Recall the formal definition of Big-O notation:

A function \( f(n) \) is \( O(g(n)) \) if there exist positive constants \( c \) and \( n_0 \) such that \( f(n) \leq cg(n) \) for all \( n \geq n_0 \)

Example

Let’s show that \( f(n) = 10n + 100 \) is \( O(g(n)) \) where \( g(n) = n \)

We need to give constants \( c \) and \( n_0 \) such that \( f(n) \leq cg(n) \) for all \( n \geq n_0 \)

In other words, we need constants \( c \) and \( n_0 \) such that \( 10n + 100 \leq cn \) for all \( n \geq n_0 \)
Example

- We can solve for appropriate constants:
  \[ 10n + 100 \leq cn \quad (1) \]
  \[ 10 + 100/n \leq c \quad (2) \]
- So if \( n > 1 \), then \( c \) should be greater than 110.
- In other words, for all \( n > 1 \), \( 10n + 100 \leq 110n \)
- So \( 10n + 100 \) is \( O(n) \)

Relatives of big-O

Recall the following relatives of big-O:

- \( O “\leq” \) This algorithm is \( O(n^2) \) (i.e. worst case is \( \Theta(n^2) \))
- \( \Theta “\equiv” \) This algorithm is \( \Theta(n) \) (best and worst case are \( \Theta(n) \))
- \( \Omega “\geq” \) Any comparison-based algorithm for sorting is \( \Omega(n \log n) \)
- \( o “<” \) Can you write an algorithm for sorting that is \( o(n^2) \)?
- \( \omega “>” \) This algorithm is not linear, it can take time \( \omega(n) \)

Rule of Thumb

- Let \( f(n), g(n) \) be two functions of \( n \)
- Let \( f_1(n) \), be the fastest growing term of \( f(n) \), stripped of its coefficient.
- Let \( g_1(n) \), be the fastest growing term of \( g(n) \), stripped of its coefficient.

Then we can say:

- If \( f_1(n) \leq g_1(n) \) then \( f(n) = O(g(n)) \)
- If \( f_1(n) \geq g_1(n) \) then \( f(n) = \Omega(g(n)) \)
- If \( f_1(n) = g_1(n) \) then \( f(n) = \Theta(g(n)) \)
- If \( f_1(n) < g_1(n) \) then \( f(n) = o(g(n)) \)
- If \( f_1(n) > g_1(n) \) then \( f(n) = \omega(g(n)) \)
More Examples

The following are all true statements:

- $\sum_{i=1}^{n} i^2$ is $O(n^3)$, $\Omega(n^3)$ and $\Theta(n^3)$
- $\log n$ is $o(\sqrt{n})$
- $\log n$ is $o(\log^2 n)$
- $10,000n^2 + 25n$ is $\Theta(n^2)$

Formal Defns

- $O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0 \}$
- $\Theta(g(n)) = \{ f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}$
- $\Omega(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0 \}$

Problems

True or False? (Justify your answer)

- $n^3 + 4$ is $\omega(n^2)$
- $n \log n^3$ is $\Theta(n \log n)$
- $\log^3 5n^2$ is $\Theta(\log n)$
- $10^{-10}n^2 + n$ is $\Theta(n)$
- $n \log n$ is $\Omega(n)$
- $n^3 + 4$ is $o(n^4)$

Formal Defns (II)

- $o(g(n)) = \{ f(n) : \text{for any positive constant } c > 0 \text{ there exists } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0 \}$
- $\omega(g(n)) = \{ f(n) : \text{for any positive constant } c > 0 \text{ there exists } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0 \}$
Another Example

- Let \( f(n) = 10 \log^2 n + \log n \), \( g(n) = \log^2 n \). Let’s show that \( f(n) = \Theta(g(n)) \).
- We want positive constants \( c_1, c_2 \) and \( n_0 \) such that \( 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \) for all \( n \geq n_0 \)
  
\[
0 \leq c_1 \log^2 n \leq 10 \log^2 n + \log n \leq c_2 \log^2 n
\]

Dividing by \( \log^2 n \), we get:

\[
0 \leq c_1 \leq 10 + 1/ \log n \leq c_2
\]

- If we choose \( c_1 = 1 \), \( c_2 = 11 \) and \( n_0 = 2 \), then the above inequality will hold for all \( n \geq n_0 \)

In-Class Exercise

Show that for \( f(n) = n + 100 \) and \( g(n) = (1/2)n^2 \), that \( f(n) \neq \Theta(g(n)) \)

- What statement would be true if \( f(n) = \Theta(g(n)) \)?
- Show that this statement can not be true.

Recurrence Relation Review

“Oh how should I not lust after eternity and after the nuptial ring of rings, the ring of recurrence” - Friedrich Nietzsche, Thus Spoke Zarathustra

- \( T(n) = 2 \cdot T(n/2) + n \) is an example of a recurrence relation
- A Recurrence Relation is any equation for a function \( T \), where \( T \) appears on both the left and right sides of the equation.
- We always want to “solve” these recurrence relation by getting an equation for \( T \), where \( T \) appears on just the left side of the equation

Recurrence Relations

- Whenever we analyze the run time of a recursive algorithm, we will first get a recurrence relation
- To get the actual run time, we need to solve the recurrence relation
Substitution Method

- One way to solve recurrences is the substitution method aka “guess and check”
- What we do is make a good guess for the solution to $T(n)$, and then try to prove this is the solution by induction

Example

- Let’s guess that the solution to $T(n) = 2 \cdot T(n/2) + n$ is $T(n) = O(n \log n)$
- In other words, $T(n) \leq cn \log n$ for all $n \geq n_0$, for some positive constants $c, n_0$
- We can prove that $T(n) \leq cn \log n$ is true by plugging back into the recurrence

Todo

- Read Syllabus
- Visit the class web page: www.cs.unm.edu/~saia/362/
- Sign up for the class mailing list (cs362)
- Read Chapter 3 and 4 in the text

Proof

- We prove this by induction, By I.H.: $T(n/2) \leq cn/2 \log(n/2)$
  
  $$T(n) = 2T(n/2) + n$$
  $$\leq 2(cn/2 \log(n/2)) + n$$
  $$= cn \log(n/2) + n$$
  $$= cn \log n - cn + n$$
  $$\leq cn \log n$$

last step holds for all $n > 0$ if $c \geq 1$