جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قسم الانظمة الطبية الذكية

Intelligent Medical Systems Department

Lap: (8)

Searching Algorithms in Python

Subject: Data Strectuer

Class: second

Lecturer: Prof.Dr. Mehdi Ebady Manaa

Programmer:- Fatima Hussein Jawad

Study Year: 2024-2025

Objectives:

1. Understand the concept of searching.
2. Learn the differences between the basic algorithms:
   - Linear Search
   - Binary Search
3. Implement simple Python examples to demonstrate the algorithms.

1. Linear Search:

Concept:

- Sequentially checks each element in the list until the desired element is found or the list ends.

Steps:

1. Start at the first element.
2. If it matches the target, stop.
3. If not, move to the next element.
4. Repeat until the target is found or the list ends.

 # Example

```
def linear_search(arr, target):
    for index in range(len(arr)):
        if arr[index] == target:
            return f"Element {target} found at index {index}."
    return f"Element {target} not found in the list."
```

```
numbers = [10, 20, 30, 40, 50]

print(linear_search(numbers, 30))  # Found

print(linear_search(numbers, 60))  # Not Found
```

output :

Element 30 found at index 2.

Element 60 not found in the list.

2. Binary Search:

Concept:

- Only works with sorted lists.
- Splits the list into halves repeatedly, narrowing the search range.

Steps:

1. Find the middle element.
2. If it matches the target, stop.
3. If the target is smaller, search in the left half.
4. If the target is larger, search in the right half.
5. Repeat until the target is found or the search range is empty.

 # Example

```
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1
    while left <= right:
```

```
    mid = (left + right) // 2  # Find the middle element
    if arr[mid] == target:
        return f"Element {target} found at index {mid}."
    elif arr[mid] < target:
        left = mid + 1  # Search in the right half
    else:
        right = mid - 1  # Search in the left half
  return f"Element {target} not found in the list."
sorted_numbers = [10, 20, 30, 40, 50]
print(binary_search(sorted_numbers, 40))  # Found
print(binary_search(sorted_numbers, 15))  # Not Found
```

output :

Element 40 found at index 3.

Element 15 not found in the list.

3. Comparison: Linear Search VS Binary Search

Aspect	Linear Search	Binary Search
Requirement	Any list	Sorted list
Efficiency	Slow: O(n)	Fast: O(log n)
Ease of Implementation	Easy	More complex

4. Additional Examples:

Practical Example - Searching in a List of Names:

Code

```
print(linear_search(numbers, 30))  # Found
print(linear_search(numbers, 60))  # Not Found
def binary_search(arr, target):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid = (left + right) // 2  # Find the middle element
        if arr[mid] == target:
            return f"Element {target} found at index {mid}."
        elif arr[mid] < target:
            left = mid + 1  # Search in the right half
        else:
            right = mid - 1  # Search in the left half
    return f"Element {target} not found in the list."
sorted_numbers = [10, 20, 30, 40, 50]
print(binary_search(sorted_numbers, 40))  # Found
print(binary_search(sorted_numbers, 15))  # Not Found
```

```
names = ["Ali", "Hassan", "Fatima", "Sara", "Ahmed"]

# Linear Search

print(linear_search(names, "Fatima"))

print(linear_search(names, "Zainab"))

# Binary Search (after sorting the list)

names.sort()

print(binary_search(names, "Fatima"))

print(binary_search(names, "Zainab"))
```

output :

```
Python 3.12.6 (tags/v3.12.6:a4a2d2b, Sep  6 2024, 20:11:23) [MSC v.1940 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\ALFA\Desktop\python_3\lap_A.py
Element 30 found at index 2.
Element 60 not found in the list.
Element 40 found at index 3.
Element 15 not found in the list.
Element Fatima found at index 2.
Element Zainab not found in the list.
Element Fatima found at index 2.
Element Zainab not found in the list.
```

Linear Search Code Without Built-in Functions:

Here is a Python implementation of linear search without using built-in functions like `find` or `index`:

Code :

```
def custom_linear_search(arr, target):
    # Traverse each element one by one
    for i in range(len(arr)):
        if arr[i] == target:  # If the target is found
            return f"Element {target} found at index {i}."
    return f"Element {target} not found in the list."  # If the target is not found
numbers = [5, 12, 7, 19, 25]
print(custom_linear_search(numbers, 7))  # Found
print(custom_linear_search(numbers, 20))  # Not Found
```

output :

Element 7 found at index 2.

Element 20 not found in the list.

Binary Search Code Without Built-in Functions:

Note: The list must be sorted before performing a binary search.

```
def custom_binary_search(arr, target):
    left = 0
    right = len(arr) - 1
    while left <= right:
        mid = (left + right) // 2  # Find the middle element
```

```
    if arr[mid] == target:  # If the middle element is the target
        return f"Element {target} found at index {mid}."
    elif arr[mid] < target:  # If the target is greater than the middle element
        left = mid + 1  # Search in the right half
    else:  # If the target is smaller
        right = mid - 1  # Search in the left half
  return f"Element {target} not found in the list."  # If the target is not found
sorted_numbers = [3, 8, 15, 20, 36]
print(custom_binary_search(sorted_numbers, 15))  # Found
print(custom_binary_search(sorted_numbers, 10))  # Not Found
```

output :

Element 15 found at index 2.

Element 10 not found in the list.

Key Notes:

1. **The above code does not rely on any built-in functions.**
   - Elements are manually checked using loops or calculations.
2. **Binary search requires the list to be sorted beforehand.**
   If you want, you can implement a sorting algorithm like Bubble Sort to sort the list first.

Home work

- How can you optimize the linear search code to make it faster?
- What is the difference between using `while` and `for` loops in these algorithms?
- Write your own algorithm to find the last element in a list.