МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра комп'ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНИХ РОБІТ

з навчальної дисципліни

«Алгоритми та методи обчислень»

Тема «Алгоритми пошуку та їх складність»

Студент гр. КІ-23-1 ПІБ Кобець О. О.

Кременчук 2024

Практична робота №4

Тема. Алгоритми пошуку та їх складність

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

Завдання

1. Оцінити асимптотичну складність алгоритму лінійного пошуку у O- нотації в найгіршому і в найкращому випадку. Як можна покращити алгоритм лінійного пошуку?

2. Оцінити асимптотичну складність алгоритму бінарного пошуку у O- нотації в найгіршому і в найкращому випадку.

3. Побудувати алгоритм тернарного пошуку і оцінити його асимптотичну складність алгоритму у O-нотації в найгіршому і в найкращому випадку. Який з алгоритмів є оптимальнішим: бінарний, чи тернарний? Обґрунтувати відповідь відповідними обчисленнями.

4. Порівняти ефективність алгоритмів лінійного, бінарного та тернарного пошуку для різних розмірів вхідного списку. Для цього провести експериментальне дослідження та побудувати графіки залежності часу виконання алгоритму від розміру вхідного списку.

5. Порівняти алгоритми пошуку за їхньою здатністю працювати з відсортованими та не відсортованими списками. Провести аналіз впливу відсортованості списку на час виконання кожного алгоритму.

6. Розглянути сценарії використання кожного з алгоритмів пошуку у практичних задачах і обґрунтувати вибір кожного алгоритму в конкретному випадку.

1. **Найгірший випадок**: Шуканий елемент знаходиться в кінці списку або відсутній у ньому. У такому випадку алгоритм перегляне всі n елементів. Асимптотична складність: O(n).

Найкращий випадок: Шуканий елемент знаходиться на першій позиції. Асимптотична складність: O(1).

2. **Найгірший випадок**: Кожна ітерація ділить список навпіл, що веде до логарифмічної кількості кроків. Асимптотична складність: O(log n).

Найкращий випадок: Шуканий елемент знаходиться на середній позиції при першій перевірці. Асимптотична складність: O(1).

3.

```python
def ternary_search(arr, l, r, x):  3 usages
    if r >= l:
        mid1 = l + (r - l) // 3
        mid2 = r - (r - l) // 3

        if arr[mid1] == x:
            return mid1
        if arr[mid2] == x:
            return mid2

        if x < arr[mid1]:
            return ternary_search(arr, l, mid1 - 1, x)
        elif x > arr[mid2]:
            return ternary_search(arr, mid2 + 1, r, x)
        else:
            return ternary_search(arr, mid1 + 1, mid2 - 1, x)

    return -1
```

Найгірший випадок:

У тернарному пошуку на кожному кроці ми ділимо масив на три частини. В найгіршому випадку шуканий елемент може знаходитися в останній підмасиві кожного разу. Таким чином, кожного разу масив скорочується до 1/3 його попередньої довжини.

Для масиву розміру nnn кількість ітерацій буде приблизно $\log_3(n)$, де логарифм береться за основою 3. Це означає, що в найгіршому випадку тернарний пошук має асимптотичну складність: $O(\log_3 n)$.

Найкращий випадок:

Як і у випадку з бінарним пошуком, найкращий випадок для тернарного пошуку відбувається, коли шуканий елемент знаходиться на одній з розділових позицій при першій перевірці. Тоді ми знайдемо його за одну ітерацію. O(1)

Порівняння тернарного і бінарного пошуків

Бінарний пошук має асимптотичну складність $O(\log_2 n)$, тоді як тернарний пошук має асимптотичну складність $O(\log_3 n)$.

$$log_3 n = \frac{log_2 n}{log_2 3}$$

$$O(log_3 n) = O(\frac{log_2 n}{log_2 3})$$

$\log_2 3 \approx 1.58496$

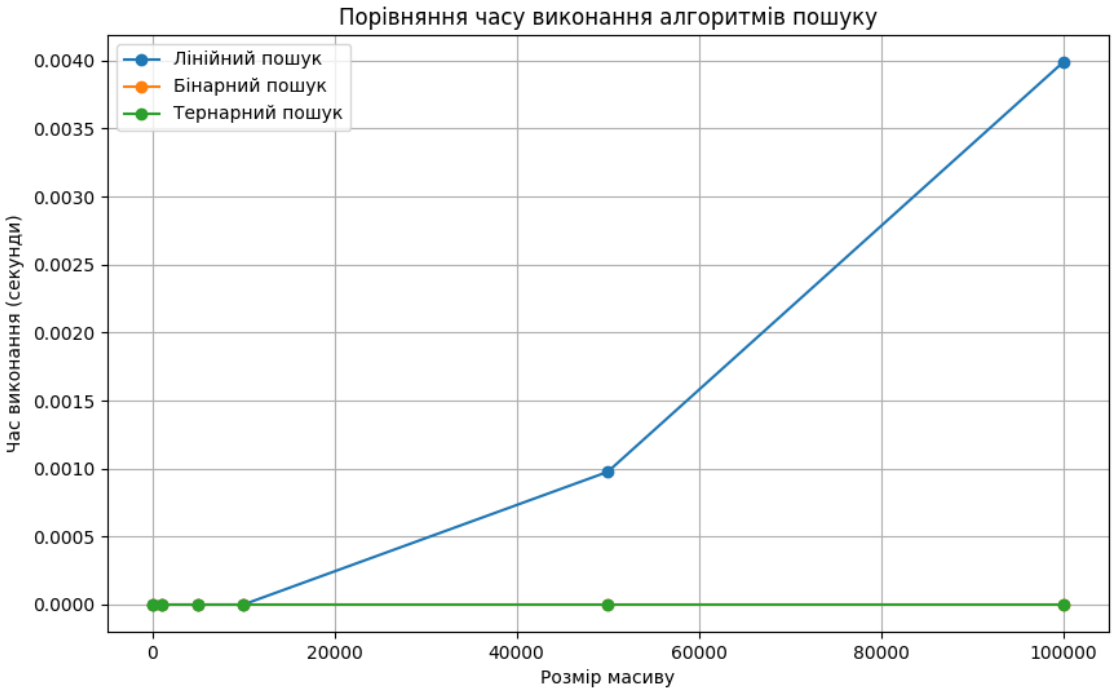
$O(\log_3 n) \approx O\ (0.631\ \log_2 n)$

Висновок

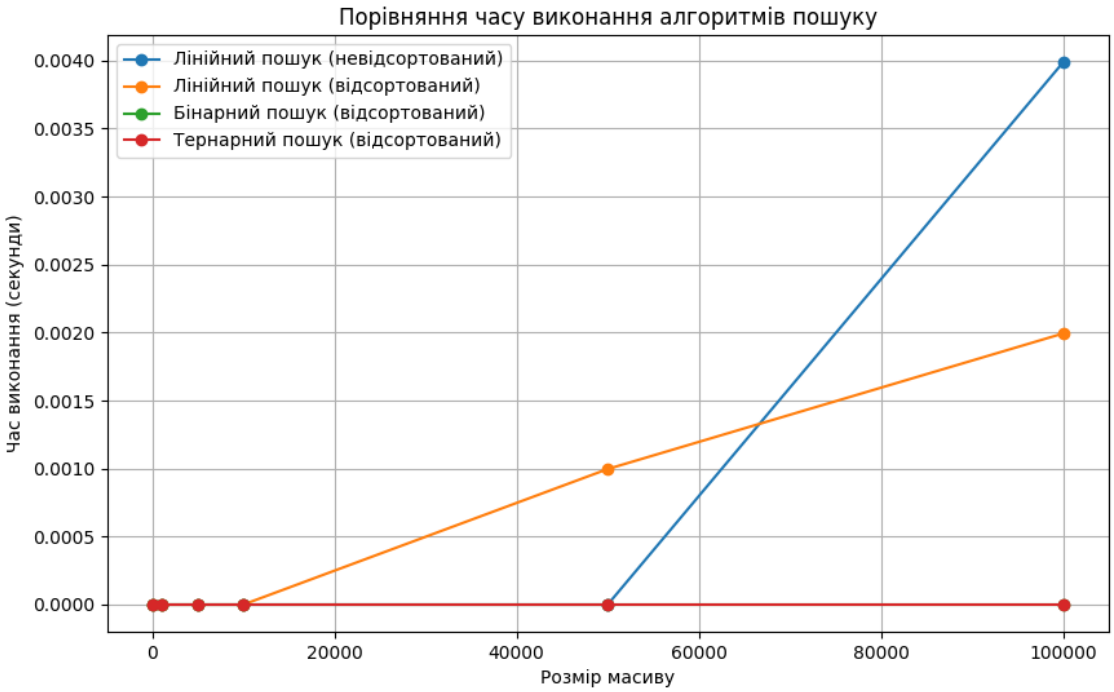
З теоретичної точки зору, бінарний пошук має перевагу над тернарним пошуком, оскільки:

1. **Кількість порівнянь на ітерацію**: Бінарний пошук робить одне порівняння на ітерацію, тоді як тернарний робить два.
2. **Час виконання**: Хоча тернарний пошук має меншу кількість ітерацій, кожна ітерація є більш затратною через більше порівнянь.

4.



5.



1. Лінійний пошук:

- На невідсортованих масивах час виконання збільшується лінійно з розміром масиву.
- На відсортованих масивах лінійний пошук також показує лінійну залежність, але може бути трохи швидшим через можливі оптимізації у відсортованих даних (наприклад, вихід з циклу при знаходженні елемента, меншого за шуканий).

2. Бінарний та тернарний пошук:

- Обидва алгоритми показують логарифмічну залежність від розміру масиву на відсортованих даних.
- Вони не застосовуються до невідсортованих масивів, тому час виконання на невідсортованих даних не вимірювався.

Висновки

- Відсортованість масиву значно впливає на час виконання пошукових алгоритмів.
- Лінійний пошук залишається ефективним для невеликих масивів або для невідсортованих даних.

- Бінарний і тернарний пошук є оптимальними для великих відсортованих масивів, показуючи значно менший час виконання порівняно з лінійним пошуком.

6.

1. Лінійний пошук

Сценарій:

- **Пошук у невідсортованому списку**: Коли дані не відсортовані і не мають спеціальної структури.
- **Малий розмір даних**: Коли кількість елементів у списку мала (наприклад, до кількох десятків або сотень).
- **Одноразовий пошук**: Коли необхідно виконати одноразовий пошук у даних.

Обґрунтування вибору:

- Лінійний пошук простий у реалізації і не потребує попереднього сортування даних.
- Він працює добре для малих масивів, де час виконання не є критичним фактором.
- Для одноразових або рідкісних запитів на пошук в невідсортованих даних, використання лінійного пошуку є виправданим.

2. Бінарний пошук

Сценарій:

- **Пошук у відсортованому масиві**: Коли дані попередньо відсортовані.
- **Великі обсяги даних**: Коли розмір масиву великий (тисячі, мільйони або більше елементів).
- **Часті запити на пошук**: Коли необхідно здійснювати пошук часто і швидко.

Обґрунтування вибору:

- Бінарний пошук значно ефективніший за лінійний пошук для великих відсортованих масивів, оскільки його час виконання зростає логарифмічно.

- Він ідеально підходить для задач, де потрібна висока швидкість пошуку, наприклад, в системах баз даних або пошукових системах.
- Якщо дані вже відсортовані або можуть бути відсортовані заздалегідь, використання бінарного пошуку суттєво скорочує час виконання порівняно з лінійним пошуком.

3. Тернарний пошук

Сценарій:

- **Пошук у відсортованому масиві**: Аналогічно до бінарного пошуку, коли дані попередньо відсортовані.
- **Особливі випадки**: Коли важливо зменшити кількість рекурсивних викликів, але можна допустити більше порівнянь на кожному кроці.

Обґрунтування вибору:

- Тернарний пошук може бути корисним у випадках, де дані відсортовані і потрібно уникнути глибокої рекурсії.
- Він підходить для задач, де порівняння є менш затратними порівняно з рекурсивними викликами, наприклад, в середовищах з обмеженою глибиною рекурсії.
- Проте, на практиці тернарний пошук рідко перевершує бінарний пошук за загальною ефективністю, тому його використання варто розглядати лише в специфічних випадках.

Контрольні питання

1. Що таке алгоритм пошуку і чому він важливий у контексті комп'ютерних наук?

Алгоритм пошуку — це послідовність кроків або правил, які використовуються для знаходження певного елемента або набору елементів у структурі даних, наприклад, у масиві, списку або базі даних. Він важливий у контексті комп'ютерних наук тому, що пошук є однією з найпоширеніших операцій при роботі з даними. Ефективний алгоритм пошуку дозволяє швидко отримати необхідну інформацію, зменшивши час виконання програм і ресурсні витрати.

2. Які основні критерії оцінки ефективності алгоритмів пошуку?

- Асимптотичну складність: Оцінка алгоритму в термінах великих O, що показує, як змінюється час виконання або використання пам'яті в залежності від розміру вхідних даних.
- Час виконання: Реальний час, необхідний для виконання алгоритму.
- Використання пам'яті: Кількість пам'яті, необхідної для виконання алгоритму.
- Простота реалізації: Наскільки легко реалізувати і зрозуміти алгоритм.
- Стабільність і коректність: Наскільки алгоритм надійний і чи завжди він знаходить правильний результат.

3. Що таке лінійний пошук, і як він працює?

Лінійний пошук — це алгоритм пошуку, який послідовно перевіряє кожен елемент структури даних до тих пір, поки не знайде потрібний елемент або не обробить всі елементи. Він працює наступним чином:

- Починаючи з першого елемента списку, перевіряється кожен елемент на відповідність шуканому значенню.
- Якщо знайдено відповідний елемент, алгоритм завершується і повертає його позицію.
- Якщо жоден елемент не відповідає шуканому значенню, алгоритм завершується, повертаючи індикатор невдачі (наприклад, -1).

4. Які умови повинні бути виконані для успішного застосування бінарного пошуку?

- Відсортований масив: Дані повинні бути відсортовані за зростанням або спаданням.
- Доступ до середнього елемента: Алгоритм повинен мати можливість швидкого доступу до середнього елемента масиву (наприклад, через індекси).
- Стаціонарність даних: Дані не повинні змінюватися під час виконання алгоритму.

5. Які переваги та недоліки використання бінарного пошуку порівняно з іншими алгоритмами пошуку?

Переваги:

- **Ефективність**: Має логарифмічну складність O(log n), що робить його дуже швидким для великих відсортованих масивів.

- **Стабільність**: Завжди знаходить шуканий елемент, якщо він присутній у масиві.

Недоліки:

- **Вимога до відсортованості**: Потребує попереднього сортування масиву, що може бути витратним за часом.
- **Складність реалізації**: Трохи складніший у реалізації, ніж лінійний пошук.
- **Менша гнучкість**: Не може бути застосований до невідсортованих масивів або структур даних з повільним доступом до середніх елементів.

6. Що таке тернарний пошук, і в чому його відмінність від бінарного

пошуку?

Тернарний пошук — це варіант алгоритму пошуку, який працює на відсортованих масивах, аналогічно до бінарного пошуку. Відмінність полягає в тому, що тернарний пошук ділить масив на три частини, а не на дві, і таким чином виконує два порівняння на кожній ітерації:

1. Масив ділиться на три частини, вибираючи два середні елементи.
2. Порівнюються ці два середні елементи з шуканим значенням.
3. В залежності від результату порівняння, пошук продовжується в одній з трьох частин масиву.