CS5800 – ALGORITHMS

MODULE 1. REVIEW OF ASYMPTOTIC NOTATION

Lesson 2: Asymptotics & Order Notation

Ravi Sundaram

Topics

-
- Asymptotic worst-case complexity
- Tour of common running times
- Big-O
- Big-Ω
- Big-Θ
- Little-o & Little-ω
- Summary

Worst-case, asymptotic

- Why worst-case (for given size/family of inputs)?
  - Different algorithms may be better on different instances
  - Worst-case gives a uniform way to compare
  - Does not require consensus on “average” case

- Why asymptotic (as input size goes to infinity)?
  - Input parameterized by n
  - Care about complexity as $n \to \infty$
  - For scalability

Some common runtimes

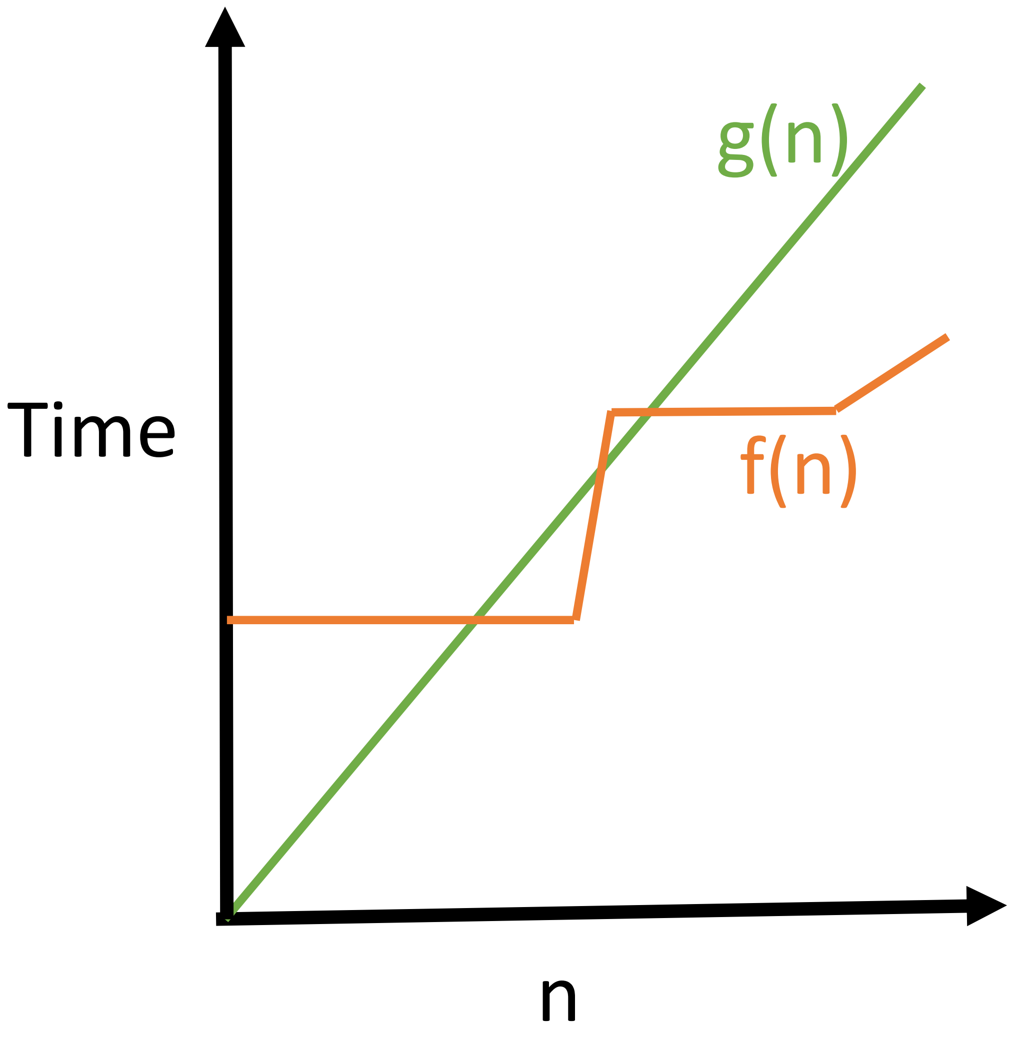
- Linear time
  - The algorithm takes time proportional to the size of the input — time taken is Cn for some constant C
  - Example: Finding the maximum of a set of numbers — keep the biggest number seen so far
- Logarithmic time
  - The time taken is proportional to $C \log_2 n = C \lg n$ for some constant C
  - Because different log bases are different only by a constant, we
  - typically omit the base: $C \log n$
  - Example: Binary search on a sorted list of numbers, throws out half the data at each step

Other common runtimes

- Polynomial time
  - A broad category that includes constant, linear, logarithmic, and quadratic time, as well as any time bounded by Cn^k for some constant k
  - Often considered theoretically “efficient” or “tractable”; n^{100} would be terrible in practice, but such exponents don’t tend to arise naturally
- Exponential time
  - The size of the input is in the exponent, for example, 2^n
  - Is terrible - if n >= 50 or so for all practical purposes same as running forever
  - Often arises when you’re trying all possible combinations of things
  - Example: try all possible numbers in a sudoku puzzle
  - Example: factoring by trying all possible factors
- Wealth of other possible running times e.g. $n^{\log n}$, $2^{n!}$

Big-O

- f = O(g(n)) if, for some positive c and n_0, $f(n) \le cg(n)$ for all $n \ge n_0$
- Core idea: f(n) = O(g(n)) if $f(n) \le cg(n)$ as n gets large
- f(n) can be greater than g(n) for a little while, as long as g(n) passes it in the long run.
- This allows us to perform expensive setups that pay off in the long term. Also constants don't matter.



Big-O

- Upper bound of function relations
- Big-O is analogous to ≤.
- It holds when two growth rates are essentially equal:
  - $2n = O(n)$. (Both linear)

It holds when the first growth rate is asymptotically less than the second:
  - $2n = O(2^n)$. (Linear vs exponential)
- Always put big-O on the right of the equals sign: $5n = O(n)$ – is saying that the function on the LHS is a member of the category on the RHS.
- Most commonly used of the bounds because with algorithms, we usually want an upper bound on the worst case running time.

Big-Omega Ω

- Lower bound of function relations
- Ω is analogous to $\geq$.

- Used when we may want to say, “This algorithm must take at least this much time”

- $f = \Omega(g(n))$ if, for some positive c and n_0, $f(n) \geq cg(n)$ for all $n \geq n_0$

- $f = \Omega(g(n))$ iff $g = O(f(n))$

Theta Θ

- Equality of function relations
- Θ is analogous to =.
- Used when we may want to say, “This is the exact growth rate of the algorithm’s time complexity”
- f = Ω(g(n)) and f = O(g(n)) => f = $\Theta(g(n))$

Little-o & Little-ω

- If we want to say one growth rate is strictly faster or slower than another, we use little-o and little-ω:
- $n = o(n^2)$
- $n = \omega(\log n)$
- These are analogous to < and > for growth rates.
- The technical definition of little-o is $f(n) = o(g(n))$ if $\text{Lim}_{n\to\infty} f(n)/g(n) = 0$
- Similarly, $f(n) = \omega(g(n))$ if the limit is infinite.

Summary

- We describe algorithm speed by the growth in number of operations required as a function of n, the input size. We want that function to grow as slowly as possible!
- big-O: an upper bound on a growth rate, ignoring constants
- big-Ω: a lower bound on a growth rate, ignoring constants
- big-Θ: a tight bound on a growth rate, ignoring constants
- o, ω: "strictly less than," "strictly greater than"
- Each polynomial degree $\Theta(n^k)$ is its own category of growth rate
- A problem is efficiently solvable or tractable when it has a polynomial time algorithm