CS140: Algorithms

Z Sweedyk
Lecture 1
1/18/01

Last class
The two important questions we consider in CS140:
– Is the computational procedure correct?
– Is the algorithm fast?

How do we measure speed?

• What to measure
  • Big-O notation/rate of growth
  • Loop counting
  • Series

Running Time
Where to measure?

Measurement is easy
Measurement is meaningful

A useful assumption

$T_A$ and $T_M$ differ by no more than a multiplicative constant

More formally
There is some constant $c$ such that for any input $Z$

$T_A(Z)/c \leq T_M(Z) \leq cT_A(Z)$
Running Time
Where to measure?

- Algorithm: Measurement is easy and meaningful
- Program
- Machine code: Input
- Machine: Measurement is meaningful

Running Time:
What to measure?

- Run time depends on input size
- Run time can vary on different inputs of size n.

Pick special case

- Run time depends on input size
- Run time can vary on different inputs of size n.
- Choose case:
  - Worst case (show in bold)
  - Best case
  - Average case
  - Etc.

Worst case performance of algorithm ▲

- We can compute this function at a finite number of points.
- Better yet, we can model this function for all input sizes.

A general problem …

- Question: How can we give a succinct description of an arbitrary function?
- Answer: Big-O notation.

Today

- What to measure
- Big-O notation/rate of growth
- Loop counting
- Series
Upper Bounds

- \( f : \mathbb{N} \to \mathbb{N} \) and \( g : \mathbb{N} \to \mathbb{N} \) are positive-valued, monotonically increasing functions.
- \( O(g(n)) = \{ f(n) : \text{there are constants } c \text{ and } M \text{ such that } f(n) \leq c \cdot g(n) \text{ for all } n \geq M \} \)

Proving \( f(n) = O(g(n)) \)

Consider \( h(n) = f(n)/g(n) \) as \( n \) goes to infinity
- \( h(n) \) converges
- \( h(n) \) diverges
- \( h(n) \) oscillates

Some useful observations about Big-O

- If \( f(n)/g(n) \) converges then \( f(n) \_\_ O(g(n)) \).
- If \( f(n)/g(n) \) diverges then \( f(n) \_\_ O(g(n)) \).
- If \( f(n)/g(n) \) oscillates then \( f(n) \_\_ O(g(n)) \).

Logarithms

For which pairs \( f(n) \), \( g(n) \) is \( f(n) = O(g(n)) \)?
- \( \log n \)
- \( \log n^2 \)
- \( \log^2 n \)
- \( \log 10000 n \)

Example

\[
\lim_{n \to \infty} \frac{\log n}{\log n^2} = \frac{(\log 10)/2}{2}\]

(Useful observation: \( \log n^2 = 2(\log 10 \cdot \log n) \))

Limits

1. \( \lim_{n \to \infty} \frac{\log n^2}{\log n} \)
2. \( \lim_{n \to \infty} \frac{\log n}{\log^2 n} \)
3. \( \lim_{n \to \infty} \frac{\log^2 n}{\log n} \)
4. \( \lim_{n \to \infty} \frac{\log n}{\log 10000 n} \)
Polynomials
For which pairs \( f(n) \), \( g(n) \) is \( f(n) = O(g(n)) \)?
- \( n \)
- \( n^2 \)
- \( 1000n^2 + n \)

Exponentials
For which pairs \( f(n) \), \( g(n) \) is \( f(n) = O(g(n)) \)?
- \( 2^n \)
- \( 3^n \)
- \( 2(n^2) \)
- \( (2^n)^2 \)

Some rules of thumb
- Polylogs are slower growing than polynomials
  For any \( k, j > 0 \):
    - \( \log^j n = O(n^k) \) and \( n^k \neq O(\log^j n) \)
- Polynomials are slower growing than exponentials
  For any \( k > 0 \) and \( r > 1 \):
    - \( n^k = O(r^n) \) and \( r^n \neq O(n^k) \)

L’hospital’s rule
- \( \lim_{n \to \infty} \log^n n / r^k = 0 \)
- \( n^k / \log^n n \) diverges as \( n \) goes to infinity

Polynomially bounded functions
\( f(n) \) is polynomially bounded if there is a constant \( k \) such that \( f(n) = O(n^k) \)

Logs, Polys, and Exps
Which of the following functions are polynomially bounded?
- \( \log n \)
- \( n^3 \)
- \( 2^n \)
Other functions

- Factorial: \( n! = n \cdot (n-1)! \), 0! = 1
- Tower of 2s: \( 2^{\uparrow\uparrow}n = 2^{2^{\uparrow\uparrow}(n-1)} \), \( 2^{\uparrow\uparrow}0 = 1 \)
- Iterated log: \( \log^*(n) = m \) such that \( 2^{\uparrow\uparrow}((m-1)) < n \leq 2^{\uparrow\uparrow}m \)
- Ceiling: \( \lceil \log n \rceil = 2^m \) such that \( m-1 < \log n \leq m \)

Logs, polys, exps, and others

Compare the rates of growth of the following functions:
\( \lg n \), \( n^3 \), \( 2^n \), \( n! \), \( 2^{\uparrow\uparrow}n \), \( \log^*(n) \), \( \lceil \log n \rceil \)

Another useful observation

- If \( \frac{f(n)}{g(n)} \) diverges then so does \( 2^{f(n)}/2^{g(n)} \)
- If \( \frac{\lg(f(n))}{\lg(g(n))} \) diverges then so does \( \frac{f(n)}{g(n)} \)

Beyond O

<table>
<thead>
<tr>
<th>Real numbers</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \leq )</td>
<td>( O )</td>
</tr>
<tr>
<td>( \geq )</td>
<td>( \Omega )</td>
</tr>
<tr>
<td>( = )</td>
<td>( \Theta )</td>
</tr>
<tr>
<td>( &lt; )</td>
<td>( o )</td>
</tr>
<tr>
<td>( &gt; )</td>
<td>( \omega )</td>
</tr>
</tbody>
</table>

Lower Bounds

- \( f: \mathbb{N} \rightarrow \mathbb{N} \) and \( g: \mathbb{N} \rightarrow \mathbb{N} \) are positive-valued, monotonically increasing functions.
- \( \Omega(g(n)) = \{ f(n) : \) there are constants \( c \) and \( M \) such that \( f(n) \geq c \cdot g(n) \) for all \( n \geq M \) \}

Definition: \( \Theta \)

\( f(n) = \Theta(g(n)) \) if the following hold:
1. \( f(n) = O(g(n)) \), and
2. \( f(n) = \Omega(g(n)) \)
Definition: little-o, little-ω

- \( f(n) = o(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \)
- \( f(n) = \omega(g(n)) \) if \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \)

Logs, polys, exps, and others

Compare the following functions. Which of \( O, \Omega, \Theta, o, \) and \( \omega \) apply?

\( \lg n, \ n^3, \ 2^n, \ n!, \ 2 \uparrow\uparrow n, \ \log^*(n), \ \lceil n \rceil \)

A slight twist…

Is \( f(2n) = O(f(n)) \)?
1. \( f(n) = 1 \): Is \( 2n = O(n) \)?
2. \( f(n) = 3n \): Is \( 6n = O(3n) \)?
3. \( f(n) = n^2 \): Is \( 4n^2 = O(n^2) \)?
4. \( f(n) = 2^n \): Is \( 4^n = O(2^n) \)?
5. \( f(n) = n! \): Is \( (2n)! = O(n!) \)?

Today

- What to measure
- Big-O notation/rate of growth
- Loop counting
- Series

CS140 pragmatism

What is the asymptotic behavior of the worst-case running time of the algorithm?

CS140 pragmatism

Big-O

What is the asymptotic behavior of the worst-case running time of the algorithm?
What is the asymptotic behavior of the worst-case running time of the algorithm?

Special case input

What is the asymptotic behavior of the worst-case running time of the algorithm?

Chosen resource

What is the asymptotic behavior of the worst-case running time of the algorithm?

Remember our assumption

The running time of $A$ is $O(n^3)$.

The worst case running time of $A$ is $O(n^3)$.

$A$ is $O(n^3)$.

Run time bounds for algorithm $A$

Rate of growth of common functions

• Review of properties/notation
• See CLR pp 32–37 for details

KNOW THIS STUFF

Today

• How should we measure the speed of an algorithm?
• Big-O notation/rate of growth
• Loop counting
• Series
Types of Algorithms

- Recursive Algorithm: one that calls itself
- Purely Iterative Algorithm: one that doesn’t

Run Time Analysis

- Iterative algorithm → Loop counting
- Recursive algorithm → Recurrence relations

Iterative Sorting Algorithms

- Insertion-sort
- Bubble-sort
- Modified Bubble-sort

Insertion-sort(S)

(in pseudo-code) S is an array of n integers: S(1), S(2), …, S(n)

For j = 2 to n
  key = S(j)
i = j - 1
While i > 0 and S(i) > key
  S(i+1) = S(i--)
S(i+1) = key

Correctness

- Inductive proof with loop invariant:
  When the for loop executes for the kth time, S(1), S(2), …, S(k) are sorted in ascending order.

Loop Counting: Insertion-sort(S)

\[
\sum_{j=2}^{n} (1 + 1 + \sum_{i=0}^{j-1} (1 + 1 + 1)) = O(n^2)
\]
Bubble-sort(S)

Bubble-sort(S)
For i=n down to 2
    For j=1 to i-1
        If S(j) > S(j+1) then swap(S(j), S(j+1))
    Return

Correctness

- Inductive Proof with loop invariant:
  When the i-loop completes its k\textsuperscript{th} execution,
  - S(n-k+1), S(n-k+2), ..., S(n) is sorted in ascending order, and
  - the max(S(1), ..., S(n-k)) \leq S(n-k+1).

Does Bubble-sort do too much work?

1. 3. 2. 4. 5
1. 2. 3. 4. 5
1. 2. 3. 4. 5
1. 2. 3. 4. 5
1. 2. 3. 4. 5

- Repeat on smaller list

Modified Bubble-sort

Modified-Bubble-sort(S)
SWAP=T
For i=n down to 2
    If SWAP=F then return
    SWAP=F
    For j=1 to i-1
        If S(j) > S(j+1) then swap(S(j), S(j+1)) and set SWAP=T
    Return

Example

1. 3. 2. 4. 5
1. 2. 3. 4. 5
1. 2. 3. 4. 5

- Repeat on smaller list unless no swaps are made

Loop counting: M-Bubble-sort

\[1 + \sum_{i=2}^{n}(4 + \sum_{j=1}^{i-1} 3) = O(n^2)\]
Summation

\[ \sum_{i=2}^{n} \sum_{j=1}^{i-1} c = c \left( \sum_{i=2}^{n} (i-1) \right) - c \]
\[ = c \left( \sum_{i=2}^{n} i - n \right) - c \]
\[ = O(n^2) \]

Series

- A series is a summation of terms
- Common series:
  - Arithmetic series: \( 1+2+\ldots+n \)
  - Geometric series: \( 1+a+a^2+\ldots+a^n \)

Series

Things we want to do:

- Solve exactly
- Bound above or below
- Prove that a solution (or bound) is correct

Closed form solutions to some common series

- \( f(n) = 1+2+\ldots+n = n(n+1)/2 \)
- \( f(n) = 1^2 + 2^2 + \ldots + n^2 = (2n^3 + 3n^2 + n)/6 \)
- \( f(n) = 1+a+a^2+\ldots+a^n = \frac{a^{n+1}-1}{a-1} \) if \( a=1 \)
  \( = 1/(1-a) \) if \( 0 \leq a < 1 \)
- \( f(n) = 1+a+a^2+\ldots = 1/(1-a) \) if \( 0 \leq a < 1 \)

Series

Things we want to do:

- Solve exactly
- Bound above or below
- Prove that a solution (or bound) is correct

Upper Bounds on series

For any constant \( k \):

\[ \sum_{i=1}^{n} i^k \leq \frac{\sum_{i=1}^{n} n^k}{(n^{k+1})} = O(n^{k+1}) \]

Is this a good upper bound?
Lower Bounds on series

For any constant k:

\[ \sum_{i=1}^{n} i^k \geq \sum_{i=\lceil n/2 \rceil}^{n} i^k \geq \frac{n}{2} \left( \frac{n}{2} + 1 \right)^{k+1} \]

\[ = \Omega(n^{k+1}) \]

So \( \sum_{i=1}^{n} i^k = \Theta(n^{k+1}) \)

Series

Things we want to do:

• Solve exactly
• Bound above or below
• Prove that a solution (or bound) is correct

Proving correctness

• Claim: \( \sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6} \)
• Claim holds for \( n=1 \).
• If the claim holds for \( n \) then it holds for \( n+1 \):

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

\[ = (n+1)^2 + \frac{2n^3 + 3n^2 + n}{6} \]

\[ = \frac{2(n+1)^3 + 3(n+1)^2 + (n+1)}{6} \]

Next time

• Recursive algorithms
• Recurrence relations