МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ

ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра комп’ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНИХ РОБІТ

з навчальної дисципліни

«Алгоритми та методи обчислень»

Тема «Алгоритми пошуку.»

Студентка гр. КН-23-1 ПІБ Варич А.І

Викладач к. т. н., доц. В. М. Сидоренко

Кременчук 2024

**ЗМІСТ**

[1 Завдання. Лінійний пошук. 3](#_Toc164602382)

[2 Завдання. Бінарний пошук. 4](#_Toc164602383)

[3 Завдання. Алгоритм тернарного пошуку 4](#_Toc164602384)

[4 Завдання. Порівняння деяких алгоритмів. 5](#_Toc164602385)

[5 Завдання. Порівняння алгоритмів з відсортованими та не відсортованими списками. 8](#_Toc164602386)

[6 Завдання. Використання кожного з алгоритмів. 9](#_Toc164602387)

[7 Контрольні запитання 10](#_Toc164602388)

# Завдання. Лінійний пошук.

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

У найгіршому випадку алгоритм лінійного пошуку пройде через усі елементи списку, тому його складність у цьому випадку буде O(n), де n - кількість елементів у списку.

У найкращому випадку, якщо елемент знаходиться на початку списку, алгоритм здійснить лише одну перевірку. Таким чином, складність у найкращому випадку також буде O(1).

А покращити алгоритм лінійного пошуку можна, якщо використовувати бінарний пошук у впорядкованому списку. Це дозволяє значно ефективніше знаходити елемент, оскільки на кожному кроці алгоритму відкидається половина списку.

# Завдання. Бінарний пошук.

**Постановка задачі:** Оцінити асимптотичну складність алгоритму бінарного пошуку у 𝑂-нотації в найгіршому і в найкращому випадку.

У найгіршому випадку алгоритм бінарного пошуку пройде O(log n) порівнянь, де n - кількість елементів у впорядкованому списку. Це трапляється, коли елемент відсутній у списку або знаходиться в останніх елементах.

А у найкращому випадку алгоритм також пройде O(1) порівнянь. Це може бути, коли шуканий елемент знаходиться в самому середині списку.

# Завдання. Алгоритм тернарного пошуку

**Постановка задачі:** 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 є модифікацією бінарного пошуку і полягає в тому, що він ділить інтервал пошуку на три частини замість двох. По суті, це означає, що ми порівнюємо шуканий елемент з двома середніми елементами в списку, а потім визначаємо, в якій з трьох частин може знаходитися шуканий елемент, і продовжуємо пошук в цій частині.

У тернарному пошуку на кожному кроці ми ділимо інтервал пошуку на три частини. Це означає, що ми перевіряємо два середніх елементи, щоб визначити, в якій частині списку може знаходитися шуканий елемент.

В найгіршому випадку може бути, якщо елемент відсутній у списку або знаходиться в останній третині списку, тоді на кожному кроці ми перевіряємо дві третини елементів. Таким чином, кількість порівнянь буде log3n, де n - кількість елементів у списку.

А у найкращому, якщо шуканий елемент знаходиться в самому середині списку, тоді на першому кроці ми знаходимо його за одне порівняння. Тому складність буде O(1).

Якщо розглядати лише основу логарифма, то log 3n більше, ніж log 2n для всіх n>1. Тому тернарний пошук зазвичай буде трохи менш ефективним, ніж бінарний пошук. Однак різниця в ефективності буде дуже незначною.

# Завдання. Порівняння деяких алгоритмів.

**Постановка задачі:** 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(рис. 1 - результат).

import time

import random

import matplotlib.pyplot as plt

def linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

def binary\_search(arr, target):

left, right = 0, len(arr) - 1

while left <= right:

mid = (left + right) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

left = mid + 1

else:

right = mid - 1

return -1

def ternary\_search(arr, target):

left, right = 0, len(arr) - 1

while left <= right:

mid1 = left + (right - left) // 3

mid2 = right - (right - left) // 3

if arr[mid1] == target:

return mid1

elif arr[mid2] == target:

return mid2

elif target < arr[mid1]:

right = mid1 - 1

elif target > arr[mid2]:

left = mid2 + 1

else:

left = mid1 + 1

right = mid2 - 1

return -1

def measure\_search\_time(search\_func, arr, target):

start\_time = time.time()

search\_func(arr, target)

end\_time = time.time()

return end\_time - start\_time

sizes = [100, 500, 1000, 5000, 10000]

linear\_times, binary\_times, ternary\_times = [], [], []

for size in sizes:

arr = sorted(random.sample(range(10 \* size), size))

target = random.randint(0, size \* 10)

linear\_times.append(measure\_search\_time(linear\_search, arr, target))

binary\_times.append(measure\_search\_time(binary\_search, arr, target))

ternary\_times.append(measure\_search\_time(ternary\_search, arr, target))

plt.plot(sizes, linear\_times, marker='o', label='лінійний пошук')

plt.plot(sizes, binary\_times, marker='o', label='бінарний пошук')

plt.plot(sizes, ternary\_times, marker='o', label='тернарний пошук')

plt.xlabel('розмір списку вхідних даних')

plt.ylabel('час(секунди)')

plt.title('порівняння пошукових алгоритмів')

plt.legend()

plt.grid(True)

plt.show()

На малюнку показано результат (графік):

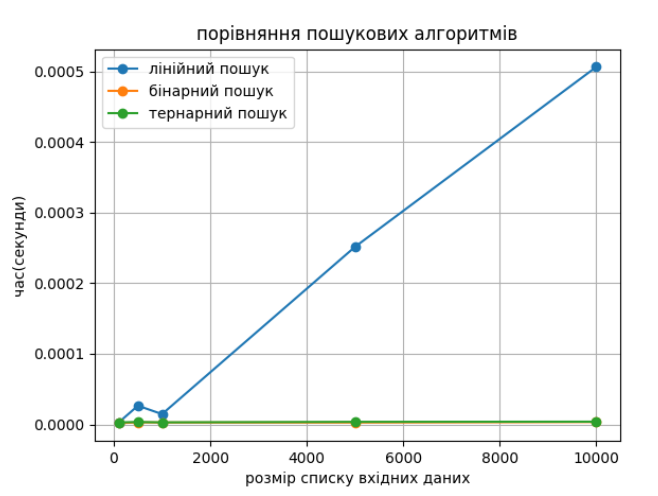


рис. 1 – результат

# Завдання. Порівняння алгоритмів з відсортованими та не відсортованими списками.

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

Алгоритм лінійного пошуку, здатний працювати з будь-якими списками, як відсортованими, так і не відсортованими, в часовій складності не залежить від відсортованості списку і залишається лінійним, проходить через кожен елемент списку, тому відсортованість не впливає на час його виконання.

Бінарний пошук, працює ефективно лише з відсортованими списками. Це означає, що перед використанням бінарного пошуку необхідно впорядкувати список, часова складність бінарного пошуку O(logn), де n - кількість елементів у відсортованому списку. Тому, таким чином, впорядкування списку перед пошуком може займати додатковий час, але сам пошук буде більш ефективним.

А тернарний пошук, аналогічно до бінарного пошуку, працює ефективно лише з відсортованими списками, Часова складність тернарного пошуку у найгіршому випадку O(log 3n), що також залежить від відсортованості списку.

Тому, аналіз впливу відсортованості списку на час виконання кожного алгоритму показує, що для бінарного та тернарного пошуку відсортованість є ключовим фактором, що впливає на ефективність. У той час як для лінійного пошуку цей фактор не має значення, оскільки алгоритм працює однаково з будь-якими списками.

# Завдання. Використання кожного з алгоритмів.

**Постановка задачі:** Розглянути сценарії використання кожного з алгоритмів пошуку у практичних задачах і обґрунтувати вибір кожного алгоритму в конкретному випадку.

У лінійному пошуку, якщо список невеликий або немає потреби в оптимальності алгоритму, лінійний пошук може бути простим та ефективним варіантом, якщо порядок елементів у списку не має значення, або список не впорядкований, то лінійний пошук є найбільш практичним варіантом.

У бінарному пошуку, якщо маємо великий впорядкований список, то бінарний пошук може значно скоротити час пошуку порівняно з лінійним. Наприклад, пошук у великих базах даних, а якщо потрібно виконати багато пошуків у великому впорядкованому списку, то вартує витратити час на сортування списку один раз, а потім використовувати бінарний пошук для кожного пошуку.

А у тернарному, може бути вибраний, якщо він виявиться ефективнішим або легше реалізовується у мові програмування Пайтон, у деяких випадках, де потрібно виконати пошук у складних структурах даних, які не завжди ідеально впорядковані, тернарний пошук може забезпечити більшу гнучкість порівняно з бінарним. Наприклад, у деяких графах або деревах, тернарний пошук може бути ефективним рішенням.

# Контрольні запитання

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

Алгоритм пошуку - це процедура або набір кроків, які використовуються для знаходження конкретного елемента (або набору елементів) в певній структурі даних, такій як масив, список, дерево, граф і т. д. Цей процес включає в себе ітерацію через елементи структури даних і порівняння їх з шуканим елементом з метою знайти його місце або визначити, чи існує він взагалі.

1. **Які основні критерії оцінки ефективності алгоритмів пошуку?**

Це може бути: Часова складність, просторова складність, швидкість збігу.

1. **Що таке лінійний пошук, і як він працює?**

Лінійний пошук - це алгоритм, що перевіряє кожен елемент послідовно у структурі даних до знаходження шуканого або кінця списку.

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

Успішне застосування бінарного пошуку вимагає впорядкованого списку і унікальних елементів.

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

Переваги: ефективність (швидкість пошуку), оптимальність (для великих впорядкованих списків).

Недоліки: потрібно впорядкований список, складність реалізації, висока чутливість до додавання або видалення елементів.

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

Тернарний пошук ділить список на три рівні частини, що може призводити до меншої кількості порівнянь, але більшої кількості кроків порівняно з бінарним пошуком.