МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

Виконала: Групи

Пугачова Д. В. КІ-23-1

Кременчук 2024

Практична робота №3

Тема. Алгоритми сортування та їх складність. Порівняння алгоритмів сортування

Мета: опанувати основні алгоритми сортування та навчитись методам аналізу їх асимптотичної складності

Завдання

1. Вивчити самостійно і записати (будь-яким способом) алгоритм бульбашкового сортування. Оцінити асимптотику алгоритму сортування методом бульбашки в найгіршому і в найкращому випадку. Порівняти за цими показниками бульбашковий алгоритм з алгоритмом сортування вставлянням. Чому на практиці бульбашковий алгоритм виявляється менш ефективним у порівнянні з сортуванням методом зливанням?

***Алгоритм бульбашкового сортування*** *— це один із найпростіших алгоритмів сортування. Він працює наступним чином:*

***Ідея****: Алгоритм порівнює сусідні елементи масиву і, якщо вони розташовані в неправильному порядку (для зростання — якщо перший більший за другий), міняє їх місцями. Цей процес повторюється по всьому масиву, поки масив не буде відсортований.*

***Алгоритм****:*

*-Пройдіться по всіх елементах масиву від початку до кінця.*

*-Порівняйте кожну пару сусідніх елементів.*

*-Якщо елементи знаходяться в неправильному порядку (для сортування за зростанням — перший елемент більший за другий), обміняйте їх місцями.*

*-Повторюйте ці кроки для всіх елементів масиву, повторюючи процес кілька разів, поки всі елементи не будуть відсортовані.*

*-Кожен наступний прохід по масиву може бути коротшим, оскільки після кожного проходу найбільший елемент «спливає» на останнє місце.*

**Приклад коду :**

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

# Якщо під час цього проходу не було обмінів, можна припинити сортування

swapped = False

for j in range(0, n - i - 1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j] # обмін

swapped = True

if not swapped:

break # якщо не було обміну, масив вже відсортований

return arr

***Оцінка асимптотики***

***1) Найгірший випадок***

*У найгіршому випадку масив повністю незасортований, і алгоритм повинен пройти через кожну пару елементів і виконати обміни. Кількість порівнянь і обмінів дорівнює:*

*-Перший прохід: n−1 порівнянь*

*-Другий прохід: n−2 порівнянь*

*-І так далі, до останнього проходу, який складається з одного порівняння.*

*-загальна кількість порівнянь:*

(n-1)+(n-2)+…+1=

*Тому* ***часова складність в найгіршому випадку****: O(n2).*

***2) Найкращий випадок***

*У найкращому випадку масив вже відсортований. Алгоритм ще буде виконувати всі проходи, але жодного обміну не відбудеться, і після першого проходу буде виявлено, що масив уже відсортований.*

*Тому* ***часова складність в найкращому випадку****: O(n), оскільки необхідно лише один раз пройти через масив для перевірки.*

***Порівняння з алгоритмом сортування вставками***

***Алгоритм сортування вставками***

*Алгоритм сортування вставками полягає в тому, що елементи по черзі вбудовуються в відсортовану частину масиву. Ось як це працює:*

1. *Початково вважаємо, що перший елемент масиву вже відсортований.*
2. *Потім кожен наступний елемент порівнюється з елементами відсортованої частини і вставляється на відповідну позицію.*

***Оцінка складності сортування вставками:***

***1)Найгірший випадок****: Масив відсортований у зворотному порядку, що вимагає для кожного елемента переміщення всіх попередніх елементів. Складність: O()*

***2)Найкращий випадок****: Масив вже відсортований, і для кожного елемента потрібно лише перевірити, що він більший за попередній. Складність: O(n).*

***Порівняння бульбашкового сортування та сортування вставками:***

*1)У* ***найгіршому випадку*** *обидва алгоритми мають складність O(), однак алгоритм сортування вставками може бути дещо ефективнішим, оскільки він робить менше обмінів.*

*2)У* ***найкращому випадку*** *сортування вставками працює за O(n), що швидше за O() у бульбашковому сортуванні, оскільки бульбашковий алгоритм навіть при відсортованому масиві може виконувати зайві проходи.*

***Чому бульбашковий алгоритм менш ефективний у порівнянні з алгоритмом злиття?***

*Алгоритм* ***сортування злиттям*** *є набагато ефективнішим, оскільки:*

***-Часова складність*** *сортування злиттям завжди становить O(n log n), незалежно від того, чи є масив відсортованим чи ні.*

***-Процес****: Алгоритм злиття розбиває масив на менші частини, сортує їх і зливає назад у відсортований масив. Це дозволяє працювати з підмасивами, що значно зменшує кількість порівнянь порівняно з бульбашковим сортуванням, яке постійно виконує обміни елементів.*

*У бульбашковому сортуванні кожен елемент може бути переміщений на значну відстань в масиві, тоді як в алгоритмі злиття елементи обробляються по частинах, що дозволяє зменшити кількість операцій для великих масивів.*

2. Оцінити асимптотичну складність алгоритму сортування зливанням, скориставшись основною теоремою рекурсії.

*Алгоритм сортування злиттям працює за принципом розділення масиву на дві частини, сортування кожної частини окремо, а потім злиття їх в один відсортований масив.*

*-****Як працює алгоритм сортування злиттям?***

1. *Якщо масив має більше одного елемента, ми ділимо його на дві частини.*
2. *Кожну частину сортуємо рекурсивно (тобто сортуємо кожну половину, викликаючи алгоритм для кожної з частин).*
3. *Коли кожна частина відсортована, ми об'єднуємо ці дві частини в один відсортований масив.*

*-****Як оцінити складність алгоритму?***

*Алгоритм має рекурсивну структуру, тобто він викликає сам себе для кожної з половин масиву. Тому можна написати рівняння для часу, який потрібен для сортування масиву розміру n:*

T(n)=2T()+O(n)

*Це означає, що час для сортування масиву nnn елементів залежить від часу сортування двох частин масиву і часу, необхідного для їх об'єднання, що займає O(n).*

*-****Як знайти складність?***

*Тепер скористаємося* ***основною теоремою рекурсії****, яка дозволяє оцінити складність таких алгоритмів. Вона працює, коли у нас є рівняння такого вигляду:*

*T(n)=aT()+O()*

*У нашому випадку:*

* *a=2 (ми ділимо масив на дві частини),*
* *b=2 (кожну частину розміром n/2),*
* *d=1 (об'єднання займає O(n) часу).*

*Згідно з цією теоремою, якщо a= то складність буде O(n log n).*

***-Результат***

*Отже,* ***складність алгоритму сортування злиттям*** *дорівнює O(n log n). Це означає, що час сортування злиттям збільшується зі збільшенням розміру масиву, але набагато повільніше, ніж у алгоритмів типу бульбашкового сортування чи сортування вставками, у яких складність O().*

3. Вивчити і записати (будь-яким способом) самостійно алгоритм швидкого сортування. Оцінити асимптотичну складність алгоритму швидкого сортування, скориставшись основною теоремою рекурсії.

***-Алгоритм швидкого сортування (Quick Sort)***

***Швидке сортування*** *— це один з найбільш ефективних алгоритмів сортування в середньому випадку, який базується на принципі «розділяй і володарюй». Алгоритм працює за принципом вибору* ***опорного елемента****, поділу масиву на дві частини (по цей бік менші елементи, по той більші або рівні опорному) і рекурсивного сортування цих частин.*

***-Кроки алгоритму:***

1. ***Вибір опорного елемента****:*
   * *Для масиву вибираємо один елемент (наприклад, перший, останній, або випадковий).*
2. ***Розбиття масиву****:*
   * *Переставляємо елементи масиву так, щоб всі елементи менші або рівні опорному, знаходились лівіше, а більші — правіше.*
   * *Опорний елемент тепер на своєму фінальному місці.*
3. ***Рекурсивне сортування****:*
   * *Тепер сортуємо дві частини масиву (ліву та праву) рекурсивно.*
4. ***Завершення****:*
   * *Алгоритм завершується, коли частини масиву не можна поділити далі (масиви містять один елемент або порожні).*

-**Псевдо код алгоритму:**

*def quick\_sort(arr):*

*if len(arr) <= 1:*

*return arr*

*pivot = arr[len(arr) // 2] # вибір опорного елемента (середина масиву)*

*left = [x for x in arr if x < pivot] # елементи менші за опорний*

*middle = [x for x in arr if x == pivot] # елементи рівні опорному*

*right = [x for x in arr if x > pivot] # елементи більші за опорний*

*return quick\_sort(left) + middle + quick\_sort(right) # рекурсивне сортування*

*-****Оцінка асимптотичної складності алгоритму швидкого сортування***

*Алгоритм швидкого сортування має рекурсивну структуру, тому для оцінки складності можна використати* ***основну теорему рекурсії****.*

*-****Рекурсивне співвідношення для швидкого сортування***

*Якщо ми позначимо час сортування масиву з nnn елементів як T(n), то можна записати рекурсивне співвідношення для алгоритму:*

*T(n)=T(k)+T(n−k−1)+O(n)*

*де:*

* *k — це кількість елементів, що менші за опорний (або більші, в залежності від вибору),*
* *O(n) — це час на поділ масиву (перестановку елементів і вибір опорного елемента).*

*У найкращому випадку масив ділиться на дві рівні частини, тобто k= Тоді рекурсивне співвідношення буде виглядати так:*

*T(n)=2T()+O(n)*

***Оцінка складності:***

*Тепер скористаємося* ***основною теоремою рекурсії****, щоб оцінити складність цього рекурсивного співвідношення.*

*Основна теорема рекурсії для співвідношення T(n)=aT()+O() дає різні варіанти складності залежно від значень aaa, bbb і ddd. У нашому випадку:*

* *a=2 (дві рекурсивні частини),*
* *b=2 (масив ділиться навпіл),*
* *d=1(операції поділу займають O(n).*

*Тепер перевіряємо умови теореми:*

a=, тобто 2=.

*Згідно з основною теоремою рекурсії, коли a=, складність буде:*

*T(n)* *=O(n log n)*

***Оцінка складності в найгіршому випадку:***

*У найгіршому випадку, коли вибір опорного елемента не дає хорошого розподілу (наприклад, кожного разу вибирається найменший або найбільший елемент), алгоритм працюватиме так, ніби кожен поділ дає масив, що містить один елемент. У такому випадку рекурсивне співвідношення буде виглядати так:*

T(n)=T(n−1)+O(n)

*Тут кожен раз обробляється лише одна частина, що дає лінійно зменшувану кількість елементів.*

*Розв'язання цього рівняння дає:*

*T(n)=O()*

***Підсумок:***

***-Найкращий випадок****: O(n log n) (коли кожен поділ дає рівні частини).*

***-Найгірший випадок****: O() (коли вибір опорного елемента не оптимальний і кожен поділ дає майже відсутній розподіл).*

*Зазначимо, що на практиці середнє виконання алгоритму швидкого сортування зазвичай ближче до O(n log n), тому що вибір опорного елемента часто оптимізується (наприклад, шляхом вибору опорного елемента випадковим чином або за допомогою стратегій, як «медіана трьох»).*

Контрольні питання

**1. Що таке асимптова складність алгоритму сортування і чому вона важлива для порівняння алгоритмів?**

***Асимптотична складність*** *алгоритму — це оцінка часу виконання алгоритму в залежності від розміру вхідних даних. Вона дозволяє визначити, як змінюється час виконання алгоритму при збільшенні кількості елементів. Асимптотична складність дає загальне уявлення про ефективність алгоритму в умовах великих даних. Це важливо, оскільки ми можемо порівняти різні алгоритми за їхньою здатністю працювати з великими наборами даних та оцінити, який з них найшвидший або найбільш оптимальний для конкретної задачі.*

**2. Які алгоритми сортування мають квадратичну складність у найгіршому випадку? Поясніть, чому це може бути проблемою для великих обсягів даних.**

*До алгоритмів з* ***квадратичною складністю*** *в найгіршому випадку належать:*

* ***Бульбашкове сортування (Bubble Sort)***
* ***Сортування вставлянням (Insertion Sort)***
* ***Сортування вибором (Selection Sort)***

*Ці алгоритми мають складність O(n2)O(n^2)O(n2) у найгіршому випадку, що означає, що час їх виконання зростає квадратично відносно розміру вхідного масиву. Це може бути* ***проблемою для великих обсягів даних****, оскільки з кожним збільшенням кількості елементів час виконання значно зростає. Для великих наборів даних ці алгоритми стають дуже повільними, тому їх не рекомендують використовувати в практичних застосунках для великих обсягів даних.*

**3. В чому полягає перевага сортування злиттям над сортуванням вставками для великих наборів даних?**

***Сортування злиттям (Merge Sort)*** *має складність O(n log n), що робить його значно ефективнішим для великих наборів даних порівняно з* ***сортуванням вставками (Insertion Sort)****, яке має складність O() у найгіршому випадку. Перевага сортування злиттям полягає в тому, що його* ***часова складність не залежить від того, чи відсортовані дані частково чи ні****, тобто він завжди працює з одною ефективністю, навіть якщо масив частково відсортований. Це дозволяє сортуванню злиттям працювати набагато швидше на великих масивах даних, де сортування вставками буде значно повільнішим.*

**4. Які алгоритми сортування використовуються для сортування списків у стандартних бібліотеках мов програмування, таких як Python, Java або C++?**

*У* ***стандартних бібліотеках*** *мов програмування часто використовуються* ***оптимізовані алгоритми сортування****, які забезпечують хорошу продуктивність в більшості випадків:*

* *У* ***Python*** *використовується алгоритм* ***Timsort****, який є гібридом сортування злиттям і сортування вставками. Він має складність O(n log n) у найгіршому випадку, і дуже ефективний для реальних даних.*
* *У* ***Java*** *використовуються два основних алгоритми:* ***Merge Sort*** *для сортування масивів об'єктів і* ***Dual-Pivot Quick Sort*** *для сортування примітивних типів даних (наприклад, int, double).*
* *У* ***C++*** *стандартна бібліотека використовує* ***IntroSort****, який поєднує сортування швидким сортуванням (Quick Sort), сортування вставками і сортування злиттям. Він починає сортування за допомогою швидкого сортування, а коли рекурсивна глибина стає занадто великою, переходить до сортування злиттям для покращення стабільності.*

**5. Яка різниця між алгоритмами сортування злиттям і швидким сортуванням? У яких випадках краще використовувати кожен з цих алгоритмів?**

***Різниця*** *між алгоритмами сортування злиттям і швидким сортуванням:*

* ***Сортування злиттям (Merge Sort)*** *завжди має складність O(n log n) і підходить для* ***великої кількості даних****, особливо коли важлива стабільність сортування. Це також хороший вибір для масивів, які важко поділяти або які знаходяться на зовнішніх носіях (наприклад, файли).*
* ***Швидке сортування (Quick Sort)****, в середньому, також має складність O(n log n), але в найгіршому випадку може бути O(). Однак воно часто працює швидше за сортування злиттям завдяки меншій кількості використаних додаткових ресурсів (наприклад, пам'яті), тому* ***швидке сортування*** *зазвичай використовується для* ***молодших або середніх обсягів даних*** *та коли важлива швидкість у середньому випадку.*

***Кращий вибір:***

* *Використовувати* ***сортування злиттям*** *для стабільності і гарантованої ефективності на великих даних.*
* *Використовувати* ***швидке сортування****, якщо потрібно швидше сортувати дані в середньому випадку, але потрібно бути готовим до можливого найгіршого сценарію.*

**6. Які фактори слід враховувати при виборі алгоритму сортування для конкретної задачі?**

*При виборі алгоритму сортування варто враховувати такі фактори:*

1. ***Розмір даних****: Для великих масивів найкращі алгоритми — це ті, що мають складність O(n log n), наприклад, сортування злиттям або швидке сортування.*
2. ***Тип даних****: Якщо важлива стабільність сортування (збереження порядку однакових елементів), тоді краще використовувати сортування злиттям.*
3. ***Вид доступної пам'яті****: Алгоритми на основі злиття потребують додаткової пам'яті для допоміжних масивів, що може бути проблемою для великих даних. Швидке сортування зазвичай використовує менше пам'яті.*
4. ***Попередній стан даних****: Для вже частково відсортованих даних алгоритм сортування вставками або бульбашкове сортування можуть бути ефективнішими.*
5. ***Простота реалізації****: Якщо потрібен простий і швидкий результат для невеликих даних, можна вибрати прості алгоритми, такі як сортування вставками.*