МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА
ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНОЇ РОБОТИ №4

з навчальної дисципліни
«Алгоритми та методи обчислень»

Тема «Алгоритми пошуку та їх складність»

Студент гр. КІ-24-1, Варакута О.О
Викладач, Сидоренко В. М

Кременчук 2025

Тема. Алгоритми пошуку та їх складність

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

Хід роботи

Задачі для самостійного розв’язання

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

```
def linear_search(arr, key):
    for i in range(len(arr)):
        if arr[i] == key:
            return i
    return -1
```

Найкращий випадок: $O(1)$

Середній випадок: $O(n)$

Найгірший випадок: $O(n)$

Аналіз найгіршого випадку

У найгіршому випадку шуканий елемент відсутній у масиві, або знаходиться в кінці.

$T(n)=C_1+C_2\cdot n+C_3\cdot n+C_4$

де:

$C1$ -ініціалізація

$C2\cdot n$- умова циклу

$C3\cdot n$ - перевірка елемента

$C4$ - повернення результату

$T(n)=(C_2+C_3)\cdot n+(C_1\to C_4)=a\cdot n+b$

При $n\to\infty$: $T(n)=O(n)$

Найкращій випадок $O(1)$. У найкращому випадку шуканий елемент знаходиться на першій позиції:

$$T(n)=C_1+C_2+0+C_4=C_1+C_2+C_4=a$$

$$T(n)=O(1)$$

Ймовірність знайти елемент на позиції i: $\frac{1}{n}$

$$T(n)=\frac{1}{n}\sum_{i=1}^{n} i=\frac{1}{n}\cdot\frac{n(n+1)}{2}=\frac{n+1}{2}$$

$$T(n)=O(n)$$

Лінійний пошук є простим, але неефективним для великих масивів через лінійну залежність часу виконання від розміру вхідних даних. Алгоритм лінійного пошуку можна покращити, якщо попередньо впорядкувати масив.

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

```
def binary_search(arr, left, right, key):
    if left > right:
        return -1
    mid = (left + right) // 2
    if arr[mid] == key:
        return mid
    elif arr[mid] > key:
        return binary_search(arr, left, mid - 1, key)
    else:
        return binary_search(arr, mid + 1, right, key)
```

Час виконання описується рекурентною формулою:

$$T(n)=T\left(\frac{n}{2}\right)+O(1)$$

$T\left(\frac{n}{2}\right)$ – час пошуку в половині масиву

$O(1)$ — операції порівняння та обчислення середини

Розв’язок:

$$T(n)=aT\left(\frac{n}{b}\right)+f(n)$$

a=1- кількість підзадач

b = 2 коефіцієнт ділення

f(n) = O(1)= O(n^0) складність об'єднання результатів

$f(n)=\Theta(n^0)$ за другою теоремою:

$$T(n)=\theta(n^{\log_b a}\cdot\log n)=\Theta(1\cdot\log n)=\Theta(\log n)$$

Аналіз випадків:

Найкращій випадок O(1): елемент знадиться точно в середині масиву.

Найгірший випадок O(log n): елемент відсутній або знаходиться на краю.

Середній випадок O(log n):елемент знаходиться в довільній позиції.

Бінарний пошук є ефективним алгоритмом з логарифмічною складністю, але вимагає попереднього сортування масиву. Він значно ефективніший за лінійний пошук для великих обсягів даних.

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.

Алгоритм тернарного пошуку:

```
def ternary_search(arr, left, right, key):
    if left > right:
        return -1
    mid1 = left + (right - left) // 3
    mid2 = right - (right - left) // 3
    if arr[mid1] == key:
        return mid1
    if arr[mid2] == key:
        return mid2
    if key < arr[mid1]:
        return ternary_search(arr, left, mid1 - 1, key)
    elif key > arr[mid2]:
        return ternary_search(arr, mid2 + 1, right, key)
```

```
    else:
        return ternary_search(arr, mid1 + 1, mid2 - 1, key)
```

Аналіз асимптотичної складності:

$$T(n)=T\left(\frac{n}{3}\right)+O(1)$$

a=1 кількість підзадач

b=3 коефіцієнт ділення

$f(n)= O(1) =O(n^0)$

$n^{\log_b a}=n^{\log_3 1}=n^0=1$

$f(n)=\Theta(n^0)$

$T(n)=\theta(n^{\log_b a}\cdot\log n)=\Theta(1\cdot\log n)=\Theta(\log n)$

Найкращий випадок: $O(1)$ Відбувається, коли шуканий елемент знаходиться в одній із двох точок розбиття на першому кроці.

Найгірший випадок: $O(\log_3 n)$- Відбувається, коли елемента немає в масиві, або він знаходиться в глибині однієї з третин, і доводиться виконувати максимальну кількість рекурсивних кроків, кожного разу зменшуючи діапазон пошуку до однієї третини.$n\cdot\left(\frac{1}{3}\right)^k\le 1\rightarrow k\ge\log_3 n$

Порівняння бінарного та тернарного пошуку:

Бінарний пошук: $O(\log_2 n)$

Тернарний пошук: $O(\log_3 n)$

$$\log_2 n=\frac{\log_3 n}{\log_3 2}\approx\frac{\log_3 n}{0.6309}\approx 1{,}5849\cdot\log_3 n$$

Бінарний пошук є оптимальнішим з таких причин:

1.Менша кількість порівнянь: 1 порівняння на крок проти 2 у тернарному

2.Краща константа в О-нотації: $C_{\text{бін}}\approx 1$ проти $C_{\text{терн}}\approx 2$

3.Менша складність реалізації

4.Краща локальність посилань (працює з меншими ділянками пам'яті)

Тернарний пошук доцільно використовувати лише в спеціальних випадках**:** Пошук екстремумів у унімодальних функціях, задачі оптимізації, де потрібно ділити область пошуку на три частини.

Критерій	**Бінарний пошук**	**Тернарний пошук**
Складність	$O(\log_2 n)$	$O(\log_3 n)$
Порівнянь на крок	1	2
Константа в О-нотації	1	2
Практична швидкість	Краща	Гірша
Область застосування	Універсальний	Спеціальний

Бінарний пошук є оптимальнішим для більшості практичних задач пошуку в відсортованих масивах через меншу загальну кількість порівнянь, незважаючи на більшу кількість кроків.

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.

. Порівняння ефективності алгоритмів лінійного, бінарного та

тернарного пошуку

Теоретична складність:

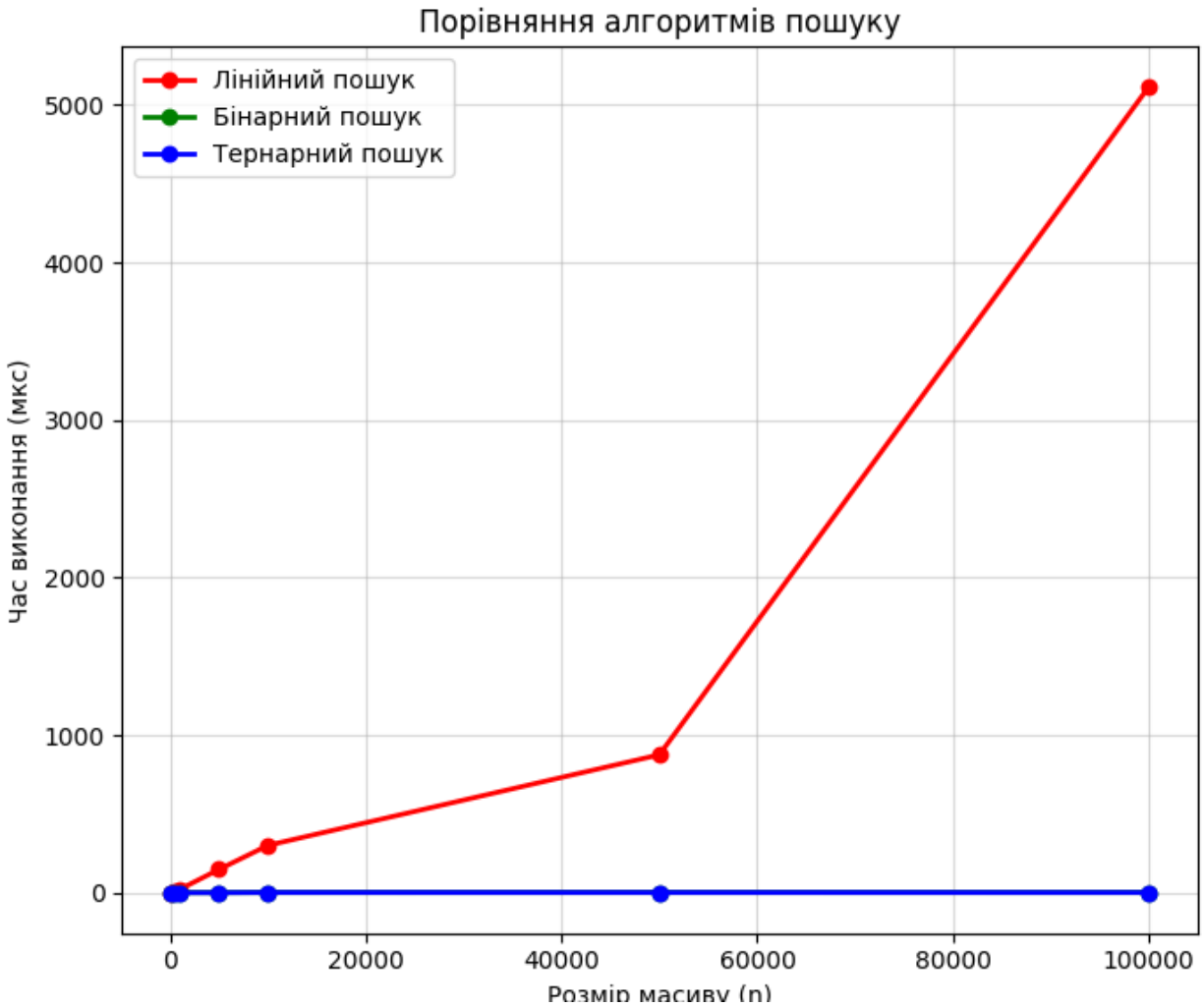
Алгоритм	Складність (найгірший випадок)
Лінійний пошук	$O(n)$
Бінарний пошук	$O(\log_2 n)$
Тернарний пошук	$O(\log_3 n)$

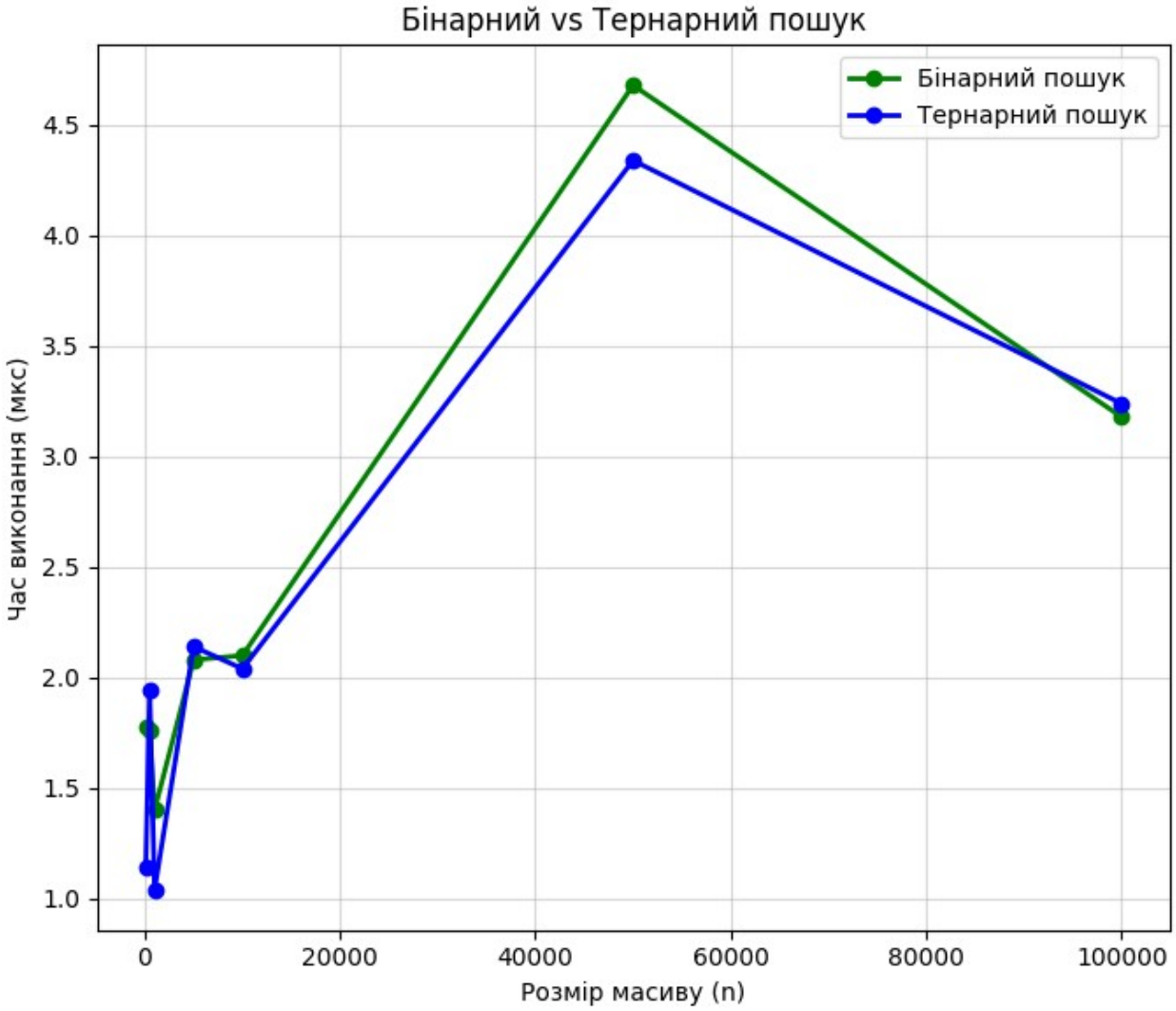
Порівняння логарифмів:

Оскільки

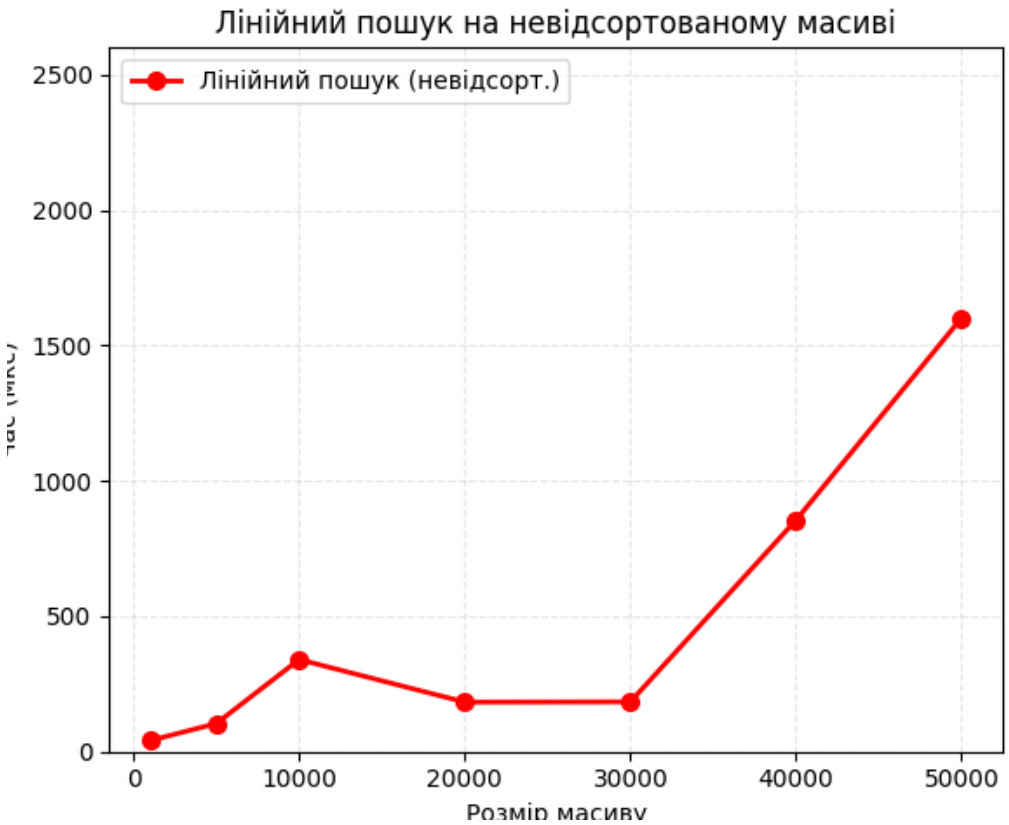
$$\log_2 n = \frac{\log_3 n}{\log_3 2} \approx 1{,}5849 \cdot \log_3 n$$

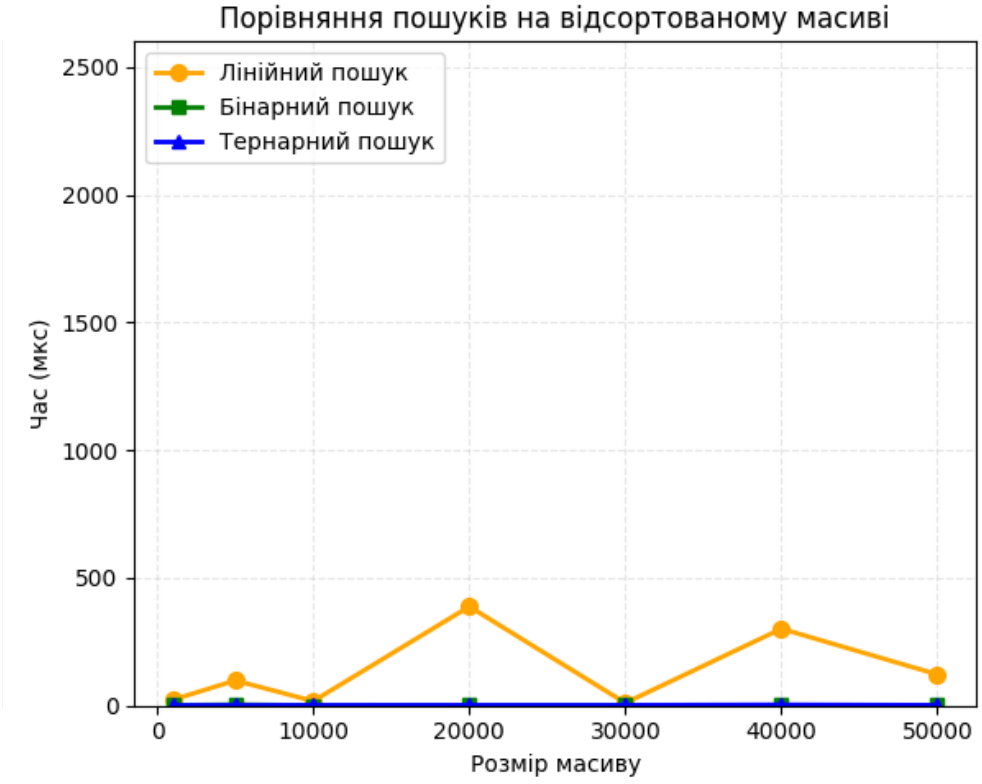
тобто бінарний пошук теоретично швидший, бо виконує менше порівнянь

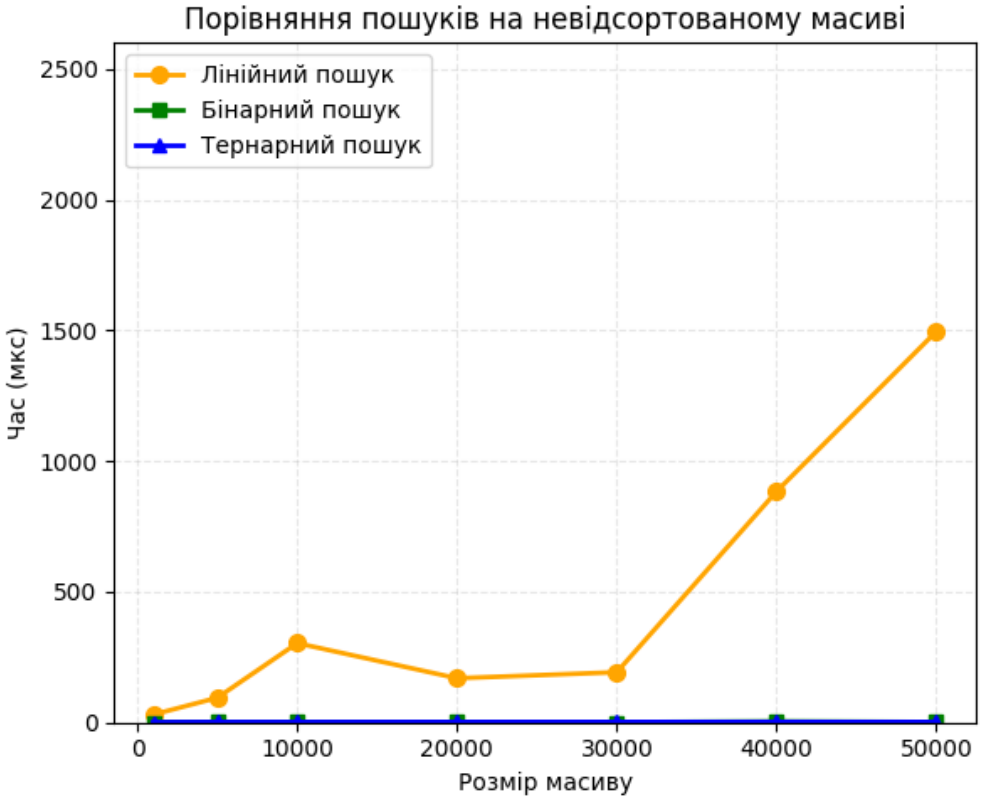




5. Порівняти алгоритми пошуку за їхньою здатністю працювати з відсортованими та не відсортованими списками. Провести аналіз впливу відсортованості списку на час виконання кожного алгоритму.


Лінійний пошук на невідсортованому масиві
Лінійний пошук (невідсорт.)
2500
2000
1500
1000
500
0
0
10000
20000
30000
40000
50000
Розмір масиву


Порівняння пошуків на відсортованому масиві
Лінійний пошук
Бінарний пошук
Тернарний пошук
Час (мкс)
2500
2000
1500
1000
500
0
0
10000
20000
30000
40000
50000
Розмір масиву



Висновок щодо ефективності:

1.Бінарний пошук виявився найоптимальнішим у практиці: Менше кроків у порівнянні з тернарним (бо ділить масив на 2 частини, а не 3, але з меншими витратами на порівняння).

2.Тернарний пошук має складність $O(\log_3 n)$, але в реальності виконує 2 порівняння на кожному кроці, тоді як бінарний — тільки 1.

3. Вплив відсортованості масиву на час пошуку

Алгоритм	Працює на несортованому?	Працює на відсортованому?
Лінійний	Так	Так
Бінарний	Ні	Так
Тернарний	Ні	Так

Бінарний пошук є оптимальнішим, ніж тернарний як теоретично, так і на практиці.

Тернарний пошук має гіршу ефективність через 2 перевірки замість 1. Відсортованість масиву значно покращує час пошуку для бінарного й тернарного пошуків. Лінійний пошук — універсальний, але неефективний на великих масивах.

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

1. Лінійний пошук

Як працює: перевіряє кожен елемент списку по черзі.

Коли використовувати: Список не відсортований; Список маленький, потрібно просте рішення

Приклади: Шукаєш своє ім’я в списку,перевіряєш, чи є помилка у текстовому файлі,шукаєш потрібну книгу в купі книжок на столі

Плюси: Працює з будь-яким списком, дуже простий

Мінуси: Повільний для великих списків

2. Бінарний пошук

Як працює: ділить відсортований список навпіл і вирішує, в якій половині шукати далі.

Коли використовувати: Список відсортований, потрібно знайти щось швидко,часто виконується пошук

Приклади: Шукаєш слово в алфавітному словнику; Вибираєш товар по ID в онлайн-магазині; Шукаєш номер у відсортованому телефонному списку.

Плюси: Дуже швидкий (особливо для великих списків)

Мінуси: Список має бути відсортований.

3. Тернарний пошук:

Як працює: ділить список на три частини замість двох, як у бінарному.

Коли використовувати: Список відсортований, потрібно знайти максимум або мінімум, задача на оптимізацію.

Приклади: Шукаєш оптимальну ціну, щоб купити товар; Потрібно знайти найменше або найбільше значення функції; Визначаєш найкращий час для поїздки, щоб уникнути заторів

Плюси: Добрий для спеціальних задач

Мінуси: У звичайному пошуку не кращий за бінарний, складніший у реалізації.

Алгоритм	Потрібно сортувати?	Працює з великими	Коли використовувати

		списками?	
Лінійний	Ні	Ні	Для простих задач
Бінарний	Так	Так	Для швидкого пошуку
Тернарний	Так	Так	Для задач з максимумом/мінімумом

Відповіді на контрольні питання :

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

Алгоритм пошуку – це метод знаходження елемента в структурі даних. Важливий для обробки великих обсягів інформації, баз даних, пошукових систем.

2. Які основні критерії оцінки ефективності алгоритмів пошуку?

Швидкодія (час), пам’ять, адаптивність до типу вхідних даних.

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

Алгоритм, що перебирає елементи один за одним. Простий, але повільний при великих об’ємах.

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

Масив має бути відсортованим, без повторень для класичної реалізації.

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

Перевага: Висока швидкість.

Недолік: Потребує попереднтого сортування.

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

Тернарний розбиває список на три частини, бінарний – на дві. На практиці тернарний зазвичай повільніший, хоча має теоретично схожу складність.