CIT 596: ALGORITHMS & COMPUTATION

Asymptotic Notation

Analysis of Algorithms

- Inputs to algorithms have **length.**
  - Naturally, an algorithm might take more time on longer inputs.
- Let $f(n)$ be the running time (number of steps) for some algorithm on inputs of length n.
  - But ***which*** inputs of length n?
- **Worst-case complexity:** $f(n)$ is the ***maximum*** time the algorithm takes on inputs of length n.
  - We don't control which inputs the algorithm will be run on!

Comparing Growth Rates of Functions

If we have functions $f(n)$ and $g(n)$ describing the worst-case running time of two algorithms, how do we know which one is better?

- *Ignore constant factors*

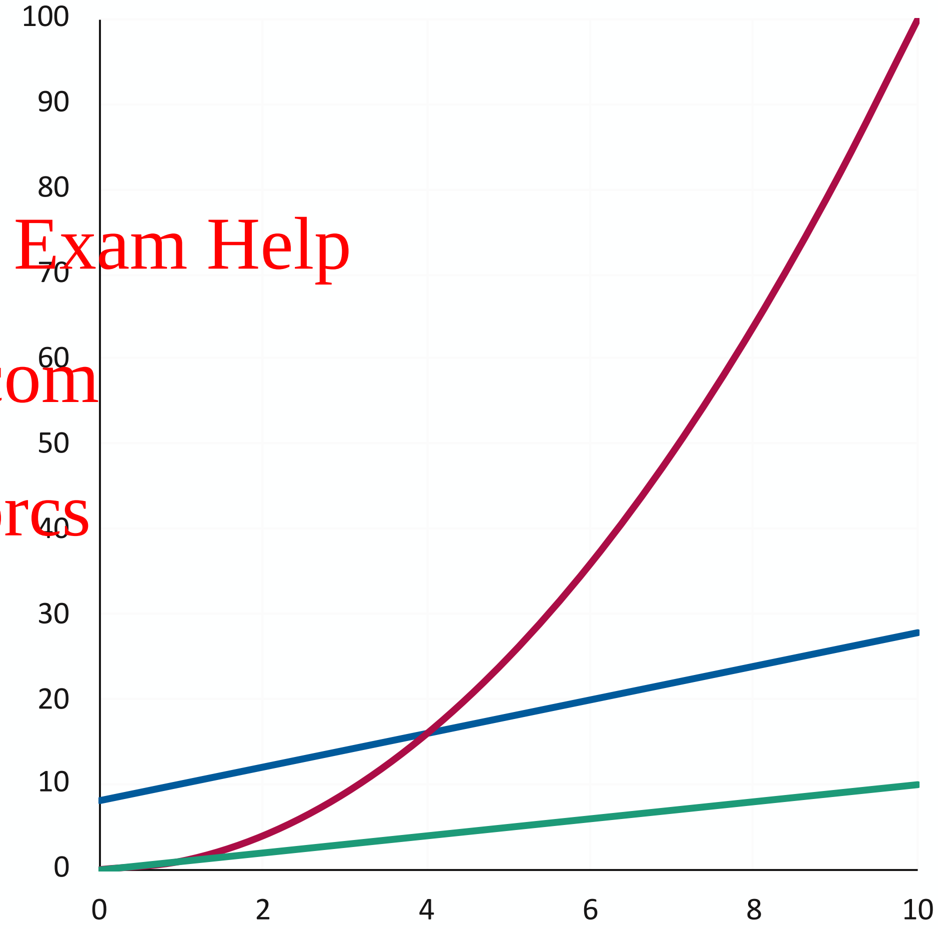
- *Ignore slower-growing terms*

To express *The function f grows no faster than the function g*:

- Say "$f(n)$ is **big O** of $g(n)$."
- Write either $f(n) = O\big(g(n)\big)$ or $f(n) \in O(g(n))$.

Examples

- $n = O(n^2)$
- $2n + 8 = O(n^2)$
- $\cancel{2}n + \cancel{8} = O(n)$
- $n^2 \neq O(n)$



Defining Big O

Definition: $f(n) = O\big(g(n)\big)$ if there exist positive constants n_0 and c such that for all $n \geq n_0$,

$$f(n) \leq c \cdot g(n).$$

Informally, "f is ***eventually*** at most a constant multiple of g."

Remember, a **constant** is a number that does not depend on n.

Examples

Valid choices of n_0 and c are not unique!

- $n = O(n^2)$
  - For all $n \geq 1$, $n \leq 1 \cdot n^2$. $1 \leq n$
- $2n + 8 = O(n^2)$
  - For all $n \geq 4$, $2n + 8 \leq 1 \cdot n^2$. $n^2 - 2n - 8 \geq 0$ $(n+2)(n-4) \geq 0$
- $2n + 8 = O(n)$
  - For all $n \geq 8$, $2n + 8 \leq 3 \cdot n$. $8 \leq n$
- $n^2 \neq O(n)$ $n = \max(n_0, c+1) \Rightarrow n^2 \geq (c+1)n > cn$
  - No matter how you choose c and n_0, I can find an $n \geq n_0$ such that

$$n^2 > c \cdot n.$$

Asymptotic Lower Bounds

Big-O notation is a way to express that an algorithm is fast. What if we want to say that it is slow?

Definition: $f(n) = \Omega(g(n))$ ("$f(n)$ is **big omega** of $g(n)$") if $g(n) = O(f(n))$.

Definition: $f(n) = \Theta(g(n))$ ("$f(n)$ is **big theta** of $g(n)$") if $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

- $5n = \Omega(n)$
- $n^2 + 3n = \Omega(n)$
- $5n \neq \Omega(n^2)$

- $5n = \Theta(n)$
- $n^2 + 3n \neq \Theta(n)$
- $5n \neq \Theta(n^2)$

The Limit of the Ratio

Theorem: If

$$\lim_{n\to\infty} \frac{f(n)}{g(n)}$$

is defined, then

$f(n) = O\big(g(n)\big) \Leftrightarrow$ the limit is finite, and

$f(n) = \Omega\big(g(n)\big) \Leftrightarrow$ the limit is positive.

- $n = O(n^2), n \neq \Omega(n^2)$
- $2n + 8 = \Theta(n)$
- $3n^5 + n \neq O(n), 3n^5 + n = \Omega(n)$

Multiplying and Adding with Big O

You will often need to multiply or add the running time of different components of an algorithm. Some useful rules:

- $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$

for $i = 1$ **to** $6n$
  for $j = 0$ **to** $3n^2 + 4$
    print "foo"

$O(n)$
$\times\, O(n^2)$
$= O(n^3)$

- If $f(n) = O(g(n))$, then
  $O(f(n)) + O(g(n)) = O(g(n))$.

for $i = 1$ **to** $2n$
  print "foo"
for $i = 1$ to n
  for $j = 1$ **to** n
    print "bar"

$O(n)$
$+\, O(n^2)$
$= O(n^2)$