Search

Common search problems:

Does the list contain x? Return True or False

Where is x located in the list? Return the index, or -1 if not found

How many times is x in the list? Returns a non-negative integer

Find and replace an item

What is the closest item to x?
Search

Python examples:

“apple” in [“banana”, “orange”, “carrot”] -> returns False

99 in [23, 77, -34, 45, 45, 99, -4] -> returns True

[23, 77, -34, 45, 45, 99, -4].index(99) -> Returns 5

We will implement our own search algorithms!
Linear Search

Idea:

Go through each item in a list and check if it matches our search key

Solution 1:

How many steps does it take to find 99?

How many steps does it take to find 77?
Linear Search

Solution 2: How is this solution different from the previous solution?

How many steps does it take to find 99?

How many steps does it take to find 77?
Linear Search

Why doesn’t this program work?
Performance

How long does it take a program to run?

We can measure it using time

```python
import time
...
startTime = time.time()  # Returns seconds
runAlgorithm()
endTime = time.time()
algorithmDuration = endTime - startTime
```

Performance is “platform-specific”, e.g. depends on hardware

Newer hardware is faster than slower hardware
Running time

How can we have measure the speed of an algorithm in a platform-independent way?

Idea: Count the number of steps the algorithm has to take

How do the number of steps increase as we increase the input size?

Ex: For linear search, the speed of the algorithm depends on the list size

-> a list with more elements in it will take longer to search

-> specifically, the number of steps grows **linearly** with the list size
Linear search time grows linearly with list size

This graph is based on simulations of solution 2!

Why is it so jaggy?

Size of the list from 100 to 10K
Big-O notation

We use the term “big-oh” to indicate the rough number of steps for an algorithm.

Linear search is an “order N algorithm”, signified as O(N).

- N represents the number of items in the list.
- Big-O notation ignores whether the number of steps is actually 3N or N+10.

In practice constants matter (N is 3 times faster than 3N) but for understanding how an algorithm scales with data, we only care about the dominant term. E.g. when N is really big, the constants become insignificant!
Can we do better than $O(N)$?

Yes! If the list is sorted we can use **Binary Search**

(If the list isn’t sorted, linear search is the only option!)
Binary Search

Idea: Eliminate half the data from consideration each step

Example: Search for 44 in the numbers between 1 and 100

Step 1: Is 44 bigger or smaller than 50? Smaller -> Check left half [1, 49]
Step 2: Is 44 bigger or smaller than 25? Bigger -> Check right [26,49]
Step 3: Is 44 bigger or smaller than 37? Bigger -> Check right [38, 49]
Step 4: Is 44 bigger or smaller than 43? Bigger -> Check right [44,49]
Step 5: Is 44 bigger or smaller than 46? Smaller -> Check left [44,45]
Step 6: Is 44 bigger or smaller than 44? It’s equal!!! FOUND IT

How many steps would this take using linear search? 44!

NOTE: The midpoint in an interval \([a,b]\) is \((a+b)/2\). To convert to an integer, cast to an int!
Example: Search for 99 in [-20,-4,44,58,99,145]

To perform binary search in a list, we use list indices to keep track of intervals.

Let $\textbf{low}$ be the beginning of an interval.

Let $\textbf{high}$ be the end of an interval.

Let $\textbf{mid} = \text{int}((\text{high}+\text{low})/2)$ be the middle of the interval.

How many steps does it take?
Example: Search for 99 in [-20,-4,44,58,99,145]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>-4</td>
<td>44</td>
<td>58</td>
<td>99</td>
<td>145</td>
</tr>
</tbody>
</table>

What is **low**, **mid**, and **high** to start?
Example: Search for 99 in [-20,-4,44,58,99,145]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>-4</td>
<td>44</td>
<td>58</td>
<td>99</td>
<td>145</td>
</tr>
</tbody>
</table>

L[mid] = 44. What should we do next?
Example: Search for 99 in [-20,-4,44,58,99,145]

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>-4</td>
<td>44</td>
<td>58</td>
<td>99</td>
<td>145</td>
</tr>
</tbody>
</table>

L[mid] = 99. FOUND IT!
Example: Search for “b” in [“a”, “b”, “c”, “d”, “e”]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

What is **low**, **mid**, and **high** to start?
Example: Search for “b” in ["a", "b", "c", "d", "e"]

L[mid] = “c”. What should we do next?
Example: Search for “b” in [“a”, ”b”, ”c”, ”d”, ”e”]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td>e</td>
</tr>
</tbody>
</table>

low  0
high  1

What should mid be?
Example: Search for “b” in [“a”, ”b”, ”c”, ”d”, ”e”]

L[mid] = “a” What should our next step be?
Example: Search for “b” in [“a”, “b”, “c”, “d”, “e”]

mid and high point to the same index! $L[mid] = “b”$
Binary search partial algorithm

Search(x, L):
    low = 0
    high = len(L)-1
    for each step:
        mid = int((high+low)/2)
        # Check L[mid]
        if x < L[mid]: # Search left
            high = mid-1
        elif x > L[mid]: # Search right
            low = mid+1
        else: # x == L[mid]
            return True

When do we stop?
What happens if L doesn’t contain x?
Example: Search for 0 in \([-20, -4, 44, 58, 99, 145]\)

\[L[mid] = 44. \text{ What should we do next?}\]
Example: Search for 0 in [-20,-4,44,58,99,145]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>-20</td>
<td>-4</td>
<td>44</td>
<td>58</td>
<td>99</td>
</tr>
<tr>
<td>145</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example: Search for 0 in [-20,-4,44,58,99,145]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-20</td>
<td>-4</td>
<td>44</td>
<td>58</td>
<td>99</td>
<td>145</td>
</tr>
</tbody>
</table>

L[mid] = -4. What should we do next? Apply the algorithm..
Example: Search for 0 in [-20, -4, 44, 58, 99, 145]

The markers high and low switched places!!
Binary search algorithm

Search(x, L):
    low = 0
    high = len(L)-1
    while low <= high:
        mid = int((high+low)/2)
        if x < L[mid]: # Search left
            high = mid-1
        elif x > L[mid]: # Search right
            low = mid+1
        else: # x == L[mid]
            return True
    return False
Example: Binary search

python3 binariesimple.py

low/mid/high 0 2 5
low/mid/high 3 4 5
Num steps:  2
Found 99? True

Num steps:  3
Found 0? False

code view

```python
def search(x, L):
    ... def main():
    numbers = [-28, -4, 44, 58, 99, 145]
    print("Found 99", search(99, numbers))
    letters = ['a', 'b', 'c', 'd', 'e']
    print("Found b", search('b', letters))
```
Binary search runtime performance

Everytime we make a step, we divide the problem in half.

Suppose we have $N$ items in the list

Step 1: $N/2$

Step 2: $N/2/2 = N/2^2$

Step 2: $N/2/2/2 = N/2^3$

....

Step $k$: $N/2^k$

$O(\log_2 N)$ based algorithm!

[We usually just say $O(\log N)$]
Binary search time grows logarithmically with list size

Remember linear search?

Size of the list from 100 to 10K
Example: A program which simulates binary search for different sized lists and different inputs

(linear search would work similarly)