

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”

A screenshot of a Google search for "binary search". The search bar at the top shows "binary search" with a search icon. Below the search bar, there are tabs for "All", "Images", "Videos", "Forums", "Shopping", and "More". The main search results area shows a snippet from Khan Academy: "Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one. We used binary search in the guessing game in the introductory tutorial." To the right of this snippet is a diagram illustrating the binary search process on a sorted array. Below the snippet, there are links to "Khan Academy" and "Binary search (article) | Algorithms - Khan Academy". On the right side of the search results, there is a section titled "Binary search algorithm" with a sub-header "Sorting algorithm". It contains a brief description: "In computer science, binary search, also known as half-interval search, logarithmic search, or binary chop, is a search algorithm that finds the position of a target value within a sorted array. Binary search compares the target value to the middle element of the array. Wikipedia". Below this, there are sections for "Worst-case complexity: $O(\log n)$ ", "Best complexity: $O(1)$ ", "Class: Search algorithm", and "Data structure: Array". At the bottom, there is a section titled "Uses" with the question "What is binary search used for?".

Find text in an article

A screenshot of a web browser displaying a Wikipedia article on the "Binary search algorithm". The browser's address bar shows the URL "https://en.wikipedia.org/wiki/Binary_search_algorithm". A search bar overlay is visible on the left side of the article, with the text "value is found. If the search ends" and "Optimal" next to it. The article text is partially visible, showing the sentence: "Binary search runs in logarithmic time in the worst case, making $O(\log n)$ comparisons, where n is the number of elements in the array.^{[a][6]} Binary search is faster than linear search except for small arrays. However, the array must be sorted first to be able to apply binary search. There are specialized data structures designed for fast searching, such as hash tables, that can be". At the bottom of the search bar overlay, there is a text input field containing the words "worst case" and a "Highlight All" button.

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  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

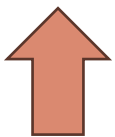
Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

Linear Search

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

21

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

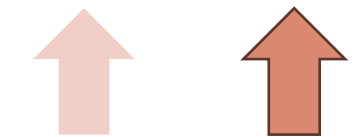


Linear Search

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

21

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

21







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

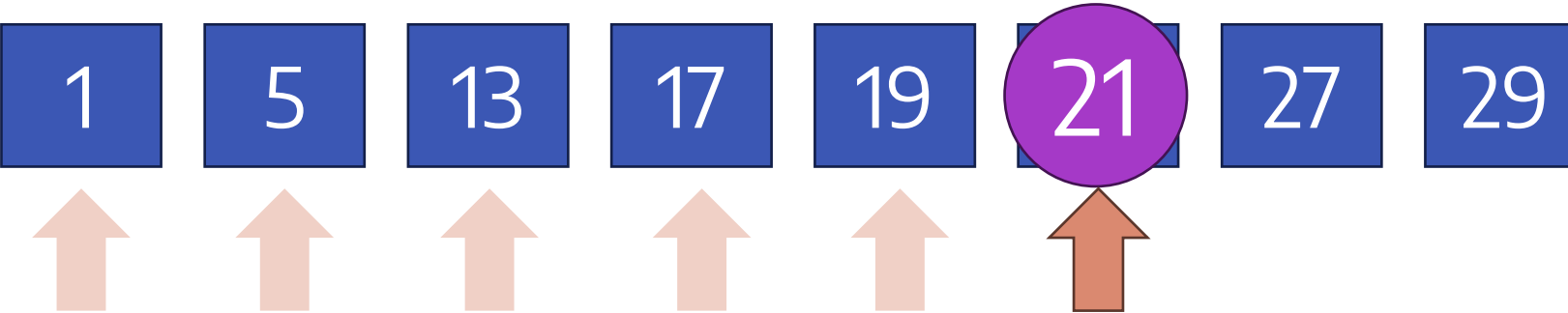
21

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: Code

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

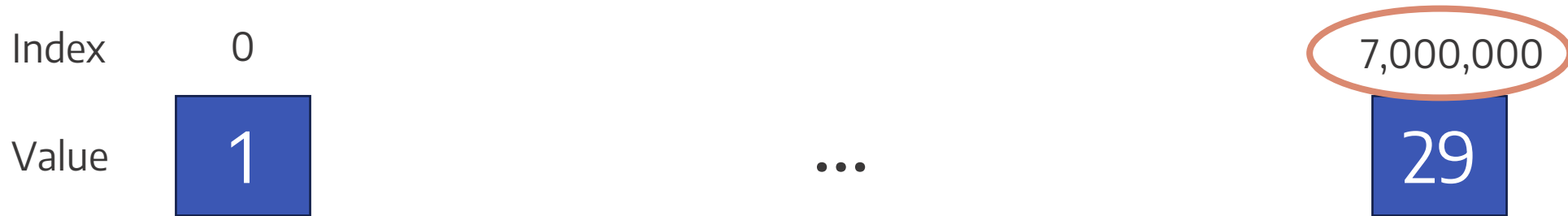
Linear Search: Pitfalls

- 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?




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

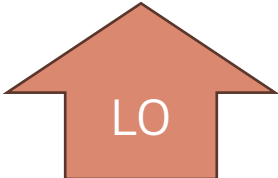
Index	0	1	2	3	4	5	6	7
Value	1	5	13	17	19	21	27	29

Binary Search

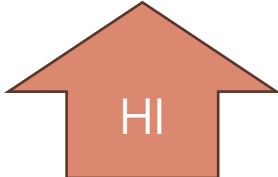
- Repeatedly divide list in half
 - Is target LESS or GREATER than the middle element
- Prereq: List MUST be sorted. 

21

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




LO

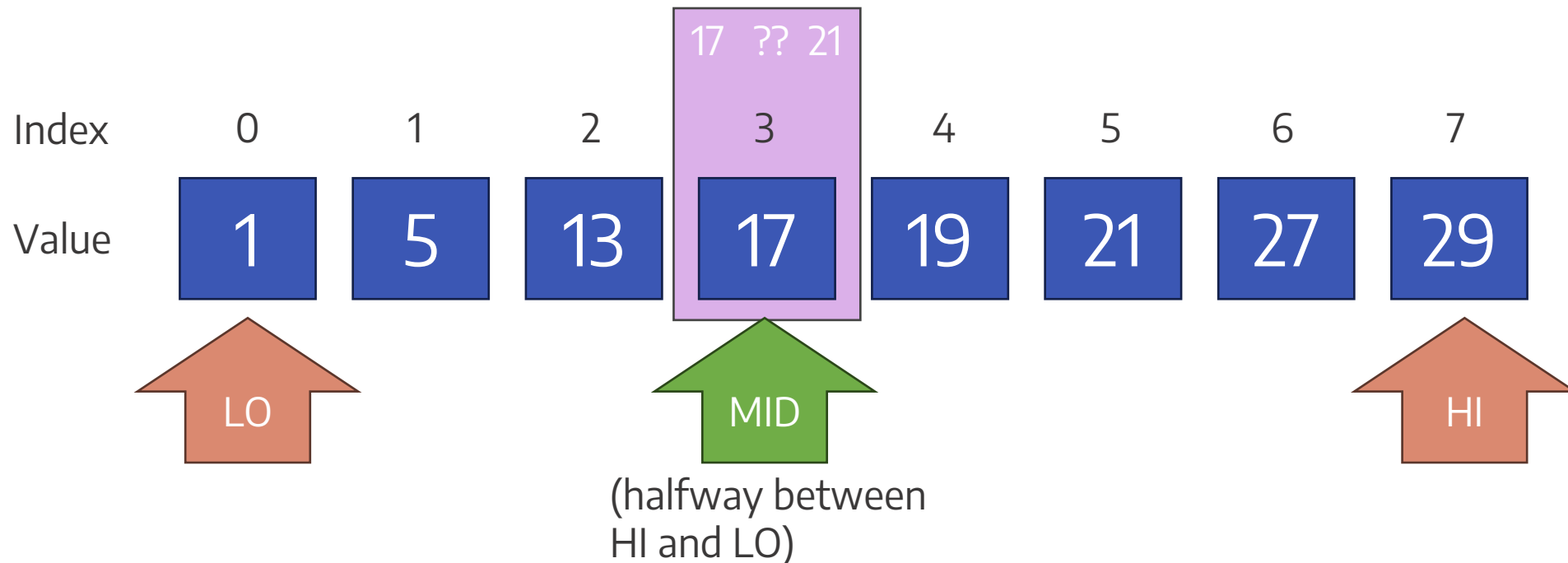


HI


Binary Search

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

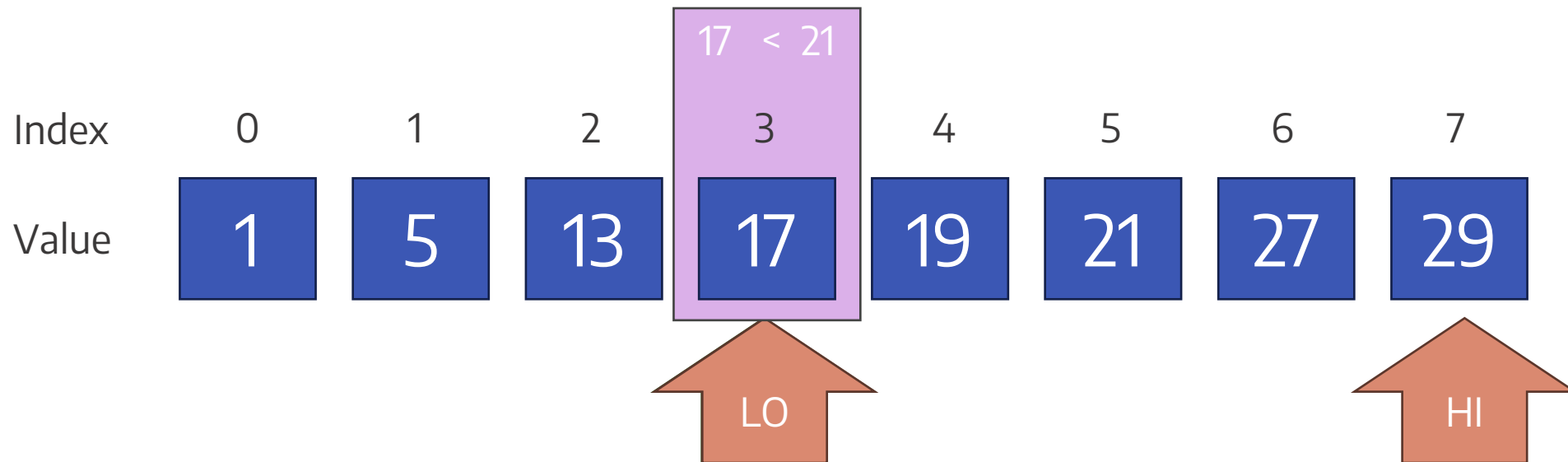
21




Binary Search

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

21

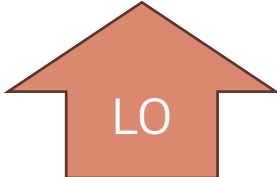


Binary Search

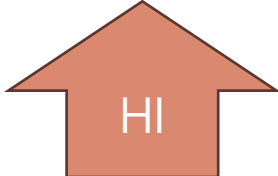
- Repeatedly divide list in half
 - Is target LESS or GREATER than the middle element
- Prereq: List MUST be sorted. 

21

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




LO

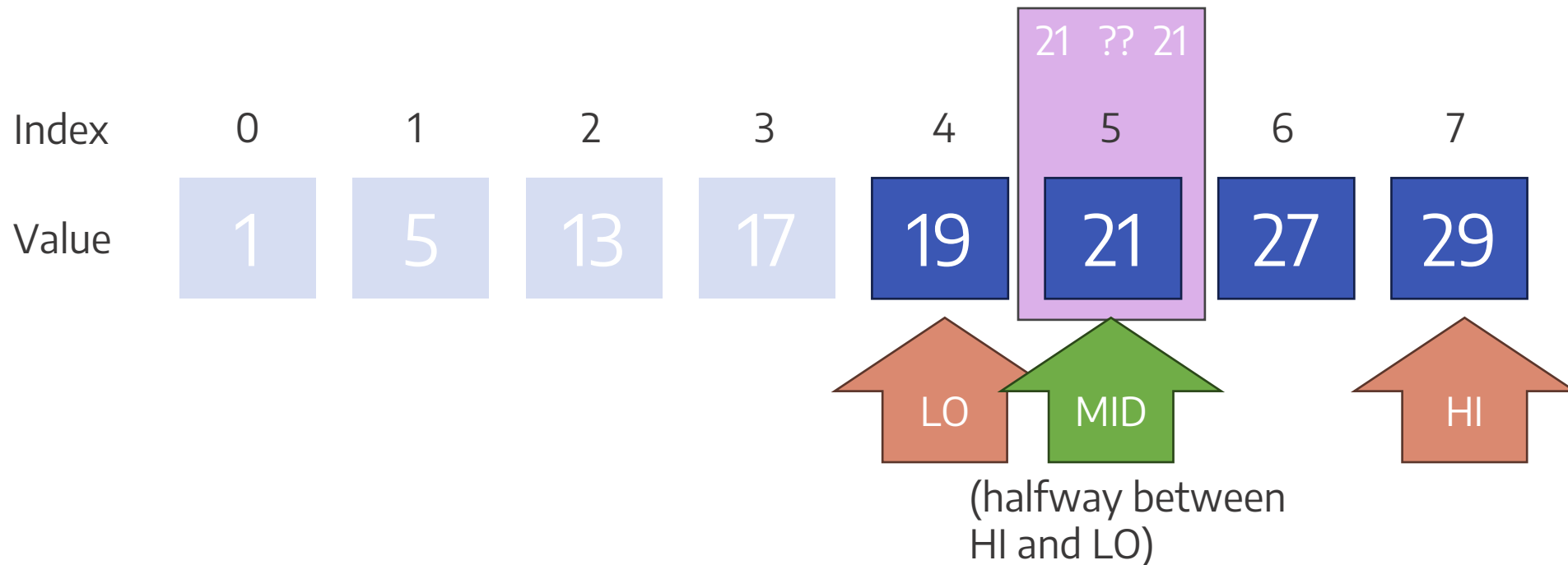


HI


Binary Search

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

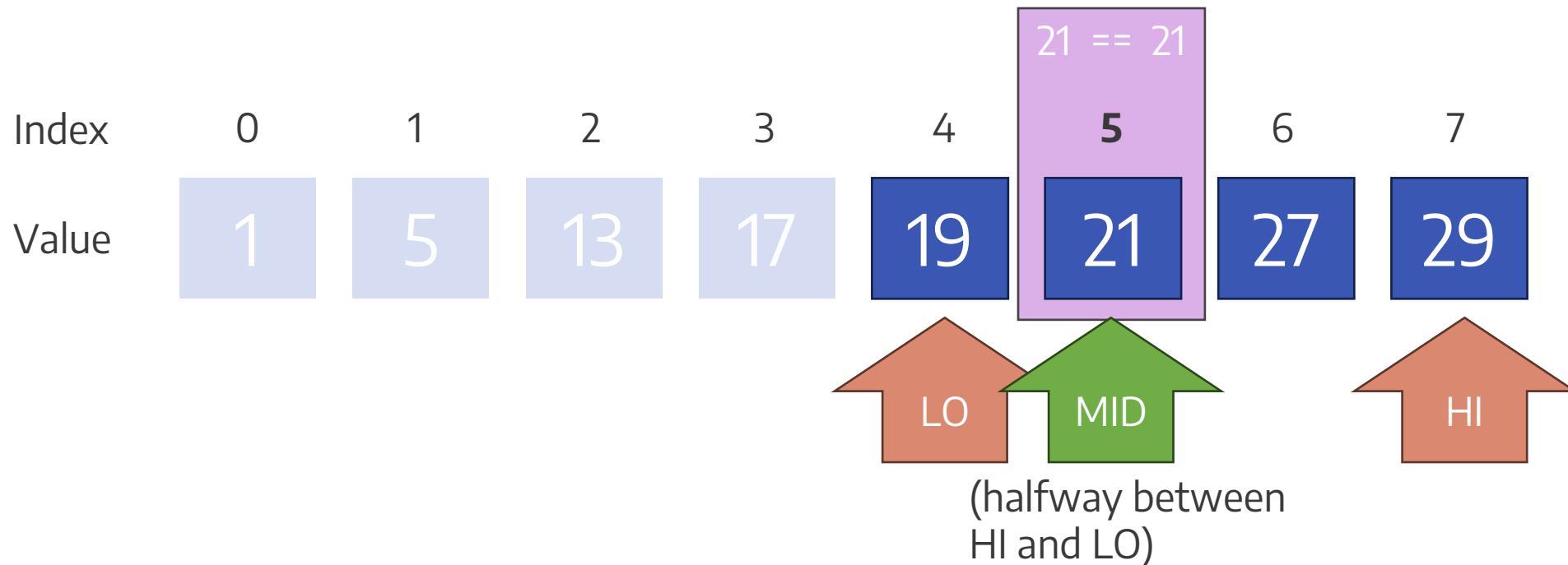
21



Binary Search

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

21



Binary Search: Algorithm

target 21

while LO <= HI

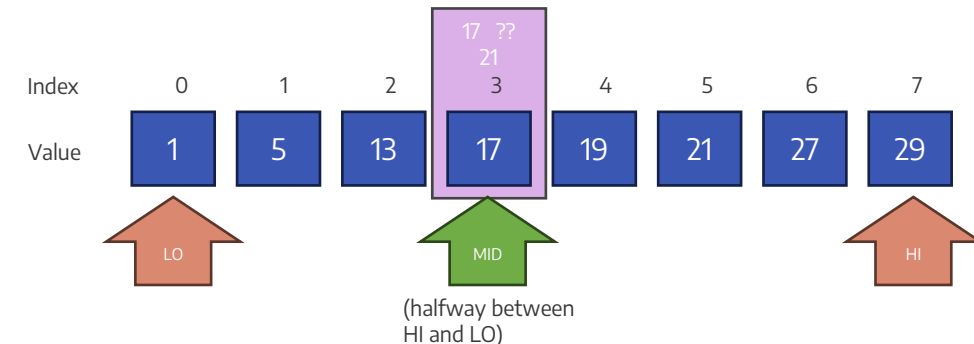
check target with



(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

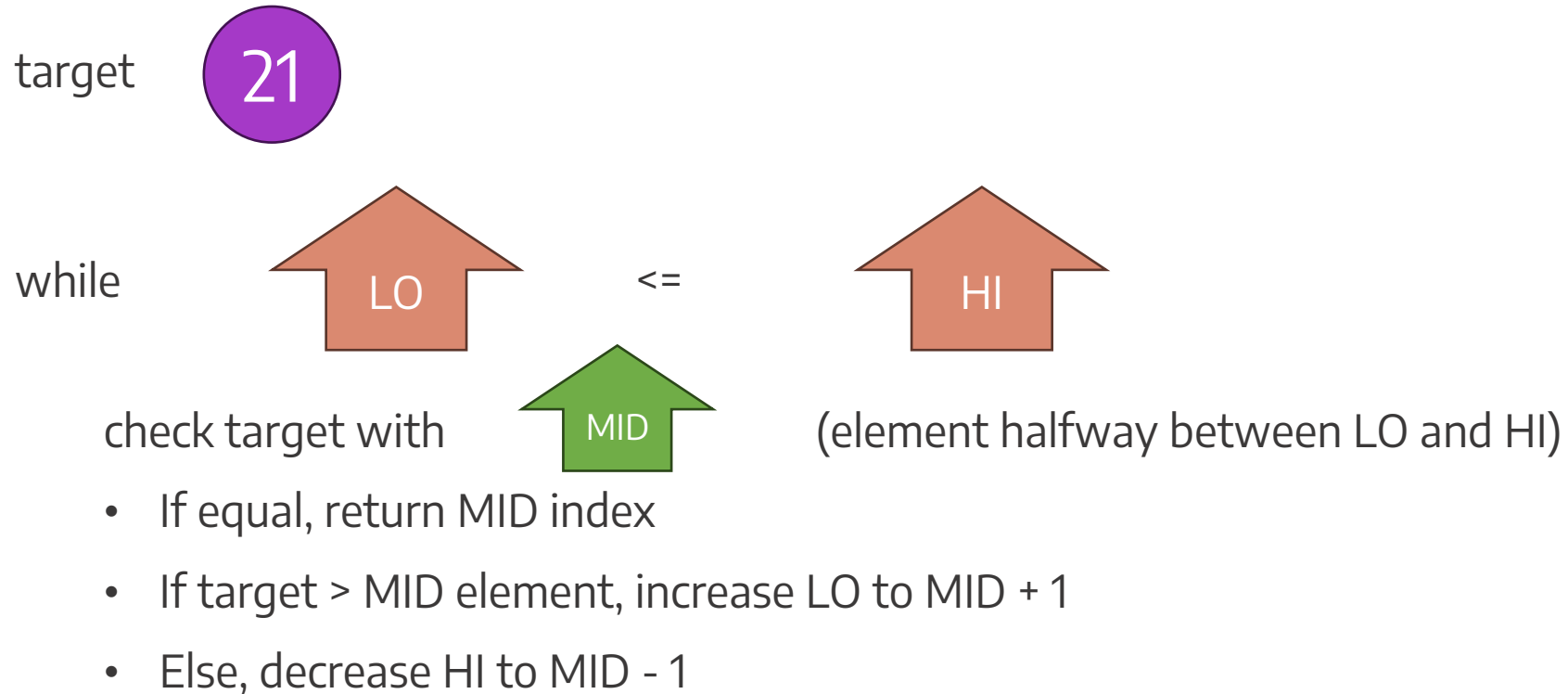
If not found, return -1



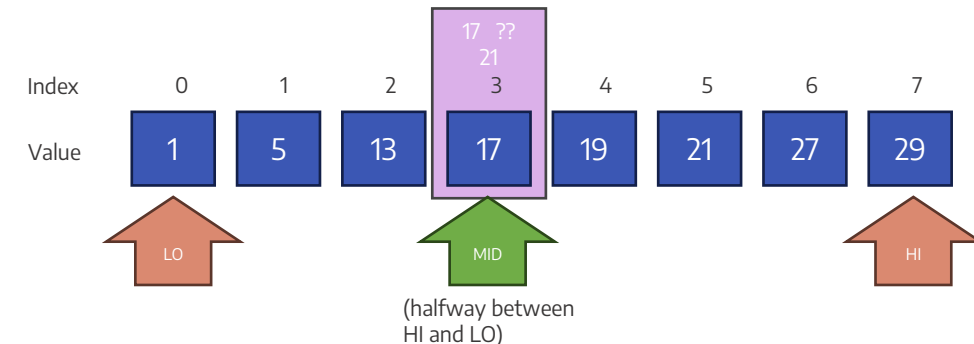
Binary Search: Code

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

Binary Search: Iterative Approach



If not found, return -1



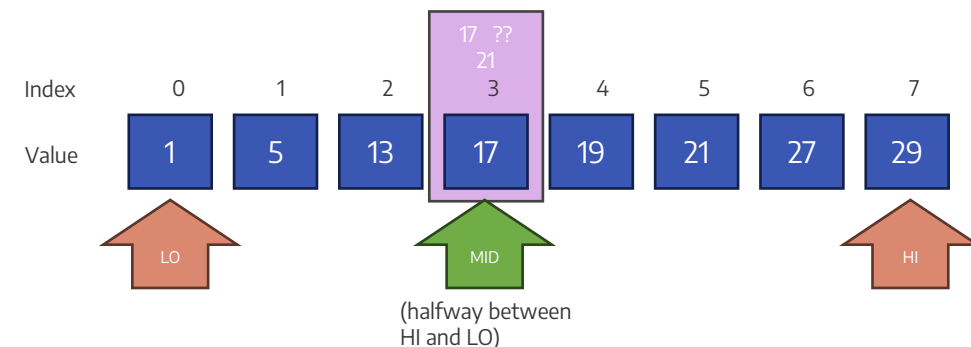
Binary Search: Recursive Approach

target 21

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

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

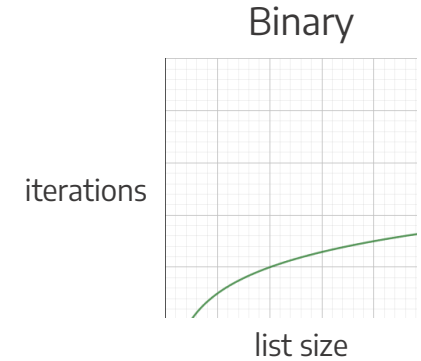
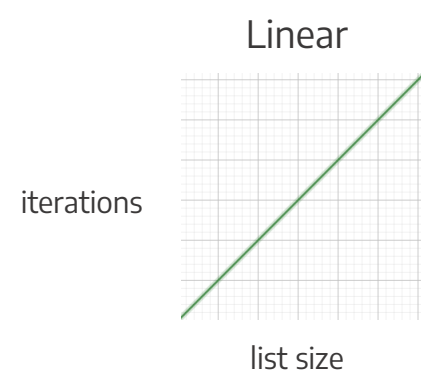
- **Base Case 2:** If equal, return MID index
- **Recursive Step 1:** If target > MID element, increase LO to MID + 1
- **Recursive Step 2:** Else, decrease HI to MID - 1



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>