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.
- But what does it mean to say that a certain program has 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)
  ```

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

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

Complexity

- What does it mean for a problem?
- What does it mean for an algorithm?
- What does it mean to hold intermediate results?
- We refer to the time complexity of space complexity of a problem.
- Certain problems take longer than others to solve, or require more...
Operation Counts and Scaling

What to Measure?

- Different programs/algorithms run on the basis of which scale best.
- This we look at how computation time scales in the worst case.
- Discussion: results are proportional to actual time for different values
- You can have different results this if the quantities are well.
- How many operations get performed by (fib)?
- How many times does fib get called recursively during computation?
- Which?

Examples:
- Choose some operations of interest and count how many times they occur.

- What do we really know?

- So what can we meaningfully say about complexity of...

- How long do...

- How long does it take to create an iterator for...

- How long is sequence...

- What kind of numbers are in...

- How many addition operations get performed by...

= 1 + (L) x (abs(x-y) <= delta)

- Assume that each operation represents some range of possible
- This looks to be exponential in t
- But these "operations" are of different kinds and complexities, so

Worst Case, Best Case, Average Case

- We can no longer get precise times, but if the operations are well...

- Can compare programs/algorithms on the basis of which scale best.
- Instead of getting precise answers in units of physical time, we...

- Choose some operations of interest and count how many times they occur.

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

- What is the worst case time to compute fib(N) as a function of the size of N, or...

- What is the average case time to compute fib(N) as a function of the size of N, or...

- Worst case, best case, average case

- And when do we...
Asymptotic Results

Expressing Approximation

The notion can be used to express the growth rate of any function.

In this course, the functions we are interested in are those that express the time a computation takes as a function of problem size.

Thus, the notion can be used to express the growth rate of any function.

In general, we need to ask about asymptotic behavior of programs: as the size of input grows to infinity.

Tests for numbers up to 1 billion will be faster than for larger numbers.

The set of primes up to 1 billion (about 50 million primes).

Looking at the number of items in each of the stages of the loop.

The precision is generally not terribly important anyway.

Approximations of execution time or space grow.

Comparing the growth rates of functions can be very complicated, and

Thus, we use the notion to mean the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

So we can write

These considerations motivate the use of order notation to express

Going back to the exact function.

We use the notation \( f = \Theta(g) \) to mean "the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

Adding or multiplying sets of functions produces sets of functions.

Sometimes, results for small values are not indicative.

We usually write things like

Thus, \( f = \Theta(g) \) means "the set of functions of \( x \) whose absolute values are eventually bounded above and below by some constant multiples of \( f \)"

Asymptotic Results

Expressing Approximation

The notion can be used to express the growth rate of any function.

In this course, the functions we are interested in are those that express the time a computation takes as a function of problem size.

Thus, the notion can be used to express the growth rate of any function.

In general, we need to ask about asymptotic behavior of programs: as the size of input grows to infinity.

Tests for numbers up to 1 billion will be faster than for larger numbers.

The set of primes up to 1 billion (about 50 million primes).

Looking at the number of items in each of the stages of the loop.

The precision is generally not terribly important anyway.

Approximations of execution time or space grow.

Comparing the growth rates of functions can be very complicated, and

Thus, we use the notion to mean the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

So we can write

These considerations motivate the use of order notation to express

Going back to the exact function.

We use the notation \( f = \Theta(g) \) to mean "the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

Adding or multiplying sets of functions produces sets of functions.

Sometimes, results for small values are not indicative.

We usually write things like

Thus, \( f = \Theta(g) \) means "the set of functions of \( x \) whose absolute values are eventually bounded above and below by some constant multiples of \( f \)".

Asymptotic Results

Expressing Approximation

The notion can be used to express the growth rate of any function.

In this course, the functions we are interested in are those that express the time a computation takes as a function of problem size.

Thus, the notion can be used to express the growth rate of any function.

In general, we need to ask about asymptotic behavior of programs: as the size of input grows to infinity.

Tests for numbers up to 1 billion will be faster than for larger numbers.

The set of primes up to 1 billion (about 50 million primes).

Looking at the number of items in each of the stages of the loop.

The precision is generally not terribly important anyway.

Approximations of execution time or space grow.

Comparing the growth rates of functions can be very complicated, and

Thus, we use the notion to mean the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

So we can write

These considerations motivate the use of order notation to express

Going back to the exact function.

We use the notation \( f = \Theta(g) \) to mean "the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

Adding or multiplying sets of functions produces sets of functions.

Sometimes, results for small values are not indicative.

We usually write things like

Thus, \( f = \Theta(g) \) means "the set of functions of \( x \) whose absolute values are eventually bounded above and below by some constant multiples of \( f \)".

Asymptotic Results

Expressing Approximation

The notion can be used to express the growth rate of any function.

In this course, the functions we are interested in are those that express the time a computation takes as a function of problem size.

Thus, the notion can be used to express the growth rate of any function.

In general, we need to ask about asymptotic behavior of programs: as the size of input grows to infinity.

Tests for numbers up to 1 billion will be faster than for larger numbers.

The set of primes up to 1 billion (about 50 million primes).

Looking at the number of items in each of the stages of the loop.

The precision is generally not terribly important anyway.

Approximations of execution time or space grow.

Comparing the growth rates of functions can be very complicated, and

Thus, we use the notion to mean the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

So we can write

These considerations motivate the use of order notation to express

Going back to the exact function.

We use the notation \( f = \Theta(g) \) to mean "the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

Adding or multiplying sets of functions produces sets of functions.

Sometimes, results for small values are not indicative.

We usually write things like

Thus, \( f = \Theta(g) \) means "the set of functions of \( x \) whose absolute values are eventually bounded above and below by some constant multiples of \( f \)".

Asymptotic Results

Expressing Approximation

The notion can be used to express the growth rate of any function.

In this course, the functions we are interested in are those that express the time a computation takes as a function of problem size.

Thus, the notion can be used to express the growth rate of any function.

In general, we need to ask about asymptotic behavior of programs: as the size of input grows to infinity.

Tests for numbers up to 1 billion will be faster than for larger numbers.

The set of primes up to 1 billion (about 50 million primes).

Looking at the number of items in each of the stages of the loop.

The precision is generally not terribly important anyway.

Approximations of execution time or space grow.

Comparing the growth rates of functions can be very complicated, and

Thus, we use the notion to mean the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

So we can write

These considerations motivate the use of order notation to express

Going back to the exact function.

We use the notation \( f = \Theta(g) \) to mean "the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

Adding or multiplying sets of functions produces sets of functions.

Sometimes, results for small values are not indicative.

We usually write things like

Thus, \( f = \Theta(g) \) means "the set of functions of \( x \) whose absolute values are eventually bounded above and below by some constant multiples of \( f \)".

Asymptotic Results

Expressing Approximation

The notion can be used to express the growth rate of any function.

In this course, the functions we are interested in are those that express the time a computation takes as a function of problem size.

Thus, the notion can be used to express the growth rate of any function.

In general, we need to ask about asymptotic behavior of programs: as the size of input grows to infinity.

Tests for numbers up to 1 billion will be faster than for larger numbers.

The set of primes up to 1 billion (about 50 million primes).

Looking at the number of items in each of the stages of the loop.

The precision is generally not terribly important anyway.

Approximations of execution time or space grow.

Comparing the growth rates of functions can be very complicated, and

Thus, we use the notion to mean the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

So we can write

These considerations motivate the use of order notation to express

Going back to the exact function.

We use the notation \( f = \Theta(g) \) to mean "the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

Adding or multiplying sets of functions produces sets of functions.

Sometimes, results for small values are not indicative.

We usually write things like

Thus, \( f = \Theta(g) \) means "the set of functions of \( x \) whose absolute values are eventually bounded above and below by some constant multiples of \( f \)".

Asymptotic Results

Expressing Approximation

The notion can be used to express the growth rate of any function.

In this course, the functions we are interested in are those that express the time a computation takes as a function of problem size.

Thus, the notion can be used to express the growth rate of any function.

In general, we need to ask about asymptotic behavior of programs: as the size of input grows to infinity.

Tests for numbers up to 1 billion will be faster than for larger numbers.

The set of primes up to 1 billion (about 50 million primes).

Looking at the number of items in each of the stages of the loop.

The precision is generally not terribly important anyway.

Approximations of execution time or space grow.

Comparing the growth rates of functions can be very complicated, and

Thus, we use the notion to mean the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

So we can write

These considerations motivate the use of order notation to express

Going back to the exact function.

We use the notation \( f = \Theta(g) \) to mean "the set of all one-parameter functions whose absolute value is eventually bounded above and below by some constant multiples of their parameter, returning real numbers.

Adding or multiplying sets of functions produces sets of functions.

Sometimes, results for small values are not indicative.

We usually write things like

Thus, \( f = \Theta(g) \) means "the set of functions of \( x \) whose absolute values are eventually bounded above and below by some constant multiples of \( f \)".
Claim that \( \min\text{-fixed-cost} + N \times \min\text{-loop-cost} \leq C_{wc} \) near \((N)\) means that \( C_{wc} \) 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 \( \min\text{-loop-cost} \) and \( q \) is \( \max\text{-loop-cost} \).

It's easy to see that by tweaking \( q \) up a bit—e.g., to \( q' \), where \( q' = q + \max\text{-fixed-cost} \)—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 \( N \). The remaining columns 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. Entering the size of problem that can be solved in a second, hour, month (31 days), and century, for various relationships be- tween the time needed and problem size.

Entries show the size of problem that can be solved in a second, hour, month (31 days), and century, for various relationships between the time required and problem size. Entering the size of problem that can be solved in a second, hour, month (31 days), and century, for various relationships between the time required and problem size. Entering the size of problem that can be solved in a second, hour, month (31 days), and century, for various relationships between the time required and problem size.

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

<table>
<thead>
<tr>
<th>( C_{wc} ) near ((N) ) in ( \Theta(N) ) for ( \Theta(\cdot) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>( \Theta(lg N) )</td>
</tr>
<tr>
<td>( \Theta(N) )</td>
</tr>
<tr>
<td>( \Theta(N^2) )</td>
</tr>
<tr>
<td>( \Theta(2^N) )</td>
</tr>
</tbody>
</table>

Last modified: Fri Mar 3 19:12:24 2017 CS61A: Lecture #17