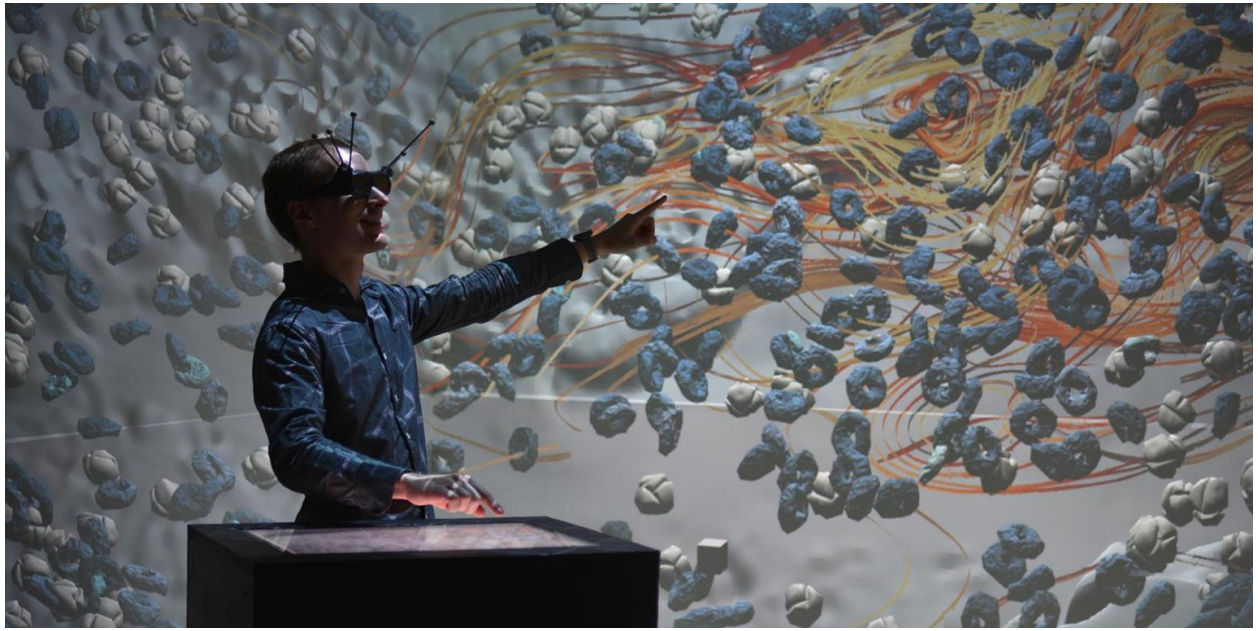
Linear & Binary Search

Tuesday, 2024-05-14

Introductions



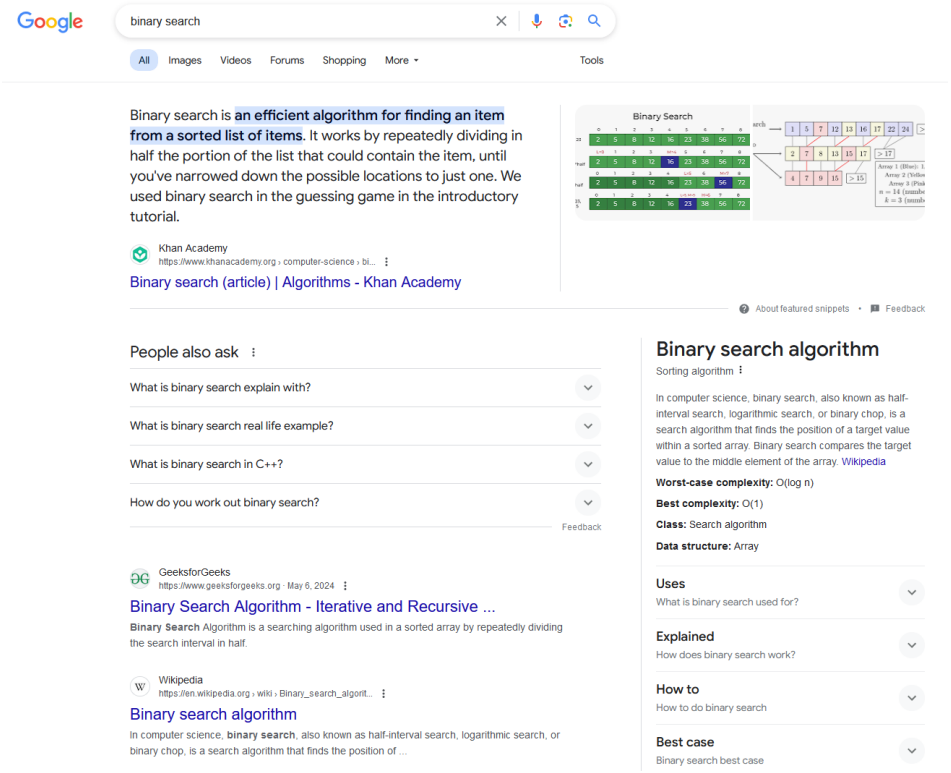
Bridger Herman

Roles, Assumptions, and Expectations

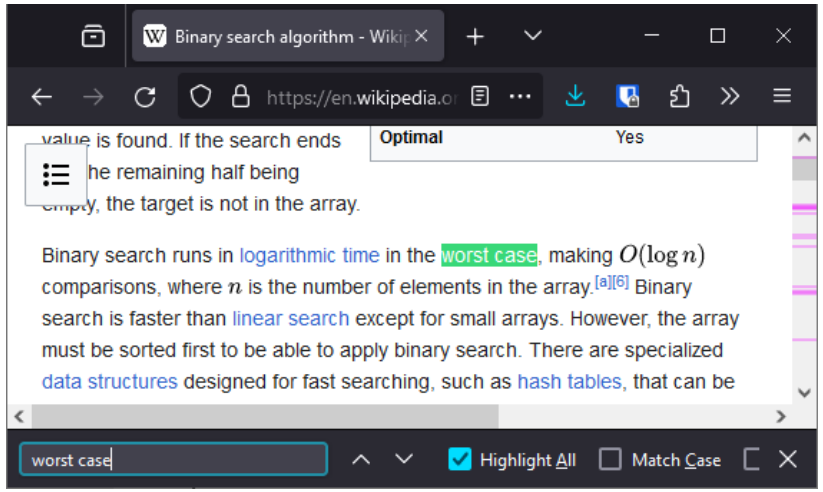
- Roles:
  - Me: Guest Lecturer for CS 111
  - You: Students in CS 111
- Assumptions:
  - We are near the end of the course
  - You’ve had a chance to solve problems with
    - Branching, Loops, Functions
    - Strings, Lists, Recursion
  - You have not explicitly worked with:
    - Any search algorithms
- Expectations:
  - There are no “wrong” answers (or questions!)
  - Be Present*

What are search algorithms?

Find something in a large collection of “stuff”



Find text in an article



Finding shortest path,
solving puzzles,
keyboard autocomplete, …

Roadmap

- Linear Search (quickly)
- Binary Search (the rest)
- Exercises & Reflections (async)

Search

How would *you* look for the item 21 in this list?

Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

Linear Search

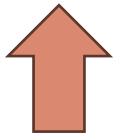
- Check one item at a time, see if it’s the same
- Repeat until we find the item

Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

Linear Search

- Check one item at a time, see if it's the same
- Repeat until we find the item

Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29



Linear Search

- Check one item at a time, see if it's the same
- Repeat until we find the item

Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

Linear Search

- Check one item at a time, see if it's the same
- Repeat until we find the item

Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

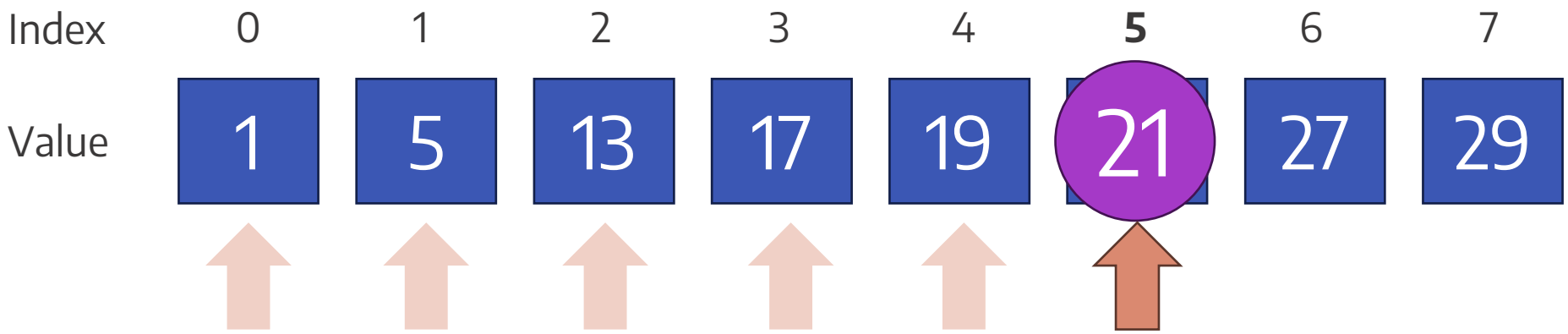
Linear Search

- Check one item at a time, see if it's the same
- Repeat until we find the item

Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

Linear Search

- Check one item at a time, see if it's the same
- Repeat until we find the item



Linear Search: Code

(you are welcome to follow along and write code, or not – the examples will be posted)

Linear Search: Pitfalls

- What if our list of values is REALLY long?

Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

Linear Search: Pitfalls

- What if our list of values is REALLY long?

Index	0		7,000,000
Value	1	...	29

Worst Case: iterations = number of elements in list.

Binary Search

- Repeatedly divide list in half
    - Is target LESS or GREATER than the middle element
- Prereq: List MUST be sorted.

Binary Search

- Repeatedly divide list in half
  - Is target LESS or GREATER than the middle element
- Prereq: List MUST be sorted. ✅

21

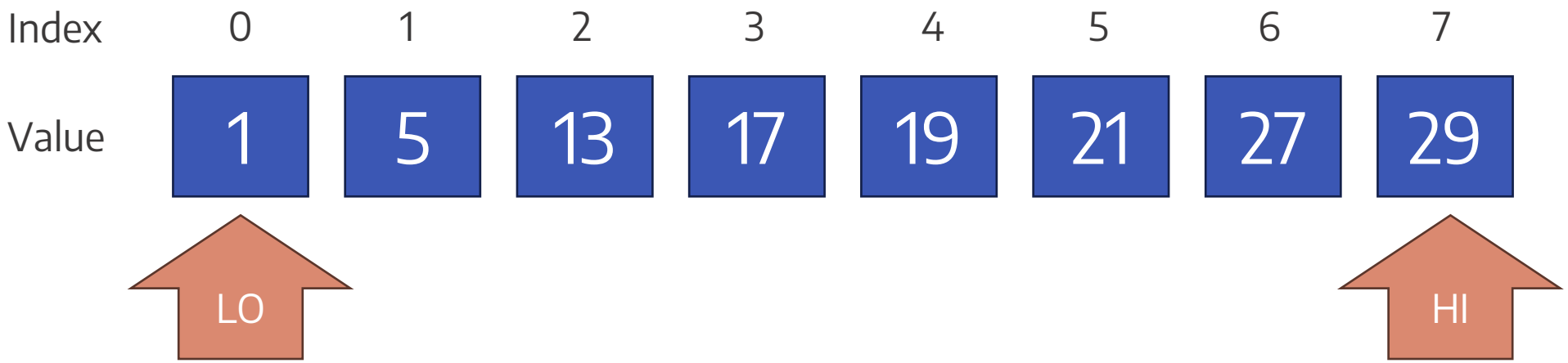
Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

Binary Search

- Repeatedly divide list in half
  - Is target LESS or GREATER than the middle element

21

- Prereq: List MUST be sorted. ✅

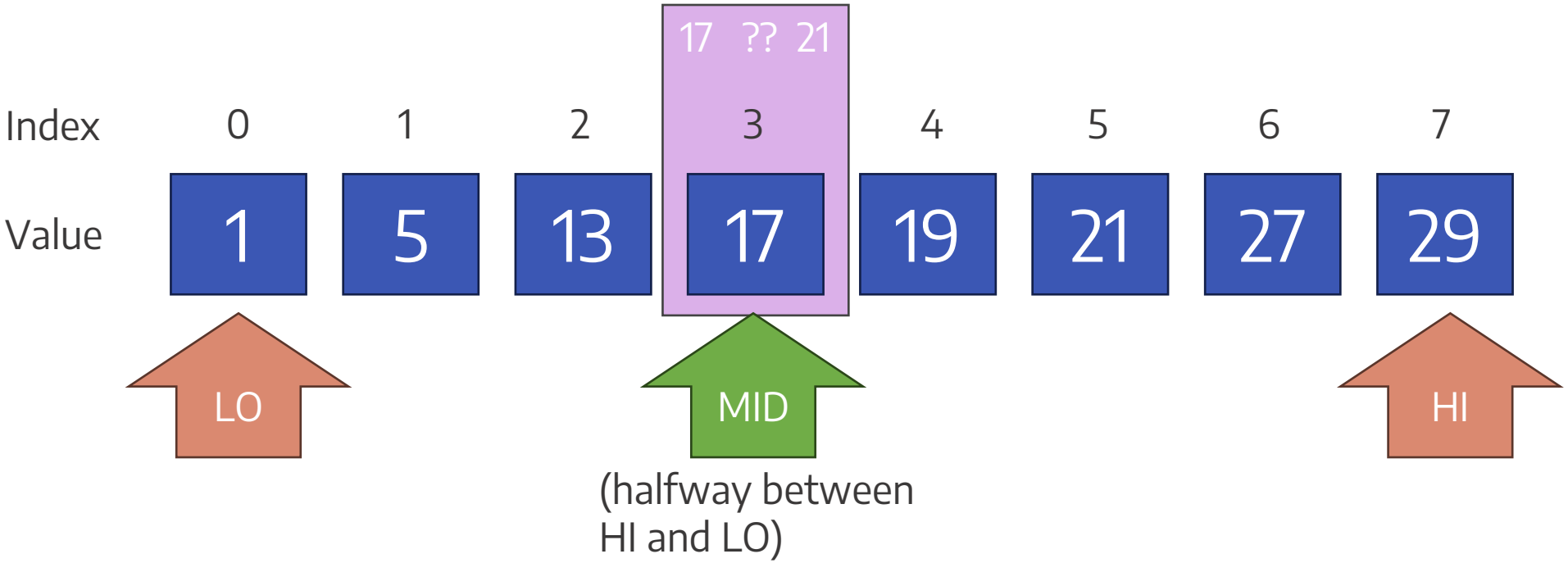


Binary Search

- Repeatedly divide list in half
  - Is target LESS or GREATER than the middle element

21

- Prereq: List MUST be sorted. ✅

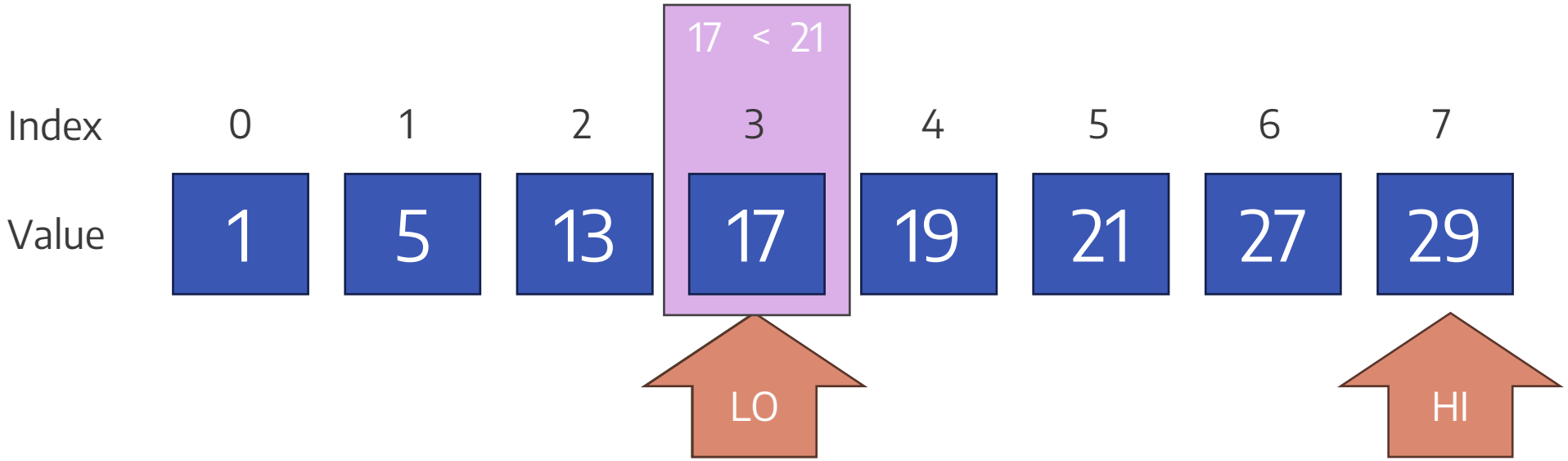


Binary Search

- Repeatedly divide list in half
  - Is target LESS or GREATER than the middle element

21

- Prereq: List MUST be sorted. ✅



Binary Search

- Repeatedly divide list in half
  - Is target LESS or GREATER than the middle element
- Prereq: List MUST be sorted. ✅

21

Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

LO

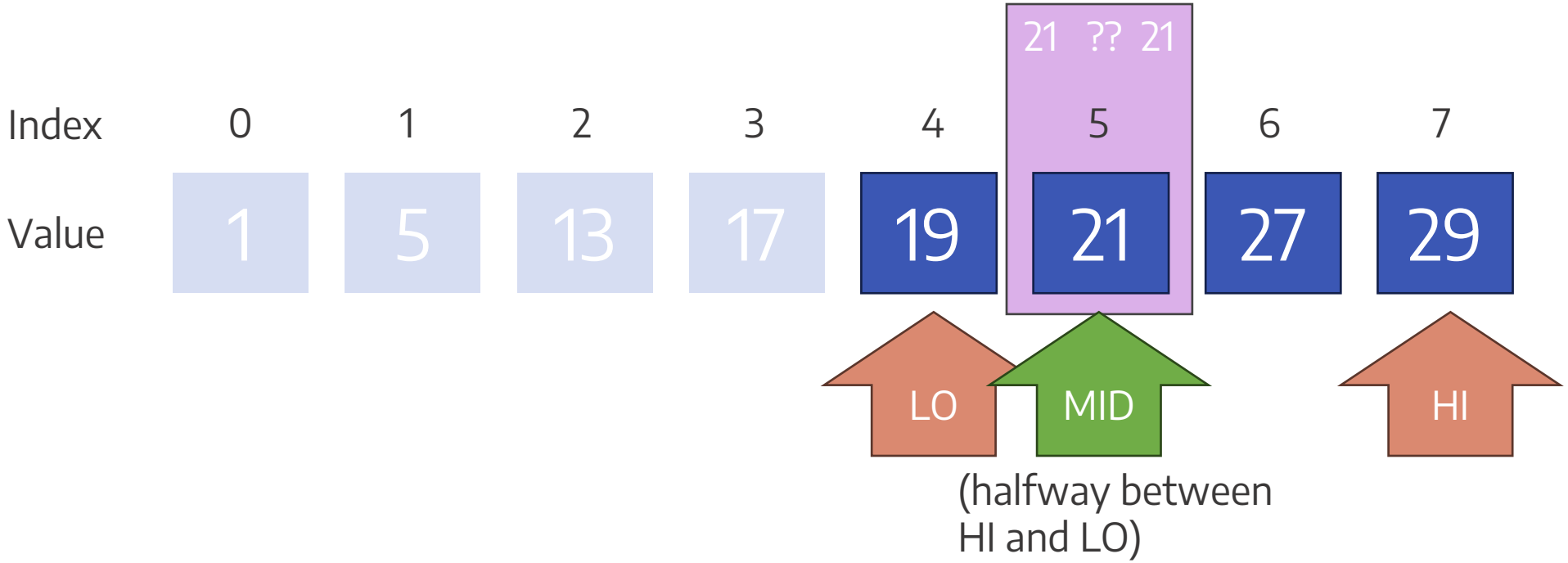
HI

Binary Search

- Repeatedly divide list in half
  - Is target LESS or GREATER than the middle element

21

- Prereq: List MUST be sorted. ✅

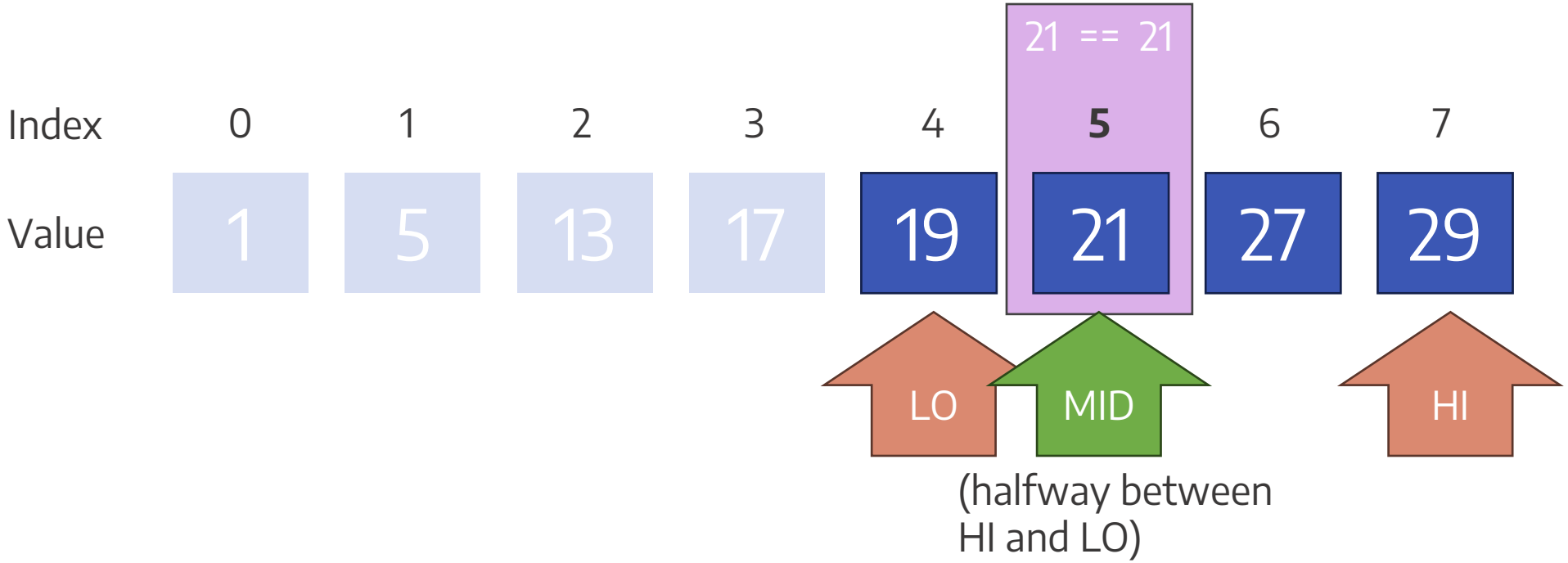


Binary Search

- Repeatedly divide list in half
  - Is target LESS or GREATER than the middle element

21

- Prereq: List MUST be sorted. ✅



Binary Search: Algorithm

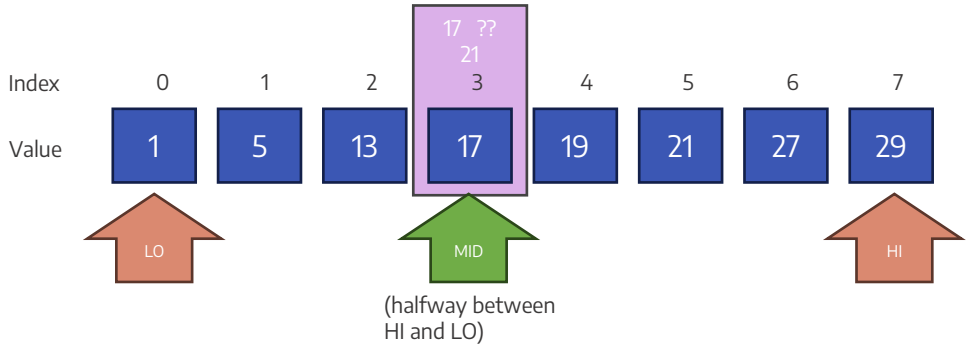
target 21

while LO <= HI

check target with MID (element halfway between LO and HI)

- If equal, return MID index
- If target > MID element, increase LO to MID + 1
- Else, decrease HI to MID - 1

If not found, return -1



Binary Search: Code

(you are welcome to follow along and write code, or not – the examples will be posted)

Binary Search: Iterative Approach

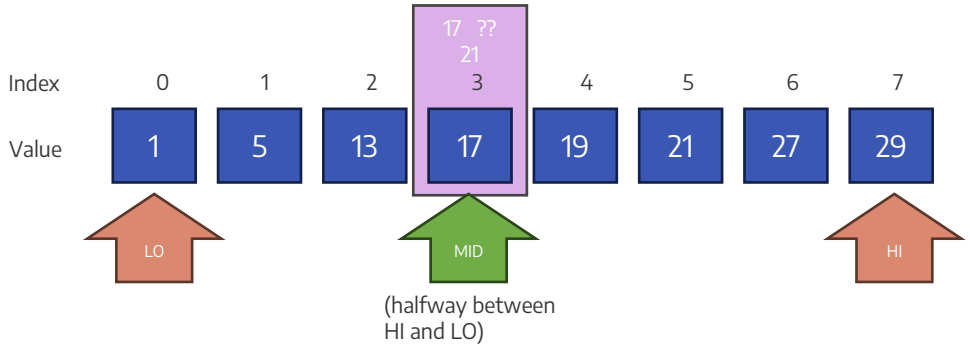
target 21

while LO <= HI

check target with MID (element halfway between LO and HI)

- If equal, return MID index
- If target > MID element, increase LO to MID + 1
- Else, decrease HI to MID - 1

If not found, return -1



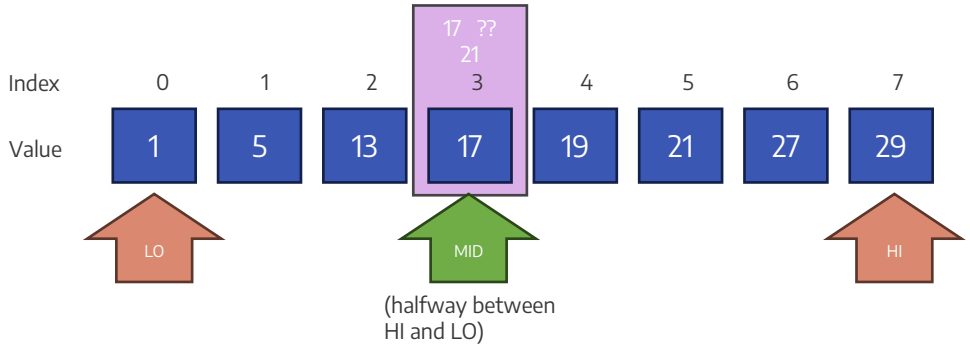
Binary Search: Recursive Approach

target 21

Base Case 1: if LO > HI : return -1

check target with MID (element halfway between LO and HI)

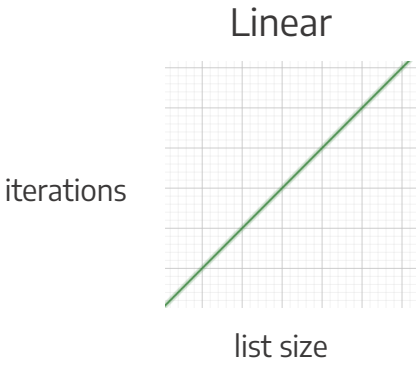
- **Base Case 2:** If equal, return MID index
- **Recursive Step 1:** If target > MID element, increase LO to MID + 1
- **Recursive Step 2:** Else, decrease HI to MID - 1

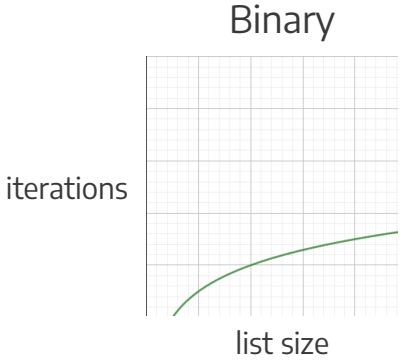


Worst Case Performance: Linear vs. Binary

(assuming **n** elements in list)

- Linear: **n** iterations
- Binary: **log(n)** iterations





Example (10,000 element list):

```
linear search: max 10000 iterations                1.5991158 sec
binary search: max 14 iterations                   0.0187128 sec
binary search (recursive): max 14 iterations       0.0263507 sec*
```

*exact timing may change based on what language you use (i.e., if it supports tail recursion)

Also works with strings!

(and anything else that can be assigned a numeric “key”)

Wrapping Up

- Code examples & slides: posted on GitHub
  - https://github.com/bridger-herman/search-lecture
- Exercises & Reflections:

https://forms.gle/vNvfVevihq3AvchS6