Important data structures and when to use them

or, why is my program so slowwww

@jessicamckellar

A seemingly simple task: looking up a bunch of words

```
my_words = [elt.strip() for elt in \
    file("my_words.txt", "r").readlines()]
word_list = [elt.strip() for elt in \
    file("all_words.txt", "r").readlines()]

counter = 0
for word in my_words:
    if word in word_list:
        counter += 1
print counter
```

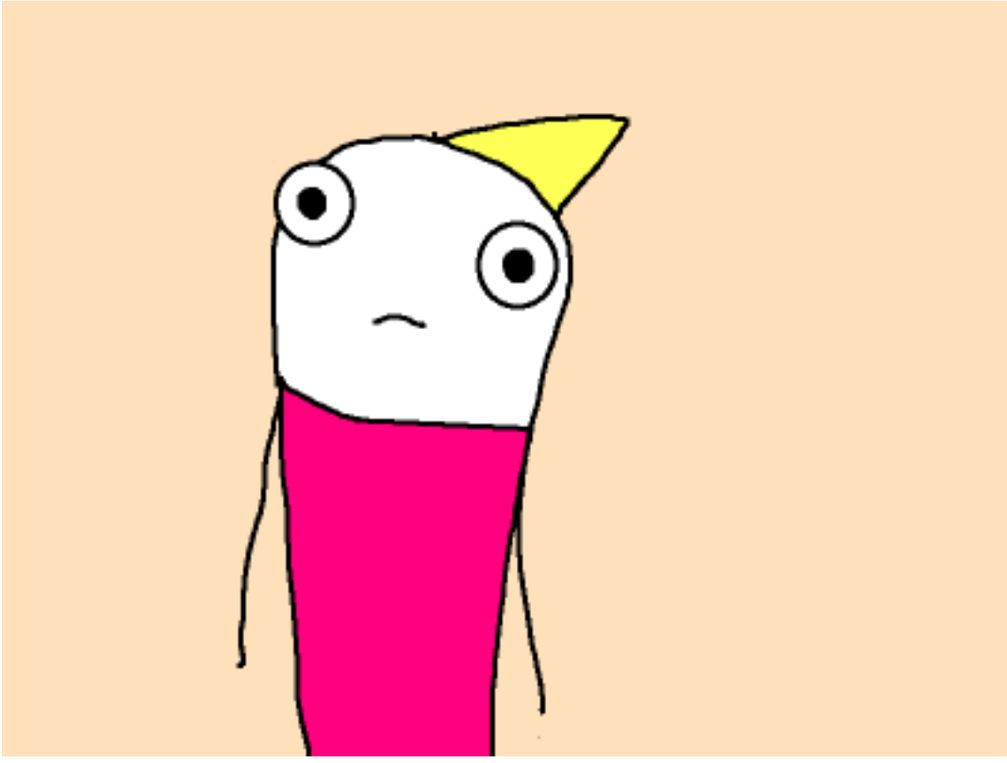
$

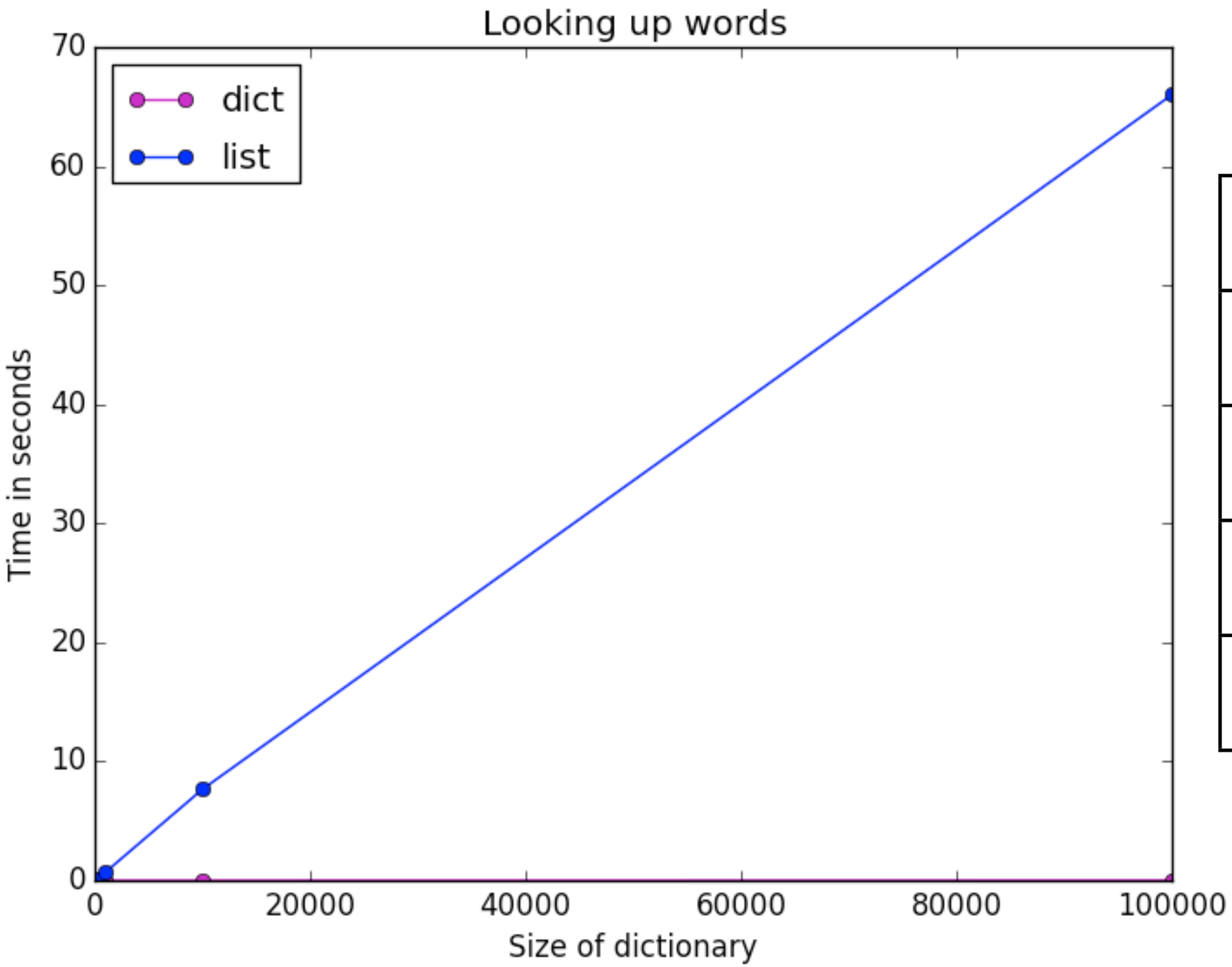
```python
import time

start = time.time()

counter = 0
for word in my_words:
    if word in word_list:
        counter += 1
print counter

print "Elapsed time:", time.time() - start
```

Elapsed time: 65.9971

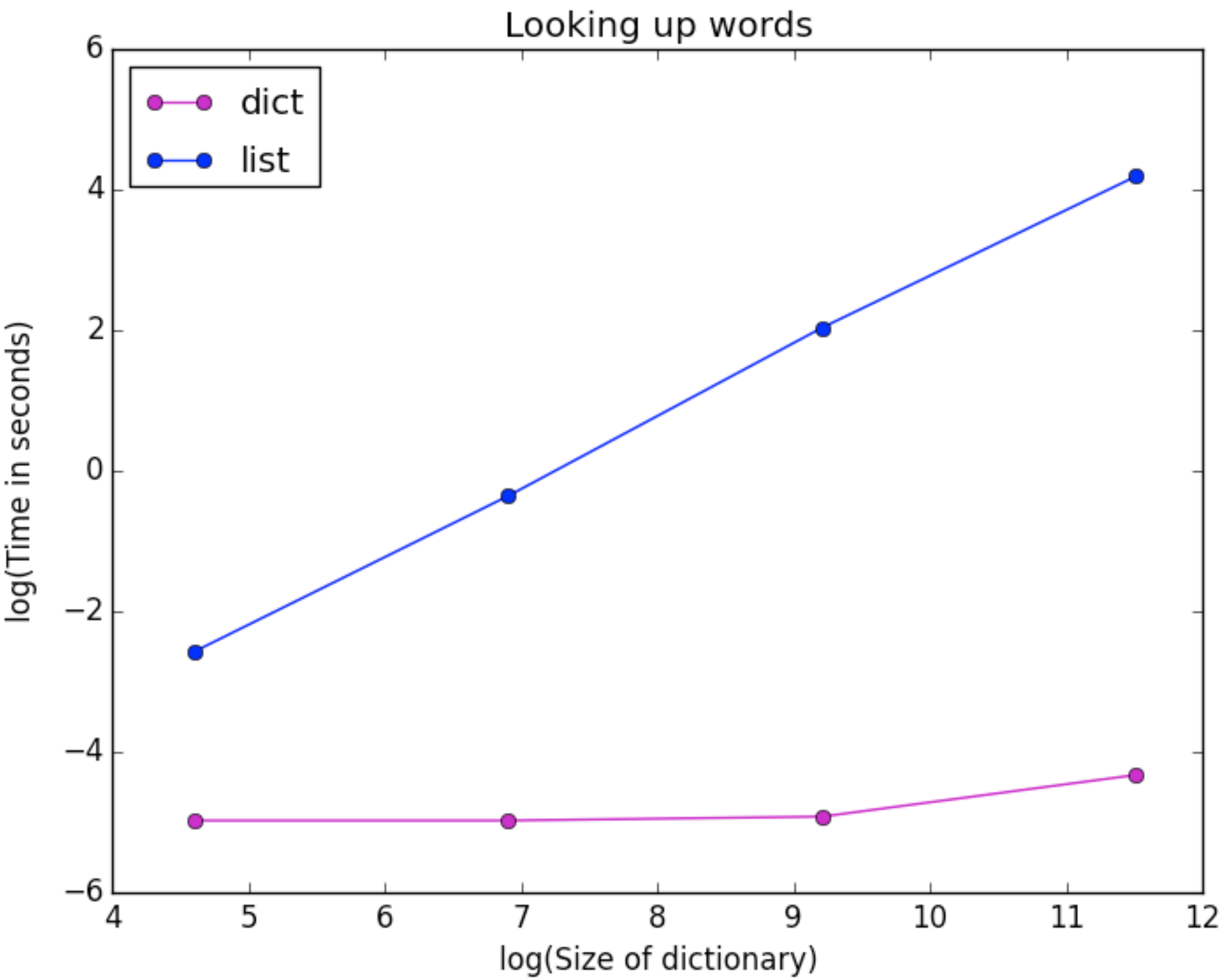
```
import time

word_dict = {}
for word in word_list:
    word_dict[word] = True

start = time.time()

counter = 0
for word in my_words:
    if word in word_dict:
        counter += 1
print counter

print "Elapsed time:", time.time() - start
```

Elapsed time: 0.0132



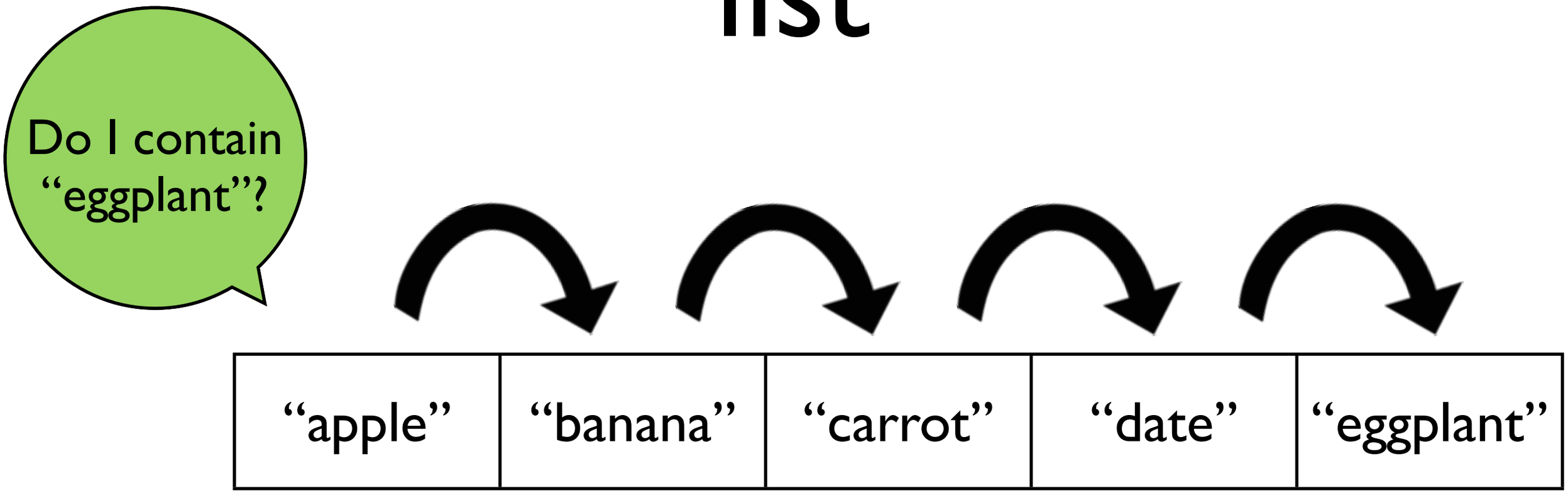


time	dict	list
100	0.0069	0.0764
1000	0.0069	0.6995
10000	0.0073	7.6593
100000	0.0132	65.9971



time	dict	list
100	0.0069	0.0764
1000	0.0069	0.6995
10000	0.0073	7.6593
100000	0.0132	65.9971

Looking up an item in a list



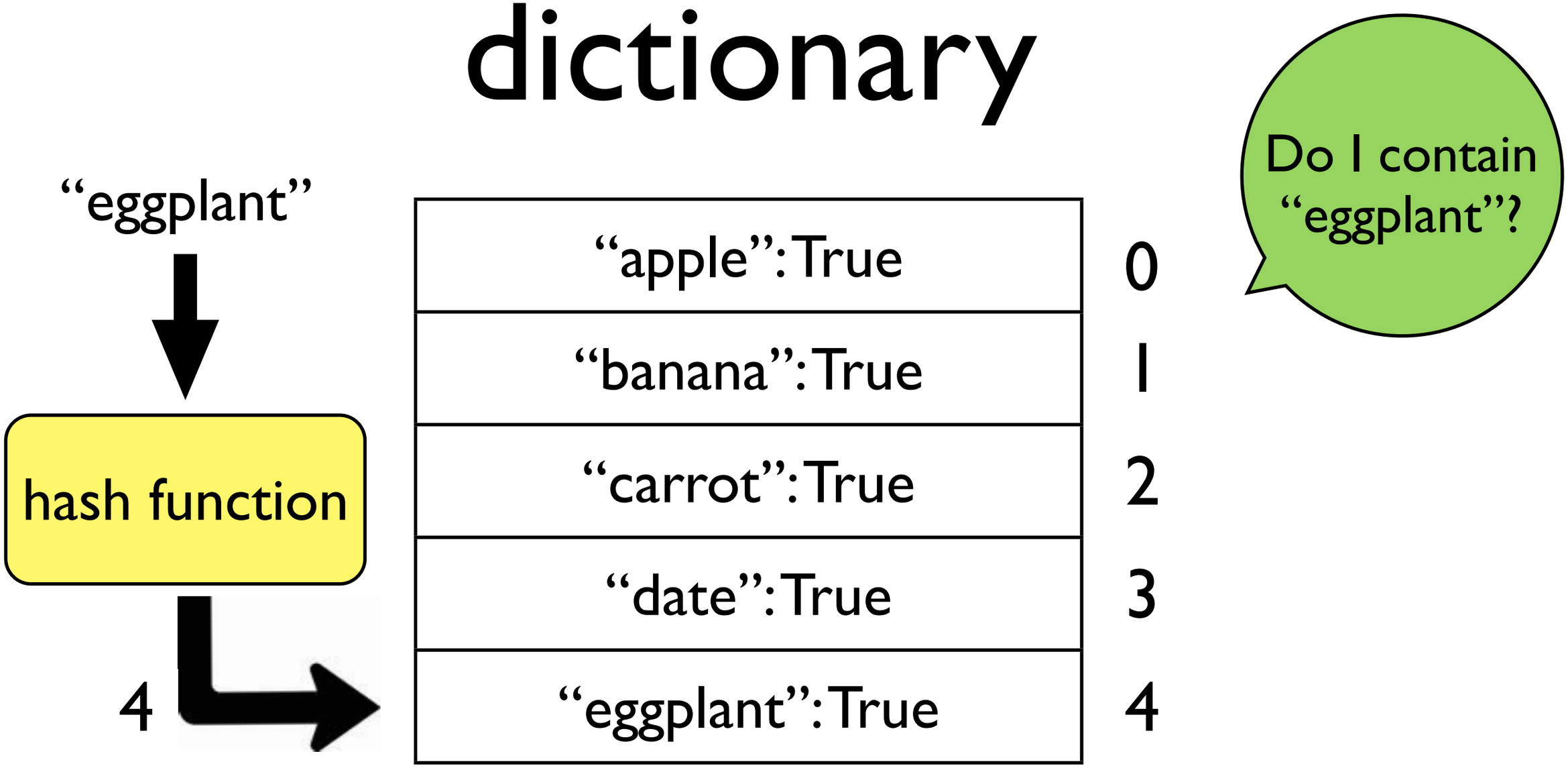
Looking up an item in a list

"apple"	"banana"	"carrot"	"date"	"eggplant"

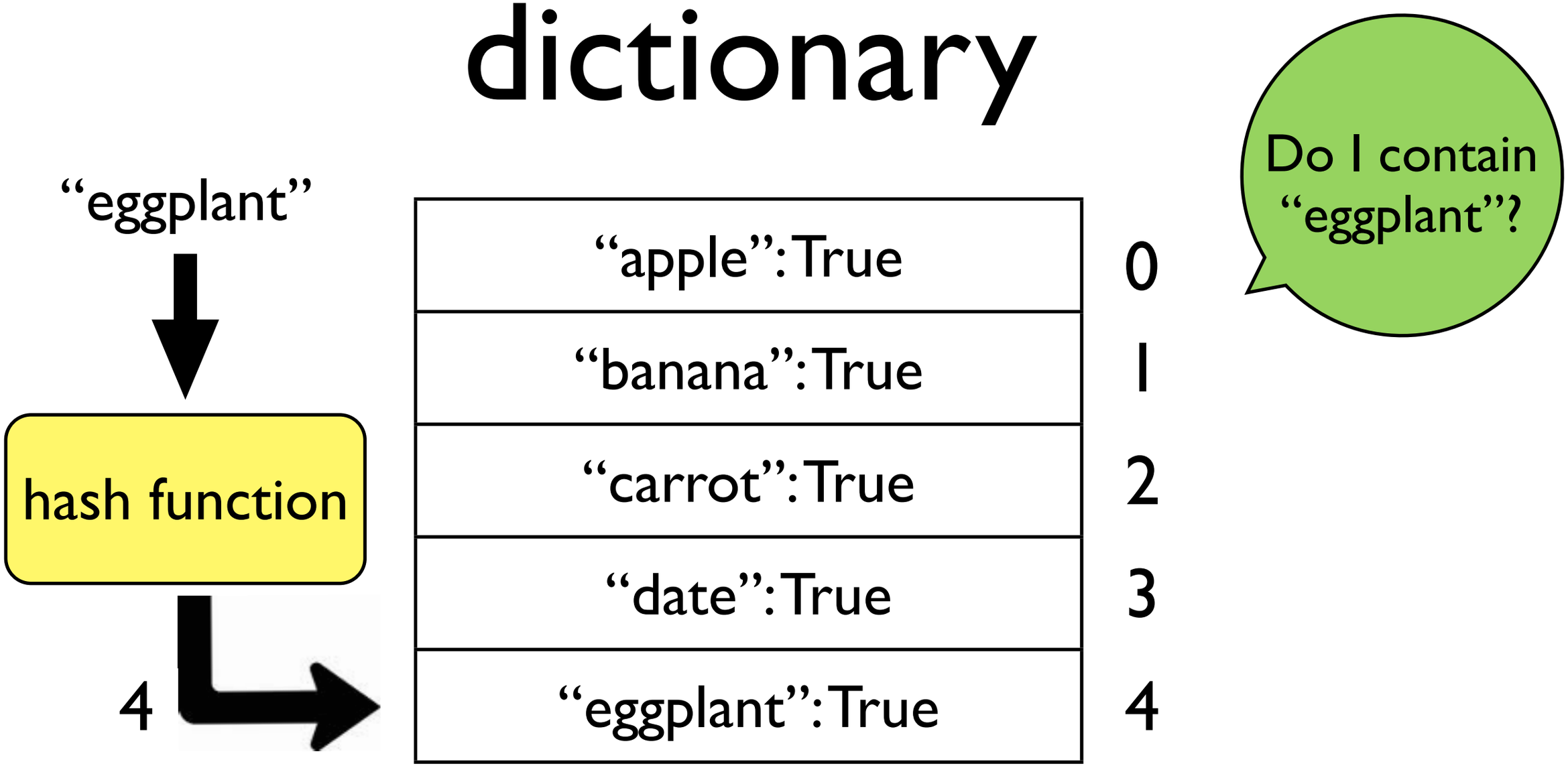
The amount of work we do is proportional to the length of the list (call it "n").

Finding an item in a list is **O(n)**

Looking up an item in a dictionary



Looking up an item in a dictionary



The amount of work we do is constant!

Finding an item in a dictionary is **O(1)**

Analyzing our word project: list case

```
for word in my_words:
    if word in word_list:
        counter += 1
```

"m" items in my_words

O(n) lookup

O(m * n) work

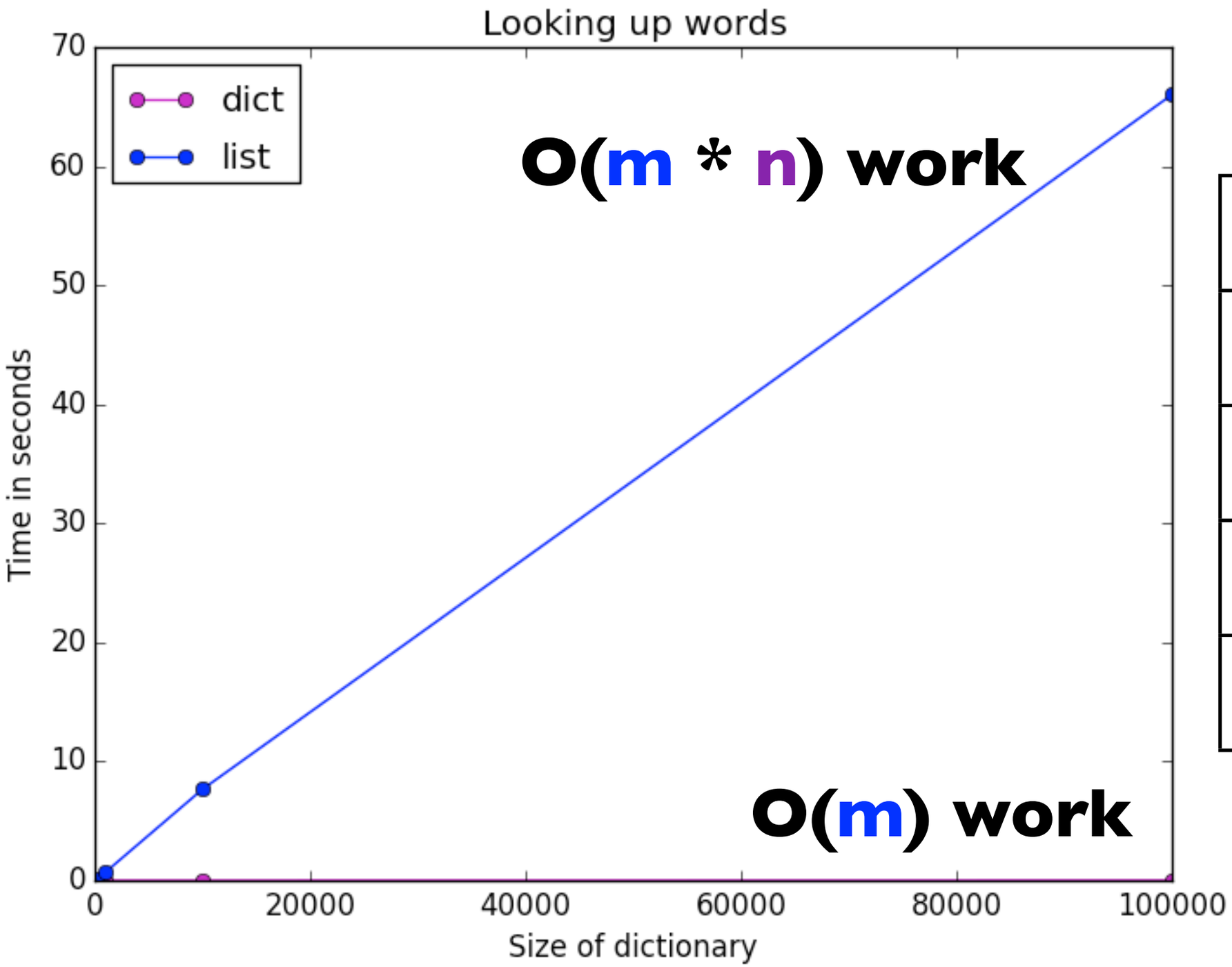
Analyzing our word project: dictionary case

```
for word in my_words:
    if word in word_dict:
        counter += 1
```

"m" items in my_words

O(1) lookup

O(m) work



time	dict	list
100	0.0069	0.0764
1000	0.0069	0.6995
10000	0.0073	7.6593
100000	0.0132	65.9971

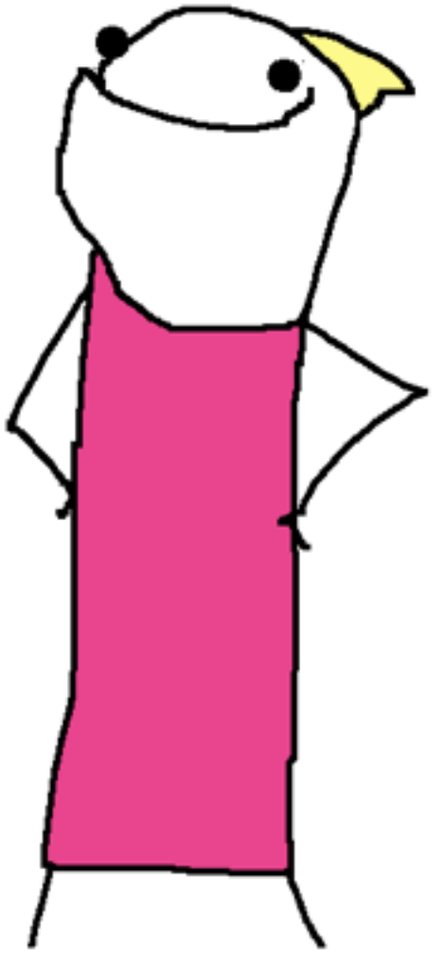
Costs of common operations

operation	list	dict
find	O(n)	O(1)
insert	O(n)	O(1)
append	O(1)	n/a
sort	O(n log n)	n/a

When choosing a data structure

1. What operations do I use a lot?

2. What data structure makes those operations fast?



Repeated lookups = dictionary

Sorting / caring about order = list

Intermediate investigations

- sets: like dictionaries, but for unique elements
- tuples: like lists, but immutable (saves space)
- numpy arrays
- Book: Think Complexity by Allen Downey
- Coursera: Analysis of Algorithms
- Coursera: Algorithms, Design and Analysis, Part II

Take-home points

- When picking a data structure and algorithm, ask:
    1. What operations do I use a lot?
    2. What data structure makes those operations fast?
- If the code is slow, do a basic run time analysis
- Time your code with different input sizes

Thank you!
Questions?

All images are courtesy of the webcomic Hyperbole and a Half: http://hyperboleandahalf.blogspot.com/