# 4.1

Searching

## Lesson Plan

- [10] Icebreaker
- [5] Linear Search
- [10] Binary Search Review
- [10] Binary Search Bug Hunt
- [5] Logarithms Review
- [10] idk whatever we need

- [45] Practice!



## Icebreaker

## Linear Search

## How do you determine if 31 is in this list?

[4, 1, 20, 15, 27, 15, 18, 28, 31, 23, 50, 39, 10, 46, 4, 23, 10, 49, 32, 18, 43]

## What's the best case?

[31, 4, 1, 20, 15, 27, 15, 18, 28, 23, 50, 39, 10, 46, 4, 23, 10, 49, 32, 18, 43]

## What's the worst case?

[4, 1, 20, 15, 27, 15, 18, 28, 23, 50, 39, 10, 46, 4, 23, 10, 49, 32, 18, 43, 31]

Or not there at all!

## What if the list is sorted?

[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

But now we can use this information to our advantage!

## Binary Search

## What if the list is sorted?

[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

Pick the middle element. Is our number bigger or smaller?

## What if the list is sorted?

[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

Now go again!

## What if the list is sorted?

[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

Keep going!

## What if the list is sorted?

[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

Keep going!

## What if the list is sorted?

[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

We found it!

## Let's try this again with some variables!

## What if the list is sorted?

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

low = 0
high = len(lst) - 1 = 20
mid = (low + high) // 2 = 10

target = 31
target > lst[mid], so we restrict to the right half
Our new low will be mid + 1

## What if the list is sorted?

11 12 13 14 15 16 17 18 19 20
[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

low = 11
high = len(lst) = 20
mid = (low + high) // 2 = 31 // 2 = 15

target = 31
target < lst[mid], so we restrict to the left half
Our new high will be mid - 1

## What if the list is sorted?

11 12 13 14
[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

low = 11
high = len(lst) = 14
mid = (low + high) // 2 = 25 // 2 = 12

target = 31
target > lst[mid], so we restrict to the right half
Our new low will be mid + 1

## What if the list is sorted?

13 14
[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

low = 13
high = len(lst) = 14
mid = (low + high) // 2 = 27 // 2 = 13

target = 31
target > lst[mid], so we restrict to the right half
Our new low will be mid + 1

## What if the list is sorted?

14
[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 31, 32, 39, 43, 46, 49, 50]

low = 14
high = len(lst) = 14
mid = (low + high) // 2 = 27 // 2 = 14

target = 31
target == lst[mid], so we found it!!

## What if 31 hadn't been in the list after all?

Everything else up to this point would have been exactly the same.

14
[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 30, 32, 39, 43, 46, 49, 50]

low = mid = high = 14

target = 31
target > lst[mid], so we restrict to the left half
Our new low will be mid + 1

## Stop searching when low > high

[1, 4, 4, 10, 10, 15, 15, 18, 18, 20, 23, 23, 27, 28, 30, 32, 39, 43, 46, 49, 50]

low = 15
high = 14

Low > high ?????

## Binary Search Algorithm

## Binary Search - Review

Basic idea:

1. Compare the target to the middle element of the list.
2. If target equals the middle element, return the index of the middle element.
3. If the target is less than the middle element, then restrict to the lower half of the array
4. If the target is greater than the middle element, restrict to the upper half.
5. **Repeat steps 1-4 on the reduced portion of the list**
6. If restricted array becomes empty, return None.

## Binary Search Bug Hunt

```python
# Buggy Binary Search
def binary_search(target, values):
 return binary_search_helper(target, values, 0, len(values) - 1)

def binary_search_helper(target, values, left, right):
 mid = (left + right) // 2
 if (values[mid] == target):
   return mid
 if (values[mid] < target):
   return binary_search_helper(target, values, left, mid)
 else:
   return binary_search_helper(target, values, mid, right)
```

## Binary Search Bug Hunt: Answers

Debugging Strategies:

- Establish expected outputs & compare to actual output
- Trace behavior with the debugger
  - Consider adding extra local variables to view intermediate state
- Add print statements to view intermediate states

What bugs did you find?

- Missing the base case for no target in list
- Recursive conditional is flipped, goes left when it should go right
- OBOB in the left recursive calls, should recur on mid - 1 or mid + 1

## Iterative Binary Search

```python
def binary_search(target, values):
 return binary_search_helper(target, values, 0, len(values) - 1)

def binary_search_helper(target, values, left, right):
  if (left > right):
    return False
  mid = (left + right) // 2
  if (values[mid] == target):
    return mid
  if (values[mid] > target):
    return binary_search_helper(target, values, left, mid - 1)
  else:
    return binary_search_helper(target, values, mid + 1, right)
```

```python
def binary_search_iterative(target, values):
  left = 0
  right = len(values) - 1

  while left < right:
    mid = (left + right) // 2
    if (values[mid] == target):
      return mid
    if (values[mid] > target):
      right = mid - 1
    else:
      left = mid + 1
  return False
```

## What is the runtime of binary search?

# log(n)

## Review: Logarithms

## Logarithms in CS

In CS, we almost always mean $\log_2(x)$

$\log_2(x)$ –> $2^{what} = x$

How many times can we split x in half before we're only left with 1?

## Effect of Doubling Input Size

| n | log(n) | $n^2$ |
|---|---|---|
| 2 | 1 | 4 |
| 4 | 2 | 16 |
| 8 | 3 | 64 |
| 16 | 4 | 256 |
| 32 | 5 | 1024 |
| x2 | +1 | x4 |

## Effect of 10x'ing Input Size

| n | log(n) | $n^2$ |
|---|---|---|
| 10 | ~3 | 100 |
| 100 | ~7 | 10,000 |
| 1000 | ~10 | 1,000,000 |
| 10000 | ~13 | 100,000,000 |
| 100000 | ~17 | 10,000,000,000 |
| x10 | + ~3 | Double the number of zeros! |

## Aside: Logarithm Rules

Why + ~3?

## Why + ~3?

**log(a*b) = log(a) + log(b)**

And some others:
log(a / b) = log(a) - log(b

$\log(a^b) = b * \log(a)$
→ $\log(a^b) = \log(a * a * \dots a) = \log(a) + \log(a) + \dots + \log(a) = b * \log(a)$

$\log_b(a) = \log_x(a) / \log_x(b)$

## Why + ~3?

Let's say $\log_2(n) = x$

Then $\log_2(10*n) = \log_2(10) + \log_2(n) = \log_2(10) + x$

$\log_2(10) \sim 3$

## Practice Problems

You will be working in teams of 3 or 4. Use the table below to figure out what your role is. [replit]

| Role | Responsibilities | Assignment Criteria |
|---|---|---|
| Captain | Share screen, write code, keep track of time, ensure all team members participate | Person who has been assigned this role the least number of times |
| Tester | Plays devil's advocate, design test cases, determine algorithm complexity (time and space) | Person who has been assigned this role the least number of times |
| Presenter | Explain solution to the class, present the team's algorithm design decisions, state solution's complexity (time and space), share one thing the team learned from the problem | Person who has been assigned this role the least number of times |

If there are ties, get creative and come up with a way to break them (i.e., sort yourselves by last name, distance to Google Austin, etc.)
If there are 4 members in your team, you should have two Testers

# Binary Search with duplicates

Next problem:

- In a list with some duplicate values find the *leftmost* occurrence of the target value if any.

# Binary Search with duplicates

```
def binary_search(target, lst):
 '''Returns the index of target in lst, or None if target is
   not present in lst'''
 return binary_search_helper(target, lst, 0, len(lst) - 1)

def binary_search_helper(target, lst, left, right):
 if left > right:
   return None
 mid = (left + right) // 2
 if (lst[mid] == target and (mid == 0 or lst[mid - 1] != target)):
   return mid
 if (lst[mid] >= target):
   return binary_search_helper(target, lst, left, mid - 1)
 else:
   return binary_search_helper(target, lst, mid + 1, right)
```

Restrict this base case. Only return the index if the value to the left isn't target or there is no value to the left.

Less-than-or-equal-to because we should go left if mid was == target but not the leftmost instance of target