How long does a function take to run?

- It depends on what computer it is run on!

Assumptions
- We want something independent of the speed of the computer
- We typically use \( N \) which is some reference to the "size" of the data.
  - It might be the variable \( N \) (in factorial)
  - It might be the size of your sentence
- We don’t care about constant factors
  - This is pretty silly in some cases but makes the math more beautiful
- We typically think about the worst case!
- It only matters for BIG input!

Colleen’s Morning Routine
- Eat breakfast (10 minutes)
- Brush teeth (5 minutes)
- Go swimming
- Read Piazza (0 minutes – ???)

Takes a consistent or “constant” amount of time
Depends “linearly” upon the number of laps \( L \)
Depends upon the number of posts \( P \) and the words per post \( W \)

Constant Runtime
- The runtime doesn’t depend upon the “size” of the input

\[
\text{(define (square x)} \quad (* x x))
\]
\[
\text{(define (average a b)} \quad (/ (+ a b) 2))
\]
\[
\text{(define (max val1 val2)} \quad (if (> val1 val2) val1 val2)
\]

\( O(1) \)
Write a constant runtime procedure? (Brushing your teeth)

Linear Runtime

• The runtime DOES depend upon the “size” of the input – but just with a linear relationship

```
(define (sum-up-to x)
  (if (< x 0)
      0
      (+ x (sum-up-to (- x 1)))))
```

```
(define (count sent)
  (if (empty? sent)
      sent
      (+ 1 (count (bf sent)))))
```

Write a linear runtime procedure? (Swimming laps)

What is the runtime of this?

```
(define (factorial n)
  (if (= n 0)
      1
      (* n (factorial (- n 1)))))
```

A) Constant runtime  
B) Linear runtime  
C) Linear*Linear (quadratic)  
D) Something else  
E) No clue!

Problem Solving Tip!

• THINK: Does this depend upon the “size” of the input?  
  – Constant vs. Not Constant

• Think about this for ANY function that is called!
Linear * Linear (Quadratic) Runtime

\[ O(N^2) \]

**Linear * Linear Runtime (quadratic)**
(Reading Piazza)

- The runtime DOES depend upon the “size” of the first input
- For each thing in the first input, we call another linear thing

```scheme
(define (factorial-sent sent)
  (if (empty? sent)
      sent
      (se (factorial-sent (first sent))
          (factorial-sent (bf sent)))))
```

Runtime:
\[ O(\text{sent-length} \times \text{value-of-elements}) \]

**Write a linear*linear (quadratic) runtime procedure?**
(Reading Piazza)

**Bad Variable Names**
(Slight break from runtimes)
What is the runtime of this?

(define (mystery a b)
  (cond
    ((empty? b) (se a b))
    ((< a (first b)) (se a b))
    (else (se (first b)
              (mystery a (bf b))))))

A) Constant runtime
B) Linear runtime
C) Linear*Linear (quadratic)
D) Something else
E) No clue

Bad variable names!

- Make your code harder to read!
  - Don’t do this!
- Even though we do
  - We will use bad variable names on exams so that you have to read/understand the code

Better variable names!

(define (insert num sent)
  (cond
    ((empty? sent) (se num sent))
    ((< num (first sent)) (se num sent))
    (else (se (first sent)
              (insert num (bf sent))))))

STK>(insert 7 '())
()  
STK>(insert 1 '(2 5 8))
(1 2 5 8)

Insert

STK>(insert 8 '(2 5 6 9))
(2 5 6 8 9)

Best vs. Worst Case

(define (insert num sent)
  (cond
    ((empty? sent) (se num sent))
    ((< num (first sent)) (se num sent))
    (else (se (first sent)
              (insert num (bf sent))))))

STK>(insert 1 '(2 3 4 5 6 7 8))
(1 2 3 4 5 6 7 8)  
STK>(insert 9 '(2 3 4 5 6 7 8))
(2 3 4 5 6 7 8 9)

What is the runtime of this?

(define (sort sent)
  (if (empty? sent)
      sent
      (insert (first sent)
              (sort (bf sent)))))

A) Constant runtime
B) Linear runtime
C) Linear*Linear (quadratic)
D) Something else
E) No clue
Sort

\[
STk> \text{(sort '(8 3 9))}
\]
\[
(3 8 9)
\]

\[
\text{(sort '(8 3 9))}
\]
\[
\text{(sort '(3 9))}
\]
\[
\text{(insert 3)}
\]
\[
\text{(sort '(9))}
\]
\[
\text{(insert 9)}
\]
\[
\text{()}\]
\[
\text{()}\]
\[
\text{(insert 8 (insert 3 (insert 9 '())))}
\]

\[
\text{(sort sent) if sent length is N}
\]
\[
1 + 2 + 3 + 4 + \ldots + (N-1) + N
\]

Some Algebra to calculate:

\[
\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + (n-2) + (n-1) + n
\]

\[
\sum_{i=1}^{n} i = n + (n-1) + (n-2) + \ldots + 3 + 2 + 1
\]

How long does it take you to read Piazza posts and check your email and shower?

- \( P \) is the number of posts
- \( \bar{W}_p \) is the number of words per post \( P \)
- \( E \) is the number of emails
- \( \bar{W}_e \) is the number of words per email \( E \)
- \( S \) is the number of minutes you shower

\[
a) \ P \bar{W}_p \cdot E \bar{W}_e \cdot S
\]
\[
b) \ P + \bar{W}_p + E + \bar{W}_e + S
\]
\[
c) \ P \bar{W}_p + E \bar{W}_e + S
\]
\[
d) \ P + E + S
\]
\[
e) \ P \cdot E \cdot S
\]
### Exponential Runtime

\[ 2^n \]

### Logarithmic Runtime

\[ \log_2(N) \]

### Number Guessing Game

- I’m thinking of a number between 1 and 100
- How many possible guesses could it take you? (WORST CASE)

- Between 1 and 10000000?
- How many possible guesses could it take you? (WORST CASE)
Take the log of both sides:

\[ \log_2(n) = \log_2(2^{h+1} - 1) \]

// Remember:

\[ \log_b m^n = n \cdot \log_b m \]

\[ \log_2(n) \approx (h + 1)(\log_2 2) \]

\[ \log_2 n \approx h \]

---

**Log\(_2\)(N)**

- When we’re able to keep dividing the problem in half (or thirds etc.)
- Looking through a phone book

---

**Asymptotic Cost**

- We want to express the speed of an algorithm *independently* of a specific implementation on a specific machine.

- We examine the cost of the algorithms for large input sets i.e. *the asymptotic cost*.

- In later classes (CS70/CS170) you’ll do this in more detail

---

**Formal definition**

\[ T(n) \in O(f(n)) \]

if and only if

\[ T(n) \leq c \cdot f(n) \]

for all \( n > N \)
Which is fastest after some big value $N$?

$\text{time} = \sqrt{n} + 50$

$\text{time} = (0.14n)^2$

$\text{time} = n + 15$

<table>
<thead>
<tr>
<th>Important Big-Oh Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
<tr>
<td>$O(1)$</td>
</tr>
<tr>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$O(\log^2 n)$</td>
</tr>
<tr>
<td>$O(\sqrt{n})$</td>
</tr>
<tr>
<td>$O(n)$</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>$O(n^3)$</td>
</tr>
<tr>
<td>$O(n^4)$</td>
</tr>
<tr>
<td>$O(2^n)$</td>
</tr>
<tr>
<td>$O(e^n)$</td>
</tr>
</tbody>
</table>

Which is fastest after some value $N$?

$\text{time} = \sqrt{n} - 1000$

$\text{time} = 0.75\sqrt{n} + 50$

$\text{time} = \sqrt{n}$

WAIT – who cares?

These are all proportional!

Sometimes we do care, but for simplicity we ignore constants.

Formal definition

$T(n) \in O(f(n))$

if and only if

$T(n) \leq c \times f(n)$

for all $n > N$

Simplifying stuff is important

$f(n) \in O(5n^3 + 10n^2 + 1000n)$

$T(n) \in O(n^3)$

Write sum-up-to to generate an iterative process!

(define (sum-up-to x)
  (define (sum-up-to-iter x answer)
    (if (= x 0)
        answer
        (sum-up-to-iter (- x 1)
            (+ answer x))))
  (sum-up-to-iter x 0))