Lecture #17: Complexity and Orders of Growth

Public-Service Announcement

"Cal Habitat for Humanity is a service club dedicated to giving back to our wonderful community through volunteering. Between building houses, assisting with soup kitchens, gardening, and selling food, we strive to make a positive impact for our neighborhood and beyond. You can commit as little (0) or as many (10,000) hours as you would like to the club, but we could really use the help of volunteers such as you. If this seems like something you'd be interested in, join us at our next meeting on March 7th at 126 Barrows 7pm or visit our website at calhabitat.org."

Complexity

• Certain problems take longer than others to solve, or require more storage space to hold intermediate results.

• We refer to the time complexity or space complexity of a problem.

• What does it mean for a program to have a particular complexity?

• What does it mean for an algorithm?

• What does it mean for a problem?

A Direct Approach

• Well, if you want to know how fast something is, you can time it.

• Python happens to make this easy:

```python
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-2) + fib(n-1)
```

```python
from timeit import repeat

>>> repeat('fib(10)', 'from main import fib', number=5)
[0.000491, 0.000486, 0.000487, 0.000488, 0.000487]

>>> repeat('fib(20)', 'from main import fib', number=5)
[0.06012, 0.06012, 0.06012, 0.06012, 0.06012]

>>> repeat('fib(30)', 'from main import fib', number=5)
[7.744, 7.812, 7.812, 7.812, 7.812]
```

• `repeat(Stmt, Setup, number=N)` says Execute Setup (a string containing Python code), then execute Stmt (a string containing Python code) `N` times. Repeat this process 3 times and report the time required for each repetition.

A Direct Approach, Continued

• You can also use this from the command line:

```bash
$ python3 -m timeit --setup='from fib import fib' 'fib(10)'
10000 loops, best of 3:
97 usec per loop
```

• This command automatically chooses a number of executions of fib to give a total time that is large enough for an accurate average.

• Good: Every actual time answers question completely for given input.

Strengths and Problems with Direct Approach

• Good: Gives actual times; answers question completely for given input.

• Bad: Results apply only to particular programs and platforms.

• Bad: Results apply only to tested inputs.

• Good: Every actual time answers question completely for given input.

Complacency

• What does it mean for a problem to have a complexity?

• What does it mean for an algorithm to have a complexity?

• We refer to the time complexity or space complexity of a problem.

• Certain problems take longer than others to solve, or require more storage space to hold intermediate results.

• Certain problems take longer than others to solve, or require more storage space to hold intermediate results.

BORROWER/TIMELINE/PUBLICATION INFORMATION

- Borrowed from Wikipedia, "List of webbats of California orangutans"
- "List of webbats of California orangutans"
- "List of webbats of California orangutans"
- "List of webbats of California orangutans"
- "List of webbats of California orangutans"
Operation Counts and Scaling

What to Measure?
- Size of the data, or the length of the list?
- The number of times the loop is going to be executed?
- Whether or not the loop body is executed once or not at all?

Instead of looking at times, we can consider number of *operations*.

Where does the time come in?
- The total time consists of
  - Some cost to run the loop.
  - Some pre-execution costs: allocation, benchmarks.
We can consider the number of operations.

If we want general answers, we have to introduce some strategic approximations.

**Examples:**

What does an "operation" cost?

- Some fixed overhead to start the function and begin the loop.
- Some cost to run the loop.
- Some pre-execution costs: allocation, benchmarks.

The total time consists of

\[
\text{time} = t + \text{setup} + \text{iterations}.
\]

How many times does the function get called recursively?
- To get worst-case times, we need to introduce some strategic approximations.
- If we want general answers, we have to introduce some strategic approximations.

**Consider the following search function:**

\[
\text{def search(L, x, delta):}
\]

\[
\text{if x in L: return True}
\]

\[
\text{return False}
\]

What is the worst case to compute \( f(x) \)?
- What is the worst case to compute \( f(x) \)?
- We usually look at more general questions, such as:

- To avoid the problem of getting results only for particular inputs,
  we usually look at more general questions, such as:

- We usually look at more general questions, such as:

- To avoid the problem of getting results only for particular inputs,
  we usually look at more general questions, such as:

**Worst Case, Best Case, Average Case**

- In the worst case, the program has to do a lot of work.
- In the best case, the program has to do very little work.
- In the average case, the program has to do something in between.

**What Does an "Operation" Cost?**

- How long do *operations* take?
- What kind of numbers are in *operations*?
- Where in *operations* do operations take place?

- What Does an "Operation" Cost?
- How long does its take?
- What if the program special-cases some inputs?
- Why can't we extrapolate?
- But these "operations" are of different kinds and complexities, so
  what do we really know?
- ...and this still applies for a particular program and machine.
Asymptotic Results

The Notation

Using Asymptotic Estimates

Informal Cutoffs

Expressing Approximation

Asymptotic Results
Using Asymptotic Estimates

Claim that
\[ \text{min-fixed-cost} + N \times \text{min-loop-cost} \leq C_{wc} \text{near}(N) \leq \text{max-fixed-cost} + N \times \text{max-loop-cost} \]
means that
\[ C_{wc} \text{near}(N) \in \Theta(N) \]

Why?

Well, if we ignore the two fixed costs (assume they are 0), we obviously fit the definition, since for \( N \geq 0 \),
\[ p \cdot N \leq C_{wc} \text{near}(N) \leq q \cdot N, \]
where \( p \) is min-loop-cost and \( q \) is max-loop-cost.

It's easy to see that by tweaking \( q \) up a bit—e.g., to \( q' \), where
\[ q' = q + \text{max-fixed-cost} \]
we can arrange that when \( N \) is big enough (\( N > 1 \) for this particular \( p' \)), we cover the necessary range.

Some Intuition on Meaning of Growth

• How big a problem can you solve in a given time?

In the following table, left column shows time in microseconds to solve a given problem as a function of problem size \( N \) (assuming perfect scaling and that problem size 1 takes 1 \( \mu \) sec).

Entries show the size of problem that can be solved in a second, hour, month (31 days), and century, for various relationships between time required and problem size. (Assuming perfect scaling and problem size \( N \) times larger.)

In the following table, left column shows time in microseconds.

How big a problem can you solve in a given time?

Typical \( \Theta() \) Estimates from Programs

<table>
<thead>
<tr>
<th>Time (( \mu )sec) for Max ( N ) Possible in</th>
<th>( N ) Possible in 1 second</th>
<th>( N ) Possible in 1 hour</th>
<th>( N ) Possible in 1 month</th>
<th>( N ) Possible in 1 century</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lg N )</td>
<td>( 10^{10} )</td>
<td>( 10^{10} )</td>
<td>( 10^{11} )</td>
<td>( 10^{12} )</td>
</tr>
<tr>
<td>( N )</td>
<td>( 10^6 )</td>
<td>( 3 \cdot 10^7 )</td>
<td>( 7 \cdot 10^9 )</td>
<td>( 3 \cdot 10^{11} )</td>
</tr>
<tr>
<td>( N \lg N )</td>
<td>( 63000 )</td>
<td>( 1.4 \cdot 10^6 )</td>
<td>( 7 \cdot 10^9 )</td>
<td>( 6 \cdot 10^{11} )</td>
</tr>
<tr>
<td>( 2^N )</td>
<td>( 1000 )</td>
<td>( 60000 )</td>
<td>( 1.6 \cdot 10^7 )</td>
<td>( 5 \cdot 10^9 )</td>
</tr>
<tr>
<td>( N^2 )</td>
<td>( 100 )</td>
<td>( 1500 )</td>
<td>( 14000 )</td>
<td>( 150000 )</td>
</tr>
<tr>
<td>( N^3 )</td>
<td>( 20 )</td>
<td>( 32 )</td>
<td>( 41 )</td>
<td>( 51 )</td>
</tr>
</tbody>
</table>

Example

Typical \( \Theta() \) Estimates from Programs

Using Asymptotic Estimates