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?

simpleSearch.py — ~/classes/cs21/f18/library.git/inclass/w09 — Atom

File Edit View Selection Find Packages Help

simpleSearch.py

```python
"""
Write a function that searches through a list of numbers


"""

def search(x, L):
    """
    Search the list L for the element x
    param L: a list of integers
    param x: an integer to find
    Returns True if the item is found; False otherwise
    """
    isFound = False
    for val in L:
        if val == x:
            isFound = True
    return isFound

def main():
    numbers = [23, 77, -34, 45, 99, -4]
    found = search(99, numbers)
    if found:
      print("Found!")
    else:
      print("Not found!")

main()
```

simpleSearch.py 12:55 LF UTF-8 Python updates Fetch 12 files

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?

```python
"""
Write a function that searches through a list of numbers


"""

def search(x, L):
    """
    Search the list L for the element x
    param L: a list of integers
    param x: an integer to find
    Returns True if the item is found; False otherwise
    """
    for val in L:
        if val == x:
            return True
    return False

def main():
    numbers = [23, 77, -34, 45, 99, -4]
    found = search(99, numbers)
    if found:
      print("Found!")
    else:
      print("Not found!")

main()
```

Linear Search

Why doesn't this program work?

```python
"""
Write a function that searches through a list of numbers


"""

def search(x, L):
    """
    Search the list L for the element x
    param L: a list of integers
    param x: an integer to find
    Returns True if the item is found; False otherwise
    """
    for val in L:
        if val == x:
            return True
        else:
            return False

def main():
    numbers = [23, 77, -34, 45, 99, -4]
    found = search(99, numbers)
    if found:
      print("Found!")
    else:
      print("Not found!")

main()
```

Performance

How long does it a take a program to run?

We can measure it using time

```
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

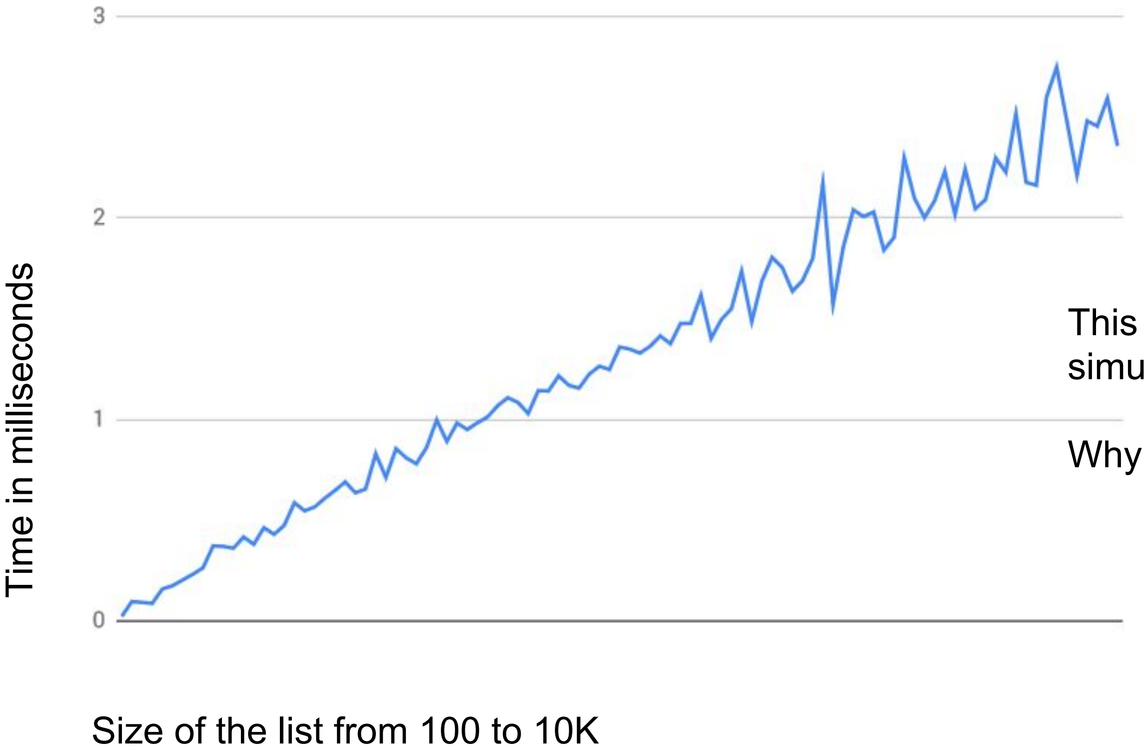
How do the number of steps increase as we increase the input size?

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?

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 are only option!)

Binary Search

NOTE: The midpoint in an interval [a,b] is (a+b)/2. To convert to an integer, cast to an int!

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

Already checked 50, so exclude it from next interval

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

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 **low** be the beginning of an interval

Let **high** be the end of an interval

Let **mid** = int((high+low)/2) be the middle of the interval

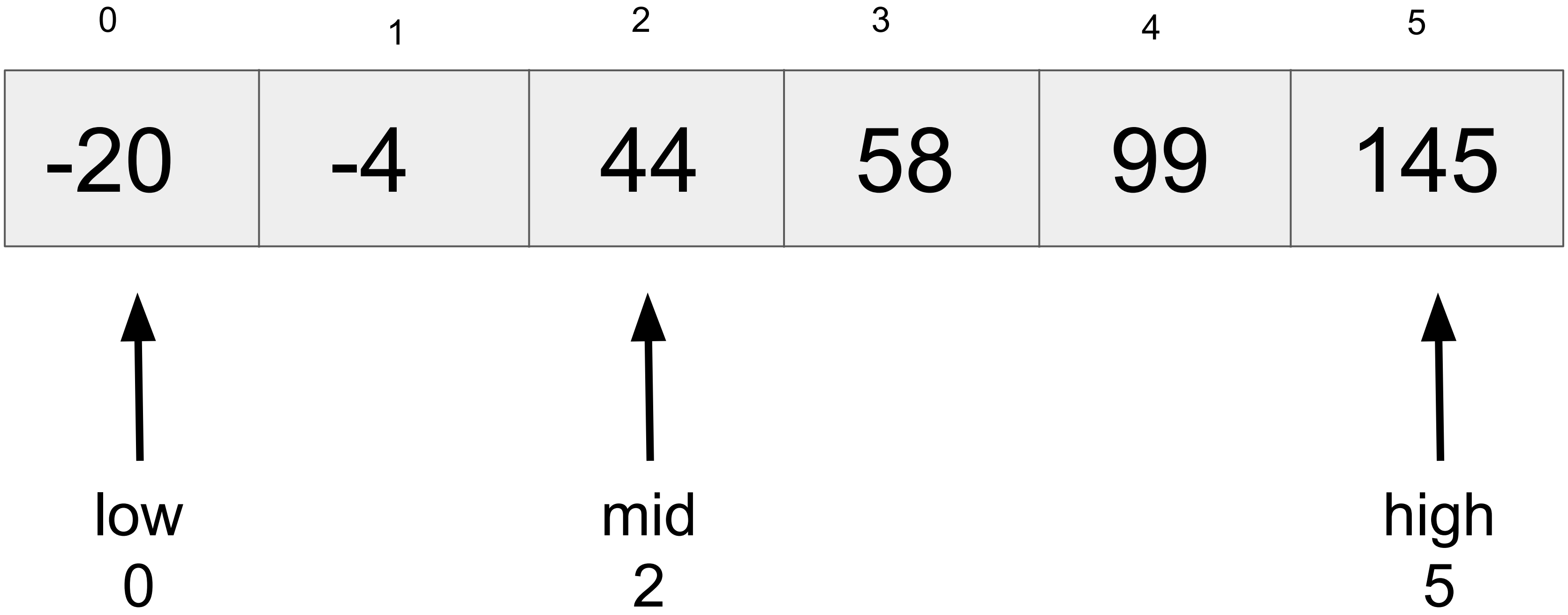
How many steps does it take?

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

0	1	2	3	4	5
-20	-4	44	58	99	145

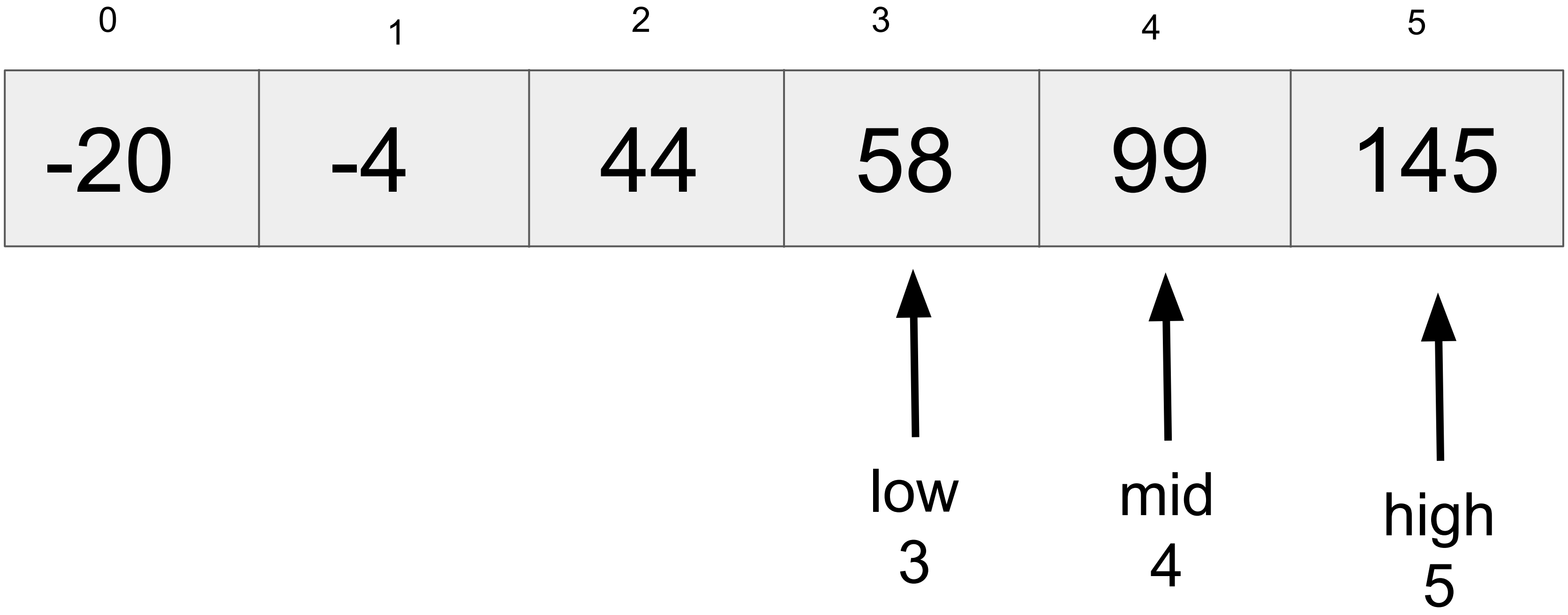
What is **low**, **mid**, and **high** to start?

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



L[mid] = 44. What should we do next?

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



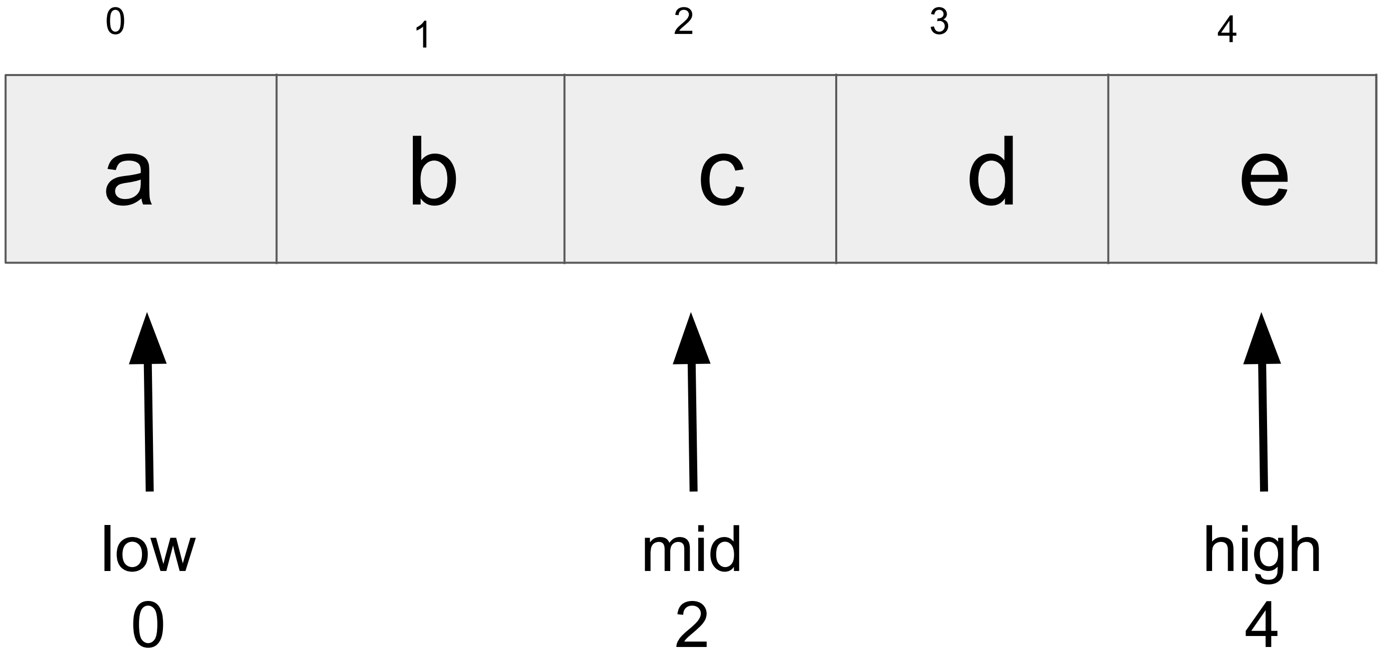
L[mid] = 99. FOUND IT!

Example: Search for “b” in [“a”,”b”,”c”,”d”,”e”]

0	1	2	3	4
a	b	c	d	e

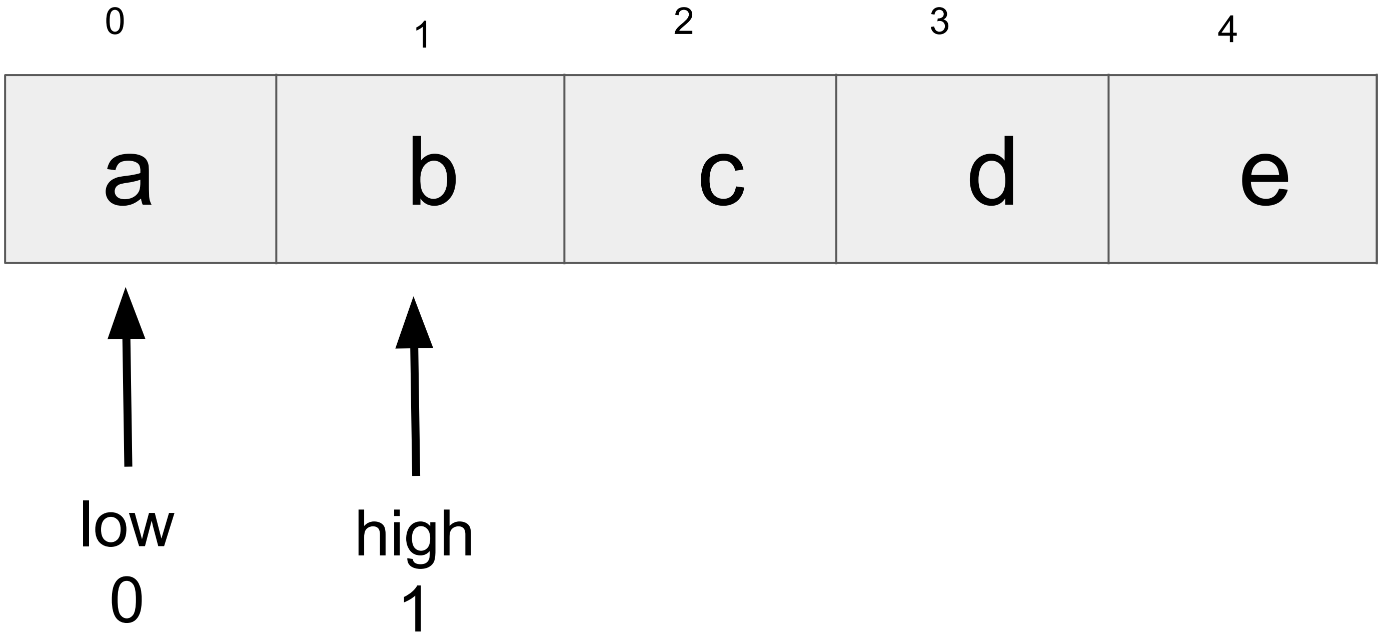
What is **low**, **mid**, and **high** to start?

Example: Search for “b” in [“a”,”b”,”c”,”d”,”e”]



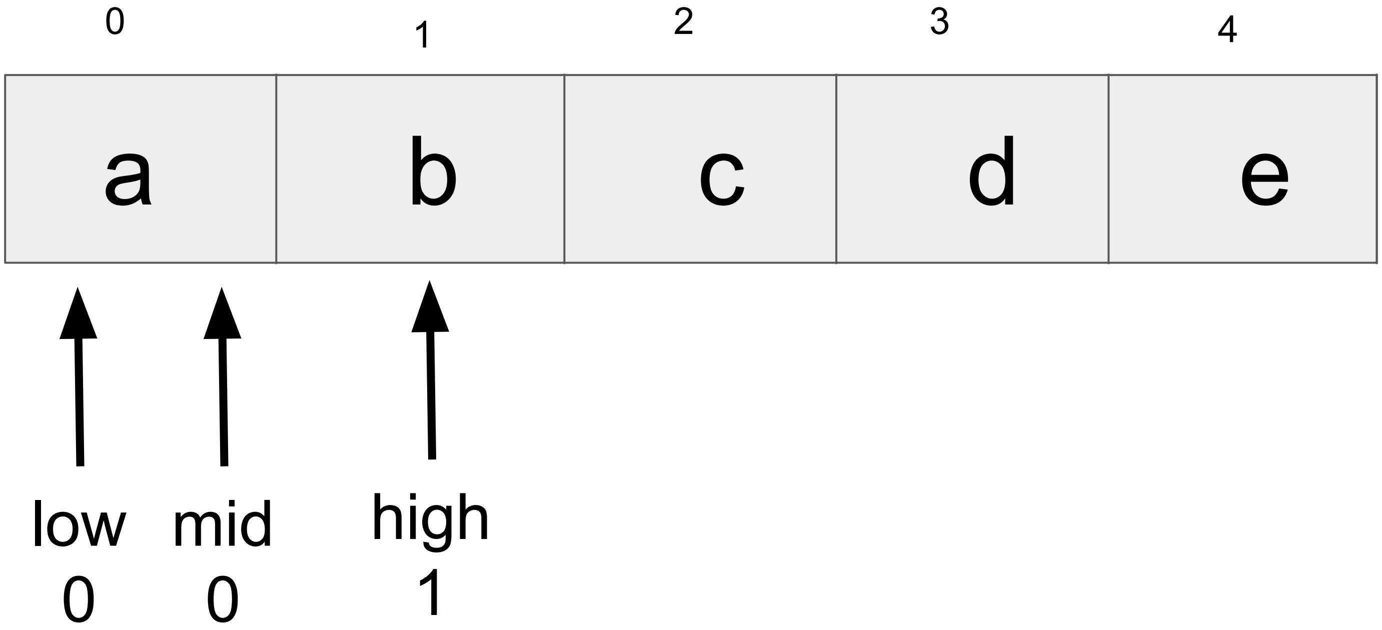
L[mid] = “c”. What should we do next?

Example: Search for “b” in [“a”,”b”,”c”,”d”,”e”]



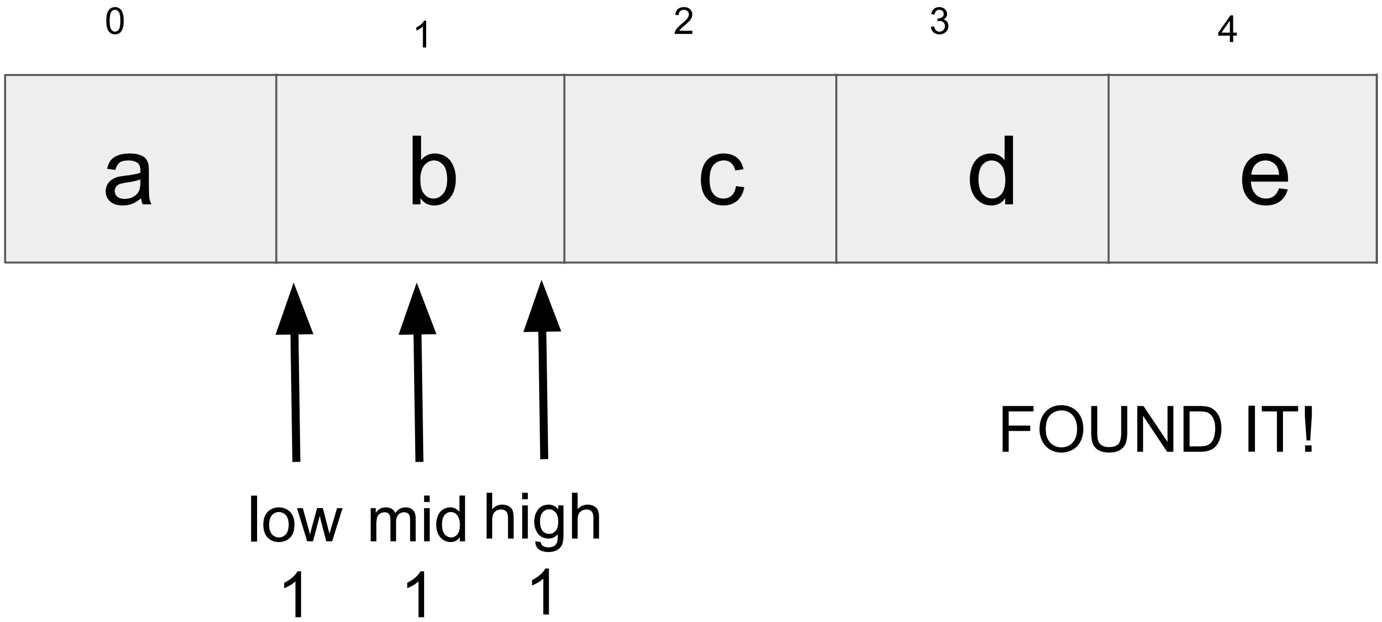
What should mid be?

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”]



FOUND IT!

mid and high point to the same index! L[mid] = “b”

Binary search partial algorithm

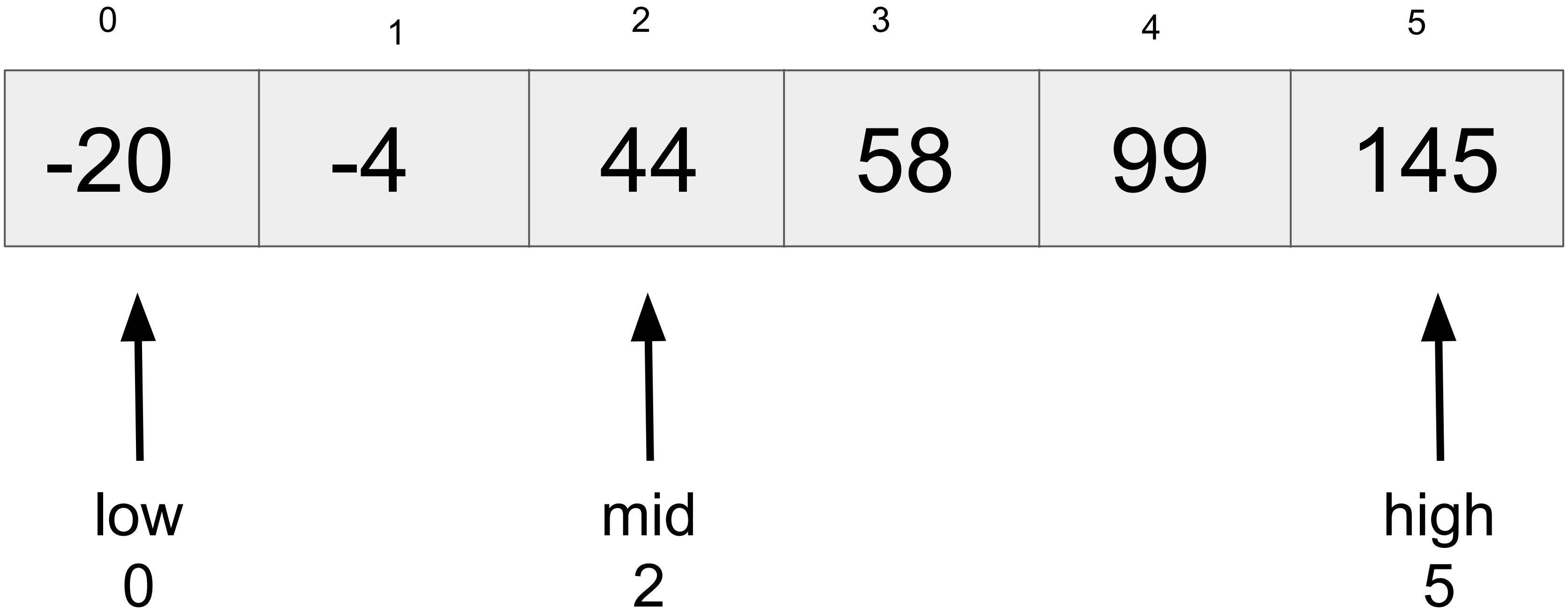
```
Search(x, L):
    low = 0
    high = len(L)-1
    for each step:
        mid = int((high+low)/2)
        #Check L[mid]
        if x < L[mid]: # Search left
            high = mid-1
        elif x > L[mid]: # Search right
            low = mid+1
        else: # x == L[mid]
            return True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

When do we stop?

What happens if L doesn't contain x?

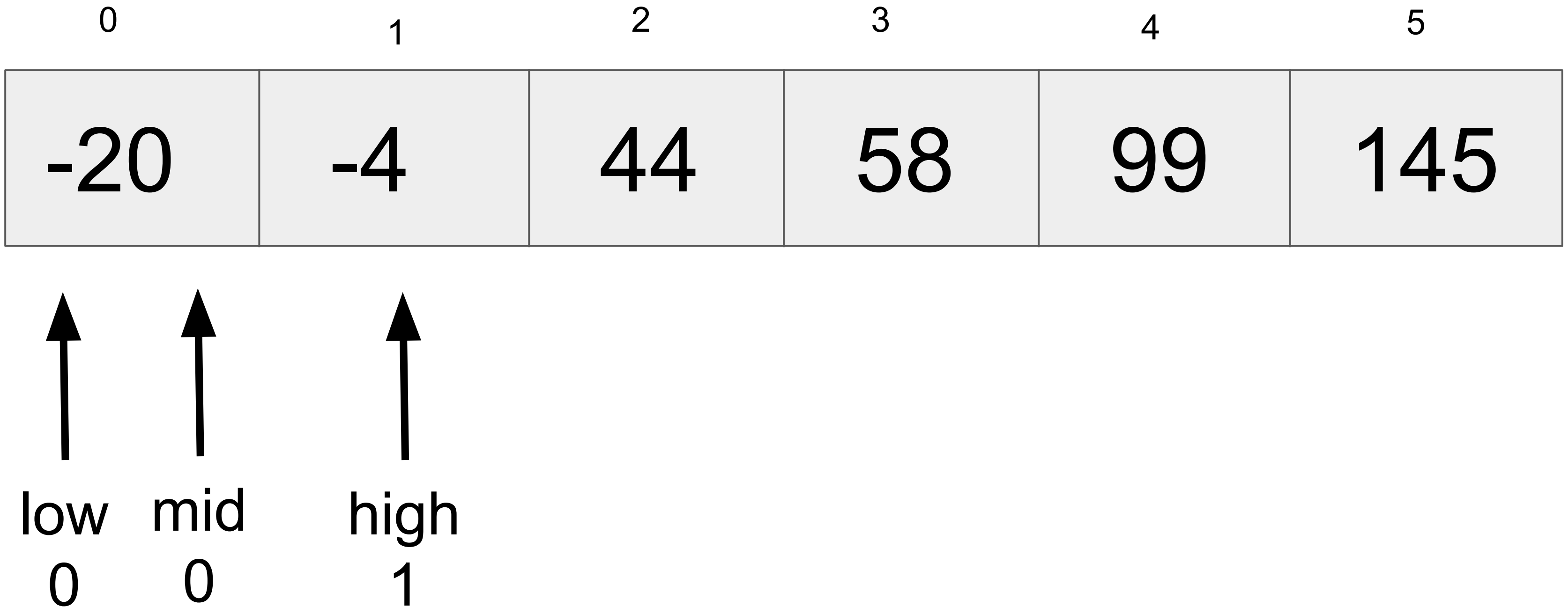
What happens when an item isn't in the list?!

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



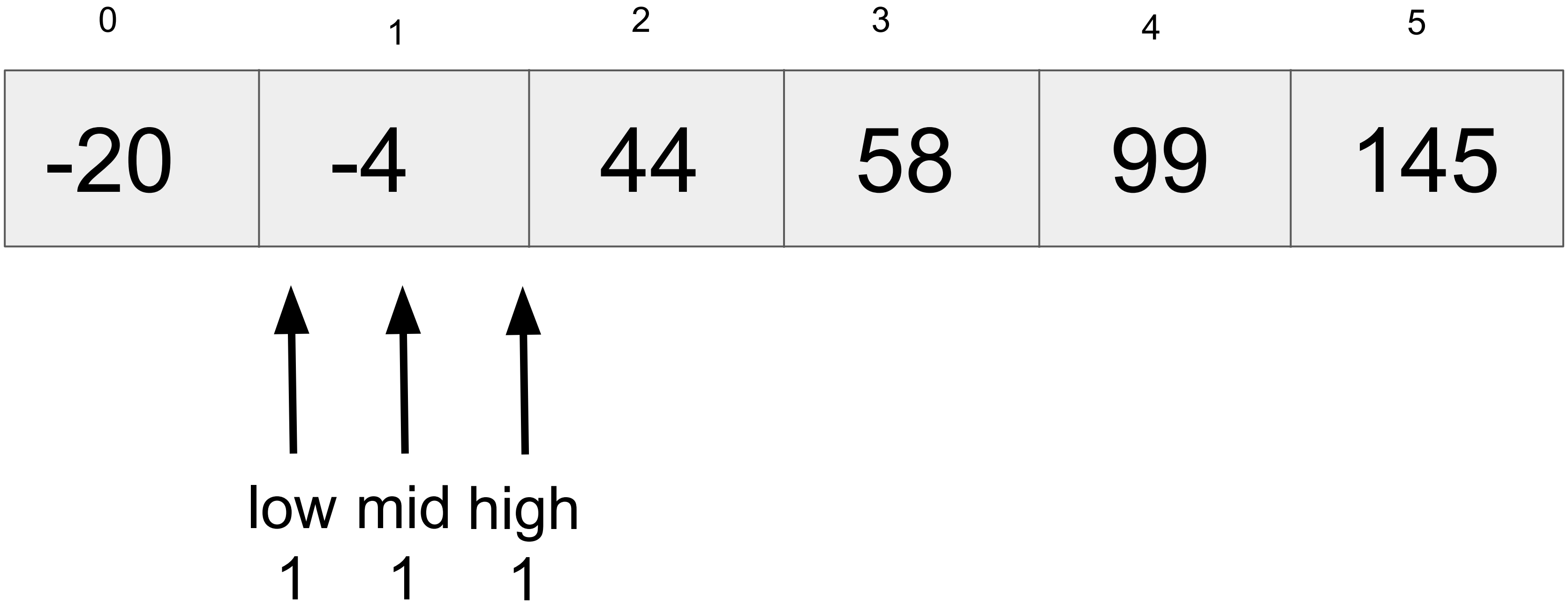
L[mid] = 44. What should we do next?

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



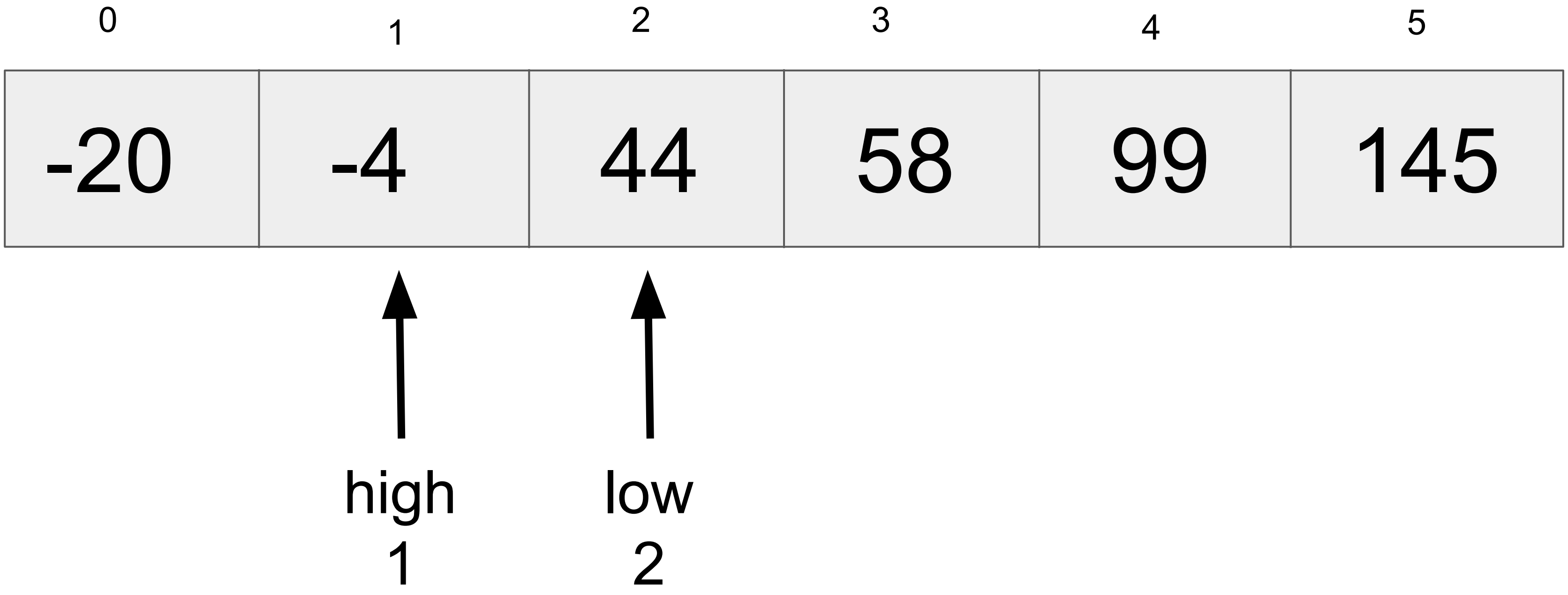
L[mid] = -20. What happens next? Apply the algorithm..

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



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
```

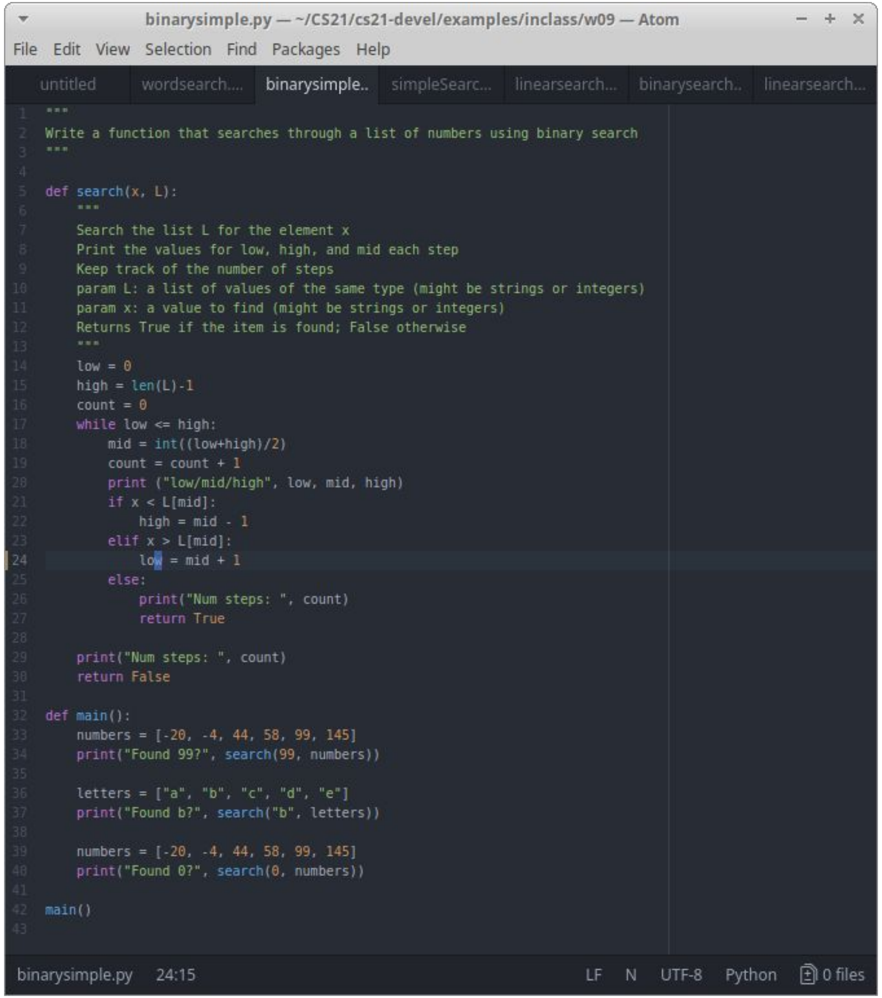
Example: Binary search

python3 binarysimple.py

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

low/mid/high 0 2 4
low/mid/high 0 0 1
low/mid/high 1 1 1
Num steps: 3
Found b? True

low/mid/high 0 2 5
low/mid/high 0 0 1
low/mid/high 1 1 1
Num steps: 3
Found 0? False




```python
"""
Write a function that searches through a list of numbers using binary search
"""

def search(x, L):
    """
    Search the list L for the element x
    Print the values for low, high, and mid each step
    Keep track of the number of steps
    param L: a list of values of the same type (might be strings or integers)
    param x: a value to find (might be strings or integers)
    Returns True if the item is found; False otherwise
    """
    low = 0
    high = len(L)-1
    count = 0
    while low <= high:
        mid = int((low+high)/2)
        count = count + 1
        print ("low/mid/high", low, mid, high)
        if x < L[mid]:
            high = mid - 1
        elif x > L[mid]:
            low = mid + 1
        else:
            print("Num steps: ", count)
            return True

    print("Num steps: ", count)
    return False

def main():
    numbers = [-20, -4, 44, 58, 99, 145]
    print("Found 99?", search(99, numbers))

    letters = ["a", "b", "c", "d", "e"]
    print("Found b?", search("b", letters))

    numbers = [-20, -4, 44, 58, 99, 145]
    print("Found 0?", search(0, numbers))

main()
```

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 = $N/2^3$

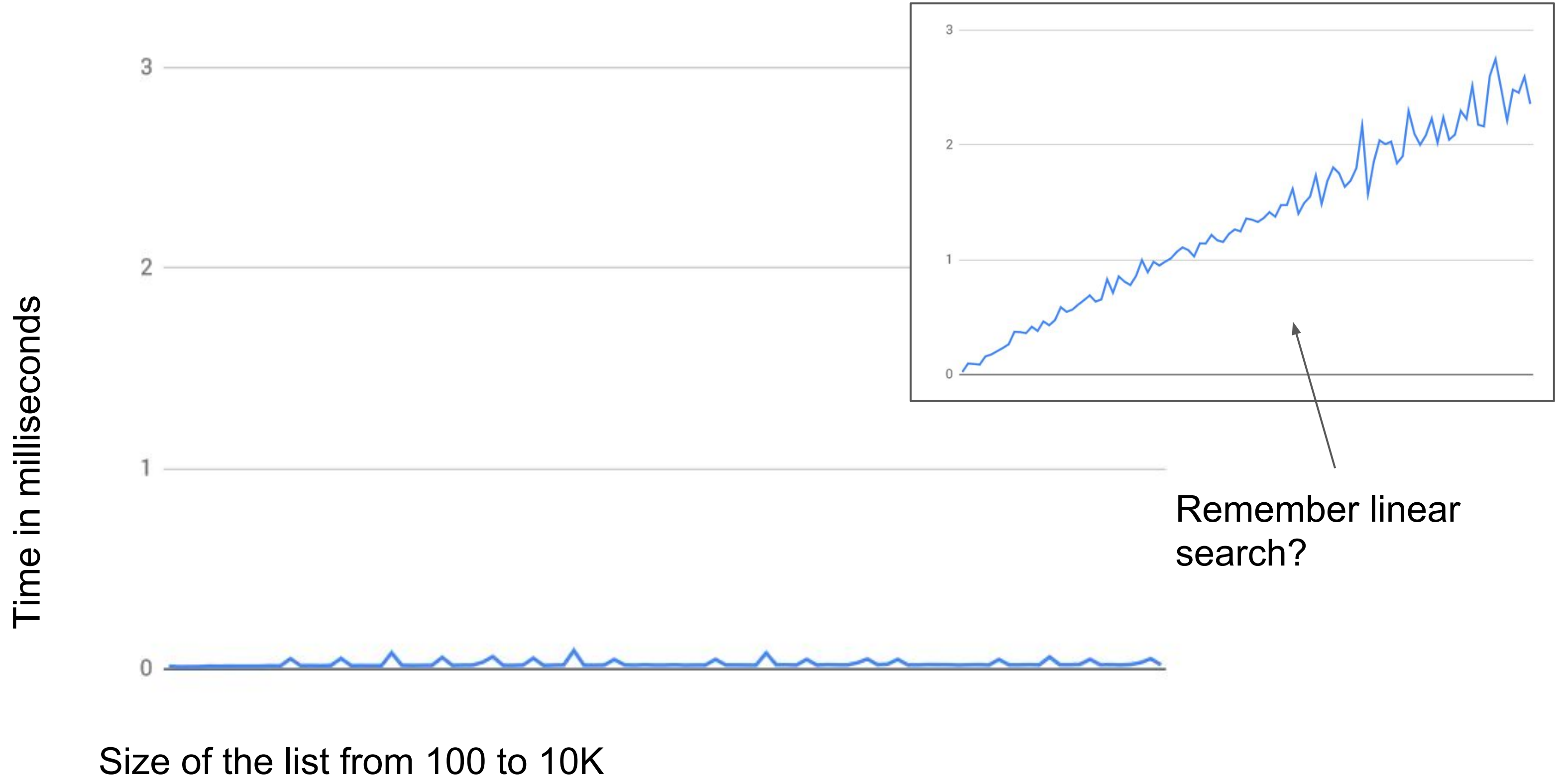
….

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



Example: A program which simulates binary search for different sized lists and different inputs

(linear search would work similarly)

binarysearch-sim.py — ~/CS21/cs21-devel/examples/inclass/w09 — Atom

```python
"""
Simulate searches for different sized lists and print their average
"""

import random
import time
from binarysearch import search

def simulateListSearch(L, numTimes):
    """
    Repeat a series of searchs numTimes times
    Compute the average time the search takes
    Param L: a list of integers
    Param numTimes: the number of times to search for values
    Returns the average time search takes
    """
    durationTotal = 0
    for i in range(numTimes):
        val = random.randrange(len(L))
        startTime = time.time()
        index = search(val, L)
        duration = (time.time() - startTime) * 1000
        durationTotal = durationTotal + duration

    return durationTotal / numTimes

def main():

    for i in range(100, 10000, 100):
        nums = range(i)
        average = simulateListSearch(nums, 100)
        print(average)

main()
```