ECS 32B - Searching

Aaron Kaloti

UC Davis - Summer Session #2 2020

UC DAVIS
COMPUTER SCIENCE

This content is protected and may not be shared, uploaded, or distributed.
CC BY-NC-SA 4.0 International Licence

This will be the hardest lecture in this course

- Ask all the questions you have.

This will be the hardest lecture in this course

- Ask all the questions you have.
- Just kidding.[1]

1. Still though, you should ask all the questions you have.

Overview

- Algorithms for **searching** for a target element in a list of elements.
  - Linear/sequential search.
  - Binary search.
    - Only works on ordered/sorted list.
- We will assume all elements are unique.
- Can either determine membership (return `True`/`False`) or location (return index).

Linear Search

Overview

- a.k.a. sequential search.
- You've seen something like this earlier:

```
def lin_search(lst, target):
    for i in range(len(lst)):
        if lst[i] == target:
            return True
    return False
```

- Worst-case time complexity: $\Theta(n)$.
  - Also: $O(n)$, $O(n^2)$, $O(n!)$, etc.
- Worst-case space complexity: $\Theta(1)$.

Linear Search

Recall: `in` Operator (on a List)

- With lists, although no loop can be seen, the `in` operator takes linear time in worst case.
  - Worst case: target element is not in list.

```
from timeit import Timer
nums = None
def check_in(vals):
    return -1 in vals

def time_list_in():
    import_line = "from __main__ import check_in, nums"
    global nums
    for n in range(10000, 80001, 10000):
        nums = list(range(n))
        timer = Timer("check_in(nums)", import_line)
        print("n={}: {} milliseconds".format(n, round(timer.timeit(number=1000), 4)))
```

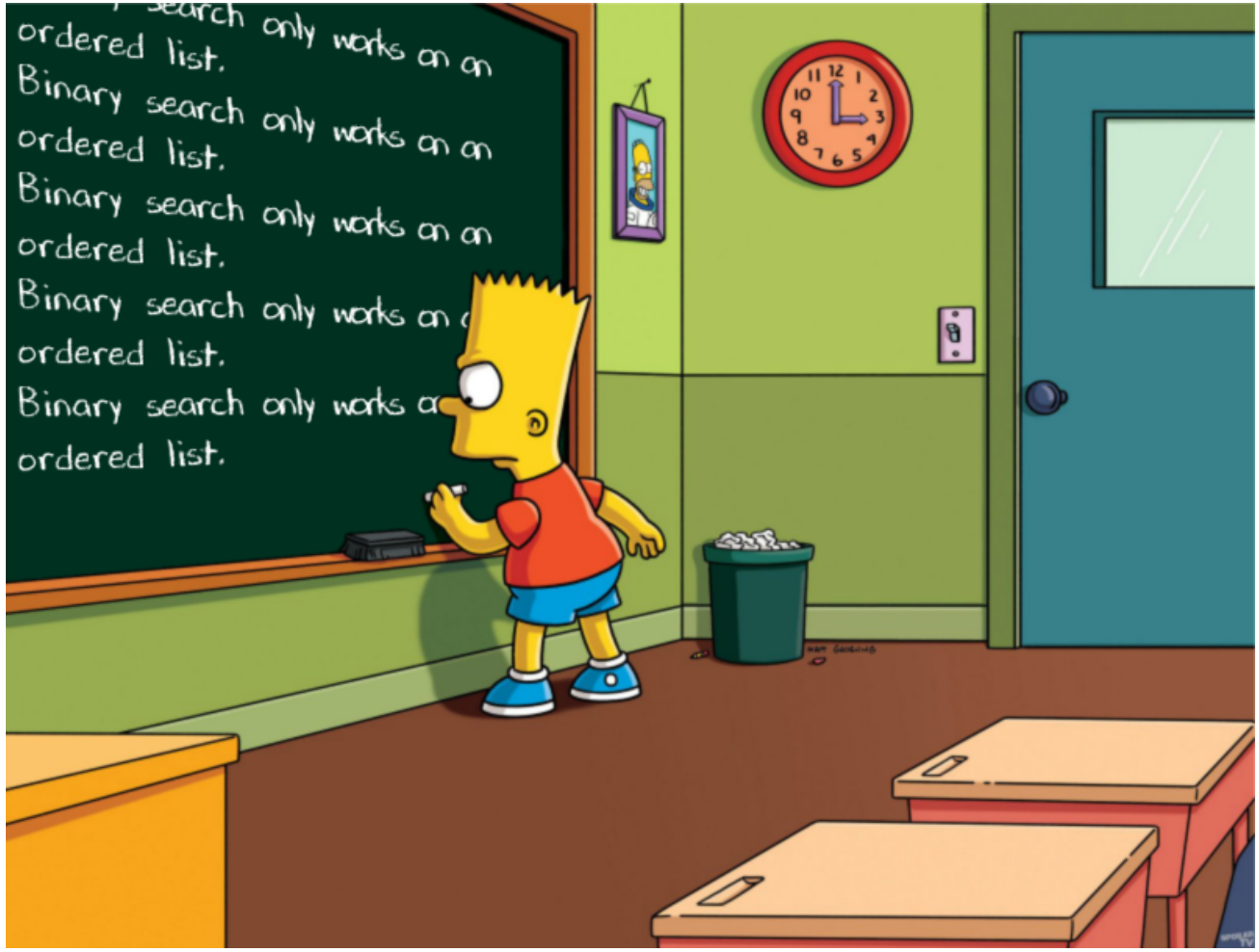
```
n=10000: 0.0584 milliseconds
n=20000: 0.1189 milliseconds
n=30000: 0.1838 milliseconds
n=40000: 0.2474 milliseconds
n=50000: 0.3084 milliseconds
n=60000: 0.369 milliseconds
n=70000: 0.4282 milliseconds
n=80000: 0.4928 milliseconds
```

- Uses linear search (hence linear time).

Binary Search

Prerequisite

- ***Binary search only works on an ordered list.***
- ***Binary search only works on an ordered list.***
- ***Binary search only works on an ordered list.***



Binary Search

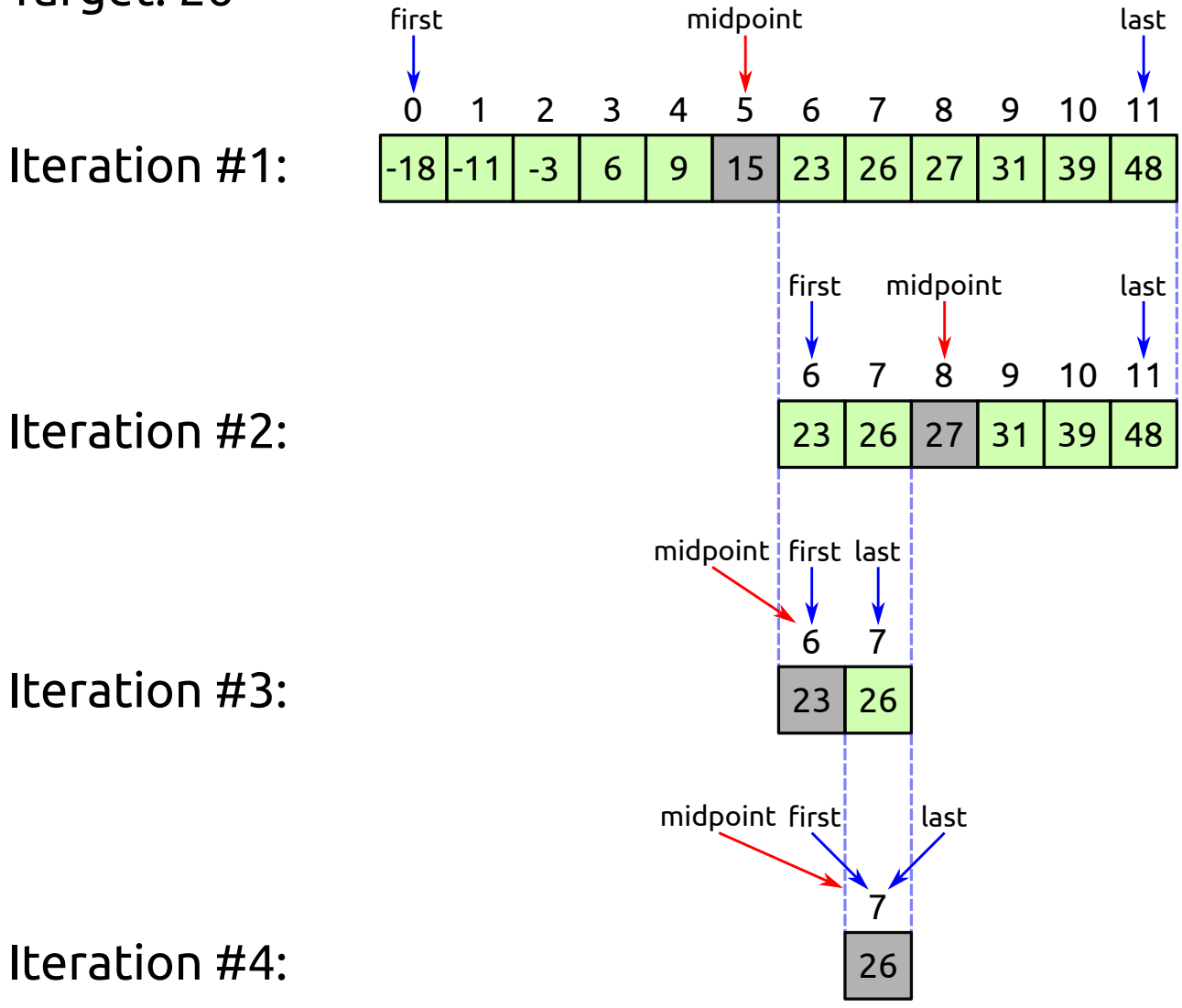
Algorithm: `bin_search(lst, target)`

- Check if `target` equals middle element of `lst`.
- If so, return `True`.
- If not:
  - If `target` < middle, repeat process on lower half of elements.
  - If `target` > middle, repeat process on upper half of elements.
- Keep going until find `target` or are zero remaining elements to look at.

Binary Search

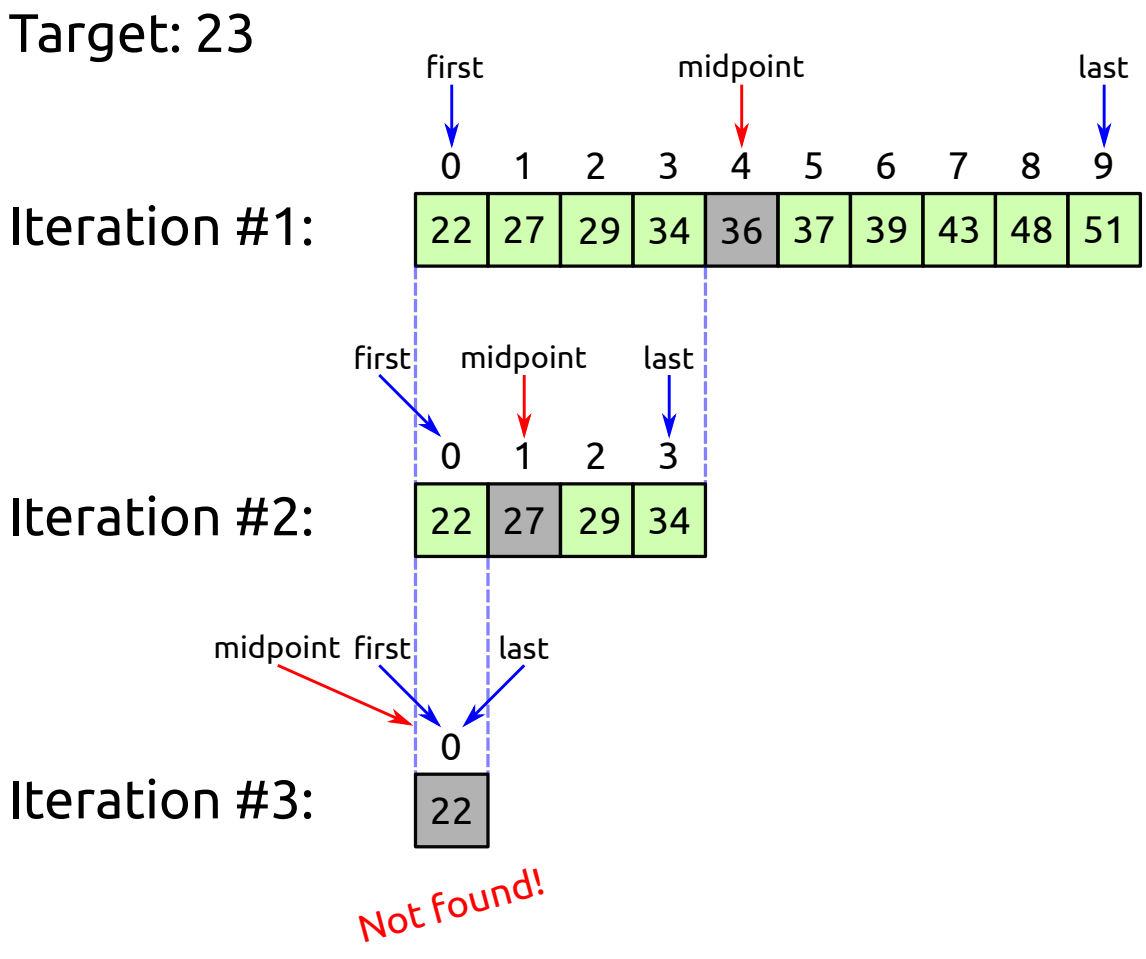
Example #1

Target: 26



Binary Search

Example #2



1. In this case, regarding the code on the next slide, `first` will be incremented to 1, and the condition `first<=last` will become `False`.

Binary Search

Implementation[1]

```
def binarySearch(alist, item):
    first = 0
    last = len(alist)-1
    found = False

    while first<=last and not found:
        midpoint = (first + last)//2
        if alist[midpoint] == item:
            found = True
        else:
            if item < alist[midpoint]:
                last = midpoint-1
            else:
                first = midpoint+1

    return found
```

Worst-Case Analysis

- Target not found.
- Time complexity: $\Theta(\lg n)$.
  - Also: $O(\lg n)$, $O(n)$, $O(n^2)$, $O(n!)$, etc.
- Space complexity: $\Theta(1)$.

Explanation

- Number of iterations scales logarithmically as n increases.

n	# Iterations (Worst-Case)
3	2
4	3
5	3
6	3
7	3
8	4
9	4
10	4
...	
15	4
16	5
17	5

1. From section 6.4 of *Problem Solving with Algorithms and Data Structures using Python* by Brad Miller and David Ranum.

Binary Search

Other Remarks

- Binary search seems to be a common interview problem, relatively speaking.
- Implementing it *perfectly* can be unexpectedly difficult.
- In programming languages where arithmetic overflow is possible (e.g. C, C++), the line `midpoint = (first + last)//2` can cause trouble.

References / Further Reading

- Sections 6.2-6.4 of *Problem Solving with Algorithms and Data Structures using Python* by Brad Miller and David Ranum.