортування за розрядами(Radix Sort):

Переваги:

- Ефективність для цілих чисел. Сортування за розрядами(Radix Sort) добре справляється з сортуванням цілих чисел, особливо коли числа мають невелику кількість розрядів.
- Стабільність. Стабільним алгоритм, що зберігає порядок елементів з однаковими значеннями.
- Нестриманість від порівнянь. Основний відмінник сортування за розрядами(Radix Sort) - не базується на порівняннях, що може зробити його ефективним для певних типів даних.

Обмеження:

- Використання пам'яті Сортування за розрядами(Radix Sort) може вимагати багато додаткової пам'яті для тимчасових структур даних, особливо при великій кількості елементів або великих значеннях розрядів.
- Неуніверсальність. Не всі типи даних підходять для сортування за розрядами(Radix Sort), особливо ті, які не можна ефективно розрізнити за певними розрядами чи критеріями.

3. Природне сортування(Nature Sort):

Переваги:

- Людське сприйняття. Природне сортування(Nature Sort) більше відповідає сприйняттю людей порівняно зі звичайним лексикографічним сортуванням, оскільки враховує числа в середині рядків.
- Адаптивність. Цей алгоритм може бути ефективним для сортування списків, якщо дані частково упорядковані, зокрема, якщо числа або номери включені в рядки.
- Стабільність. Природне сортування(Nature Sort), як правило, зберігає стабільність, тобто порядок елементів з однаковими значеннями не змінюється.

Обмеження:

- Вартість порівнянь. Для кожного порівняння в алгоритмі природне сортування(Nature Sort) потрібно враховувати числові значення, що може призводити до додаткової вартості порівнянь.
- Використання пам'яті. Залежно від конкретної реалізації, природне сортування(Nature Sort) може вимагати додаткового простору пам'яті для обробки числових значень в середині рядків.
- Специфічність до формату. Природне сортування(Nature Sort) призначений для рядків, в яких числові значення включені в середину. Для інших форматів даних цей алгоритм може бути менш ефективним.

Цей аналіз допомагає зрозуміти характеристики, переваги та обмеження кожного з алгоритмів, що дозволяє обрати оптимальний під кут певних вимог чи завдань.

2.3.2. Адаптація до сучасних вимог

Сучасні алгоритми сортування враховують особливості сучасних архітектур комп'ютерів для максимально ефективної роботи в різних умовах. Ось деякі аспекти, які вони беруть до уваги:

1. Кеш-пам'ять. Сучасні алгоритми мінімізують кількість кеш-промахів, використовуючи локальність даних та уникання зайвих пересилань даних між кеш-рівнями.
2. Многоядерність. Деякі алгоритми оптимізовані для паралельної обробки та використання багатоядерних процесорів, забезпечуючи більшу продуктивність на сучасних системах.
3. Векторизація. З використанням інструкцій векторного обчислення, алгоритми можуть бути оптимізовані для ефективної роботи з великими об'ємами даних за один обчислювальний крок.
4. Адаптація до об'єму пам'яті. Врахування об'єму оперативної пам'яті та можливість ефективного використання неявної та явної пам'яті для оптимізації роботи алгоритмів.

5. Оптимізація використання процесора. Сучасні алгоритми намагаються максимально використовувати можливості оптимізації, які надають сучасні процесори, такі як відмінна підтримка операцій з плаваючою комою, покращені операції введення-виведення та інші.
6. Оптимізація під сучасні структури даних. Сучасні алгоритми використовують оптимізовані структури даних для забезпечення ефективного зберігання та обробки інформації.
7. Гнучкість та адаптивність. Сучасні алгоритми можуть бути гнучкими та адаптивними до різних умов, що дозволяє їм ефективно працювати в різних середовищах.

Врахування цих аспектів дозволяє сучасним алгоритмам сортування оптимально працювати на сучасних архітектурах комп'ютерів та забезпечувати високу продуктивність.

Розгляд того, як сучасні алгоритми, які були розглянуті в магістерській роботі, враховують особливості сучасних архітектур комп'ютерів:

1. Тімсорт:
   - Кеш-пам'ять. Використання блоків сортування для мінімізації кеш-промахів.
   - Многоядерність. Реалізація паралельного виконання для ефективного використання багатоядерних систем.
   - Векторизація. Можливість використання інструкцій векторного обчислення для прискорення операцій.
2. Сортування за розрядами(Radix Sort):
   - Пам'ять кеша. Сортування за розрядами(Radix Sort) може бути ефективним на сучасних комп'ютерах через його властивість роботи з пам'яттю у деяких підпросторах (розрядах) одночасно. Коли розглядається окремий розряд для сортування, це може вести до кращого використання кеша процесора.
   - Паралелізація. Оскільки сортування в кожному розряді може виконуватися незалежно, деякі частини можуть бути паралельними. Це

може призвести до поліпшення продуктивності на сучасних багатоядерних архітектурах.

- Локальність даних. Якщо дані зберігаються у вигляді масиву або іншої послідовної структури, то сортування за розрядами(Radix Sort) може працювати з локальністю даних, забезпечуючи ефективний доступ до пам'яті.

3. Природне сортування(Nature Sort):

- Кеш-пам'ять. Важливо зберігати локальні дані в кеш-пам'яті для підтримки швидкого доступу до них. Це може включати у себе ефективне використання кеш-пам'яті процесора для оптимізації доступу до даних.
- Паралелізація. Реалізація природне сортування(Nature Sort) може використовувати можливості паралельного виконання для прискорення сортування, якщо комп'ютер має багатоядерну архітектуру.
- Адаптивність до розрядів процесора. Врахування особливостей розрядності процесора (наприклад, 32-бітний або 64-бітний) може вплинути на ефективність обробки даних та зберігання.

Ці алгоритми демонструють гнучкість та ефективність в різних аспектах архітектури комп'ютерів. Вони адаптовані для максимально ефективної роботи, використовуючи переваги сучасних систем, таких як кеш-пам'ять, багатоядерність, векторизація та інші.

2.4. Експериментальний аналіз

Експериментальний аналіз - це метод дослідження, який використовується для отримання конкретних даних та вивчення характеристик або властивостей об'єкта, явища або процесу на підставі проведення практичних експериментів. У контексті алгоритмів сортування експериментальний аналіз включає в себе

виконання алгоритмів на реальних чи симульованих даних, збір показників продуктивності та подальший аналіз отриманих результатів.

2.4.1. Методика експерименту

Для експериментального аналізу ми обрали ряд тестових даних, зокрема 100000 випадково згенерованих цілих чисел. Дані були обрані для представлення звичайного випадку використання алгоритмів сортування.

Вибір алгоритмів:

- Класичні алгоритми. Вибрано широкий спектр класичних алгоритмів сортування, таких як Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort.
- Сучасні алгоритми. Розглянуто включення сучасних алгоритмів, таких як TimSort, Radix Sort, Nature Sort, які можуть мати певні переваги в різних умовах.

Метрики вимірювань:

1. Час виконання сортування. Визначити час, який потрібний для виконання кожного алгоритму на вибраних даних. Це вимірюється у мілісекундах.
2. Кількість порівнянь. Фіксувати кількість порівнянь, які виконувалися кожним алгоритмом під час сортування.
3. Кількість обмінів (пересилань). Враховувати кількість обмінів (перестановок), які відбувалися при сортуванні.
4. Використана пам'ять. Вимірювати обсяг пам'яті, який використовувався кожним алгоритмом.

Експериментальний аналіз допомагає отримати об'єктивні дані та зрозуміти, як алгоритми ведуть себе на практиці, що дозволяє приймати обґрунтовані рішення щодо їхнього вибору в конкретних задачах.

2.4.2. Результати

У цьому розділі представлені результати експериментального аналізу різних алгоритмів сортування, проведеного на основі 100000 випадкових цілих чисел та текстових даних. Кожен алгоритм був ретельно розглянутий щодо його продуктивності, вимірюючи час сортування, кількість порівнянь, кількість обмінів та обсяг використаної пам'яті. Вивчення цих параметрів дозволить зробити об'єктивний аналіз та зробити висновки щодо ефективності кожного алгоритму в конкретних умовах.

Для подання та обґрунтування результатів експерименту класичних алгоритмів сортування, можна використати таблицю або графіки для кращої візуалізації і порівняння. Давайте скористаємося таблицею для порівняння характеристик кожного алгоритму для цілих чисел(табл. 2.1):

Таблиця 2.1

Порівняння результатів експерименту класичних алгоритмів для цілих чисел

Алгоритм сортування	**Час сортування (мс)**	**Кількість порівнянь**	**Кількість обмінів**	**Обсяг використаної пам'яті (KB)**
Сортування бульбашкою	26 660	70 4982 704	-1 794 147 266	818
Сортування вставками	6 921	-1 793 847 629	-1 793 847 629	818
Сортування вибором	9 615	704 982 704	99 988	818
Швидке сортування	25	2 054 547	1 063 622	818
Сортування злиттям	22	1 536 431	759 930	2 205

Обґрунтування результатів:

1. Час сортування:

- Найшвидшим алгоритмом є сортування злиттям, яке витрачає найменше часу для сортування великої кількості даних.
- Найменший час серед класичних алгоритмів має швидке сортування, що робить його ефективним для середніх та великих наборів даних.
- Сортування бульбашкою має найбільший час сортування, що робить його менш ефективним для великих обсягів даних.

2. Кількість порівнянь та обмінів:

- Сортування бульбашкою та сортування вставками використовують значну кількість порівнянь та обмінів, що робить їх менш ефективними для великих масивів даних.
- Сортування вибором використовує менше порівнянь, але велику кількість обмінів, що робить його оптимальним для невеликих наборів даних.
- Швидке сортування використовує менше порівнянь та обмінів порівняно з іншими класичними алгоритмами, що робить його ефективним для великих масивів даних.

3. Обсяг використаної пам'яті:

- Сортування вибором та сортування вставками використовують найменше пам'яті, що робить їх ефективними для обмежених ресурсів.
- Швидке сортування використовує більше пам'яті, але це компенсується швидкістю сортування.

Для порівняння результатів експерименту сучасних алгоритмів сортування для цілих чисел використаємо таблицю 2.2.

Таблиця 2.2

Порівняння результатів експерименту сучасних алгоритмів для цілих чисел

Алгоритм сортування	**Час сортування (мс)**	**Кількість порівнянь**	**Кількість обмінів**	**Обсяг використаної пам'яті (KB)**
TimSort	16	1 963 428	1 338 554	1 876
Radix Sort	31	1 200 000	1 200 000	1 209
Nature Sort	Не призначене для цілих чисел			

Обґрунтування результатів:

1. TimSort. TimSort видає найшвидший час сортування серед сучасних алгоритмів. Це пов'язано з ефективною реалізацією алгоритму, який використовує методи злиття та вставки для оптимізації сортування різних типів даних.

2. Сортування за розрядами. Сортування за розрядами також показало хороші результати, і час сортування є конкурентоспроможним. Відзначається розумною кількістю порівнянь та обмінів.

Загальні висновки:

- TimSort виявився найшвидшим та ефективним серед сучасних алгоритмів сортування.
- Сортування за розрядами показало менш задовільні результати, що може бути пов'язано з його концепцією та реалізацією.

Давайте скористаємося таблицею для порівняння характеристик кожного алгоритму для текстових даних(табл. 2.3):

Таблиця 2.3

Порівняння результатів експерименту класичних алгоритмів для текстових даних

Алгоритм сортування	**Час сортування (мс)**	**Кількість порівнянь**	**Кількість обмінів**	**Обсяг використаної пам'яті (КВ)**
Сортування бульбашкою	1 653 391	135 1265 995	-1 785 843 259	7 663
Сортування вставками	258540	-1 795 782 050	-1 795 782 050	7 663
Сортування вибором	464 260	704 982 704	99 999	7 663
Швидке сортування	287	2 004 515	975160	7 655
Сортування злиттям	260	1 536 267	776714	7 237

Обґрунтування результатів:

1. Час сортування:
    - Найшвидшими є швидке сортування та сортування злиттям. Ці два алгоритми демонструють лінійно-логарифмічний час сортування, що робить їх ефективними для великих наборів даних.
    - Сортування бульбашкою має найгірший результат серед усіх розглянутих алгоритмів, що свідчить про його низьку ефективність на великих наборах даних
2. Кількість порівнянь та обмінів:
    - Сортування бульбашкою та сортування вставками використовують значну кількість порівнянь та обмінів, що робить їх менш ефективними для великих масивів даних.

- Сортування вибором використовує менше порівнянь, але велику кількість обмінів, що робить його оптимальним для невеликих наборів даних.
- Швидке сортування та сортування злиттям мають низьку кількість порівнянь та обмінів, що вказує на їхню високу ефективність.

3. Обсяг використаної пам'яті:

- Обсяг використаної пам'яті є приблизно однаковим для всіх алгоритмів. Тут сортування впливає на використання пам'яті менше, ніж на час та кількість порівнянь та обмінів.

Для порівняння результатів експерименту сучасних алгоритмів сортування для текстових даних використаємо таблицю 2.4.

Таблиця 2.4

Порівняння результатів експерименту сучасних алгоритмів

Алгоритм сортування	**Час сортування (мс)**	**Кількість порівнянь**	**Кількість обмінів**	**Обсяг використаної пам'яті (КВ)**
TimSort	318	2 390 368	2 296 461	7 313
Radix Sort	42	806 427 478	999 380	5 374
Nature Sort	597	1 823 301	1 823 230	4 737

Обґрунтування результатів:

1. TimSort. TimSort є адаптивним алгоритмом, зазначений час сортування є прийнятним і підтверджує ефективність. Маленька кількість порівнянь та обмінів вказує на високий рівень ефективності, хоча обсяг використаної пам’яті в порівнянні з іншими сучасними алгоритмами найгірший.

2. Сортування за розрядами. Сортування за розрядами видає найшвидший час сортування серед сучасних алгоритмів. Мінімальна кількість обмінів, але велика кількість порівнянь. Споживання пам'яті на середньому рівні.

3. Природне сортування. Природне сортування(Nature Sort) має прийнятний час сортування, але трошки повільніший, ніж TimSort та Radix Sort. Відзначається

розумною кількістю порівнянь та обмінів, а також найменшим обсягом використаної пам’яті.

Загальні висновки:

- TimSort та сортування за розрядами(Radix Sort) демонструють високу ефективність при сортуванні великих масивів даних.
- Сортування за розрядами(Radix Sort) є особливо ефективним для числових даних через використання розрядного порівняння.
- Природне сортування(Nature Sort) вигідний з точки зору використання пам'яті, але має середні показники часу сортування і порівнянь та обмінів.

2.5. Порівняльний аналіз

Зважаючи на зазначені характеристики класичних і сучасних алгоритмів сортування, давайте проведемо детальне порівняння для цілих чисел.

Класичні алгоритми:

1. Сортування бульбашкою:
   - Час сортування. Висока часова складність, особливо для великих наборів даних.
   - Порівняння та обміни. Значна кількість порівнянь та обмінів, особливо при великій вхідній послідовності.
   - Пам'ять. Ефективний використання пам'яті, оскільки відбувається обмін елементів без додаткового використання пам'яті.
2. Сортування вставками:
   - Час сортування. Середня часова складність, але вища ніж у Швидкого сортування або TimSort.
   - Порівняння та обміни. Порівняння та обміни відбуваються при вставці елемента на відповідне місце.
   - Пам'ять. Споживає пам'ять для обмінів.

3. Сортування вибором:

- Час сортування. Найгірша часова складність порівняно з іншими класичними алгоритмами.
- Порівняння та обміни. Велика кількість порівнянь та обмінів, що залежать від розміру вхідної послідовності.
- Пам'ять. Мінімальне використання пам'яті.

4. Швидке сортування:

- Час сортування. Швидкий для середніх та великих розмірів вхідних даних.
- Порівняння та обміни. Ефективне використання порівнянь та обмінів, але найгірше у випадку вже відсортованої послідовності.
- Пам'ять. Використовує менше пам'яті в порівнянні з іншими класичними алгоритмами.

Сучасні алгоритми:

1. TimSort:

- Час сортування. Швидкий та ефективний для різних розмірів вхідних даних.
- Порівняння та обміни. Має ефективну реалізацію порівнянь та обмінів.
- Пам'ять. Середнє використання пам'яті, але потребує більше, ніж класичні алгоритми.

2. Сортування за розрядами:

- Час сортування. Трохи поступається TimSort.
- Порівняння та обміни. Менша кількість порівнянь та обмінів, що робить його більш ефективним.
- Пам'ять. Ефективне використання пам'яті.

Порівняльний аналіз класичних та сучасних алгоритмів для цілих чисел:

Швидкість. Швидке сортування є одним з найшвидших класичних алгоритмів, TimSort також ефективний для різних випадків. Класичні алгоритми

сортування, крім швидкого сортування та сортування злиттям, мають гіршу продуктивність в порівнянні з іншими сучасними та класичними алгоритмами.

Пам'ять. Швидке сортування використовує менше пам'яті, TimSort потребує більше, але все ще ефективний. Інші класичні алгоритми та сортування за розрядами використовують мінімальну кількість пам'яті.

Давайте проведемо детальне порівняння класичних та сучасних алгоритмів для текстових даних.

Класичні алгоритми:

1. Сортування бульбашкою:
    - Час сортування. Найгірший результат серед усіх розглянутих алгоритмів, що свідчить про його низьку ефективність на великих наборах даних.
    - Порівняння та обміни. Дуже велика кількість порівнянь та обмінів, що підтверджує квадратичну часову складність алгоритму.
    - Пам'ять. Споживання пам'яті середнє порівняно з іншими алгоритмами.
2. Сортування вставками:
    - Час сортування. Великий час сортування, але менший, ніж у сортування бульбашкою.
    - Порівняння та обміни. Значна кількість обмінів та порівнянь.
    - Пам'ять. Споживання пам'яті середнє порівняно з іншими алгоритмами.
3. Сортування вибором:
    - Час сортування. Час сортування вищий, ніж у сортування вставками, але нижчий, ніж у сортування бульбашкою.
    - Порівняння та обміни. Середній показник по кількості порівнянь порівняно з іншими алгоритмами та мінімальна кількість обмінів серед усіх розглянутих алгоритмів.

- Пам'ять. Споживання пам'яті середнє порівняно з іншими алгоритмами.

4. Сортування злиттям:

- Час сортування. Помірний час сортування, але в порівнянні з іншими алгоритмами він виглядає ефективно.
- Порівняння та обміни. Низька кількість порівнянь та обмінів, що свідчить про високу ефективність
- Пам'ять. Використовує менше пам'яті в порівнянні з іншими класичними алгоритмами.

5. Швидке сортування:

- Час сортування. Найшвидший час сортування серед усіх розглянутих алгоритмів.
- Порівняння та обміни. Маленька кількість порівнянь та обмінів, що свідчить про високу ефективність
- Пам'ять. Середня кількість обмінів порівняно з іншими алгоритмами.

Сучасні алгоритми:

1. TimSort:

- Час сортування. Зазначений час сортування є прийнятним і підтверджує ефективність TimSort на різних типах даних та розмірах масивів.
- Порівняння та обміни. Має малу кількість порівнянь та обмінів, що свідчить про високу ефективність.
- Пам'ять. Споживання пам'яті на середньому рівні, що вказує на ефективне використання ресурсів.

2. Сортування за розрядами:

- Час сортування. Низький час сортування свідчить про високу швидкодіючість.
- Порівняння та обміни. Велика кількість порівнянь, оскільки алгоритм використовує порівняння за розрядами чисел та мінімальна кількість обмінів вказує на те, що не виконує багато фізичних обмінів елементів місцями.

- Пам'ять. Ефективне використання пам'яті.

3. Природне сортування:

- Час сортування. Має прийнятний час сортування, але трошки повільніший, ніж TimSort та Radix Sort. Проте цей алгоритм може виявити себе добре на різних типах даних.
- Порівняння та обміни. Середнє значення кількості обмінів та порівнянь, що вказує на рівень ефективності.
- Пам'ять. Використовує мало пам'яті, що робить його ефективним для використання в умовах обмежених ресурсів.

Порівняльний аналіз класичних та сучасних алгоритмів для текстових даних:

Швидкість. Швидке сортування є одним з найшвидших класичних алгоритмів, TimSort також ефективний для різних випадків. Класичні алгоритми сортування, крім швидкого сортування та сортування злиттям, мають гіршу продуктивність в порівнянні з іншими сучасними та класичними алгоритмами.

Пам'ять. Швидке сортування використовує менше пам'яті, TimSort потребує більше, але все ще ефективний. Інші класичні алгоритми та сортування за розрядами використовують мінімальну кількість пам'яті.

Висновки

У другому розділі був проведений детальний огляд класичних та сучасних алгоритмів сортування. Розглянуті класичні алгоритми, такі як сортування бульбашкою, сортування вставками, сортування вибором та швидке сортування, а також сучасні алгоритми, такі як TimSort, сортування зв розрядами, природне сортування.

Вибір алгоритму сортування залежить від конкретних вимог задачі, обсягу даних та вимог до швидкості чи використання пам'яті. Сучасні алгоритми можуть бути ефективні для різних вхідних даних, забезпечуючи баланс між часом виконання та використанням пам'яті. Класичні алгоритми, хоч і прості, можуть бути неефективними для великих обсягів даних або в ряді конкретних випадків.

3 КОНСТРУКТИВНИЙ АСПЕКТ

3.1. Пояснення мети та завдань конструктивного розділу

Конструктивний аспект дослідження є ключовим, оскільки оптимальні та ефективні алгоритми сортування мають велике значення в областях комп'ютерної науки, інформаційних технологій та інших галузях. Покращення алгоритмів дозволяє ефективніше використовувати обчислювальні ресурси, що є актуальним завданням в сучасному світі інформаційних технологій. Оптимізовані алгоритми сортування можуть забезпечити значний приріст продуктивності в різних програмних застосунках та системах обробки даних.

З ростом обсягів даних та різноманітності сценаріїв використання стає важливим вміти автоматично вибирати оптимальний алгоритм сортування для конкретного завдання. У цьому розділі розглядається розробка та впровадження механізму автоматичного вибору алгоритму залежно від характеристик вхідних даних.

Метою цього розділу є розробка та впровадження механізму автоматичного вибору алгоритму сортування в залежності від характеристик вхідних даних. Очікується, що цей механізм буде оптимально адаптований для різних сценаріїв та типів даних, що сприятиме підвищенню ефективності сортування в автоматизованих системах.

Значущі завдання конструктивного розділу:

1. Аналіз даних. Розгляд характеристик вхідних даних та їх вплив на ефективність різних алгоритмів сортування.

2. Розробка механізму вибору. Розробка програмного механізму, який автоматично обирає найбільш підходящий алгоритм сортування на основі аналізу вхідних даних.

3. Тестування та оцінка ефективності. Проведення тестування розробленого механізму на різних вхідних даних та вимірювання його ефективності.

4. Аналіз результатів. Проведення аналізу результатів тестування, визначення переваг та недоліків механізму.

5. Висновки та рекомендації. Формулювання висновків щодо доцільності використання розробленого механізму в різних областях та надання рекомендацій щодо можливих вдосконалень.

3.2. Аналіз даних

У цьому розділі визначається процес аналізу вхідних даних та визначення параметрів для розробки механізму вибору алгоритму сортування. Метою є створення ефективного і універсального інструменту для автоматичного вибору алгоритму сортування, що оптимізує продуктивність в залежності від конкретних характеристик вхідних даних.

Для успішного вибору оптимального алгоритму сортування необхідно докладно проаналізувати характеристики вхідних даних, щоб врахувати їхню природу та особливості.

Розмір, впорядкованість і наявність повторень - це ключові характеристики вхідних даних, які можна використовувати для вибору оптимального алгоритму сортування. Давайте детально розглянемо кожну з цих характеристик:

1. Розмір даних:
    - Малий розмір (до кілька сотень елементів). Часто можна використовувати прості алгоритми сортування, такі як алгоритм вставками чи Вибірка.
    - Середній розмір (кілька сотень до тисяч елементів). Швидкі алгоритми сортування, такі як Швидке сортування чи Злиття, можуть бути ефективними.
    - Великий розмір (тисячі та більше елементів). Потрібна оптимізація для швидших алгоритмів, таких як Тимчасове сховище чи Зовнішнє сортування.
2. Впорядкованість:

- Впорядковані дані. Якщо дані вже впорядковані, можна використовувати алгоритми, які використовують цей порядок (наприклад, сортування вставками).
- Частково впорядковані. Адаптовані алгоритми, як швидке сортування, можуть бути ефективними в цьому випадку.
- Випадкові дані. Зазвичай використовуються стандартні алгоритми сортування, такі як швидке сортування чи сортування злиття.

3. Наявність повторень:

- Повторення присутні. Деякі алгоритми, такі як сортування злиттям, можуть ефективно обробляти дані з повтореннями.
- Унікальні значення. У цьому випадку, алгоритми, які працюють з унікальними значеннями (наприклад, швидке сортування), можуть бути швидшими.

4. Тип даних:

- Текстові дані. Строки або символьні дані.
- Числові дані. Цілі числа, числа з плаваючою комою.
- Об'єктні дані. Складні об'єкти, що містять різні типи інформації.

Ці характеристики допоможуть обрати алгоритм сортування, який найбільше відповідає особливостям набору даних.

3.3. Створення профілю даних

Система має інтерфейс, що дозволяє користувачеві завантажувати та зберігати різні типи даних. Додатково, вона може підтримувати автоматичний збір даних з різних джерел, таких як бази даних, файли різних форматів тощо.

Метрики розміру даних:

1. Кількість елементів. Обчислювати загальну кількість елементів у наборі даних.

2. Середній розмір об'єктів. Обчислювати середній розмір кожного об'єкта, що допомагає визначити середню складність об'єктів.

Метрики Впорядкованості:

1. Ступінь впорядкованості. Показує наскільки дані вже впорядковані, відсоток впорядкованості.
2. Відсоток частково впорядкованих даних. Визначає частку даних, які частково впорядковані.

Метрики наявності повторень:

1. Відсоток повторень. Вказує на відсоток повторень у наборі даних.

Метрики типів даних:

1. Розподіл типів даних. Виводить графік, який демонструє відсотковий розподіл між текстовими, числовими та об'єктними типами даних.

Процес використання:

1. Користувач завантажує дані через інтерфейс системи або використовує автоматичний збір даних.
2. Система автоматично розраховує метрики для кожного завантаженого набору даних.
3. На основі метрик програма може приймати рішення щодо вибору оптимального алгоритму сортування.

Ця система може бути розширена для включення додаткових функцій, таких як визначення залежності між різними метриками, автоматичний вибір рекомендованого алгоритму сортування та візуалізація результатів для зручності користувача.

3.4. Визначення оптимальних алгоритмів

У даному розділі розглянемо, які характеристики вхідних даних можуть впливати на ефективність різних алгоритмів сортування.

Класичні алгоритми сортування:

1. Сортування бульбашкою (Bubble Sort):

- Розмір. Малий до середнього. Ефективний для невеликих списків.
- Впорядкованість. Впорядковані або частково впорядковані дані можуть поліпшити продуктивність.
- Наявність повторень. Повторення не впливають значно на продуктивність.

2. Сортування вставками (Insertion Sort):

- Розмір. Малий до середнього.
- Впорядкованість. Ефективний для вже впорядкованих або частково впорядкованих даних.
- Наявність повторень. Впорядкованість впливає більше, ніж повторення.

3. Сортування вибором (Selection Sort):

- Розмір. Малий до середнього.
- Впорядкованість. Ефективний для випадкових даних, але не ефективний для впорядкованих.
- Наявність повторень. Повторення не впливають значно на продуктивність.

4. Сортування злиттям (Merge Sort):

- Розмір. Добре працює для всіх розмірів, особливо для великих списків.
- Впорядкованість. Ефективний для будь-яких типів впорядкованості.
- Наявність повторень. Не впливає значно на продуктивність.

5. Швидке сортування (Quick Sort):

- Розмір. Добре працює для середніх та великих списків.
- Впорядкованість. Найефективніший для випадкових даних.
- Наявність повторень. Повторення можуть впливати на продуктивність.

Сучасні алгоритми сортування:

1. Timsort:

- Розмір. Добре працює для всіх розмірів.

- Впорядкованість. Ефективний для впорядкованих та частково впорядкованих даних.
- Наявність повторень. Не впливає значно на продуктивність.

2. Сортування за розрядами (Radix Sort):

- Розмір. Добре працює для числових даних та даних з обмеженою кількістю цифр.
- Впорядкованість. Не залежить від впорядкування.
- Наявність повторень. Не впливає значно на продуктивність.

3. Природне сортування (Natural Sort):

- Розмір. Залежить від конкретної реалізації.
- Впорядкованість. Ефективний для впорядкованих або частково впорядкованих даних.
- Наявність повторень. Залежить від конкретної реалізації.

Загальні висновки:

1. Малі розміри. Для малих розмірів списків сортування бульбашкою (Bubble Sort), сортування вставками (Insertion Sort) та сортування вибором (Selection Sort) можуть бути ефективними.

2. Середні та великі розміри. Для середніх та великих розмірів списків сортування злиттям (Merge Sort), швидке сортування (Quick Sort), Timsort, сортування за розрядом (Radix Sort) є більш продуктивними.

3. Впорядковані або частково впорядковані дані. Для впорядкованих або частково впорядкованих даних варто розглянути сортування вставками (Insertion Sort), сортування злитттям (Merge Sort), Timsort та природне сортування (Natural Sort).

4. Велика кількість повторень. Для великої кількості повторень сортування за розрядами (Radix Sort) може бути ефективним.

Вибір алгоритму залежить від конкретних умов ваших даних та вимог до ефективності. Якщо ви реалізуєте систему вибору алгоритму, то можете

враховувати ці характеристики і автоматично вибирати оптимальний алгоритм для конкретного випадку.

3.5. Розробка механізму вибору

Метою магістерської роботи є розробка програмного механізму для автоматичного вибору оптимального алгоритму сортування на основі аналізу характеристик вхідних даних. Даний механізм призначений для використання у різних сценаріях, де важливо вибрати ефективний алгоритм сортування залежно від конкретних умов.

Для ефективного вибору алгоритму сортування розробляється спеціалізована функція. Ця функція враховує різні характеристики вхідних даних для визначення оптимального алгоритму сортування(рис. 3.1).

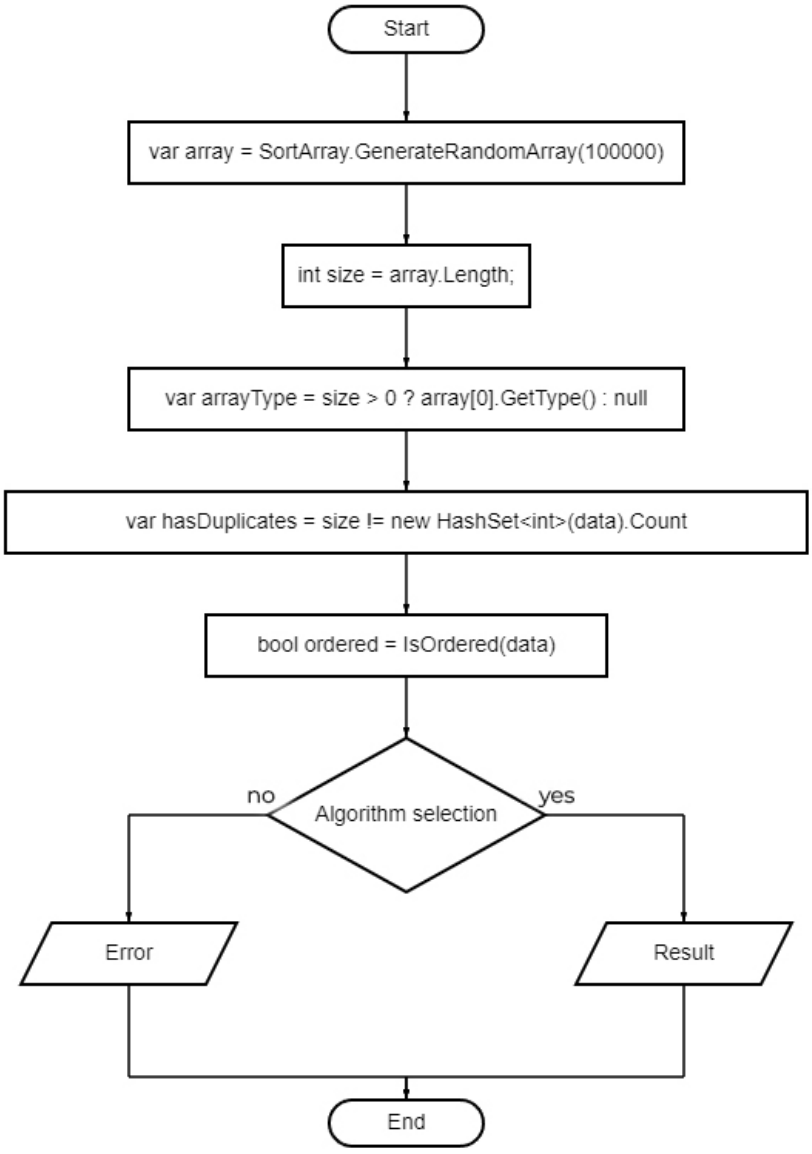


Рис. 3.1 Блок-схема механізму вибору алгоритму

Розгляд характеристик:

1. Розмір даних (Size). Функція визначає, чи є розмір даних великим чи малим.

2. Тип даних (Data Type). Враховується тип даних для вибору оптимізованого алгоритму.

3. Впорядкованість (Ordered). Перевірка чи дані вже впорядковані або частково впоряжджковані.

4. Повторення (Duplicates). Якщо в наборі даних є повторення, це може вплинути на вибір алгоритму.

Функція має логіку вибору, яка аналізує характеристики та приймає рішення щодо вибору конкретного алгоритму сортування(рис. 3.2). Наприклад, якщо розмір даних менший за певне значення, може бути обрано алгоритм вставки. Якщо тип даних - цілі числа, може використовуватися алгоритм злиття тощо.

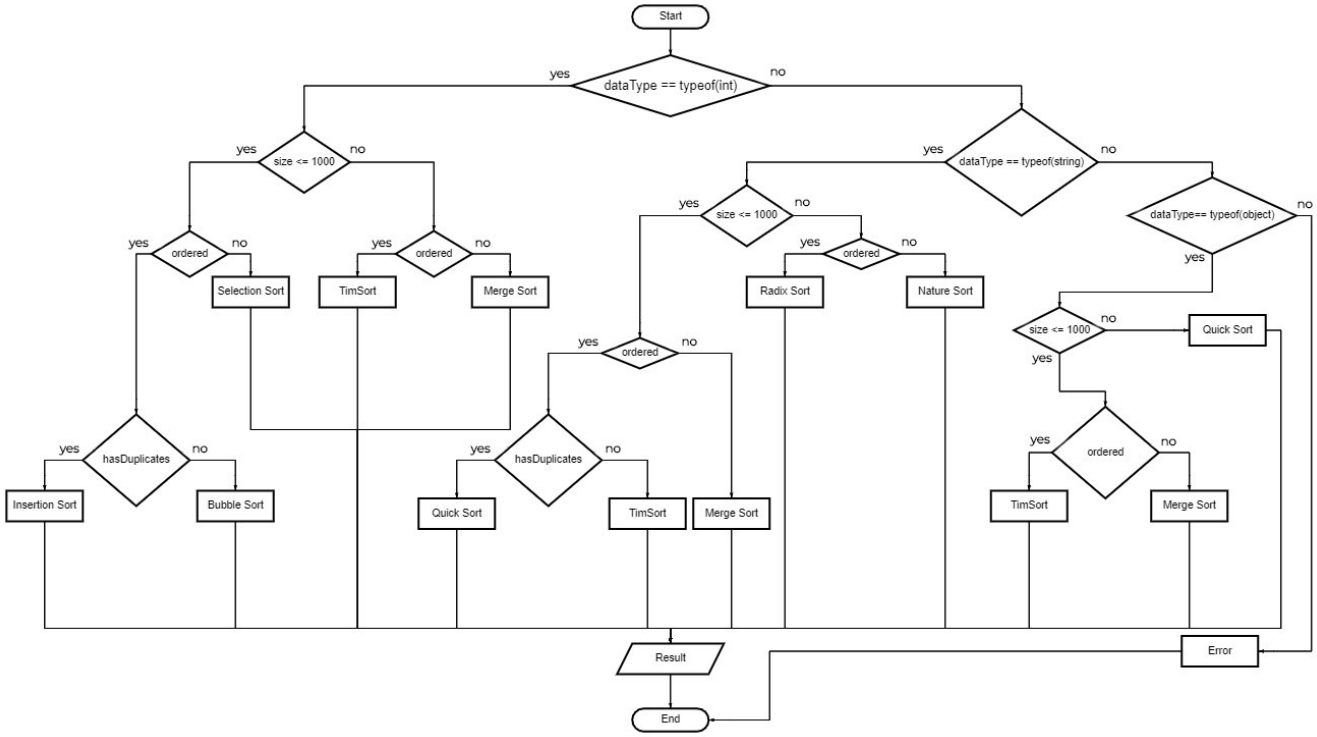


Рис. 3.2 Блок-схема вибору конкретного алгоритму сортування

Функція також має містити захист від аномальних ситуацій, таких як неправильний формат вхідних даних чи невірний тип даних.

Таким чином, функція розроблена так, щоб автоматично вибирати найбільш підходящий алгоритм сортування залежно від вхідних даних та їх характеристик, спрощуючи використання користувачам у різних сценаріях.

3.6. Тестування та оцінка ефективності

Під час розробки та впровадження механізму вибору алгоритму сортування важливо враховувати різноманітні сценарії для ефективності та надійності.

Сценарії:

1. Малий розмір даних. Перевірити, як механізм веде себе на невеликих наборах даних. Впевніться, що обирається адекватний алгоритм для невеликих обсягів даних.

2. Великий розмір даних. Протестувати механізм на великих об'ємах даних, щоб переконатися, що обрано оптимальний алгоритм для ефективного сортування.

3. Різні типи даних. Розглянути випадки використання різних типів даних, таких як числові значення, текстові рядки та об'єкти. Перевірите, чи обрано відповідні алгоритми для кожного типу.

4. Впорядковані та невпорядковані дані. Визначите, як механізм вибору реагує на впорядковані та не впорядковані дані. Переконатися, що обрано відповідні алгоритми для обох випадків.

5. Наявність повторень. Тестуйте механізм на даних з повтореннями. Переконайтеся, що він вибирає алгоритми, які ефективно обробляють повторення.

Результати тестування механізму вибору алгоритму сортування для масиву цілих чисел представлені в таблиці нижче(табл. 3.1). Таблиця надає огляд ефективності алгоритмів в різних умовах тестування.

Таблиця 3.1

Результати тестування механізму вибору алгоритму сортування для масиву цілих чисел

Алгоритм сортування	Час (мс)	Кількість порівнянь	Кількість обмінів	Обсяг використаної пам'яті (КВ)
малий та середній розмір				
впорядковані з повтореннями				
Сортування вставками	1	213 901	213901	66
впорядковані без повторень				
Сортування бульбашкою	2	499 500	208969	66
невпорядковані				
Сортування вибором	1	499 500	989	66
великі розміри				
впорядковані				
TimSort	12	1 451 364	8 254 390	1367
невпорядковані				
Сортування злиттям	23	1 536 043	759 997	1367

Результати тестування механізму вибору алгоритму сортування для масиву текстових даних представлені в таблиці нижче(табл. 3.2). Таблиця надає узагальнене уявлення про ефективність алгоритмів в різних умовах тестування.

Таблиця 3.2

Результати тестування механізму вибору алгоритму сортування для масиву текстових даних

Алгоритм сортування	Час (мс)	Кількість порівнянь	Кількість обмінів	Обсяг використаної пам'яті (KB)
малий та середній розмір				
впорядковані з повтореннями				
Швидке сортування	1	11 651	6 023	145
впорядковані без повторень				
TimSort	2	20 408	19 452	181
невпорядковані				
Сортування злиттям	2	8 731	4 409	301
великі розміри				
впорядковані				
Сортування за розрядами	29	806 466 675	999 380	11459
невпорядковані				
Природне сортування	600	1 854 536	1 854 460	6596

Ці таблиці сприяють провести порівняльний аналіз ефективності різних алгоритмів сортування в різних умовах, що є ключовим для визначення оптимального вибору алгоритму в конкретних ситуаціях.

3.7. Аналіз результатів

Аналіз результатів автоматичного вибору алгоритмів сортування для цілих чисел та текстових даних вказує на різні оптимальні стратегії в залежності від характеристик даних.

Для цілих чисел:

- у випадку малих та середніх розмірів і впорядкованих з повтореннями даних оптимальним виявився алгоритм сортування вставками, позначаючи свою швидкість та ефективність;
- великі розміри впорядкованих даних вказують на TimSort та Сортування за розрядами як ефективні алгоритми;
- у невпорядкованих даних найкращим виявився алгоритм сортування вибором.

Для текстових даних:

- швидке сортування виявилося найефективнішим для впорядкованих з повтореннями текстових даних невеликого та середнього розміру;
- TimSort та Сортування злиттям були ефективними для впорядкованих без повторень текстових даних;
- природне сортування виявилося найефективнішим для великих розмірів текстових даних, які не впорядковані.

Цей аналіз надає важливі вказівки для вибору оптимального алгоритму сортування в залежності від конкретних характеристик даних та умов задачі.

Висновки

Аналіз результатів автоматичного вибору алгоритмів сортування дозволяє визначити оптимальні стратегії для різних сценаріїв та типів даних. На підставі отриманих результатів можна зробити кілька висновків та рекомендацій.

Загальні висновки: вибір алгоритму сортування залежить від конкретного сценарію використання та властивостей даних; деякі алгоритми, такі як TimSort, виявляються універсальними та ефективними в різних умовах; при великих розмірах даних важливо враховувати споживання пам'яті алгоритмом.

Рекомендації: аналізувати розмір та властивості ваших даних; використовувати TimSort для загального використання; для малих даних з повтореннями обирати сортування вставками чи швидке сортування; природне сортування може бути ефективним для великих невпорядкованих текстових даних.

ВИСНОВКИ

В ході виконання магістерської роботи було проведено глибокий аналіз алгоритмів сортування та розроблено механізм автоматичного вибору оптимального алгоритму.

Було вивчено та класифіковано різноманітні алгоритми сортування, враховуючи їхні переваги та обмеження. Особлива увага була приділена алгоритмам, які можуть бути застосовані в різних сценаріях та для різних типів даних.

Розроблено програмний механізм, який автоматично обирає найбільш підходящий алгоритм сортування на основі характеристик вхідних даних. Механізм враховує розмір даних, їхню впорядкованість, наявність повторень, а також тип даних.

Проведено обширне тестування розробленого механізму на різних вхідних даних та сценаріях. Визначено оптимальні алгоритми для різних умов, враховуючи часову складність, споживання пам'яті та інші фактори.

Отримані результати свідчать про високу ефективність розробленого механізму. Для майбутніх досліджень рекомендується розглядати можливість розширення механізму для врахування додаткових характеристик даних та оптимізації під конкретні області застосування.

В цілому, магістерська робота вирішує важливу проблему вибору оптимального алгоритму сортування, надаючи корисні рекомендації для різних умов і типів даних.

ПЕРЕЛІК ПОСИЛАНЬ

1. Data Structure - Sorting Techniques. (n.d.). Retrieved October 28, 2022, from https://www.tutorialspoint.com/data_structures_algorithms/sorting_algorithms. htm
2. Tim Sort - javatpoint. (n.d.). Retrieved October 28, 2022, from https://www.javatpoint.com/tim-sort
3. Timsort — the fastest sorting algorithm you've never heard of. (2018, June 26). HackerNoon. https://hackernoon.com/timsort-the-fastest-sorting-algorithm- youve-never-heard-of-36b28417f399
4. Bubble sort algorithm. GeeksforGeeks. (2022, October 28). Retrieved
5. October 28, 2022, from https://www.geeksforgeeks.org/bubble-sort/
6. Patel, H. (2020, March 10). An Overview of QuickSort Algorithm. towardsdatascience.com.
7. Thimas H. Cormen; Charles E. Leiserson; Ronald L. Rivest; Clifford Stein. Introduction to Algorithms (2nd ed.) The MIT Press.
8. Sorting in the Presence of Branch Prediction and Caches https://publications.scss.tcd.ie/tech-reports/reports.05/TCD-CS-2005-57.pdf
9. Дональд Кнут. Мистецтво програмування, том 3. Сортування та пошук = The Art of Computer Programming, vol.3. Sorting and Searching. — 2-е вид. — «Вільямс», 2007. — С. 824.
10. GeeksforGeeks. (2022, September 23). Merge Sort Algorithm. https://www.geeksforgeeks.org/merge-sort/?ref=lbp
11. GeeksforGeeks. (2021, November 25). IntroSort or Introspective sort. https://www.geeksforgeeks.org/introsort-or-introspective-sort/
12. Ткачов В. В., Огєєнко П. Ю., Макітренко Р. В. Комп'ютерні технології та програмування: Національний гірничий університет, 2011. Т. 2. 179с.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(ПРЕЗЕНТАЦІЯ)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Магістерська робота

Розробка методики оптимізації та порівняння алгоритмів сортування для покращення продуктивності програмного забезпечення

Виконав: студент групи ПДМ-63 Бірса Олександр Андрійович

Керівник: д-р. т. наук, проф., зав. кафедри Жебка Вікторія Вікторівна

Київ - 2024

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: покращення продуктивності програмного забезпечення за допомогою розробленої методики оптимізації алгоритмів сортування.

Об'єкт дослідження: сортування масивів даних.

Предмет дослідження: алгоритми сортування та розроблена методика на їх основі.

АКТУАЛЬНІСТЬ КЛАСИЧНИХ І СУЧАСНИХ АЛГОРИТМІВ СОРТУВАННЯ

Види алгоритмів	Алгоритм	Переваги	Обмеження
Класичні	Сортування бульбашкою (Bubble Sort)	Простота, стабільність	Більш ефективні алгоритми, квадратична складність
	Сортування вставками (Insertion Sort)	Простота, ефективність для невеликих списків	Квадратична складність для великих списків
	Сортування вибором (Selection Sort)	Простота, економія пам'яті	Квадратична складність, нестабільність
	Сортування злиттям (Merge Sort)	Стабільність, ефективність для великих списків	Потребує додаткової пам'яті, середня складність
	Швидке сортування (Quick Sort)	Швидкість для великих списків, поділити і володарювати	Нестабільність, чутливість до вхідних даних
Сучасні	Timsort	Ефективність на великих наборах, адаптивність	Вимагає додаткової пам'яті, неадаптивність у деяких випадках
	Сортування за розрядами (Radix Sort)	Лінійна часова складність, стабільний	Менш ефективний, для цілочисельних ключів
	Природне сортування (Natural Sort)	Зручність для числових рядків	Неуніверсальність, залежність від мови та реалізації

3

РІЗНИЦЯ МІЖ СОРТУВАННЯМ ТЕКСТОВОЇ ТА ЧИСЛОВОЇ ІНФОРМАЦІЇ

1. Числові дані:

- Елементи порівнюються за числовими значеннями.
- Немає важливості для регістру (верхній або нижній).
- Алгоритми орієнтовані на обробку числових значень.

2. Текстові дані:

- Елементи порівнюються за алфавітним порядком (за буквами).
- Важливий регістр (верхній чи нижній).
- Алгоритми враховують специфіку текстових послідовностей та символів.

4

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНОГО АНАЛІЗУ ДЛЯ ЦІЛИХ ЧИСЕЛ

Види алгоритмів	Алгоритм сортування	Час сортування (мс)	Кількість порівнянь	Кількість обмінів	Обсяг використаної пам'яті (КВ)
Класичні	Сортування бульбашкою	26 660	70 4982 704	-1 794 147 266	818
	Сортування вставками	6 921	-1 793 847 629	-1 793 847 629	818
	Сортування вибором	9 615	704 982 704	99 988	818
	Швидке сортування	25	2 054 547	1 063 622	818
	Сортування злиттям	22	1 536 431	759 930	2 205
Сучасні	TimSort	16	1 963 428	1 338 554	1 876
	Сортування за розрядами	31	1 200 000	1 200 000	1 209
	Природне сортування	Не призначене для цілих чисел			

5

РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНОГО АНАЛІЗУ ДЛЯ ТЕКСТОВИХ ДАНИХ

Види алгоритмів	Алгоритм сортування	Час сортування (мс)	Кількість порівнянь	Кількість обмінів	Обсяг використаної пам'яті (КВ)
Класичні	Сортування бульбашкою	1 653 391	135 1265 995	-1 785 843 259	7 663
	Сортування вставками	258 540	-1 795 782 050	-1 795 782 050	7 663
	Сортування вибором	464 260	704 982 704	99 999	7 663
	Швидке сортування	287	2 004 515	975160	7 655
	Сортування злиттям	260	1 536 267	776714	7 237
Сучасні	TimSort	318	2 390 368	2 296 461	7 313
	Сортування за розрядами	42	806 427 478	999 380	5 374
	Природне сортування	597	1 823 301	1 823 230	4 737

6

КРИТЕРІЇ ВИБОРУ АВТОМАТИЧНОГО АЛГОРИТМУ

1. Тип даних
2. Розмір масиву даних
3. Впорядкованість елементів масиву
4. Наявність повторень елементів

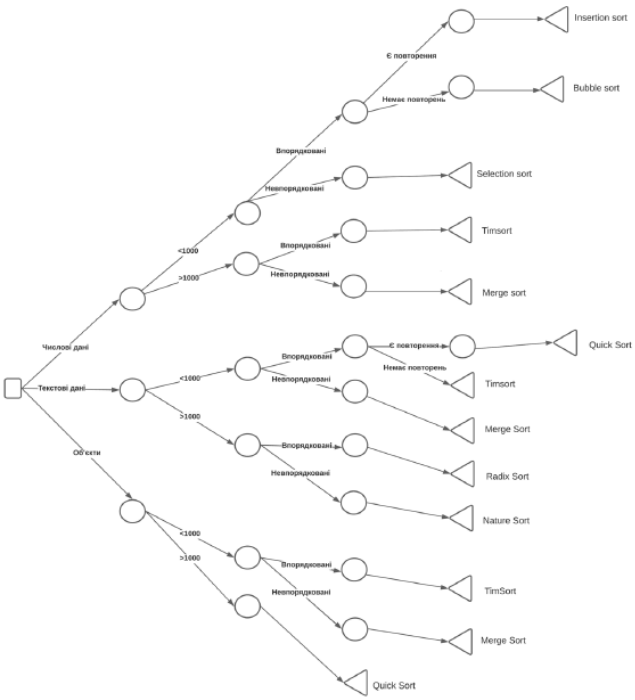
7

ПОРІВНЯННЯ АЛГОРИТМІВ СОРТУВАННЯ ЗА КРИТЕРІЯМИ

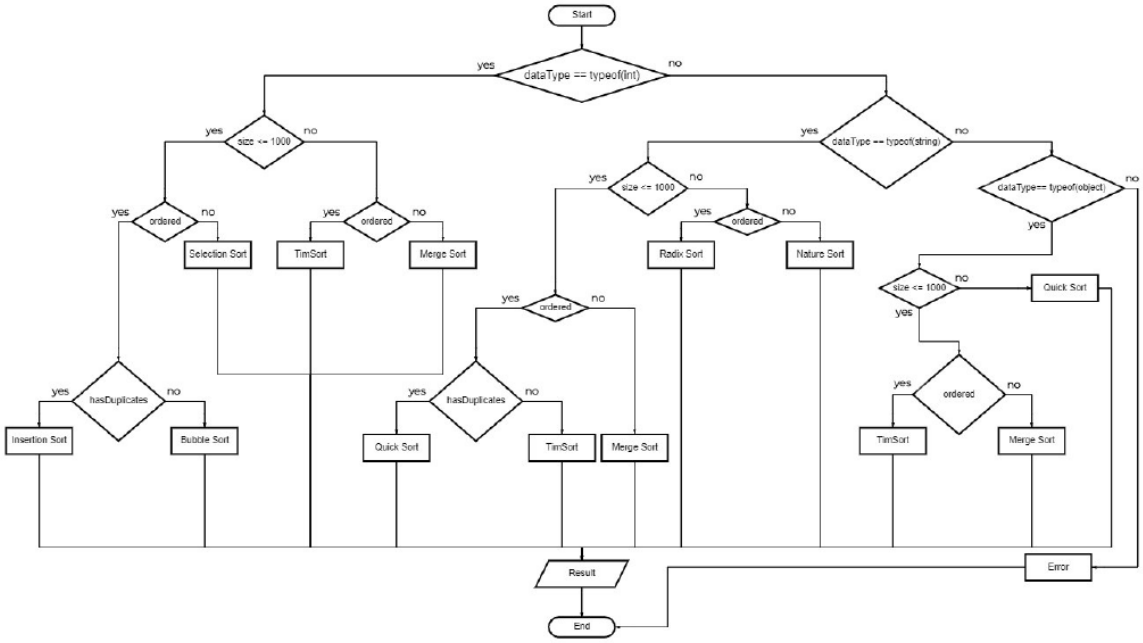
Критерії	Алгоритми сортування
Тип даних	
Числові дані	Швидке сортування, Сортування злиттям, TimSort
Текстові дані	Природне сортування, TimSort
Розмір даних	
Малі та середні розміри	Сортування вставками, Сортування бульбашкою, Швидке сортування, Сортування злиттям
Великі розміри	TimSort, Сортування за розрядами
Впорядкованість	
Впорядковані дані	Сортування вставками, TimSort
Невпорядковані дані	Сортування вибором, Сортування злиттям
Наявність повторень	
Повторення присутні	TimSort

8

ДЕРЕВО РІШЕНЬ ВИБОРУ АЛГОРИТМУ



БЛОК-СХЕМА ВИБОРУ КОНКРЕТНОГО АЛГОРИТМУ СОРТУВАННЯ



ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Алгоритм сортування	Час сортування (мс)	Кількість порівнянь	Кількість обмінів	Обсяг використаної пам'яті (КВ)
малий та середній розмір				
впорядковані з повтореннями				
Сортування вставками (Insertion sort)	1	213 901	213901	66
впорядковані без повторень				
Сортування бульбашою (Buble sort)	2	499 500	208969	66
невпорядковані				
Сортування вибором (Selection Sort)	1	499 500	989	66
великі розміри				
впорядковані				
TimSort	12	1 451 364	8 254 390	1367
невпорядковані				
Сортування злиттям(Merge sort)	23	1 536 043	759 997	1367

11

ПРАКТИЧНИЙ РЕЗУЛЬТАТ

Алгоритм сортування	Час (мс)	Кількість порівнянь	Кількість обмінів	Обсяг використаної пам'яті (КВ)
малий та середній розмір				
впорядковані з повтореннями				
Швидке сортування	1	11 651	6 023	145
впорядковані без повторень				
TimSort	2	20 408	19 452	181
невпорядковані				
Сортування злиттям	2	8 731	4 409	301
великі розміри				
впорядковані				
Сортування за розрядами	29	806 466 675	999 380	11459
невпорядковані				
Природне сортування	600	1 854 536	1 854 460	6596

12

ВИСНОВКИ

1. Вивчено та проаналізовано класичні та сучасні алгоритми сортування з метою визначення їхніх переваг та обмежень
2. Розглянуто та систематизовано теоретичні концепції оптимізації алгоритмів сортування
3. Проведено аналіз та критичний огляд наукових публікацій, присвячених оптимізації алгоритмів сортування
4. Розробка та систематизація методів та підходів до оптимізації алгоритмів сортування
5. Здійснено аналіз продуктивності різних алгоритмів сортування
6. Розробка конструктивних рекомендацій для вибору та тестування оптимальних стратегій сортування
7. Проведено емпіричне підтвердження ефективності запропонованої методики
8. Формулювання висновків на основі отриманих результатів та визначення можливостей подальших досліджень

13

ПУБЛІКАЦІЇ ТА АПРОБАЦІЯ РОБОТИ

Тези доповідей:

1. Бірса О. А. Особливості розробки програмного забезпечення для цифровізації життя і промисловості. // Науково-практична конференція «Проблеми комп’ютерної інженерії»» – Київ: ДУТ, 2023. – С. 82-84.
2. Бірса О. А. Роль та перспективи програмного забезпечення в комп'ютерній інженерії. // IV Всеукраїнська студентська конференція Науковий простір: аналіз, сучасний стан, тренди та перспективи – Київ: ДУТ, 2023. – С. 379-381.

14

ДЯКУЮ ЗА УВАГУ!