Let us reminisce…

CS140: Two questions

Is it correct? Is it fast?

Computational procedure → yes → Algorithm → yes → Good algorithm

Time:
Where to measure?

Algorithm — Measurement is easy

Program

Machine code

Input

Measurement is meaningful — Machine

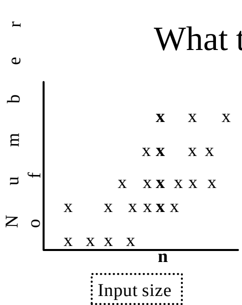
Time:
Where to measure?

Algorithm — Measurement is easy

Assumption: These running times differ by no more than a constant multiplicative factor.

Measurement is meaningful — Machine

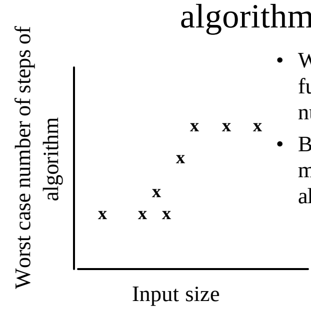
Time:
What to measure?



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

Choose special case to consider

Worst case performance of algorithm A

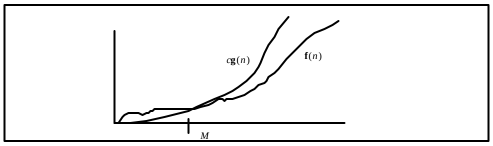


- We can compute this function at a finite number of points.
- Better yet, we can model this function for all input sizes.

Big-O notation

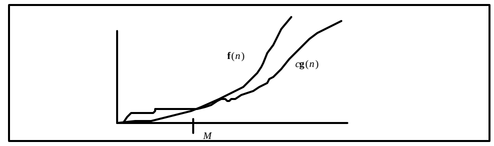
Upper Bounds

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



Lower Bounds

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



Definition: Θ

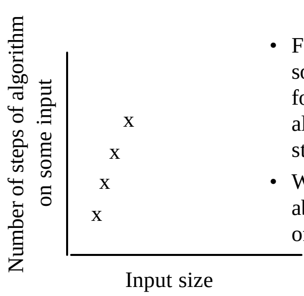
$f(n) = \Theta(g(n))$ if the following hold:

1. $f(n) = O(g(n))$, and
2. $f(n) = \Omega(g(n))$

Definition: little-o, little-ω

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

Sample Question



- For every n there is some input of size n for which the algorithm uses 2n steps
- What can you say about the running time of the algorithm?

Useful properties

- If $\lim_{n\to\infty} f(n)/g(n) \to c$ then $f(n)=O(g(n))$
- If $f(n)/g(n) \to \infty$ as $n \to \infty$ then $f(n) \neq O(g(n))$

Questions

- Order these functions by increasing rate of growth:

n^2, $\ln^2 n$, e^n, $\ln n$

Transitivity: Another useful property

- If f(n) = O(g(n)) and g(n)=O(h(n)) then f(n)=O(h(n))
- The relations o, Ω, ω, and Θ are also transitive.

Some rules of thumb

- PolyLogs are slower growing than polynomials

$\log^k(n) = O(n^\varepsilon)$ for any $k>0$, $\varepsilon>0$

- Polynomials are slower growing than exponentials

$n^k = O(r^n)$ for any $k>0$, $r>1$

Sample question

Order these functions by increasing rate of growth:

$\lg(2n)$, $\ln^2 n$, $\log(n^3)$, n^3, $n!$, 2^n, $(\lg n)^{\lg n}$

Another useful properties

- If f(n)=O(g(n)) then lg(f(n))=O(lg(g(n)))

$\Updownarrow$

- If $\lg(f(n)) \neq O(\lg(g(n)))$ then $f(n) \neq O(g(n))$

Run time analysis

- Iterative algorithms: loop counting
  - Find a series that describes the running time
  - Solve or bound the series
- Recursive algorithms: recurrence relations
  - Find a recurrence relation that describes the running time
  - Convert the recurrence relation to a series
  - Solve or bound the series

Series
Things we want to do:

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

Sample questions

- Prove that $\Sigma_{i=0\ldots n} 2^i = O(2^n)$.
- Prove that there exist constants c and M such that $\Sigma_{i=0\ldots n} 2^i \leq c2^n$ for all $n \geq M$.

Some useful series

- Arithmetic: $\Sigma_{i=1\ldots n}\, i$
- Generalized arithmetic: $\Sigma_{i=1\ldots n}\, i^k$
- Geometric: $\Sigma_{i=1\ldots n}\, a^i$

Recurrence Relations

Methods to solve or bound:
- Guess and prove
- Unwinding
- Master method
- **WORK TREES**

Sample Problem

- On instance size 1 algorithm **A** performs a constant number of steps.
- On instance size $n=3^m$, $m>0$, algorithm **A** makes 4 recursive calls, each on size n/3. Then **A** performs an additional cn^3 steps to produce it result.
- Analyze the work tree for algorithm **A**.

Algorithms we've seen

- InsertionSort
- MergeSort
- HeapSort
- Linear-time Select
- Find-min&max

Lower Bound for Sorting

Theorem: Any comparison-based sorting algorithms has a worst-case running time that is Ω(n log(n)).

Sample Problem

- Suppose the input is restricted to be a heap. Is sorting still Ω(nlg n)?

Brief detour in our journey: Sorting in O(n)

Can we do it?

Yes – but only if we can make some assumptions about the input.

Some examples:

- Counting-sort
- Radix-sort
- Bucket-sort

Upper and Lower Bounds for Problem ▲

- Upper bound:
  - Problem ▲ can be solved in at most f(n) time
  - Proof: Give an algorithm
- Lower bound:
  - Any algorithm for problem ▲ must take at least g(n) time
  - Proof: Give an adversary