CIS 1100

Searching

Python
Spring 2025
University of Pennsylvania

Overview

We often need to search for an item in a collection

- *Is this student in this recitation roster?*
- *Is this username in our user database?*
- *Is there any data point in our dataset that matches this description?*

In this module, we will learn about how to search for an element in a list.

Learning Objectives

- To be able to use **linear search** to find an element inside an sequence
- To be able to use **binary search** to find an element inside an sequence
- To be able to know when to use linear search and when to use binary search

What Is *Efficiency?*

(L11) Discuss with a partner and write down: what quality or qualities of a program make it more or less "efficient"?

Our Primary Concern: Runtime Efficiency

All code takes time to run. A simple heuristic is that a function's runtime is proportional to the number of iterations of the loops it takes to execute.

Speedy Snakes

Let's approximate "speed" with printed snakes: 🐍. Each iteration of the loop prints a snake, so "speed" ≈ runtime.

Recall that `isupper()` is a method of strings that returns `True` only when all letters are uppercase. We could write our own implementation in a couple of ways...

```
def isupper_one(word):
    all_upper = True
    for letter in word:
        print("🐍")
        if not "A" <= letter <= "Z":
            all_upper = False
    return all_upper
```

```
def isupper_two(word):
    for letter in word:
        print("🐍")
        if not "A" <= letter <= "Z":
            return False
    return True
```

(S7): How many snakes are printed if we run `isupper_one("GaRBANZO")`?

(S8): How about `isupper_two("GaRBANZO")`?

Early Returns

For problems that are just long-winded "and"/"or", you can often return early!

- `isupper()` $\rightarrow$ "are all letters uppercase?" $\rightarrow$ "the first letter is uppercase **and** the second letter is uppercase **and** the third letter is uppercase **and**...
- logical `and` is `True` only if both of its arguments are `True`
- logical `and` is always `False` if any of its arguments are `False`
- $\Rightarrow$ safe to return `False` as soon as any boolean in a chain of "ands" is `False`!

This is called *boolean short-circuiting*, and it's built into Python for safety and efficiency.

Roadmap

1. Formalize the problem of **search**
2. Propose a simple algorithm for searching and analyze its runtime in several cases
3. Propose a more complex algorithm for searching and analyze its runtime, too
4. Compare the algorithms and conclude which is more efficient.

Problem: Search

Formally, given a sequence of values and a target value, we want to determine if the target value is in the sequence, and if so, where it is located.

Solution: `.index()`

Python has a built-in solution: `sequence.index(target)` returns the position of `target` inside of the `sequence`, or raises a `ValueError` if the `target` is not present.

- You'll just use this (or `.find()` for strings) most of the time
- BUT!
  - How does it work?
  - `index()` the best solution in all cases? Are there better strategies?
    - (No, and there are.)

Problem: Search

Formally, given a sequence of values and a target value, we want to determine if the target value is in the sequence, and if so, where it is located.

- in our case, the "sequence of values" could be a list, tuple, string...
- the "target value" is the value we are searching for
- the location is the index of the value in the sequence, or `-1` if it's not present.

Concept: The "Feasible Region"

In any problem, the **feasible region** is the name for the set of possible values that might be a solution.

- In the context of search, the feasible region refers to the set of indices in the sequence that might contain the target value.
- A set of indices is functionally a region of the sequence where the target value might be found.

In our search algorithms, we repeatedly reduce the feasible region until we find the target value, or until we determine that the target value is not present in the sequence.

Quick Analogy: Lost Keys

(C12) Suppose you lose your keys in your home. You're at home, so you know they're somewhere in one of these rooms! You have a **bedroom**, **front room**, **kitchen**, and **bathroom**. Describe a procedure for searching for your keys.

CIS 1100

Linear Search

Python
Spring 2025
University of Pennsylvania

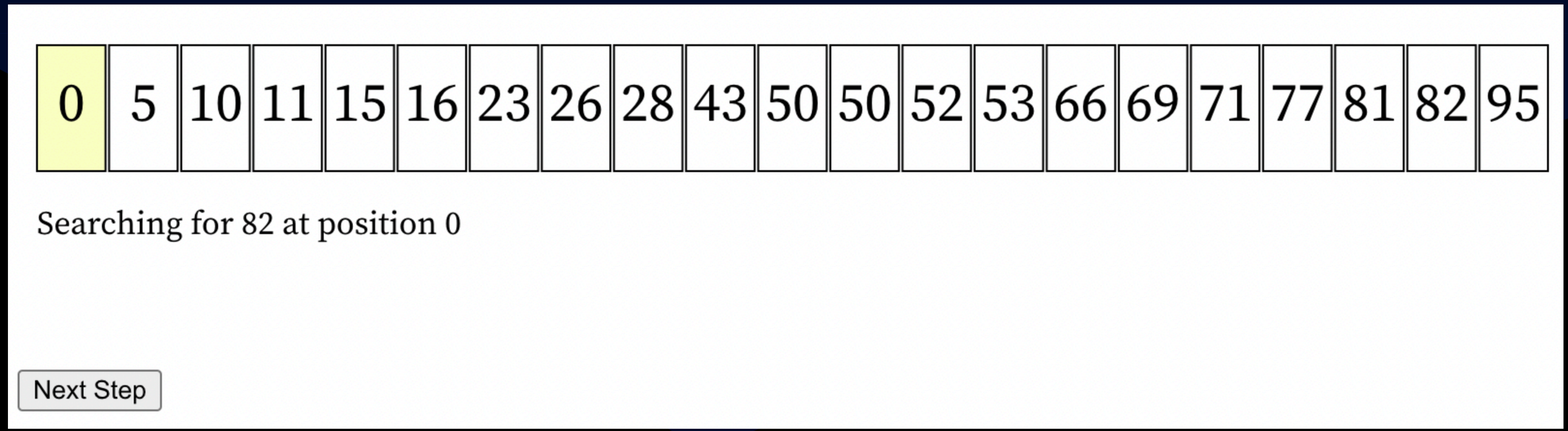
Linear Search

Used to search for a value (the target) in an **unsorted list**

- Use a loop to iterate over the values
- Start at the first element and move to the next element until the target is found
- Returns the position of the target if it was found in the sequence, or -1 if the target was not found in the sequence

With each iteration, we reduce the feasible region by one element.

Linear Search Example



(this image is a link)

Linear Search

```
def linear_search(sequence, target):
    for idx, element in enumerate(sequence):
        if element == target:
            return idx
    return -1
```

```
>>> linear_search(range(30, 300, 4), 30)
0
>>> linear_search(range(30, 300, 4), 262)
58
>>> linear_search(range(30, 300, 4), 31)
-1
```

Linear Search: Thinking Critically

```
def linear_search(sequence, target):
    for idx, element in enumerate(sequence):
        print("🐍")
        if element == target:
            return idx
    return -1
```

(S9) How many snakes get printed if...

- the target is the first element in the sequence?
- the target is the 10th element in the sequence?
- the target is not in the sequence?

Linear Search: Thinking Critically

How many iterations of the for loop will we need if...

- the target is the first element in the sequence? **1**
- the target is the 10th element in the sequence? **10**
- the target is not in the sequence? **len(sequence)**

Linear Search: Properties

Linear search is...

- **Complete:** we'll always get an answer
- **Correct:** we'll always get the right answer

These are desirable properties, but linear search is not always the most **efficient.**

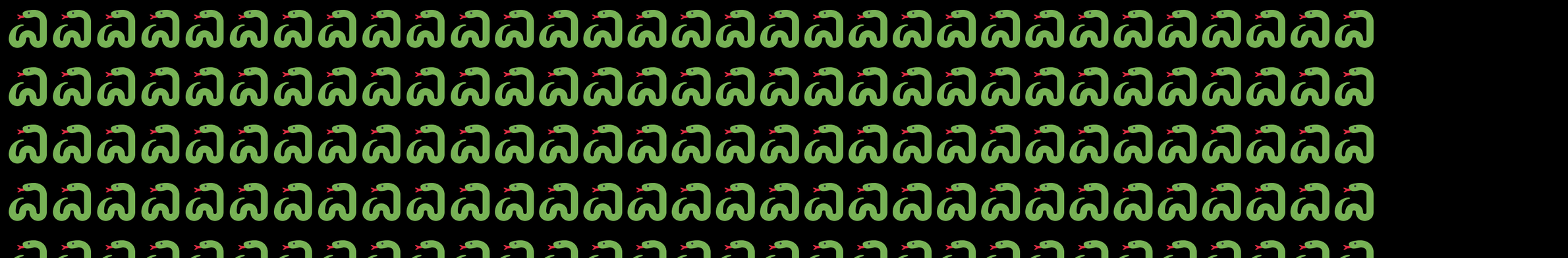
- May require more time (~more iterations) than other searching algorithms for "average use"

A Contrasting Point of View

Here's a dumb searching algorithm called **Bogo Search**

```python
from random import randrange
def bogo_search(sequence, target):
    while True:
        print("🐍")
        idx = randrange(len(sequence)) # picks a random index to look at
        if sequence[idx] == target:
            return idx
```

Not even **complete:** if we got unlucky, we could accidentally just look at the same (wrong) index infinitely many times in a row and never return.



CIS 1100

Binary Search

Python
Spring 2025
University of Pennsylvania

Can we do better than linear search? Can we be...

- complete?
- correct?
- faster?

The answer is "yes, yes, and **sometimes.**"

Binary Search

Used to search for a target value in an **(ascending) sorted sequence only**

- Compares the target with the value at the middle index (middle element)
  - If the middle element is the target element, then we're done!
  - If the target is less than the middle element, then we search for the target in the **left half of the sequence** (the positions before the middle element)
  - If the target is greater than the middle element, then we search the target in the **right half of the sequence** (the positions after the middle element)
- Repeat on the remaining search area of the sequence until
  - the element is found
  - there is no feasible search area left

Taking the Pulse

Quickly: True (A) or False (B)?

- **M1:** (2 + 4) // 2 == 3
- **M2:** (2 + 9) // 2 == 5.5
- **M3:** If a *sorted* array of numbers has the value `34.1` at index `9`, then the value `9` could be stored at index `34`.
- **M4:** If a *sorted* array of numbers has the value `34.1` at index `9`, then the value `34` could be stored at index `6`.

Binary Search

Searching for `"Dustin"` in the sequence `names`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
low			middle			high

- `middle = (low + high) // 2 = 3`
- `names[middle]` is `"Elliot"`, which comes after `"Dustin"` alphabetically.
- So, if `"Dustin"` is present, it must be between positions `0` and `middle - 1`.
  - ➡️ shift `high` one to the left of `middle`

Binary Search

Searching for `"Dustin"` in the sequence `names`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
low	middle	high				

- `middle = (low + high) /// 2 = 1`
- `names[middle]` is `"Debbie"`, which comes before `"Dustin"` alphabetically.
- So, if `"Dustin"` is present, it must be between positions `middle + 1` and `2`.
  - ➡️ shift `low` one to the right of `middle`

Binary Search

Searching for `"Dustin"` in the sequence `names`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
		low, middle, high				

- `middle = (low + high) // 2 = 2`
- `names[middle]` is `"Dustin"`, which is the target element! So, we return `middle`.

Binary Search: Searching for an Element not Present

Searching for `"Drew"` in the sequence `names`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
low			middle			high

- `middle = (low + high) // 2 = 3`
- `names[middle]` is `"Elliot"`, which comes after `"Drew"` alphabetically.
- So, if `"Drew"` is present, it must be between positions `0` and `middle - 1`.

Binary Search: Searching for an Element not Present

Searching for `"Drew"` in the sequence `names`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
low	middle	high				

- `middle = (low + high) // 2 = 1`
- `names[middle]` is `"Debbie"`, which comes before `"Drew"` alphabetically.
- So, if `"Drew"` is present, it must be between positions `middle + 1` and `2`.

Binary Search: Searching for an Element not Present

Searching for `"Drew"` in the sequence `names`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
		low, middle, high				

- `middle = (low + high) // 2 = 2`
- `names[middle]` is `"Dustin"`, which comes after `"Drew"` alphabetically.
- So, if `"Drew"` is present, it must be between positions `2` and `middle - 1`.

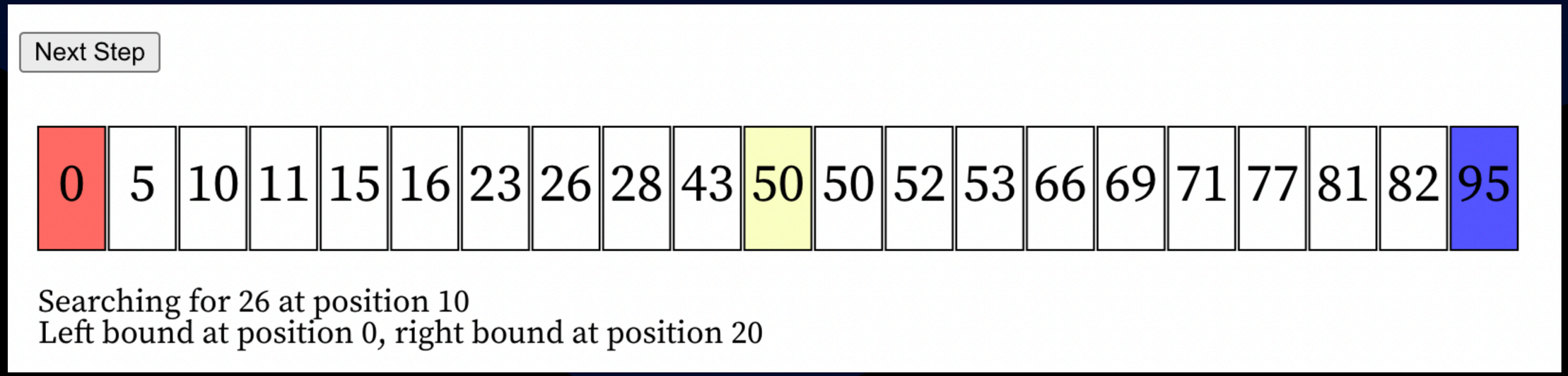
Binary Search: Searching for an Element not Present

Searching for `"Drew"` in the sequence `names`!

Caryn	Debbie	Dustin	Elliot	Jacquie	Jon	Rich
0	1	2	3	4	5	6
	high	low				

- `high` is now less than `low`. The "feasible search area" is now totally empty.
- So, we return `-1` to indicate that the target was not found in the sequence.

Binary Search, Interactive



(this image is a link)

Binary Search

```
def binary_search(sequence, target):
    low_index, high_index = 0, len(sequence) - 1
    while low_index <= high_index:
        middle_index = (low_index + high_index) // 2
        if target < sequence[middle_index]:      # middle too big:
            high_index = middle_index - 1        # discard right half
        elif target > sequence[middle_index]:    # middle too small:
            low_index = middle_index + 1         # discard left half
        else:
            return middle_index                  # found it! 😀
    return -1                                    # couldn't find it 😕
```

Properties of Binary Search

- Binary Search is **complete** since each iteration of the `while` loop shrinks our feasible search area down to a point where we'll stop, or we return the index where we find the target.
- Binary Search is **correct** since we return the index of the target when we find it and we only return `-1` when the element could not have been present in the sequence.
  - This is only guaranteed if the sequence was sorted, though!
- Is Binary Search any more **efficient** than Linear Search?

CIS 1100

Comparing Linear
& Binary Search

Python
Spring 2025
University of Pennsylvania

Linear Search vs. Binary Search

- Binary search is faster "on average" than linear search 😄🎉🆒
  - Per iteration, binary search shrinks the feasible region by half the remaining elements, linear search only by one element.
  - In both cases, max number of iterations needed is bounded above by the number of iterations needed to shrink the feasible region to empty.
  - On average, binary search requires fewer iterations of the search loop
  - (when is binary search not faster then linear search?)

Linear Search vs. Binary Search

Runtime analysis: how many iterations will it take to determine that the target is not in the sequence?

Length of the sequence	Linear Search	Binary Search
2	2	2
4	4	3
8	8	4
16	16	5
100	100	7

Linear Search vs. Binary Search

Runtime analysis: how many iterations will it take to determine that the target is the first element of the sequence?

Length of the sequence	Linear Search	Binary Search
2	1	2
4	1	3
8	1	4
16	1	5
100	1	7

Linear Search & Binary Search

Linear search is...

- Usable when your sequence is not sorted to start with
- As efficient as any search algorithm can be when you don't know anything about the sequence ahead of time

Binary search is...

- Only usable when your sequence is sorted to start with
- Significantly more efficient than linear search *on average.*