

RUNTIME COMPLEXITY

or "Big O Notation"

OR

Or, how **long does it take** my program to run
compared to **how much workload** I have?

(often (and in this presentation), workload == length of a list)

WHY IS THIS IMPORTANT?

- "A way to talk about talking about code"
-- Nick Audo
- A mathematical way to describe algorithm efficiency
- Interviews

YOU ALREADY KNOW HOW
TO DO THIS

EXAMPLE

Say we have this function:

```
def sum_nums( num_list ):  
    sum = 0  
    for num in num_list:  
        sum += num  
    return sum  
  
num_list = [1, 2, 3, 4, 5]
```

- Say each operation takes 1ms
- One operation for initialization of sum variable
- One operation for each addition
- Total operations is $1 + (1 * 5)$
- = 6 ms

EXAMPLE

Same function:

```
def sum_nums( num_list ):  
    sum = 0  
    for num in num_list:  
        sum += num  
    return sum
```

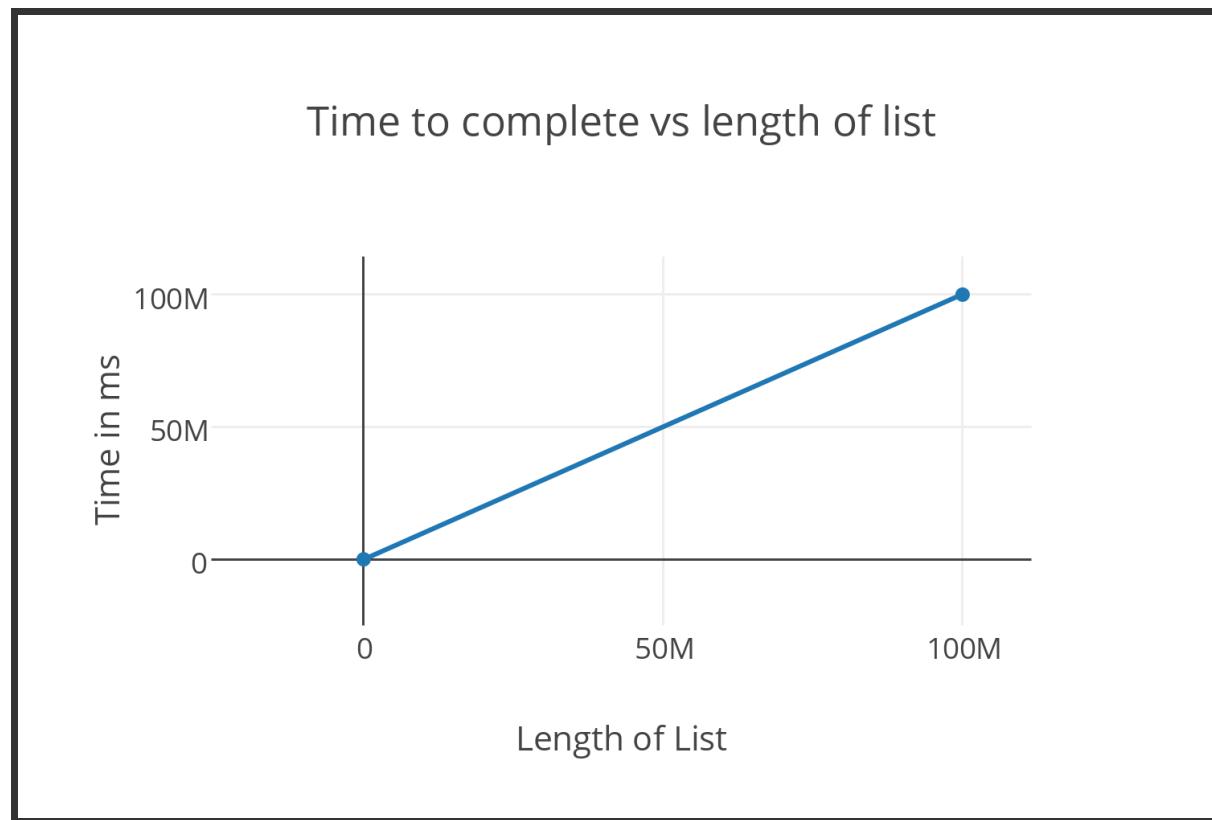
How long will this take?

- One operation for initialization of sum variable
- One operation for each addition
- Total operations is $1 + (1 * 1000000000)$
- = 1000000001 ms

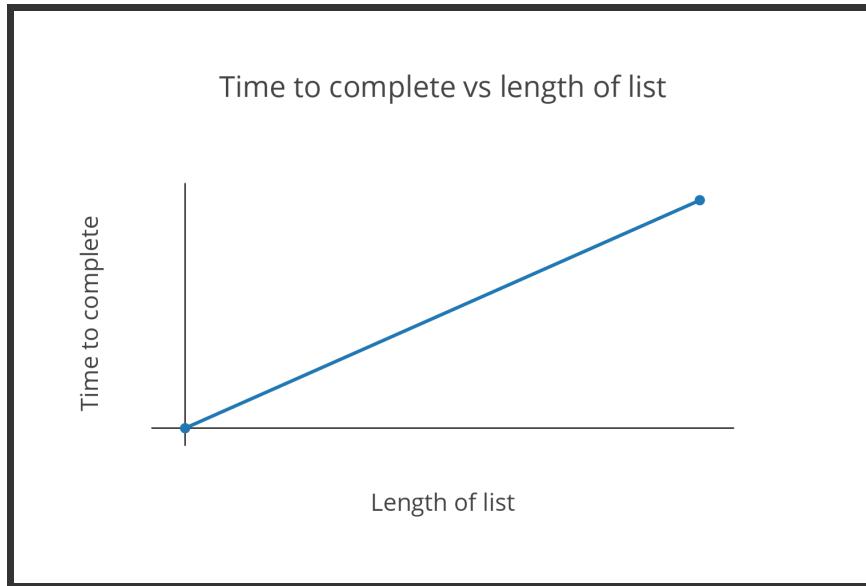
Different List

```
num_list = [1, 2, 3, ... , 1000000000  
]
```

GRAPH?



GRAPH?



It doesn't really matter how long each operation takes, it matters what the general shape of the graph is like.

$$1 + (1 * 5)$$

$$1 + (1 * 1000000000)$$

General formula: $1 + (1 * n)$

$$n + 1$$

This turns out to be linear

So if your list is `n` long,
it takes n time to do your calculations

This function is $O(n)$!

HOW DO I SAY IT?

You can say:

It has a linear runtime.

It runs linearly.

It runs in $O(n)$ (read: "oh of n") time.

Q: What is the runtime of your program? A: $O(n)$

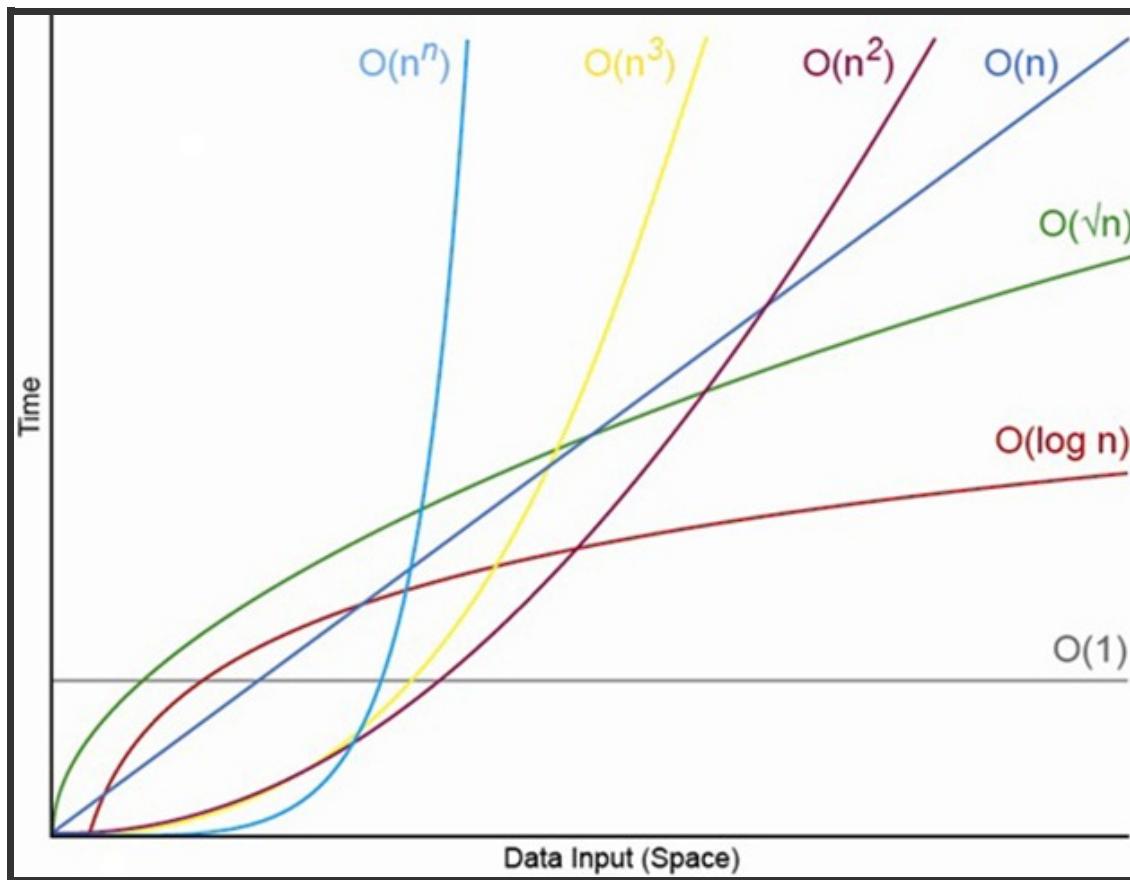
OTHERS?

Common runtimes:

- $O(n)$
- $O(n^2)$
- $O(1)$
- $O(n \log(n))$

GRAPHS OF COMMON RUN-TIMES

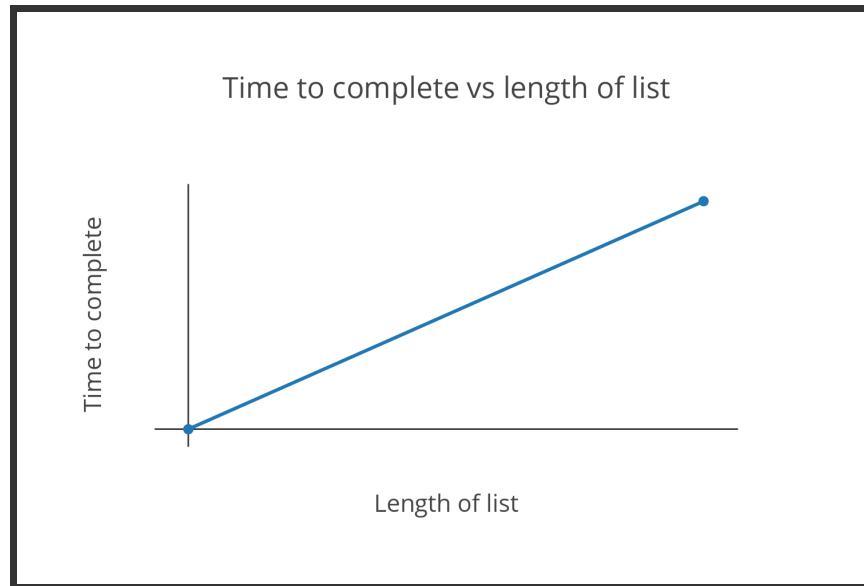
Flat slopes are good. Steep slopes are bad



WHEN/HOW DO THESE
RUNTIMES HAPPEN?

$O(n)$

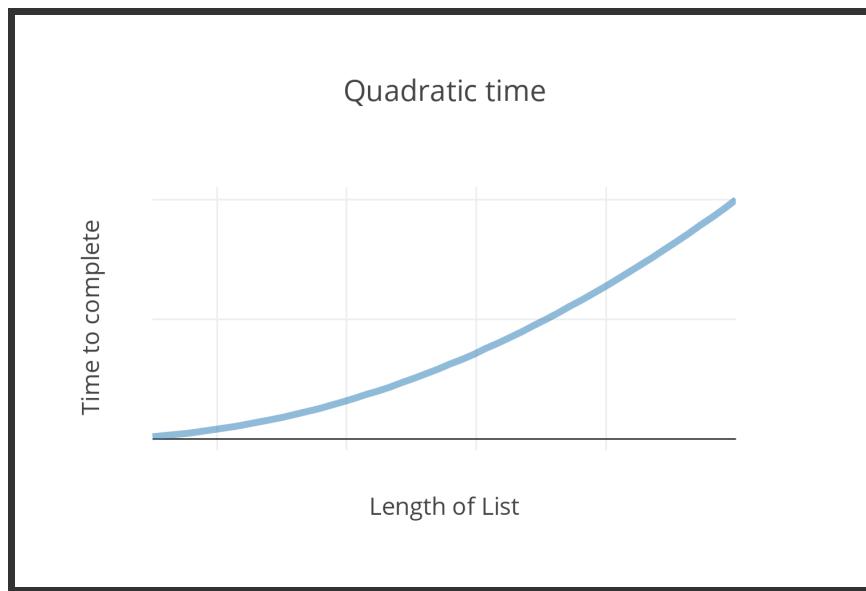
LINEAR TIME



- A single for loop

$O(n^2)$

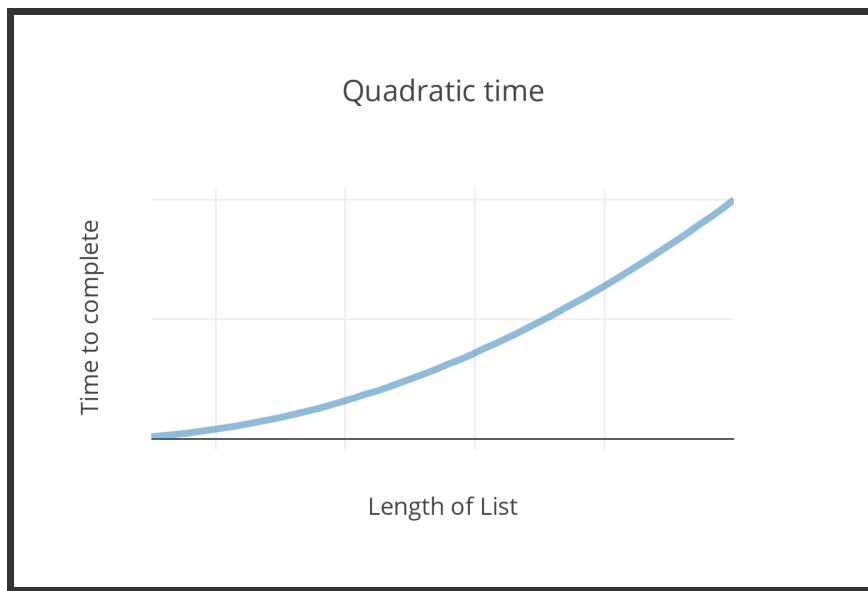
QUADRATIC TIME



- Two nested for loops
- Bubble sort
- Insertion sort

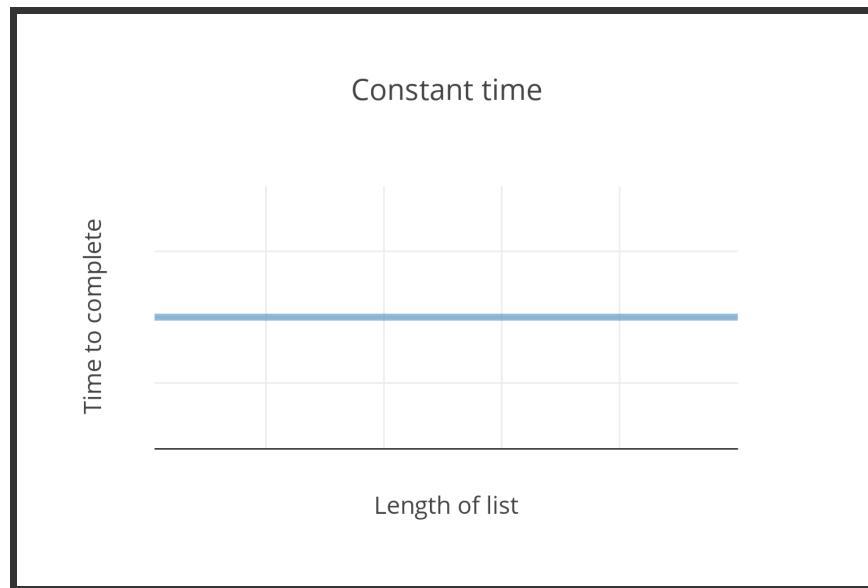
$O(n^2)$

QUADRATIC TIME



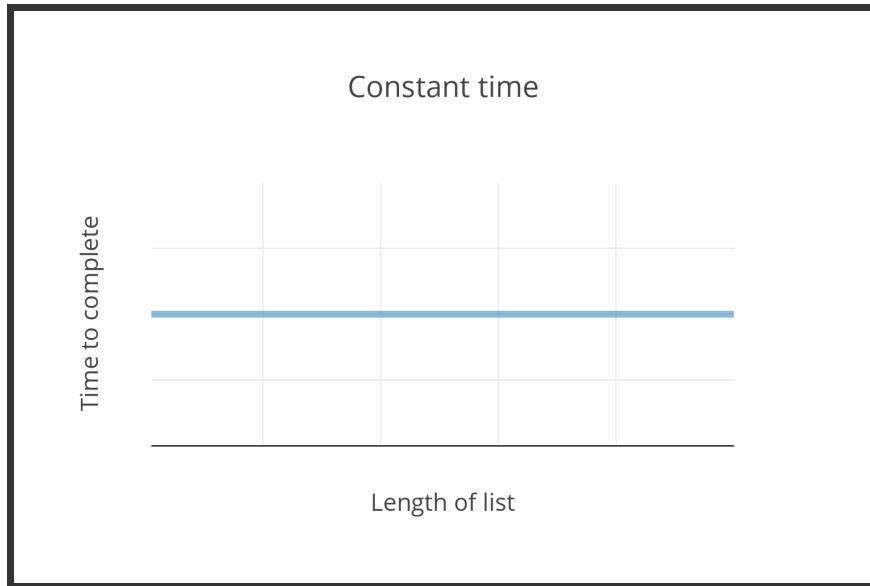
```
list_1 = [ 5, 48, 88, 24, 14 ]  
  
sum = 0  
for i in list_1:  
    for j in list_1:  
        sum += i * j
```

$O(1)$ CONSTANT TIME



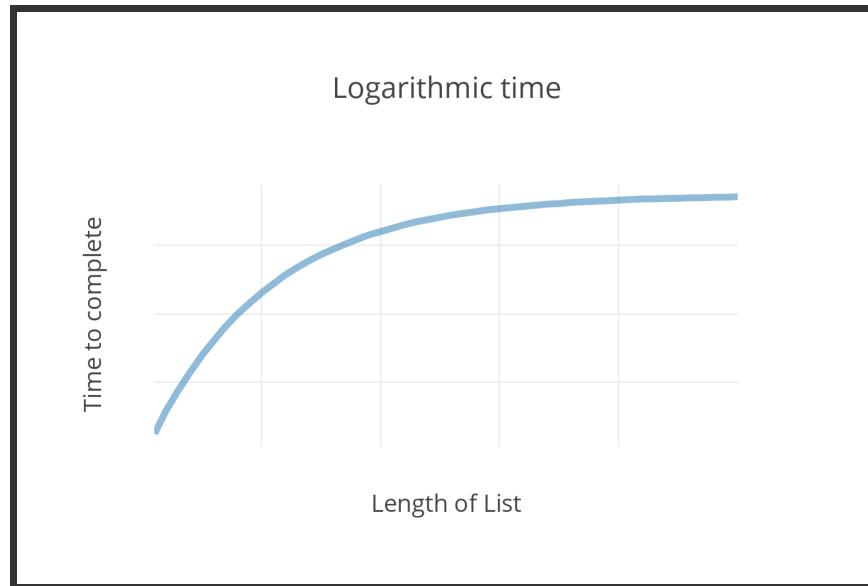
- Dictionary lookup
- Array indexing

$O(1)$ CONSTANT TIME



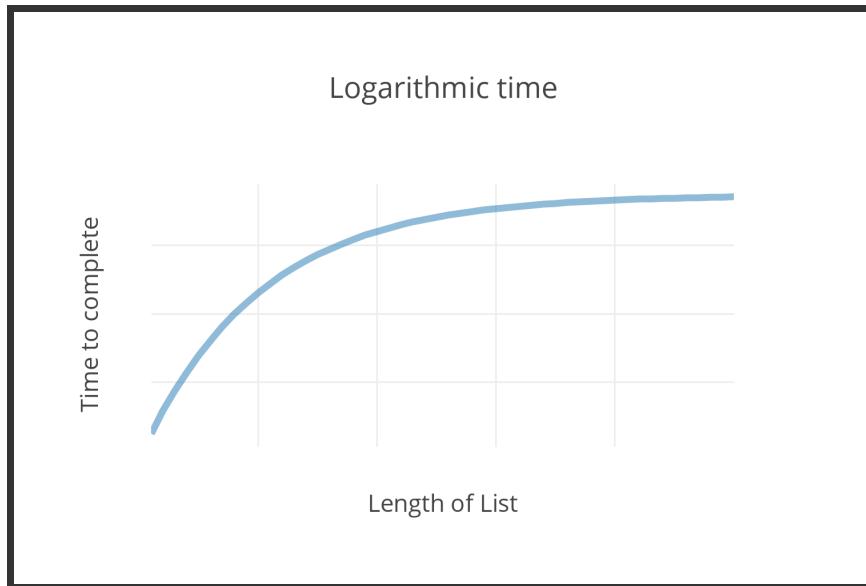
```
list_1 = [ 5, 48, 88, 24, 14 ]  
list_1[3]  
  
dict_1 = {"a": 5, "b": 6, "c": 7}  
dict_1[ "a" ]
```

$O(\log(n))$ OR $O(n \log(n))$ LOGARITHMIC TIME



- "divide and conquer"
- Binary Tree Search
- Quicksort, Mergesort

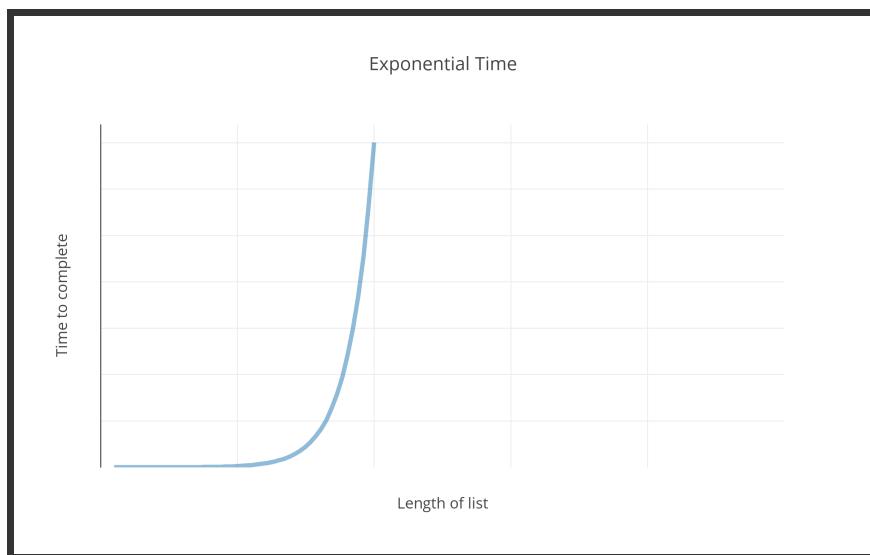
$O(\log(n))$ OR $O(n \log(n))$ LOGARITHMIC TIME



```
def find_num(min, max):
    target = random.randint(1, max)
    found = False
    while not found:
        guess = min + ((max - min) / 2)
        print "guessing %d" % guess
        if guess == target:
            found = True
        elif guess > target:
            max = guess
        else:
            min = guess
    print "The number is %d" % guess
find_num(1, 100)
```

SOME LESS COMMON BUT
POSSIBLE RUNTIMES:

$O(n^n)$ OR $O(2^n)$ EXPONENTIAL TIME



Password Cracking

Say there's 72 valid chars for a pw

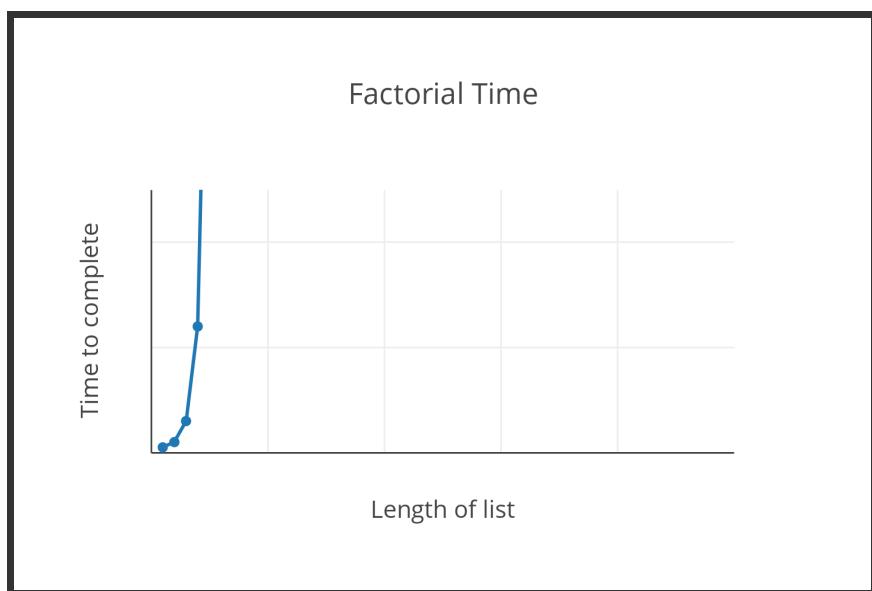
If your pw is 1 char there's 72 possibilities

For 4 chars, that's
 $72 * 72 * 72 * 72$ possibilities
 $72^4 \Rightarrow 26873856$

For 10 chars,
 $72^{10} \Rightarrow 3743906242624487424$

For 20,
 $72^{20} \Rightarrow 140168339535626072939181857587$
34155776

$O(n!)$ FACTORIAL TIME



Traveling Salesman (brute force)

To find a path that hits each node once and only once:

Must start with one node, say A
[A ...]

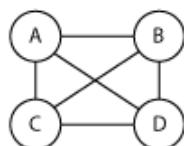
Now there are 3 left, choose 1
[A, C ...]

Now there are 2 left, choose 1
[A, C, D ...]

Now there's only 1 left,
[A, C, D, B]

$$4 * 3 * 2 * 1 = 4! = 24$$

There are 24 possible routes
to hit every node once



SUM TO ZERO

Problem: write a function that returns true if two numbers in an array sum to zero

```
list_1 = [5, 9, 10, -2, 10, -5]
```

Easiest way?

```
def sum_zero(l):
    for i in l:
        for j in l:
            if (i + j) == 0:
                return True
    return False
```

Runtime?

$O(n^2)$

SUM TO ZERO

Problem: write a function that returns true if two numbers in an array sum to zero

```
list_1 = [5, 9, 10, -2, 10, -5]
```

Efficient way.

```
def sum_zero(l):
    num_dict = {}
    for i in l:
        num_dict[i] = True

    for i in l:
        if num_dict.get(-i) == True:
            return True

    return False
```

Runtime?

$O(2n) \Rightarrow O(n)$

PRACTICE

- Ex: 5 (Letter count)
(<https://github.com/hackbrightacademy/Hackbright-Curriculum/tree/master/Exercise05>)
 - $O(n \times m)$ solution (where m is the length of another list)
 - $O(n)$ solution (simplified from $n + m$)
 - Can you implement both?
- Skills 1
(<https://github.com/hackbrightacademy/Hackbright-Curriculum/blob/master/skills1/skills1.py>) and Skills 2
(<https://github.com/hackbrightacademy/Hackbright-Curriculum/blob/master/skills2/skills2.py>)
 - All can be solved in $O(n)$

MORE RESOURCES

- Big-O cheat sheet (<http://bigocheatsheet.com/>)
- Tim Roughgarden's lectures (Asymptotic Analysis) (this is still heavy but still rigorous, ya'll)
(<https://class.coursera.org/algo-004/lecture>)