### ****Практична робота №4****

**Тема: Алгоритми пошуку та їх складність**

**Завдання №1**.

### ****1. Асимптотична складність алгоритму:****

#### Найкращий випадок:

* Потрібний елемент є першим у масиві.
* Виконується лише одна перевірка.
* Складність: O(1)

#### Найгірший випадок:

* Потрібний елемент є останнім у масиві або його взагалі немає.
* У цьому випадку потрібно перевірити всі nn елементів.
* Складність: O(n)

#### Середній випадок:

* У середньому алгоритм перебирає половину елементів масиву.
* Але в O-нотації ми враховуємо домінуючі складові, тому: O(n)

### ****2. Покращення алгоритму лінійного пошуку:****

#### ****Сентинельний пошук****:

* Додається "сентинель" (штучний елемент) у кінець масиву, щоб уникнути перевірки виходу за межі масиву.
* Це зменшує кількість перевірок меж масиву, роблячи пошук трохи швидшим.

#### ****Сортування даних перед пошуком:****

* Якщо масив відсортовано, можна використовувати методи ранньої зупинки:
  + Якщо поточний елемент перевищує шуканий, пошук припиняється.
  + Для частково відсортованих масивів це може значно скоротити час виконання.

#### ****Індексація або хешування:****

* Створення хеш-таблиці або індексаційної структури для масиву дозволяє виконувати пошук за O(1) у середньому випадку.

#### ****Паралельний пошук:****

* Якщо доступні багатоядерні процесори, масив можна розбити на частини й виконувати пошук у них одночасно.

### ****3. Чому лінійний пошук неефективний для великих даних?****

* Лінійний пошук завжди потребує O(n) часу в найгіршому випадку, тому для великих масивів це повільно.
* Для відсортованих масивів краще використовувати алгоритми, наприклад, **бінарний пошук**, який має складність O(log n).

### ****Висновки****

* Лінійний пошук ефективний для невеликих масивів або в ситуаціях, коли дані невідсортовані і потрібна проста реалізація.
* Для великих даних або масивів, що часто використовуються, можна розглянути покращення, як-от хешування, паралельний пошук або бінарний пошук (за умови попереднього сортування).