Today

- Announcements
  - Thursday 12/7: Course response forms. Bring computer.
  - Final exam: self-scheduled; 2 sheets of notes allowed

- Computational Complexity
  - Big-O notation to describe # of operations
  - Example: complexity of search algorithms
Time and space

Algorithm choice will determine the resources used at runtime.

Key resources:

- **Time** (CPU time)
- **Space** (memory)
Computational Complexity

A *basic operation* requires one time unit

- adding two values
- assigning to a variable
- comparing two values
- accessing a list element

How many *basic operations* does a given algorithm perform?
Traversing a List

Code to print all values in a list:

```python
n = len(t)
i = 0
while i < n:
    print(t[i], end='')
i += 1
```

How many operations does this perform?

→ $c \cdot n$ operations for list of $n$ elements

↑ a constant, eg, 5
Big-O notation

- Use a function to describe number of basic operations in terms of input size.
- The function includes only the dominant terms, ignoring constants.
- Example: list traversal is $O(n)$ for a list of $n$ values.
Linear Search

Find a value in a list:

\[
\begin{align*}
n &= \text{len}(t) \\
i &= 0 \\
\text{while } i < n: \\
&\quad \text{if } t[i] == \text{target}: \\
&\quad\quad \text{return } i \\
&\quad i += 1 \\
\text{return } -1
\end{align*}
\]

Number of operations for a list of \( n \) elements: \( O(n) \)
Binary Search

We can search faster if the list is sorted

→ Compare middle element to the target, then refine search to one half of list

| 2 | 5 | 8 | 11 | 15 | 16 | 21 | 24 | 29 | 41 | 45 | 58 | 71 | 85 | 92 | 95 |

Number of operations for a list of $n$ elements: $O(\log_2 n)$ or $O(\log n)$
Binary Search: $O(\log n)$

Time required to execute the algorithm

Time increasing by decreasing increments

Length increasing by uniform increments

Length of list
Order of Growth

<table>
<thead>
<tr>
<th>Problem Size (n)</th>
<th>Running Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>O(2^n)</td>
<td>Very bad.</td>
</tr>
<tr>
<td>O(n^2)</td>
<td></td>
</tr>
<tr>
<td>O(n log n)</td>
<td></td>
</tr>
<tr>
<td>O(n)</td>
<td></td>
</tr>
<tr>
<td>O(log n)</td>
<td></td>
</tr>
<tr>
<td>O(1)</td>
<td></td>
</tr>
</tbody>
</table>

Bad. Very bad.
<table>
<thead>
<tr>
<th>If time needed…</th>
<th>then we say…</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>grows proportionally with the input size</td>
<td>the algorithm runs in $O(n)$ or linear time</td>
<td>Linear search</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compute sum of list</td>
</tr>
<tr>
<td>grows only incrementally as the input size doubles</td>
<td>the algorithm runs in $O(\log n)$ or logarithmic time</td>
<td>Binary search,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fast exponentiation</td>
</tr>
<tr>
<td>doubles with a unit increment to the input size</td>
<td>the algorithm runs in $O(2^n)$ or exponential time</td>
<td>Recursive Fibonacci</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Towers of Hanoi</td>
</tr>
<tr>
<td>doesn't change with the input size</td>
<td>the algorithm runs in $O(1)$ or constant time</td>
<td>Finding max of sorted list</td>
</tr>
<tr>
<td>grows quadratically with the input size</td>
<td>the algorithm runs in $O(n^2)$ or quadratic time</td>
<td>Insertion, selection, and bubble sorts</td>
</tr>
</tbody>
</table>
Algorithms We Prefer

Polynomial time algorithms are desirable
- $O(1)$ [constant]
- $O(\log n)$ [logarithmic]
- $O(n)$ [linear]
- $O(n \log n)$
- $O(n^2)$ [quadratic]
- $O(n^3)$ [cubic]

Non-polynomial time algorithms are undesirable
- $O(2^n)$ [exponential]
- $O(n!)$ [factorial]