

# LeetCode

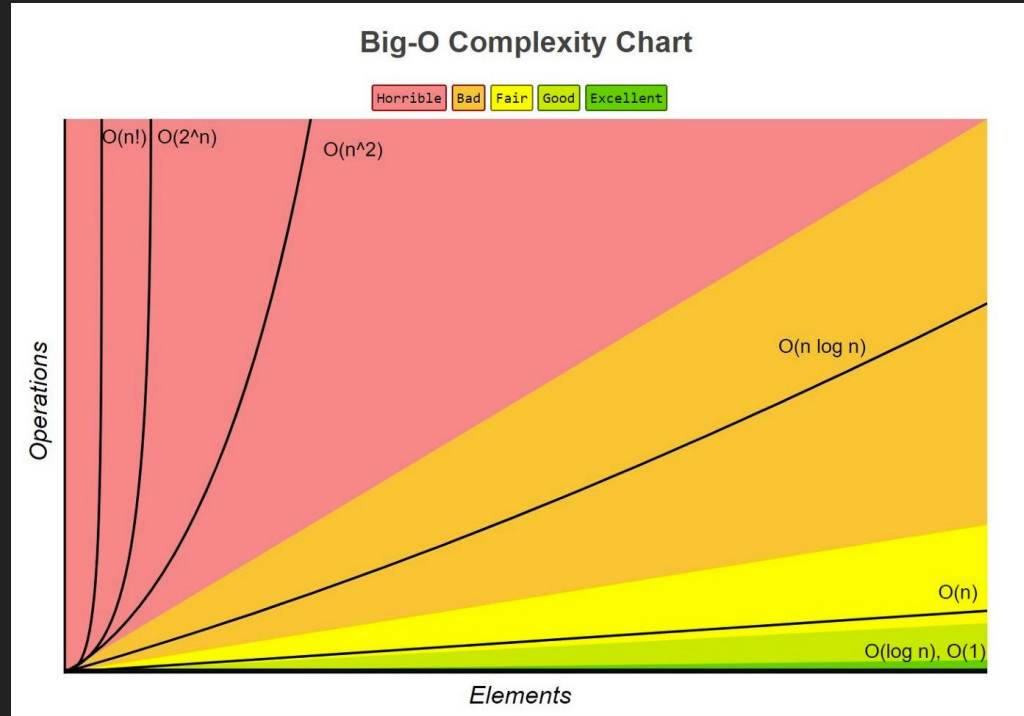
A project dedicated to DS&A

# Agenda

1. Time Complexity Review
2. Binary Search
3. Q&A Time (hopefully)

# Time Complexity (Speedrun)!

# Time Complexity - Chart



<https://www.bigocheatsheet.com/>

```
def example(nums):  
    if len(nums) > 0:  
        return nums[0]  
    else:  
        return -1
```

Time Complexity:  $O(1)$

Space Complexity:  $O(1)$

```
def example(nums):  
    for x in nums:  
        print(x)  
  
    for y in nums:  
        print(y)
```

Time Complexity:  $O(N)$   
Space Complexity:  $O(1)$

```
def containsDuplicate(nums):  
    """Determine if the list nums has duplicates."""  
    for i in range(len(nums)):  
        for j in range(len(nums)):  
            if i != j:  
                # if they are the same  
                if nums[i] == nums[j]:  
                    # because it does contain a duplicate  
                    return True  
    return False
```

Time Complexity:  $O(n^2)$

aka: Hot Garbage

Space Complexity:  $O(1)$

```
def containsDuplicate(nums):  
    """Determine if the list nums has duplicates."""  
    prev = set()  
    for x in nums:  
        if x in prev:  
            return True  
        prev.add(x)  
    return False
```

Time Complexity:  $O(n)$

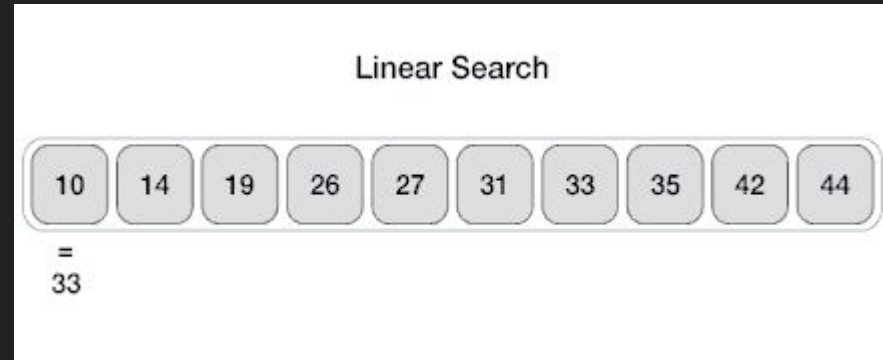
Space Complexity:  $O(n)$



# Binary Search

# What is the simplest way to search?

- **Linear search** is when you check each item in the list to find a target
- **Time Complexity:**  $O(n)$  - each item checked
- **Space Complexity:**  $O(1)$  - nothing is stored



# What is Binary Search?

- Asking **yes** or **no** questions that cut the problem in **half**
- Okay... What's the catch?
  - The elements **MUST BE SORTED!**
- **Time Complexity:**  $O(\log n)$ 
  - problem is cut in half with each question
- **Space Complexity:**  $O(1)$ 
  - nothing is stored

Search for 47

0	4	7	10	14	23	45	47	53
---	---	---	----	----	----	----	----	----

Binary search

steps: 0

37

1	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

low

mid

high

Sequential search

steps: 0

37

1	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59
---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

```
def search(self, nums: List[int], target: int) -> int:
    left = 0
    right = len(nums) - 1

    while left <= right:

        mid = (left + right) // 2

        if nums[mid] == target:
            return mid
        elif nums[mid] > target:
            right = mid - 1
        else:
            left = mid + 1

    return -1
```

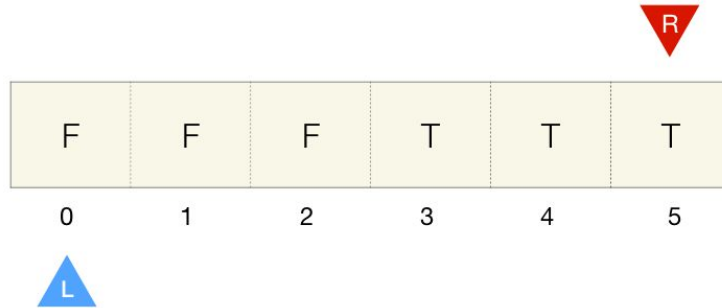
**Time Complexity:**  $O(\log n)$

**Space Complexity:**  $O(1)$

# Finding Boundaries

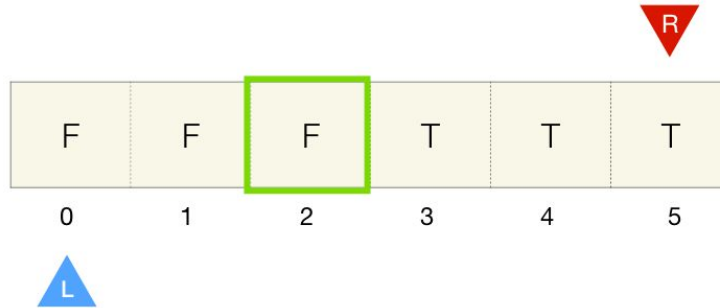
- Binary search is not just for searching! (Wha....?)
- You can also binary search to efficiently determine how data changes after a certain index
- This index is known as the “boundary”
- Comes in handy when working with implicitly sorted arrays...

# Finding Boundaries



**boundary\_index = -1**

# Finding Boundaries

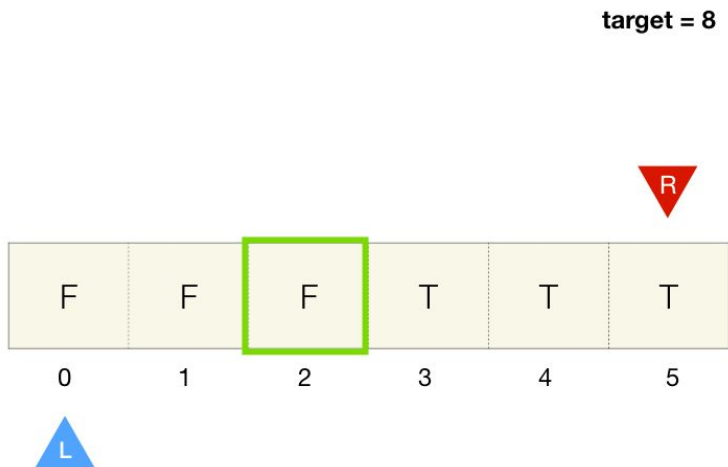


$$\text{mid} = (0 + 5) / 2 = 2$$

**boundary\_index = -1**



# Finding Boundaries

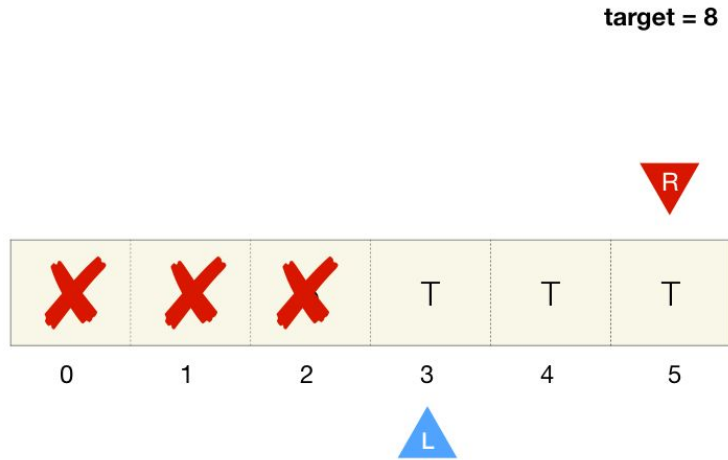


$$\text{mid} = (0 + 5) / 2 = 2$$

`arr[mid] == false`, discard everything on the left

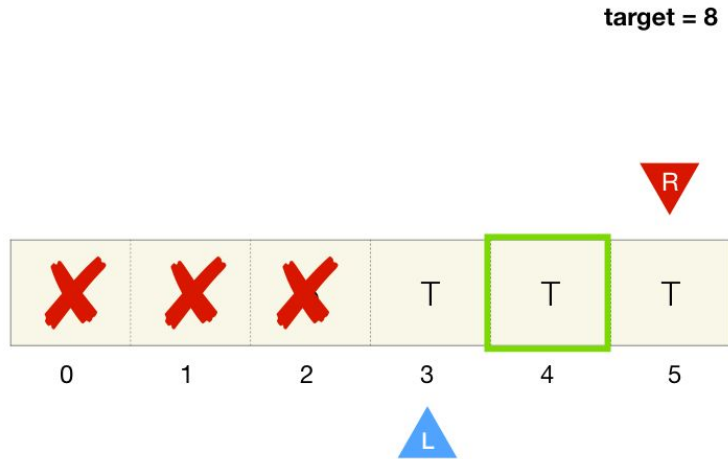
`boundary_index = -1`

# Finding Boundaries



boundary\_index = -1

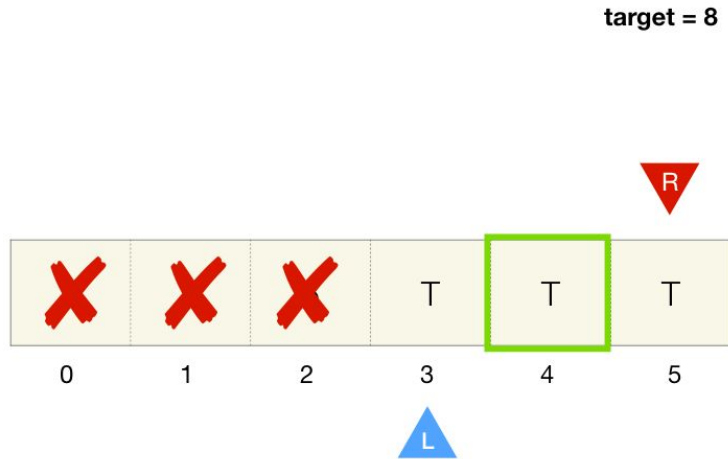
# Finding Boundaries



$$\text{mid} = (3 + 5) / 2 = 4$$

boundary\_index = -1

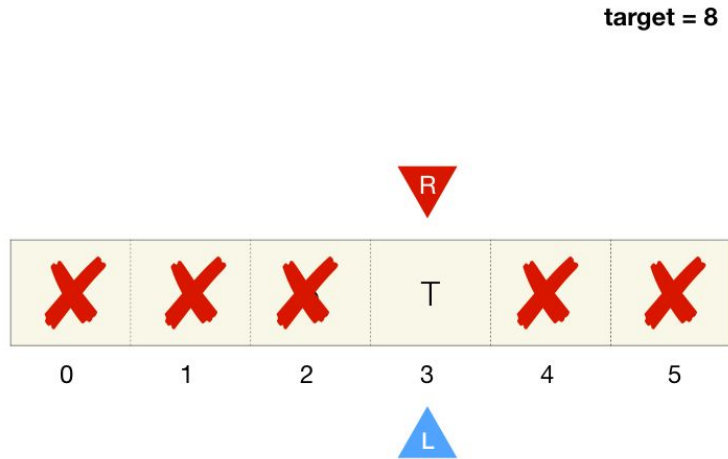
# Finding Boundaries



`arr[mid] == true`  
update `boundary_index` and discard everything on the right

`boundary_index = 4`

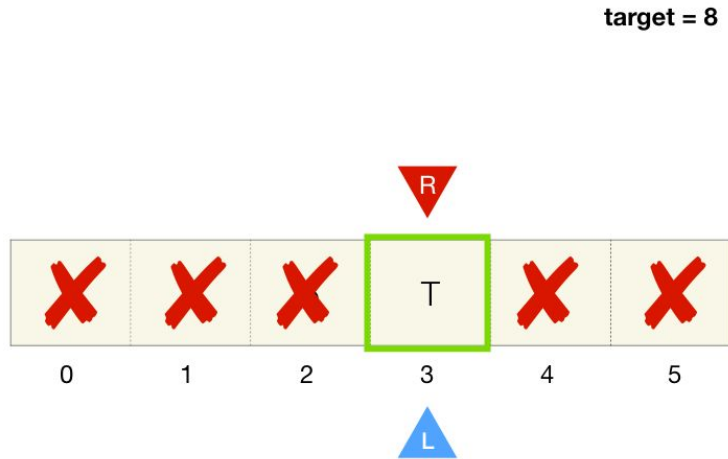
# Finding Boundaries



`arr[mid] == true`  
update `boundary_index` and discard everything on the right

`boundary_index = 4`

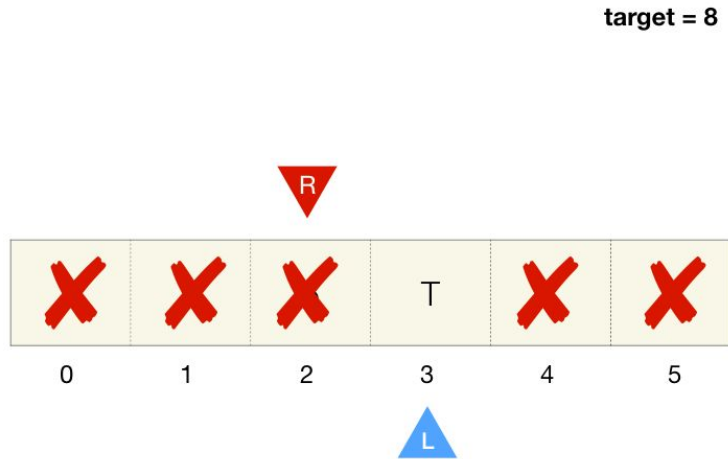
# Finding Boundaries



`arr[mid] == true`  
update `boundary_index` and discard everything on the right

`boundary_index = 3`

# Finding Boundaries



left > right, exit loop, boundary\_index is final answer

boundary\_index = 3

# “Find the Boundary” Template

- Here's a good template to follow

- Time Complexity:  $O(\log n)$
- Space Complexity:  $O(1)$

```
def find_boundary(arr):  
    left, right = 0, len(arr) - 1  
    boundary_index = -1  
  
    while left <= right:  
  
        mid = (left + right) // 2  
        if arr[mid]:  
            boundary_index = mid  
            right = mid - 1  
  
        else:  
            left = mid + 1  
  
    return boundary_index
```



# What Does “Implicitly Sorted” Mean?

- Basically a normal sorted array, but with a catch!
- The values in the array are sorted depending on:
  - A certain value in the array or....
  - A certain index in the array
- Normal sorted arrays are usually sorted in increasing / decreasing order
- Many MANY problems boil down to finding the boundary
- Example: Find the minimum in a sorted array that is pivoted:
  - For example, [10, 20, 30, 40, 50] becomes [30, 40, 50, 10, 20]
  - The list is not sorted, but there is a pattern that can be leveraged

Until next time...

Keep practicing

# More Problems

To Start:

- <https://leetcode.com/problems/binary-search/>
- <https://leetcode.com/problems/valid-perfect-square/>
- <https://leetcode.com/problems/sqrtx/>
- <https://leetcode.com/problems/find-smallest-letter-greater-than-target/>

Medium:

- <https://leetcode.com/problems/peak-index-in-a-mountain-array/>
- <https://leetcode.com/problems/capacity-to-ship-packages-within-d-days/>
- <https://leetcode.com/problems/two-sum-ii-input-array-is-sorted/>

Hard:

- <https://leetcode.com/problems/median-of-two-sorted-arrays/>

# Practice Problems

- <https://neetCode.io/practice>
- Try working on the
- Binary Search questions in the “NeetCode ALL” tab

Binary Search (0 / 17)				
Status	Problem	Difficulty	Video Solution	Code
<input type="checkbox"/>	Binary Search	Easy		<button>Python</button>
<input type="checkbox"/>	Search Insert Position	Easy		
<input type="checkbox"/>	Guess Number Higher Or Lower	Easy		
<input type="checkbox"/>	Arranging Coins	Easy		
<input type="checkbox"/>	Squares of a Sorted Array	Easy		
<input type="checkbox"/>	Valid Perfect Square	Easy		
<input type="checkbox"/>	Search a 2D Matrix	Medium		<button>Python</button>
<input type="checkbox"/>	Koko Eating Bananas	Medium		<button>Python</button>
<input type="checkbox"/>	Search In Rotated Sorted Array	Medium		<button>Python</button>
<input type="checkbox"/>	Find Minimum In Rotated Sorted Array	Medium		<button>Python</button>
<input type="checkbox"/>	Time Based Key Value Store	Medium		<button>Python</button>
<input type="checkbox"/>	Find First And Last Position of Element In Sorted Array	Medium		
<input type="checkbox"/>	Maximum Number of Removable Characters	Medium		
<input type="checkbox"/>	Populating Next Right Pointers In Each Node	Medium		
<input type="checkbox"/>	Search Suggestions System	Medium		