CS140: Algorithms

Z Sweedyk
Lecture 1

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
- Recurrence relations

Running Time
Where to measure?

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 $Z$, $\frac{T_A(Z)}{c} \leq T_M(Z) \leq cT_A(Z)$
Running Time
Where to measure?

- Measurement is easy and meaningful
- Program
- Machine code
- Measurement is meaningful
- Machine

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
- Recurrence relations
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
  - Unbounded
  - Bounded

Some useful observations about Big-O

- If \( f(n)/g(n) \) is bounded then \( f(n) \text{ ___ } O(g(n)) \).
- If \( f(n)/g(n) \) is unbounded then \( f(n) \text{ ___ } O(g(n)) \).

Logarithms

For which pairs \( f(n), g(n) \) is \( f(n) = O(g(n)) \)?

- \( \lg n \)
- \( \log n \)
- \( \log^2 n \)
- \( \lg 10000n \)

Example

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

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

Limits

1. \( \lim_{n \to \infty} \frac{\log n^2}{\lg n} \)
2. \( \lim_{n \to \infty} \frac{\lg n}{\log^2 n} \)
3. \( \lim_{n \to \infty} \frac{\lg^2 n}{\lg n} \)
4. \( \lim_{n \to \infty} \frac{\lg n}{\log 10000 n} \)
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
- $\lim_{n \to \infty} \frac{\log n}{n^k} = 0$
- $n^k / \log n$ 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)! \quad 0! = 1
- Tower of 2s: \quad 2 \uparrow\uparrow n = 2 \uparrow\uparrow(n-1), \quad 2 \uparrow\uparrow 0 = 1
- Iterated log: \quad \log^*(n) = m \quad \text{such that} \quad 2 \uparrow\uparrow (m-1) < n \leq 2 \uparrow\uparrow m
- Ceil-ceil: \quad \lceil \lceil n \rceil \rceil = 2^m \quad \text{such that} \quad m - 1 < \log n \leq m

Logs, polys, exps, and others

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

Another useful observation

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

Beyond O

\begin{align*}
\text{real numbers} & \quad \text{functions} \\
\leq & \quad \Omega \\
\geq & \quad \Theta \\
= & \quad o \\
< & \quad \omega
\end{align*}

Lower Bounds

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

Definition: \Theta

f(n) = \Theta(g(n)) \quad \text{if the following hold:}
1. \ f(n) = O(g(n)), \quad \text{and}
2. \ f(n) = \Omega(g(n))
Definition: \textit{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?

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

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?
CS140 pragmatism

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

Special case

input

CS140 pragmatism

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

Chosen resource

CS140 pragmatism

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

Remember our assumption

In this class we’ll say

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

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

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

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)

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)

\[ \sum_{j=2}^{n} (1 + \sum_{i=0}^{j-1} 1) = O(n^2) \]
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)) ≤ 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, 4

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

Loop counting: M-Bubble-sort

Modified-Bubble-sort(S)
\[
1 + \sum_{i=1}^{n-1} 1 + \sum_{j=1}^{i-1} \left( 1 + \sum_{k=1}^{j-1} 4 \right)
\]
= O(n^2)
Summation

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

Series

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

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

- \(f(n) = 1+2+...+n = \frac{n(n+1)}{2}\)
- \(f(n) = 1^2+2^2+...+n^2 = \frac{2n^3+3n^2+n}{6}\)
- \(f(n) = 1+a+a^2+...+a^n = \frac{a^{n+1}-1}{a-1}\) if \(a \neq 1\)
- \(f(n) = 1+a+a^2+... = \frac{1}{1-a}\) 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 \(k\):

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

Is this a good upper bound?
Lower Bounds on series

For any constant $k$:
\[
\sum_{i=1}^{n} i^k \geq \sum_{i=\lceil n/2 \rceil}^{n} i^k \\
\geq \sum_{i=\lceil n/2 \rceil}^{n} (n/2)^k \\
= \Omega(n^{k+1})
\]

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

---

Run Time Analysis

• Iterative algorithm → Loop counting
• Recursive algorithm → Recurrence relations
  1. Write the recurrence relation
  2. Convert the recurrence relation to a series
  3. Solve the series

---

Sort3: A Recursive Algorithm for SIAO

Sort3(S)
  If ||S|| <= 1
    Return: S
  Else
    Return: Sort3(S\max-element(S)), max-element(S)

---

1. Write the recurrence relation

Let $T(n)$ be the running time of Sort3:
\[
T(1) = c_2 \\
T(n) = c_1 n + T(n-1), n \geq 1
\]
2. Convert to series

\[ T(1) = c_2 \]
\[ T(n) = c_1n + T(n-1), n>1 \]
\[ \downarrow \]
\[ c_2 + \sum_{i=2}^{n} c_1i \]

3. Solve

\[ c_2 + \sum_{i=2}^{n} c_1i \]
\[ \longrightarrow \]
\[ c_2c_1c_i(n+1)/2 = O(n^2) \]

Recurrence Relations

Shortcuts and other tools:
- Guess and prove
- Master method
- Unwinding
- WORK TREES

Steps 2-3: Guess and Prove

- \( T(n)=c_1n + T(n-1), T(1)=c_2 \)
- Guess: \( T(n) = O(n^2) \)
- Prove: We need to show that there exists constants \( c \) and \( M \) such that \( T(n) \leq cn^2 \) for all \( n \geq M \)

Guess and Prove cont.

- \( T(1) \leq c \) provided \( c \geq c_1 \)

- Suppose \( T(n-1) \leq c(n-1)^2 \).
  \( T(n) = c_1n + T(n-1) \)
  \leq c_1n + c(n-1)^2 
  = c_1n + c(n^2 - 2n + 1) 
  = cn^2 - (2c - c_1)n + c 
  \leq cn^2 \) provided \( c \geq c_2 \) and \( n \geq 1 \)

Guess and Prove cont.

- \( T(n) \leq cn^2 \) for all \( n \geq 1 \), where 
  \( c = \max(c_1, c_2) \)
- \( T(n) = O(n^2) \)
Guess and Prove cont.

• What if you guess is wrong?

• You'll reach a contradiction in the proof step

Recurrence Relations

Shortcuts and other tools:
- Guess and prove
- Master method
- Unwinding
- WORK TREES

Steps 2-3: Master Theorem

• Read the book

Warning - only works for certain types of recurrence relations

Recurrence Relations

Shortcuts and other tools:
- Guess and prove
- Master method
- Unwinding
- WORK TREES

Step 2: Unwinding

\[ T(n) \leq c_1 n + T(n-1) \]
\[ \leq c_2 n + c_2(n-1) + T(n-2) \]
\[ \leq c_2 n + c_2(n-1) + c_2(n-2) + T(n-3) \]
... 
\[ \leq c_1 + \sum_{i=2}^{n} c_2 i \]

Recurrence Relations

Shortcuts and other tools:
- Guess and prove
- Master method
- Unwinding
- WORK TREES
Work Tree

Graphical representation of the "work" done by a recursive algorithm

Example: \( \text{Sort3}(3,1,5,2,4) \)

**Level 0**
- \( \text{Sort3}(3,1,5,2,4) \)

**Level n-1**
- \( \text{Sort3}(3,1,2) \)
- \( \text{Sort3}(2) \)
- \( \text{Sort3}(1) \)

Tree Nodes = Recursive calls

Example: \( \text{Sort3}(3,1,5,2,4) \)

Input size at level 5 = \( n \)
Input size at level 4 = \( n-1 \)
Input size at level 3 = \( n-2 \)
Input size at level 2 = \( n-3 \)
Input size at level 1 = \( n-4 \)

Work done at level 5 = \( c \)
Work done at level 4 = \( c \)
Work done at level 3 = \( c \)
Work done at level 2 = \( c \)
Work done at level 1 = \( c \)

Total work:
\[
\sum_{i=0}^{n-1} c(n-i) = cn(n+1)/2 = O(n^2)
\]

Example: \( \text{Sort3}(3,1,5,2,4) \)

Next time
- Work trees continued
- MergeSort
- A little probability theory
- QuickSort