Checking in

Roughly how much time are you spending on your CS 42 assignment each week?

(Your response)
(define (remove e L)
  (if (empty? L)
      empty
      (let* ([it (first L)]
             [lose-it (rest L)]
             [lose-it-solution (remove e lose-it)]
             [use-it-solution (cons it (remove e lose-it))])
      (if (equal? e it)
          lose-it-solution
          use-it-solution))))
Base case(s): use-it solution

F( )

“it” solution

“lose it” solution

use-it solution

lose-it solution

solution
Base case(s):

$F(\quad )$

“it” $\rightarrow$ “lose it”

use-it solution

lose-it solution

solution
Base case(s):

\[ F( \quad ) \]

- "it" solution
- "lose it" solution

use-it solution

lose-it solution

solution
Base case(s):

F( use-it solution )

“it” “lose it”

use-it solution

lose-it solution

solution
Base case(s):

F(                           )

“it”  “lose it”

use-it solution

lose-it solution

solution
How “good” are these solutions?

Are they efficient?

Do they “cost” more than they should?
Interpreting a theoretical model

Key take-away: it’s lossy!

A theory abstracts away certain details.

**cost metric:**
- corresponds to one “step”
- highlights the essence of the work
  - e.g., multiplications, comparisons, function calls…
- serves as a proxy for an empirical measurement

Instead of measuring time, we count steps.

e.g., “This algorithm costs $n^2$ multiplications.”
Asymptotic Analysis
(Big O)
Asymptotic analysis

We’re always answering the same question:

How does the cost **scale** (when we try larger and larger inputs)?

**Not:**

- Exactly how many steps will it execute?
- How many seconds will it take?
- How many megabytes of memory will it need?
The informal definition of “Big O”

A *reasonable* upper bound on (an abstraction of) a problem’s difficulty or a solution’s performance, for *reasonably* large input sizes.
In the limit (for VERY LARGE inputs)

The running time is bounded regardless of the input size. \[ O(1) \]

An input twice as big takes no more than twice as long. \[ O(n) \]

An input twice as big takes no more than four times as long. \[ O(n^2) \]

An input one bigger takes no more than twice as long. \[ O(2^n) \]
If We Only Care About Scalability...

What are the consequences?

Constant factors can be ignored.
\[ n \quad \text{and} \quad 6n \quad \text{and} \quad 200n \] scale identically ("linearly")

Small summands can be ignored.
\[ n^2 \quad \text{and} \quad n^2 + n + 999999 \] are indistinguishable when \( n \) is huge.
Grouping Algorithms by Scalability

- **O(1)**
  - Takes 6 steps
  - Takes 1 (big) step
  - No more than 4000 steps
  - Somewhere between 2 and 47 steps, depending on the input

- **O(n)**
  - Takes 100n + 3 steps
  - Takes n/20 + 10,000,000 steps
  - Anywhere between 3 and 68 steps per item, for n items.

- **O(n^2)**
  - Takes 2n^2 + 100n + 3 steps
  - Takes n^2/17 steps
  - Somewhere between 1 and 40 steps per item, for n^2 items
  - Anywhere between 1 and 7n steps per item, for n items.
How hard is the problem?

- \(O(n^n)\)
- \(O(n!)\)
- \(O(2^n)\)
- \(O(n^3)\)
- \(O(n^2)\)
- \(O(n \log(n))\)
- \(O(n)\)
- \(O(\sqrt{n})\)
- \(O(\log(n))\)
- \(O(1)\)

**Intractable problems (exponential)**

**Tractable problems (polynomial)**

**No problem!**
logs aren’t scary!

They’re our friends.

\[ \log_2 N = p \iff 2^p = N \]

log is the inverse of exponentiation.

How many times can I cut N in half?

Can I avoid looking at *all* the input?!
How hard are these problems?

<table>
<thead>
<tr>
<th>Cost Metric</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>double</strong></td>
<td>multiplications</td>
</tr>
<tr>
<td>multiply a number by 2</td>
<td></td>
</tr>
<tr>
<td><strong>sum</strong></td>
<td>additions</td>
</tr>
<tr>
<td>sum a list of numbers</td>
<td></td>
</tr>
<tr>
<td><strong>half-count</strong></td>
<td>divisions</td>
</tr>
<tr>
<td>divide a positive number by 2 until you get 1</td>
<td></td>
</tr>
</tbody>
</table>
How hard are these problems?

<table>
<thead>
<tr>
<th>cost metric</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>double</strong></td>
<td></td>
</tr>
<tr>
<td>multiply a number by 2</td>
<td>multiplications</td>
</tr>
<tr>
<td><strong>sum</strong></td>
<td></td>
</tr>
<tr>
<td>sum a list of numbers</td>
<td>additions</td>
</tr>
<tr>
<td><strong>half-count</strong></td>
<td></td>
</tr>
<tr>
<td>divide a positive number by 2 until you get 1</td>
<td>divisions</td>
</tr>
</tbody>
</table>
What's the cost, $T$, for each solution?

(double n) = $n \times 2$

(sum n) = $\begin{cases} 0 & \text{if } n = 0 \\ n + (\text{sum } (n - 1)) & \text{otherwise} \end{cases}$

(half-count n) = $\begin{cases} 0 & \text{if } n = 1 \\ 1 + (\text{half-count } (\text{quotient } n \div 2)) & \text{otherwise} \end{cases}$

<table>
<thead>
<tr>
<th>input size</th>
<th>double multiplications</th>
<th>sum additions</th>
<th>half-count divisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>T(0)</td>
<td></td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>T(1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T(2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T(3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T(4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T(n)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What’s the cost, $T$, for each solution?

<table>
<thead>
<tr>
<th>input size</th>
<th>double multiplications</th>
<th>sum additions</th>
<th>half-count divisions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(0)$</td>
<td>1</td>
<td>0</td>
<td>n/a</td>
</tr>
<tr>
<td>$T(1)$</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$T(2)$</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$T(3)$</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$T(4)$</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$T(n)$</td>
<td>1</td>
<td>$n$</td>
<td>$\lfloor \log_2 n \rfloor$</td>
</tr>
</tbody>
</table>

Can we prove it?

- (define (double n) (* n 2))
- (define (sum n) (if (= n 0) 0 (+ n (sum (- n 1)))))
- (define (half-count n) (if (= n 1) 0 (+ 1 (half-count (quotient n 2)))))
Recurrence Relations
(translating code to math)
Translating recursion to recurrence relations

For a given cost metric: additions

1. Translate the base case(s), using specific input sizes
   How many steps does this base case take?

2. Translate the recursive case(s), using input size N
   Define T(N) recursively, in terms of smaller cost.

\[ \text{(define \ (sum \ n)} \]
\[ \text{\quad (if \ (= \ n \ 0)} \]
\[ \text{\quad \quad \quad 0} \]
\[ \text{\quad (} + \ n \ (\text{sum \ (- \ n \ 1)})))) \]
Translating recursion to recurrence relations

For a given cost metric: **additions**

1. Translate the base case(s), using specific **input sizes**
   How many steps does this base case take?

2. Translate the recursive case(s), using **input size N**
   Define $T(N)$ recursively, in terms of smaller cost.

$$
T(N) = 1 + T(N-1)
$$

$$
T(N) = 1 + 1 + T(N-2)
$$

$$
T(N) = 1 + 1 + 1 + T(N-3)
$$

$$
\vdots
$$

$$
T(N) = 1 + 1 + 1 + \ldots 1 + T(N-N)
$$

Define $T(0) = 0$ and $T(N) = 1 + T(N-1)$.

The closed form is:

$$
T(N) = \sum_{i=0}^{N-1} 1 = N
$$

Asymptotic form:

$$
N \in \mathcal{O}(N)
$$

### Recurrence Relation

**Base Case:**

$$
T(0) = 0
$$

**Recursive Case:**

$$
T(N) = 1 + T(N-1)
$$

**Recurrence Relation:**

$$
T(N) = 1*1 + T(N-1) = 2*1 + T(N-2) = 3*1 + T(N-3) = \ldots = N*1 + T(N-N) = N \in \mathcal{O}(N)
$$
Translating recursion to recurrence relations

For a given cost metric: arithmetic operations and comparisons

1. Translate the base case(s), using specific input sizes
   How many steps does this base case take?

2. Translate the recursive case(s), using input size N
   Define T(N) recursively, in terms of smaller cost.

(define (sum n)
  (if (= n 0)
    0
    (+ n (sum (- n 1))))

recurrence relation

\[
T(0) = 1 \quad \Rightarrow \quad \text{input size}
\]

\[
T(N) = 2 + T(N-1)
\]
Translating recursion to recurrence relations

For a given cost metric: arithmetic operations and comparisons

1. Translate the base case(s), using specific input sizes
   How many steps does this base case take?

2. Translate the recursive case(s), using input size N
   Define T(N) recursively, in terms of smaller cost.

\[
T(N) = 3 + T(N-1) \\
= 3 + 3 + T(N-2) \\
= 3 + 3 + 3 + T(N-3) \\
... \\
= 3 + 3 + 3 + ... 3 + T(N-N) \\
\]

Base case \( T(0) = 1 \)

\[
T(N) = 3 + T(N-1) \\
= 1 \times 3 + T(N-1) \\
= 2 \times 3 + T(N-2) \\
= 3 \times 3 + T(N-3) \\
... \\
= N \times 3 + T(N-N) = 3N + 1 \in O(N)
\]
Translating recursion to recurrence relations

For a given cost metric: divisions

1. Translate the base case(s), using specific input sizes
   How many steps does this base case take?

2. Translate the recursive case(s), using input size N
   Define $T(N)$ recursively, in terms of smaller cost.

```
(define (half-count n)
  (if (= n 1)
      0
      (+ 1 (half-count (quotient n 2))))
```

**base case** → $T(1) = 1$

**recursive case** → $T(N) = 3 + T(N-1)$
Translating recursion to recurrence relations

For a given cost metric: *divisions*

1. Translate the base case(s), using specific input sizes
   How many steps does this base case take?

2. Translate the recursive case(s), using input size $N$
   Define $T(N)$ recursively, in terms of smaller cost.

\[
\text{(define (half-count n)} \quad \begin{array}{l}
\text{base case} \Rightarrow \quad T(1) = 0 \\
\text{recursive case} \Rightarrow \quad T(N) = 1 + T(N/2)
\end{array}
\]

\[
T(N) = 1 + T(N/2) = 1 + 1 + T(N/4) = 1 + 1 + 1 + T(N/8) = \ldots = 1 + 1 + 1 + \ldots 1 + T(N/N) = \log_2 N + T(N/N) = \log_2 N \in O(\log N)
\]

\[
\]
## How hard are these problems?

<table>
<thead>
<tr>
<th>cost metric</th>
<th>work to do</th>
<th>predicted cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove</td>
<td>number of comparisons made</td>
<td>visit every element</td>
</tr>
<tr>
<td>uniq</td>
<td>number of comparisons made</td>
<td>compare each element to all the other elements</td>
</tr>
<tr>
<td>sublists</td>
<td>number of sublists created</td>
<td>construct $2^n$ lists</td>
</tr>
</tbody>
</table>
The cost of \texttt{remove} measured in \texttt{list-element} comparisons

\begin{verbatim}
(define (remove e L)
  (if (empty? L)
      empty
      (let* ([it (first L)]
              [lose-it (rest L)]
              [lose-it-solution (remove e lose-it)]
              [use-it-solution (cons it lose-it-solution)])
        (if (equal? e it)
            lose-it-solution
            use-it-solution))))
\end{verbatim}

\[T(0) = 0\]
\[T(N) = 1 + T(N-1)\]
\[= 1 + 1 + T(N-2)\]
\[\in O(N)\]
The cost of uniq measured in list-element comparisons

\[
(\text{define} \ (\text{uniq} \ L) \\
\quad (\text{if} \ (\text{empty} \ ? \ L) \\
\quad \quad '()) \\\n\quad (\text{let}^* \ ([\it\ (\text{first} \ L)] \\
\quad \quad [\text{lose-it} \ (\text{rest} \ L)] \\
\quad \quad [\text{lose-it-soln} \ (\text{uniq} \ \text{lose-it})] \\
\quad \quad [\text{use-it-soln} \ (\text{cons} \ \it \ \text{lose-it-soln})]) \\
\quad \quad (\text{if} \ (\text{member} \ \it \ \text{lose-it-soln}) \\
\quad \quad \text{lose-it-soln} \\
\quad \quad \text{use-it-soln}))))
\]

\[
T(0) = 0 \\
T(N) = N-1 + T(N-1)
\]

\[
T(N) = N-1 + N-2 + T(N-2) \in O(N^2)
\]
The cost of sublists
measured in number of sublists created, i.e., calls to \texttt{cons} and \texttt{empty}

\begin{align*}
T(0) & = 1 \\
T(N) & = 2^{N-1} + 2^{N-1} + T(N-1) \\
& = 2^N + T(N-1)
\end{align*}

\[ T(N) \in O(2^N) \]

\begin{verbatim}
(define (sublists L)
  (if (empty? L)
      (list empty)
      (let* ([it (first L)]
              [lose-it (rest L)]
              [lose-it-soln (sublists lose-it)]
              [use-it-soln (map (lambda (l) (cons it l)) lose-it-soln)])
        (append use-it-soln lose-it-soln))))
\end{verbatim}
<table>
<thead>
<tr>
<th></th>
<th>cost metric</th>
<th>work to do</th>
<th>predicted cost</th>
<th>UloLI cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>remove</td>
<td>number of comparisons made</td>
<td>visit every element</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>uniq</td>
<td>number of comparisons made</td>
<td>compare each element to all the other elements</td>
<td>O(n^2)</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>sublists</td>
<td>number of sublists created</td>
<td>construct 2^n lists</td>
<td>O(2^n)</td>
<td>O(2^n)</td>
</tr>
</tbody>
</table>