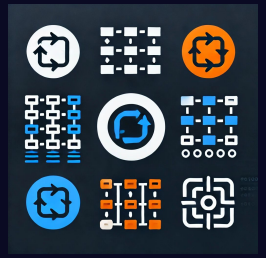


Linear Search

The Search We Use Every Day

Recap: Introduction to Algorithms

- Understanding Algorithms
  - Find student with maximum grade

Recap: Introduction to Algorithms

- Understanding Algorithms
  - Find student with maximum grade
- Expressing Algorithms
  - Pseudocode

Recap: Introduction to Algorithms

- Understanding Algorithms
  - Find student with maximum grade
- Expressing Algorithms
  - Pseudocode
  - Python Implementation

Recap: Introduction to Algorithms

- Understanding Algorithms
  - Find student with maximum grade
- Expressing Algorithms
  - Pseudocode
  - Python Implementation
  - Functions: find_max

Recap: Introduction to Algorithms

- Understanding Algorithms
  - Find student with maximum grade
- Expressing Algorithms
  - Pseudocode
  - Python Implementation
  - Functions: find_max

Recap: Introduction to Algorithms

- Understanding Algorithms
  - Find student with maximum grade
- Expressing Algorithms
  - Pseudocode
  - Python Implementation
  - Functions: find_max
- find_max

Recap: Introduction to Algorithms

- Understanding Algorithms
  - Find student with maximum grade
- Expressing Algorithms
  - Pseudocode
  - Python Implementation
  - Functions: find_max
- find_max
  - Step count, or complexity

Recap: Introduction to Algorithms

- Understanding Algorithms
  - Find student with maximum grade
- Expressing Algorithms
  - Pseudocode
  - Python Implementation
  - Functions: find_max
- find_max
  - Step count, or complexity

} **Second Video**

Recap of Complexity Analysis

```
1. def find_max(numbers):
2.     max_value = numbers[0]
3.     for number in numbers:
4.         if number > max_value:
5.             max_value = number
6.     return max_value
```

Recap of Complexity Analysis

```
1. def find_max(numbers):
2.     max_value = numbers[0]
3.     for number in numbers:
4.         if number > max_value:
5.             max_value = number
6.     return max_value
```

Recap of Complexity Analysis

```
1. def find_max(numbers):
2.     max_value = numbers[0]
3.     for number in numbers:
4.         if number > max_value:
5.             max_value = number
6.     return max_value
```

Depends on input size (i.e., length of the list of numbers)

Recap of Complexity Analysis

```
1. def find_max(numbers):
2.     max_value = numbers[0]
3.     for number in numbers:          n steps
4.         if number > max_value:
5.             max_value = number
6.     return max_value
```

Recap of Complexity Analysis

```
1. def find_max(numbers):                1 step
2.     max_value = numbers[0]            1 step
3.     for number in numbers:            n steps
4.         if number > max_value:
5.             max_value = number
6.     return max_value                  1 step
```

Recap of Complexity Analysis

1.	def find_max(numbers):	1 step
2.	max_value = numbers[0]	1 step
3.	for number in numbers:	n steps
4.	if number > max_value:	n steps
5.	max_value = number	n steps
6.	return max_value	1 step

Recap of Complexity Analysis

```
1. def find_max(numbers):          1 step
2.     max_value = numbers[0]      1 step
3.     for number in numbers:      n steps
4.         if number > max_value:  n steps
5.             max_value = number  n steps
6.     return max_value            1 step
```

1 + 1 + n + n + n + 1 = 3n + 3 steps

Recap of Complexity Analysis

1.	def find_max(numbers):	1 step
2.	max_value = numbers[0]	1 step
3.	for number in numbers:	n steps
4.	if number > max_value:	n steps
5.	max_value = number	n steps
6.	return max_value	1 step

1 + 1 + n + n + n + 1 = 3n + 3 steps

For a list of size **n**,
the total steps are **roughly n**
➜ linear complexity

Recap of Complexity Analysis

1.	def find_max(numbers):	1 step
2.	max_value = numbers[0]	1 step
3.	for number in numbers:	n steps
4.	if number > max_value:	n steps
5.	max_value = number	n steps
6.	return max_value	1 step

1 + 1 + n + n + n + 1 = 3n + 3 steps

For a list of size **n**,
the total steps are **roughly n**
→ linear complexity

3n + 3 ≈ O(n)

Recap of Complexity Analysis

1.	def find_max(numbers):	1 step
2.	max_value = numbers[0]	1 step
3.	for number in numbers:	n steps
4.	if number > max_value:	n steps
5.	max_value = number	n steps
6.	return max_value	1 step
		1 + 1 + n + n + n + 1 = 3n + 3 steps

For a list of size **n**,
the total steps are **roughly n**
→ linear complexity

3n + 3 ≈ O(n)

~~**3n + 3 ≈ O(n)**~~

Recap of Complexity Analysis

1.	def find_max(numbers):	1 step
2.	max_value = numbers[0]	1 step
3.	for number in numbers:	n steps
4.	if number > max_value:	n steps
5.	max_value = number	n steps
6.	return max_value	1 step
		1 + 1 + n + n + n + 1 = 3n + 3 steps

For a list of size **n**,
the total steps are **roughly n**
→ linear complexity

3n + 3 ≈ O(n)

~~3n + 3~~ ≈ **O(n)**

Introducing Linear Search

Algorithm FindMax

Algorithm LinearSearch

Introducing Linear Search

Algorithm FindMax
Input: A list of numbers

Algorithm LinearSearch
Input: A list of numbers, a target

Introducing Linear Search

Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Introducing Linear Search

Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Begin

Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Begin

Introducing Linear Search

Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Begin
    max ← list[0]

Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Begin
    ~~max ← list[0]~~

Introducing Linear Search

Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Begin
    max ← list[0]
    for each number in numbers do

Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Begin
    ~~max ← list[0]~~
    for each number in numbers do

Introducing Linear Search

```
Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Begin
    max ← list[0]
    for each number in numbers do
        if number > max then
```

```
Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Begin
    max ← list[0]
    for each number in numbers do
        if number == target then
```

Introducing Linear Search

```
Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Begin
    max ← list[0]
    for each number in numbers do
        if number > max then
            max ← number
```

```
Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Begin
    max ← list[0]
    for each number in numbers do
        if number == target then
            return index of number
```

Introducing Linear Search

Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

```
Begin
    max ← list[0]
    for each number in numbers do
        if number > max then
            max ← number
        end if
    end for
```

Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

```
Begin
    max ← list[0]
    for each number in numbers do
        if number == target then
            return index of number
        end if
    end for
```

Introducing Linear Search

```
Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Begin
    max ← list[0]
    for each number in numbers do
        if number > max then
            max ← number
        end if
    end for
    return max
```

```
Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Begin
    max ← list[0]
    for each number in numbers do
        if number == target then
            return index of number
        end if
    end for
    return -1
```

Introducing Linear Search

```
Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Begin
    max ← list[0]
    for each number in numbers do
        if number > max then
            max ← number
        end if
    end for
    return max
End
```

```
Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Begin
    max ← list[0]
    for each number in numbers do
        if number == target then
            return index of number
        end if
    end for
    return -1
End
```

Introducing Linear Search

```
Algorithm FindMax
Input: A list of numbers
Output: The maximum number in the list

Begin
    max ← list[0]
    for each number in numbers do
        if number > max then
            max ← number
        end if
    end for
    return max
End
```

```
Algorithm LinearSearch
Input: A list of numbers, a target
Output: The position of target or -1

Begin
    max ← list[0]
    for each number in numbers do
        if number == target then
            return index of number
        end if
    end for
    return -1
End
```

Implementing Linear Search

```
def find_max(numbers):

    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number


sample_numbers = [5, 3, 9, 1, 6]
print("The maximum number is:",
find_max(sample_numbers))
```

```
def linear_search(numbers, target):
```

Implementing Linear Search

```
def find_max(numbers):

    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number


sample_numbers = [5, 3, 9, 1, 6]
print("The maximum number is:",
find_max(sample_numbers))
```

```
def linear_search(numbers, target):


    for number in numbers:
```

Implementing Linear Search

```
def find_max(numbers):

    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number


sample_numbers = [5, 3, 9, 1, 6]
print("The maximum number is:",
find_max(sample_numbers))
```

```
def linear_search(numbers, target):


    for number in numbers:
        if number == target:
```

Implementing Linear Search

```
def find_max(numbers):

    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number


sample_numbers = [5, 3, 9, 1, 6]
print("The maximum number is:",
find_max(sample_numbers))
```

```
def linear_search(numbers, target):


    for number in numbers:
        if number == target:
            return ?
```

Implementing Linear Search

```
def find_max(numbers):

    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number


sample_numbers = [5, 3, 9, 1, 6]
print("The maximum number is:",
find_max(sample_numbers))
```

```
def linear_search(numbers, target):


    for number in numbers:
        if number == target:
            return ?
    return -1
```

Implementing Linear Search

```
def find_max(numbers):

    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number


sample_numbers = [5, 3, 9, 1, 6]
print("The maximum number is:",
find_max(sample_numbers))
```

```
def linear_search(numbers, target):


    for number in numbers:
        if number == target:
            return ?
    return -1


sample_list = [5, 3, 9, 1, 6]
target = 1
```

Implementing Linear Search

```
def find_max(numbers):

    max_number = numbers[0]
    for number in numbers:
        if number > max_number:
            max_number = number
    return max_number


sample_numbers = [5, 3, 9, 1, 6]
print("The maximum number is:",
find_max(sample_numbers))
```

```
def linear_search(numbers, target):


    for number in numbers:
        if number == target:
            return ?
    return -1


sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :",
linear_search(sample_list, target))
```

Refining Implementation

```
def linear_search(numbers, target):

    for number in numbers:
        if number == target:
            return ?
    return -1

sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :", linear_search(sample_list, target))
```

Refining Implementation

```
def linear_search(numbers, target):

    for number in numbers:        for i in range(len(numbers)):
        if number == target:
            return ?
    return -1

sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :", linear_search(sample_list, target))
```

Refining Implementation


```
def linear_search(numbers, target):

    for number in numbers:          for i in range(len(numbers)):
        if number == target:
            return ?
    return -1


sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :", linear_search(sample_list, target))
```

Refining Implementation


```
def linear_search(numbers, target):

    for number in numbers:
        if number == target:
            return ?
    return -1
```

```
for i in range(len(numbers)):
    if numbers[i] == target:
```

```
sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :", linear_search(sample_list, target))
```

Refining Implementation


```
def linear_search(numbers, target):

    for number in numbers:
        if number == target:
            return ?
    return -1
```

```
for i in range(len(numbers)):
    if numbers[i] == target:
        return i
```

```
sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :", linear_search(sample_list, target))
```

Refining Implementation


```
def linear_search(numbers, target):

    for i in range(len(numbers)):
        if numbers[i] == target:
            return i
    return -1

sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :", linear_search(sample_list, target))
```

Refining Implementation

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

sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :", linear_search(sample_list, target))
```

Refining Implementation

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

sample_list = [5, 3, 9, 1, 6]
target = 1
print("The target is found at :", linear_search(sample_list, target))
index = linear_search(sample_list, target)
```

Refining Implementation

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

sample_list = [5, 3, 9, 1, 6]
target = 1
index = linear_search(sample_list, target)
if index == -1:
    print("The target is not found!")
```

Refining Implementation

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

sample_list = [5, 3, 9, 1, 6]
target = 1
index = linear_search(sample_list, target)
if index == -1:
    print("The target is not found!")
else:
    print("The target is found at location:", index)
```

Refining Implementation

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

sample_list = [5, 3, 9, 1, 6]
target = 1
index = linear_search(sample_list, target)
if index == -1:
    print("The target is not found!")
else:
    print("The target is found at location:", index)
```

Refining Implementation

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

sample_list = [5, 3, 9, 1, 6]
target = int(input("Enter the number to search: "))
index = linear_search(sample_list, target)
if index == -1:
    print("The target is not found!")
else:
    print("The target is found at location:", index)
```

Complexity of Linear Search

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

1 step

Complexity of Linear Search

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

1 step

n steps

Complexity of Linear Search

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

1 step

n steps

Worst Case → n steps

Complexity of Linear Search

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

1 step

n steps

Worst Case → n steps

Best Case → 1 step

Complexity of Linear Search

```
def linear_search(numbers, target):          1 step

    for i in range(len(numbers)):            n steps
        if numbers[i] == target:             n steps
            return i
    return -1
```

Complexity of Linear Search

```
def linear_search(numbers, target):          1 step

    for i in range(len(numbers)):            n steps
        if numbers[i] == target:             n steps
            return i                         1 step
    return -1
```

Complexity of Linear Search

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

1 step

n steps

n steps

1 step

1 step

Complexity of Linear Search

```
def linear_search(numbers, target):          1 step

    for i in range(len(numbers)):            n steps
        if numbers[i] == target:             n steps
            return i                         1 step
    return -1                                1 step
```

1 + n + n + 1 + 1 = 2n + 3 steps

Complexity of Linear Search

```
def linear_search(numbers, target):          1 step

    for i in range(len(numbers)):            n steps
        if numbers[i] == target:             n steps
            return i                         1 step
    return -1                                1 step
```

1 + n + n + 1 + 1 = 2n + 3 steps

2n + 3 ≈ O(n)

Complexity of Linear Search

```
def linear_search(numbers, target):          1 step

    for i in range(len(numbers)):            n steps
        if numbers[i] == target:             n steps
            return i                         1 step
    return -1                                1 step
```

1 + n + n + 1 + 1 = 2n + 3 steps

2n + 3 ≈ O(n)

~~2n + 3 ≈~~ **O(n)**

Complexity of Linear Search

```
def linear_search(numbers, target):          1 step

    for i in range(len(numbers)):            n steps
        if numbers[i] == target:             n steps
            return i                         1 step
    return -1                                1 step
```

1 + n + n + 1 + 1 = 2n + 3 steps

2n + 3 ≈ O(n)

~~2n + 3 ≈~~ **O(n)**

Worst Case: O(n)

Complexity of Linear Search

```
def linear_search(numbers, target):          1 step

    for i in range(len(numbers)):            n steps
        if numbers[i] == target:             n steps
            return i                         1 step
    return -1                                1 step
```

1 + n + n + 1 + 1 = 2n + 3 steps

2n + 3 ≈ O(n)

~~2n + 3 ≈~~ **O(n)**

Worst Case: O(n)

Best Case: O(1)

Complexity of Linear Search

```
def linear_search(numbers, target):          1 step

    for i in range(len(numbers)):            n steps
        if numbers[i] == target:             n steps
            return i                         1 step
    return -1                                1 step
```

1 + n + n + 1 + 1 = 2n + 3 steps

2n + 3 ≈ O(n)

~~2n + 3 ≈~~ **O(n)**

Linear Search Is a Linear Time Algorithm

Worst Case: O(n)

Best Case: O(1)

Summary and Next Steps

- What We Covered:

Summary and Next Steps

- What We Covered:
  - Analyzed complexity of **find_max**

Summary and Next Steps

- What We Covered:
  - Analyzed complexity of **find_max**
  - Introduced **Big Oh** notation

Summary and Next Steps

- What We Covered:
  - Analyzed complexity of **find_max**
  - Introduced **Big Oh** notation
  - Explored the **linear_search** algorithm in comparison to **find_max**.

Summary and Next Steps

- What We Covered:
  - Analyzed complexity of **find_max**
  - Introduced **Big Oh** notation
  - Explored the **linear_search** algorithm in comparison to **find_max**.
  - Analyzed the best case, and worst case scenarios for complexity.

Summary and Next Steps

- What We Covered:
  - Analyzed complexity of **find_max**
  - Introduced **Big Oh** notation
  - Explored the **linear_search** algorithm in comparison to **find_max**.
  - Analyzed the best case, and worst case scenarios for complexity.
- Exercise:
  - Action Required: Visit our GitHub page (link in the description) to complete the exercise.

Summary and Next Steps

- What We Covered:
  - Analyzed complexity of **find_max**
  - Introduced **Big Oh** notation
  - Explored the **linear_search** algorithm in comparison to **find_max**.
  - Analyzed the best case, and worst case scenarios for complexity.
- Exercise:
  - Action Required: Visit our GitHub page (link in the description) to complete the exercise.
  - Objective: Analyze step counts in **different linear_search scenarios**.

Summary and Next Steps

- What We Covered:
  - Analyzed complexity of **find_max**
  - Introduced **Big Oh** notation
  - Explored the **linear_search** algorithm in comparison to **find_max**.
  - Analyzed the best case, and worst case scenarios for complexity.
- Exercise:
  - Action Required: Visit our GitHub page (link in the description) to complete the exercise.
  - Objective: Analyze step counts in **different linear_search scenarios**.

Coming Up Next:

Topic: Binary Search—An efficient algorithm for **sorted lists**.

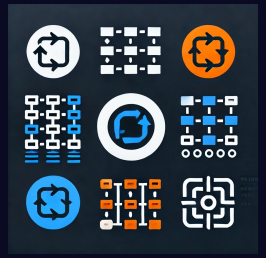
Summary and Next Steps

- What We Covered:
  - Analyzed complexity of **find_max**
  - Introduced **Big Oh** notation
  - Explored the **linear_search** algorithm in comparison to **find_max**.
  - Analyzed the best case, and worst case scenarios for complexity.
- Exercise:
  - Action Required: Visit our GitHub page (link in the description) to complete the exercise.
  - Objective: Analyze step counts in **different linear_search scenarios**.

Coming Up Next:

Topic: Binary Search—An efficient algorithm for **sorted lists**.

Highlights: Learn how Binary Search dramatically **reduces search time** by halving the search range.



Thank You for Watching!