23.03.21 06:26

CSCI 120
Introduction to CompSci and Programming I
Lec 5: Algorithms III

Jetic Gū

Overview

- Focus: Python Programming
- Architecture: von Neumann
- Core Ideas:
    1. Binary Search
    2. Lab

Binary Search

Summary

What is Search?

- The process of looking for stuff
- In algorithm design
  - Finding specific item in a data structure, with specific properties

Concept

P1
Binary Search

Search Algorithm

10	99	32	7	12	1	56	33	64	78	9	5	3	27

- An array contains n unique elements. Design an algorithm to search for the second largest number in an array.

- We discussed two algorithms for solving this problem in LS6, both linear algorithms of time complexity $O(n)$

Review

Case 2: Search Problem

1	3	5	7	9	10	12	27	32	33	56	64	78	99

- A **sorted** array contains n unique elements. Design an algorithm to search for **a specific item** in the array. (e.g. is 13 in the array?)

Concept

P1
Binary Search

Case 2: Search Problem

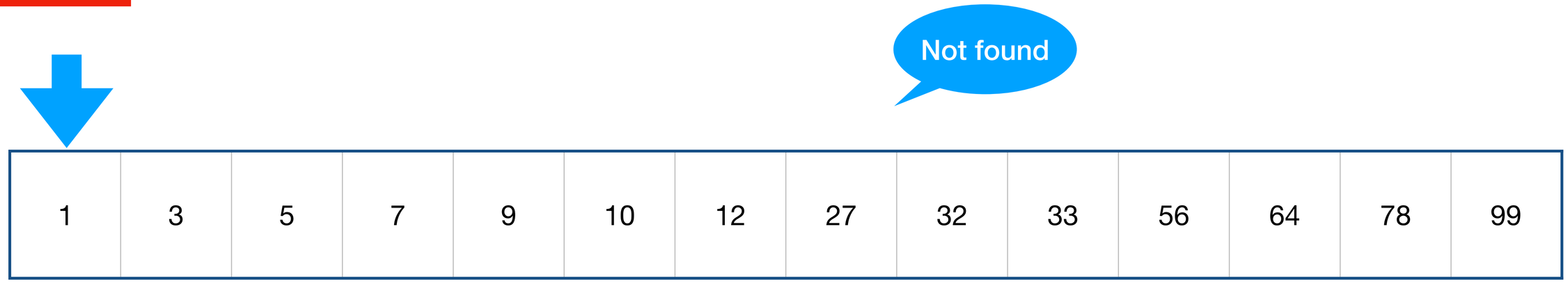

```
tgt = 13
for item in arr:
    if item == tgt:
        return FOUND
return NOT_FOUND
```

- Simple Solution: linear search, go through all elements inside the array
- What complexity is this algorithm?
- Is there a better way?

Concept

P1
Binary Search

Case 2: Search Problem




```
tgt = 13
for item in arr:
    if item == tgt:
        return FOUND
return NOT_FOUND
```

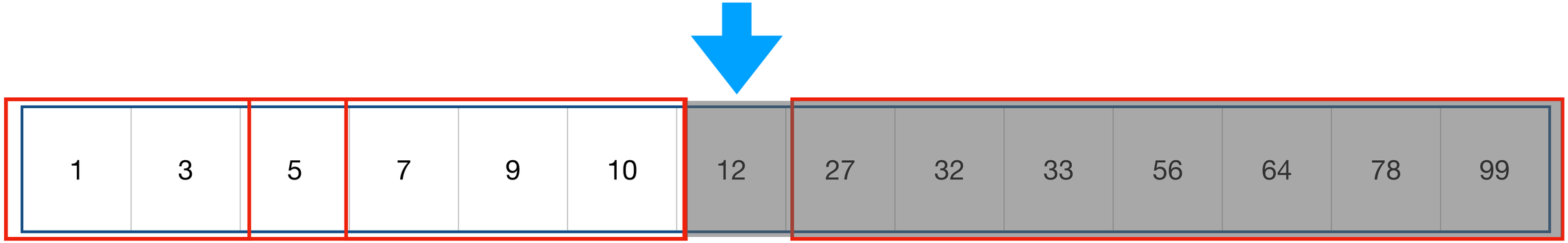
Is there an even better way?

nt 1:

- We know the array is sorted
- Once `item` is greater than `tgt`, we know it doesn't exist
- What is the time complexity?

Concept

Case 2: Search Problem

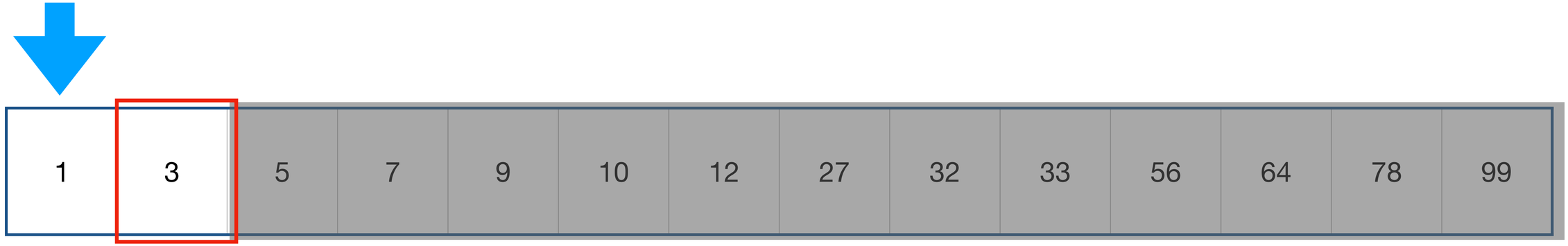


- Divide and Conquer
  - We split the array into two parts each time
  - By comparing with the number in the middle, we know which part the target must be
  - Say we are looking for 3, in this case the answer must be in the left partition
  - Now, we split again and take the middle, keep searching recursively

Concept

P1
Binary Search

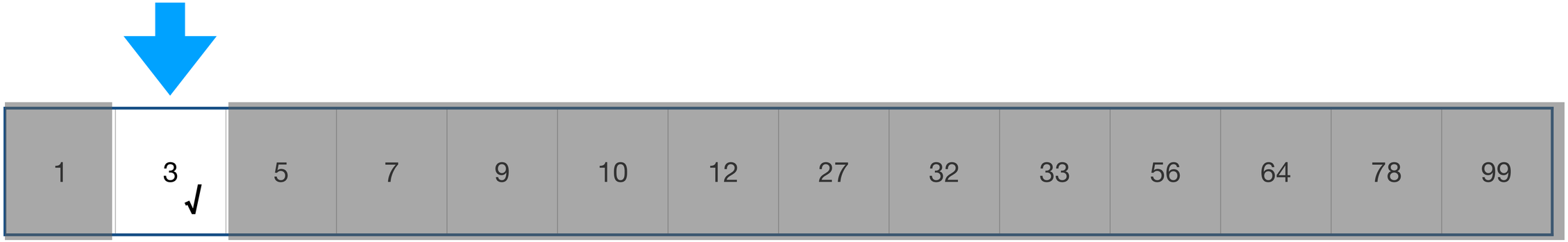
Case 2: Search Problem



- Divide and Conquer
  - We split the array into two parts each time
  - By comparing with the number in the middle, we know which part the target must be
  - Say we are looking for 3, in this case the answer must be in the left partition
  - Now, we split again and take the middle, keep searching recursively

Concept

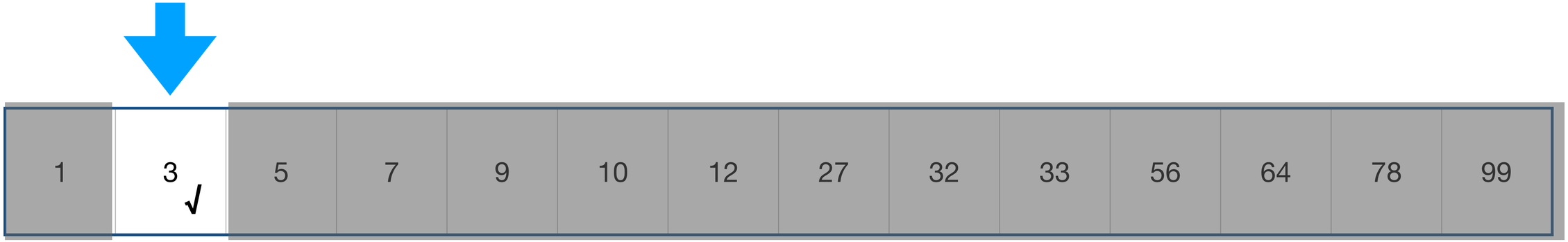
Case 2: Search Problem



- Divide and Conquer
  - We split the array into two parts each time
  - By comparing with the number in the middle, we know which part the target must be
  - Say we are looking for 3, in this case the answer must be in the left partition
  - Now, we split again and take the middle, keep searching recursively

Concept

Case 2: Search Problem



- This is called binary search, we reduce the search space by half at every step
  - Assuming the original sorted array has n elements, search space n
  - Step 1: reduce search space to $n/2$, ...
  - Step 2: reduce search space to $n/4$, ...
  - Total steps: $\log_2 n$, hence the algorithm is $O(\log n)$, significant improvement over $O(n)$

Concept

Case 2: Search Problem

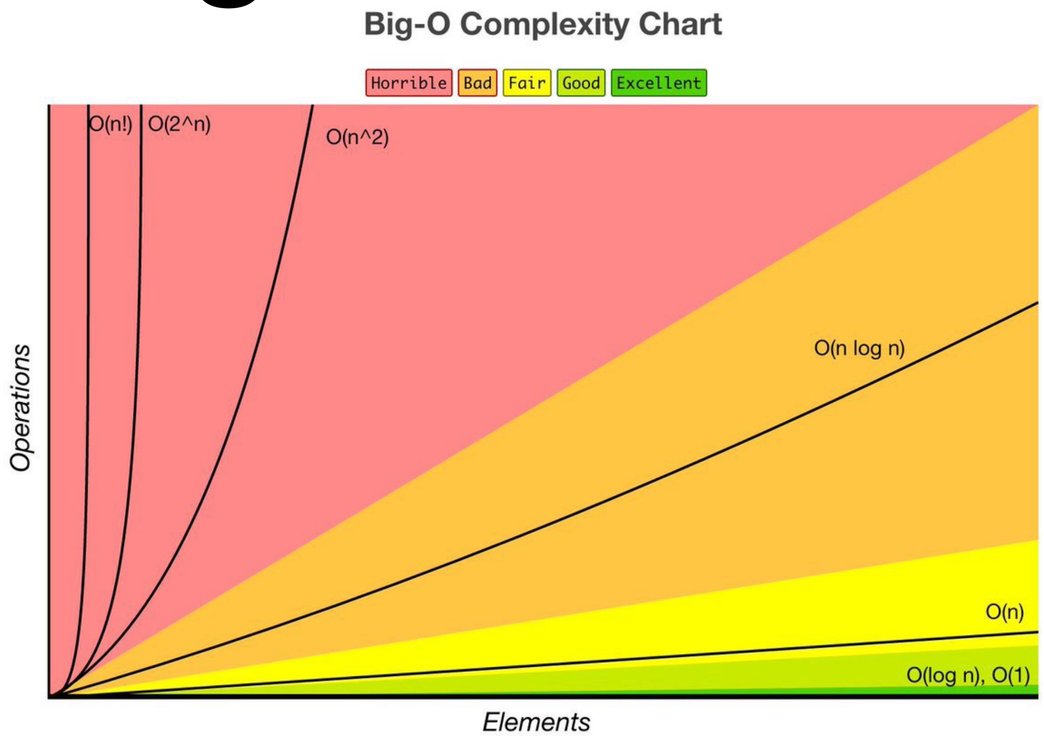
```
Func BS(arr, tgt):
    if (len(arr) == 0)
        return NOT_FOUND
    mid = len(arr)/2
    if (arr[mid] == tgt) return FOUND
    if (arr[mid] > tgt)
        return BS(arr[:mid], tgt)
    if (arr[mid] < tgt)
        return BS(arr[mid+1:], tgt)
```

- This is called pseudocode, not real computer programme but easy to understand for human
- Termination: not found
- Termination: found
- Search left
- Search right

Concept

P1
Binary Search

Why is log better than linear?



n	5	10	100	1000	10000	100000
n/2 = O(n)	2.5	5	50	500	5000	50000
50 log n = O(log n)	116	166	332	498	664	830

Concept