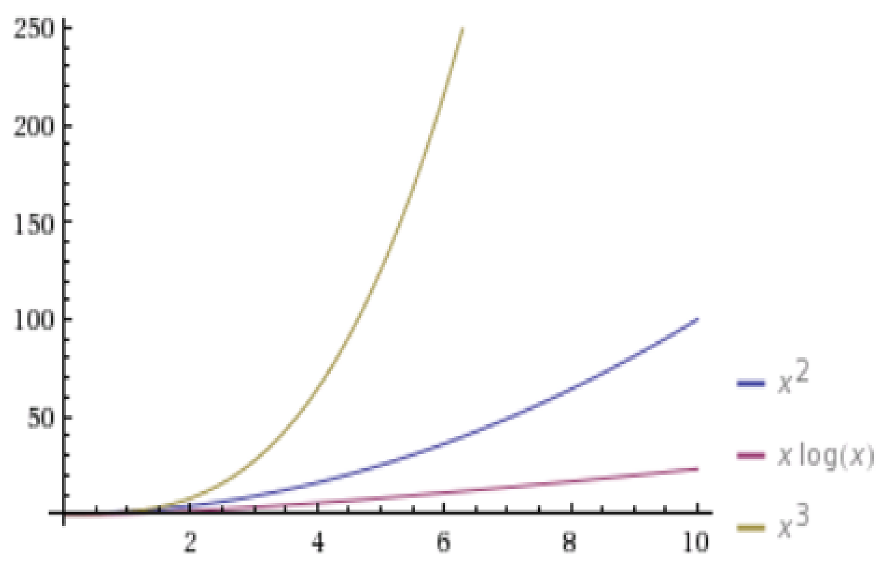
CS 325 – Asymptotic Analysis

Week 1 Part 1 Big-Oh, Omega, Theta

Problems vs Algorithms

- For any given problem there are potentially many different types of algorithms to solve it.
- Problem: Sorting a list of integers
- Algorithms: Insertion Sort, Merge Sort, Naive Sort
- Running time
  - Insertion Sort is $O(n^2)$
  - Merge Sort is O(nlgn)
  - Naive Sort is $O(n^3)$

Plot:



Benchmarking

- Algorithmic analysis is the first and best way, but not the final word
- What if two algorithms are both of the same complexity?
- Example: bubble sort and insertion sort are both $O(n^2)$
  - So, which one is the "faster" algorithm?
  - Benchmarking: run both algorithms on the same machine
  - Often indicates the constant multipliers and other "ignored" components
  - Still, different implementations of the same algorithm often exhibit different execution times – due to changes in the constant multiplier or other factors (such as adding an early exit to bubble sort)

Constant factors and domination

Suppose we have two algorithms with exact running times of:

Algorithm 1		Algorithm 2
$1{,}000{,}000 \cdot n$	versus	$2 \cdot n^2$

Is it reasonable to say that runtime of Algorithm 2 dominates (is worse) than Algorithm1?

NO for small values of n

Yes for large values of n

How do we compare algorithms?

We need to define a number of objective measures.

(1) Compare execution times?

Not good: times are specific to a particular computer and programming language!!

(2) Count the number of statements executed?

Not good: number of statements vary with the programming language as well as the style of the individual programmer.

Asymptotic Analysis

To compare two algorithms with running times *f(n)* and *g(n)*, we need a **rough measure** that characterizes **how fast each function grows.**

- Running time of an algorithm as a function of input size *n* ***for large n***.
- Compare functions in the limit, that is, **asymptotically!** (i.e., for large values of *n*)
- Worst Case Analysis

Input Size

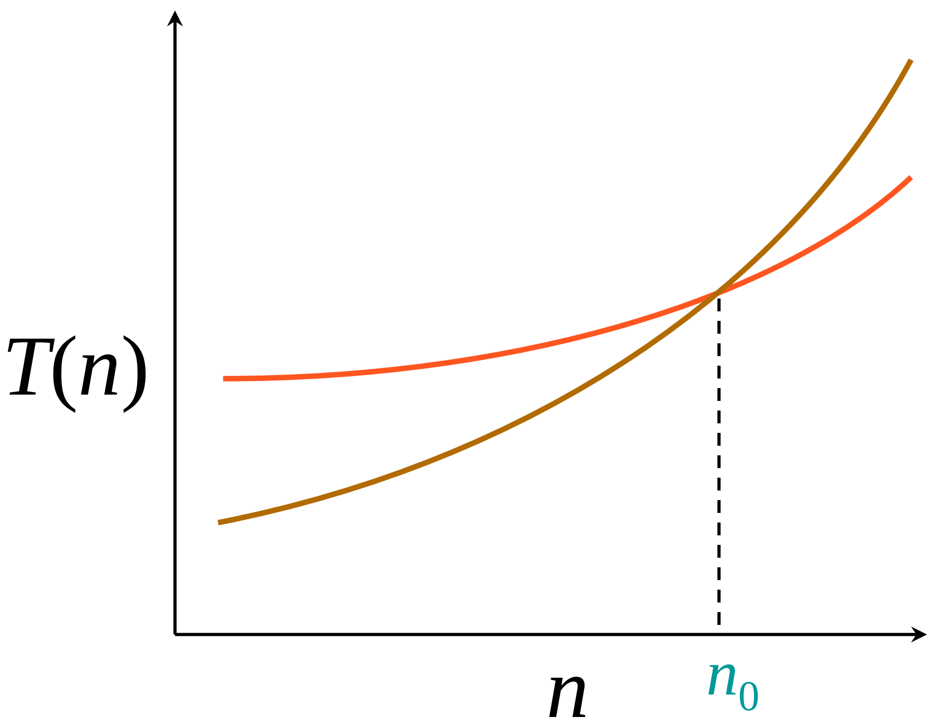
- Express running time as a function of the input size *n* (i.e., f(n)).
  - size of an array
  - # of elements in a matrix
  - # of bits in the binary representation of the input
  - vertices and edges in a graph

Types of Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute **guarantee** that the algorithm would not run longer, no matter what the inputs are
- Best case
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest
- Average case = Expected Value
  - Provides a prediction about the running time
  - Assumes that the input is random

Asymptotic performance

When n gets large enough, an n^2 algorithm always "*beats*" a n^3 algorithm.

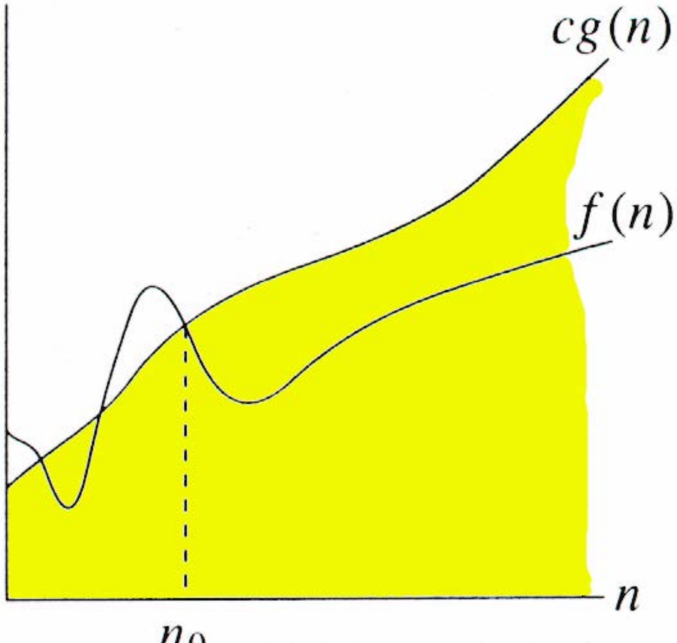


Asymptotic Notation

- Big-Oh - O notation: asymptotic "less than":
  - $f(n)=O(g(n))$ implies: $f(n)$ "$\leq$" $g(n)$
- Omega - Ω notation: asymptotic "greater than":
  - $f(n)= \Omega (g(n))$ implies: $f(n)$ "$\geq$" $g(n)$
- Theta - Θ notation: asymptotic "equality":
  - $f(n)= \Theta (g(n))$ implies: $f(n)$ "$=$" $g(n)$

O-notation

$O(g(n)) = \{f(n) :$ there exist positive constants c and n_0 such that
$0 \le f(n) \le cg(n)$ for all $n \ge n_0\}$.



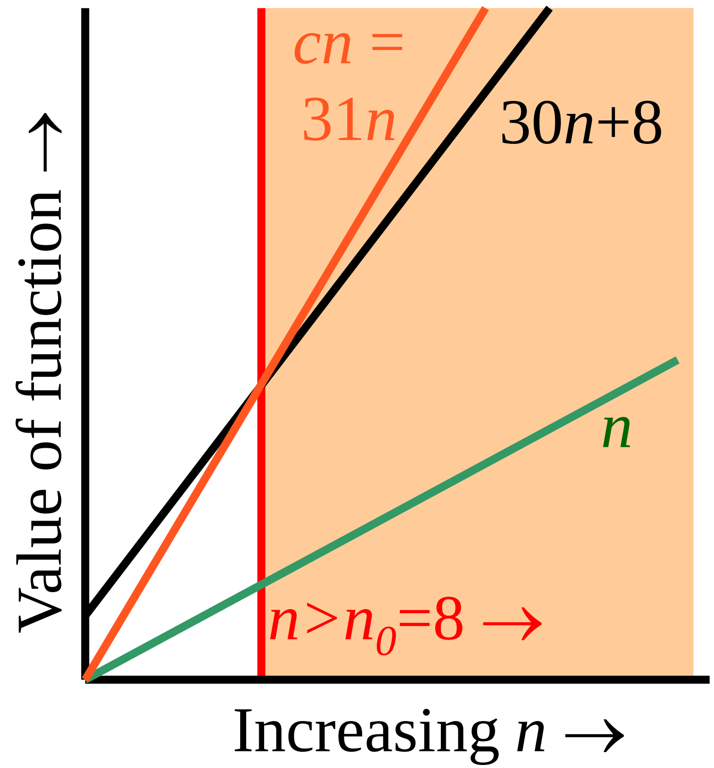
$f(n) = O(g(n))$

$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

Big-O example, graphically

- Note $30n+8$ isn't less than n *anywhere* ($n>0$).
- It isn't even less than $31n$ *everywhere*.
- But it *is* less than $31n$ <u>everywhere to the right of $n=8$</u>.

$30n+8$ is $O(n)$



Not Unique - Example

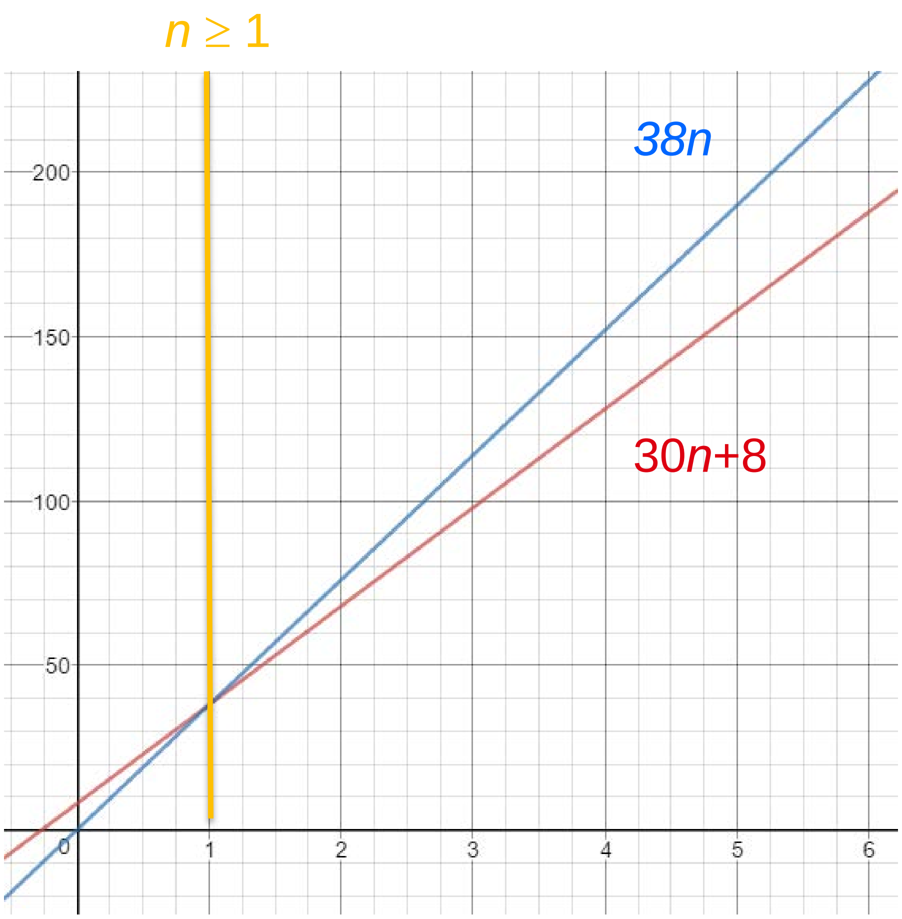
Show that $30n+8$ is $O(n)$.

Show $\exists c,n_0$: $30n+8 \le cn$, $\forall n \ge n_0$.

Let c=38, n_0=1. Assume $n \ge n_0$=1.

Then $cn = 38n = 30n + 8n \ge 30n + 8$, so

$30n+8 \le 38n$. when $n \ge 1$



A Simple Code Example

- Consider summing an array of n integers.

```
sum = 0;
for (i = 0; i < n; i++)
    sum += array[i];
return sum;
```

Executes in constant time c_1 (independent of n)

Executes in $c_2 \cdot n$ time for for some constant c_2

Executes in constant time c_3 (independent of n)

- Total running time: $c_1 + c_2 n + c_3$
  - But the constants c_1, c_2, c_3 depend on hardware, compiler, etc.
- What is the big-Oh runtime? (big-Oh ignores factors)

O(n) also known as linear time

A Simple Example

- Consider summing an array of n integers.

```
sum = 0;
for (i = 0; i < n; i++)
    sum += array[i];
return sum;
```

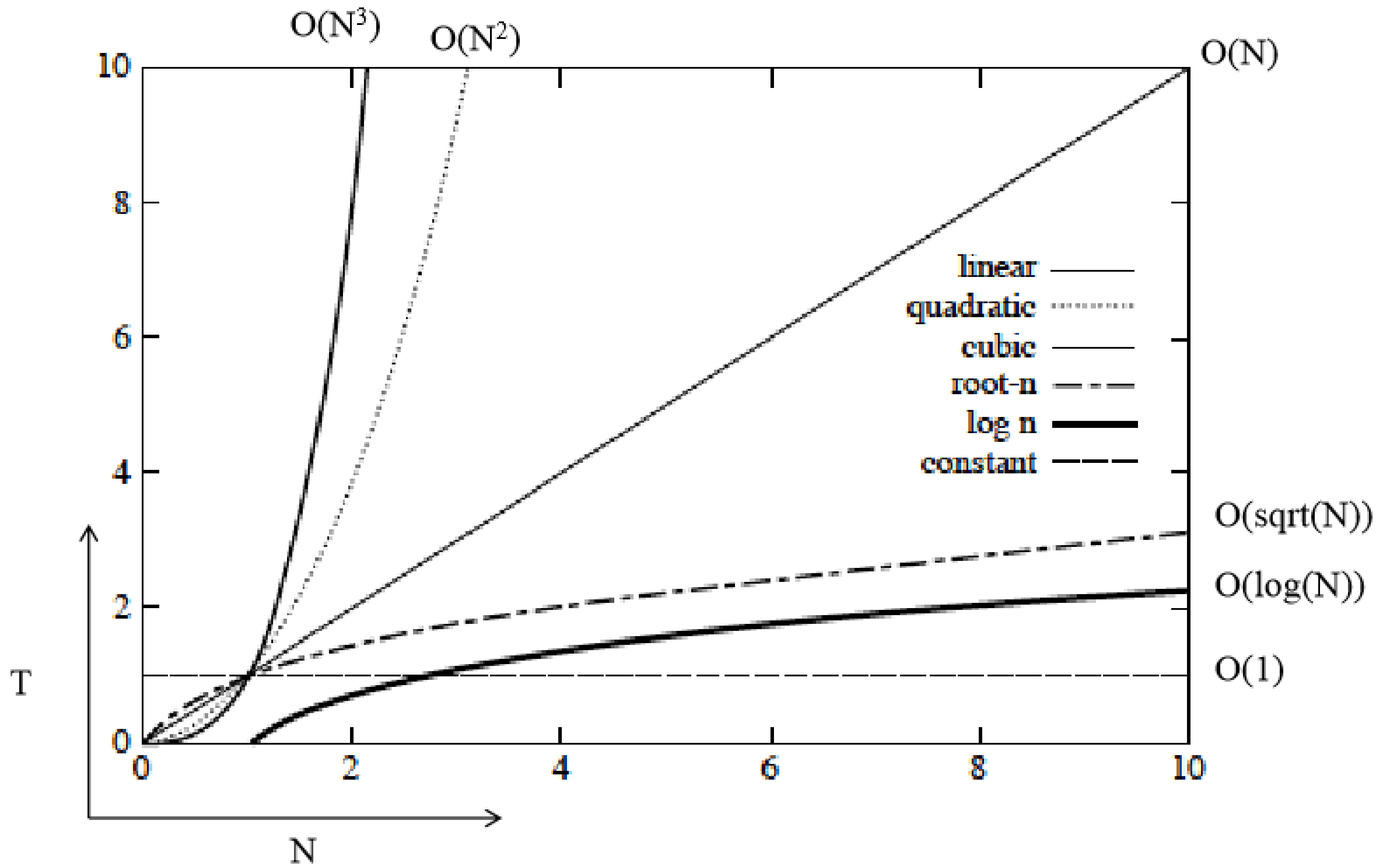
O(n)

Non-Linear Times

- Consider the BubbleSort Algorithm
  - Let n be the size of the input list to be sorted
  - Runtime is $O(n^2)$, also known as quadratic time.
- Suppose size doubles, what happens to execution time?

- It goes up by a factor of 4. Why?

Orders of Growth



Big Oh Classes

- Constant $O(1)$
- Logarithmic $O(\log (n))$
- Linear $O(n)$
- Quadratic $O(n^2)$
- Cubic $O(n^3)$
- Polynominal $O(n^k)$ for any $k>0$
- Exponential $O(k^n)$, where $k>1$
- Factorial $O(n!)$

Rank the following functions in increasing order of growth

- 10, logn, $lg(2^n)$, $3n^2$, n!

A polynomial of degree k is $O(n^k)$

Recall: f(n) is O(g(n)) if there exist positive constants c and n_0 such that f(n) $\leq c \cdot g(n)$ for all $n \geq n_0$

Proof:

Suppose $f(n) = b_k n^k + b_{k-1} n^{k-1} + \ldots + b_1 n + b_0$

Let $a_i = | b_i |$

$$f(n) \leq a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$$

$$f(n) \leq n^k \left(a_k + a_{k-1} \frac{n^{k-1}}{n^k} + \ldots + a_1 \frac{n^1}{n^k} + a_0 \frac{1}{n^k} \right)$$

$$f(n) \leq \quad n^k \sum a_i \frac{n^i}{n^k} \leq n^k \sum a_i$$

$$let \;\; c = \sum a_i$$

$$f(n) \leq \quad cn^k \quad \text{for n} \geq 1$$

Therefore all polynomial functions *f(n)* of degree k are $O(n^k)$.

What does this mean in practice?

```
for(s = 0; s < N; s++)
      sum = sum + 1;
```

$O(n)$

```
for(i = 0; i < N; i++)
  for(j =0; j < N; j++)
      printf( …)
```

$O(n^2)$

Total = $O(n) + O(n^2) = O(n + n^2) = O(n^2)$

Code Example

```
int isPrime (int n) {
  for (int i = 2; i * i <= n; i++) {
    if (0 == n % i) return 0;
  }
  return 1; /* 1 is true */
}
```

What is the “Big-Oh” running time in terms of n?

$$O(\sqrt{n})$$

Trouble with Big-Oh

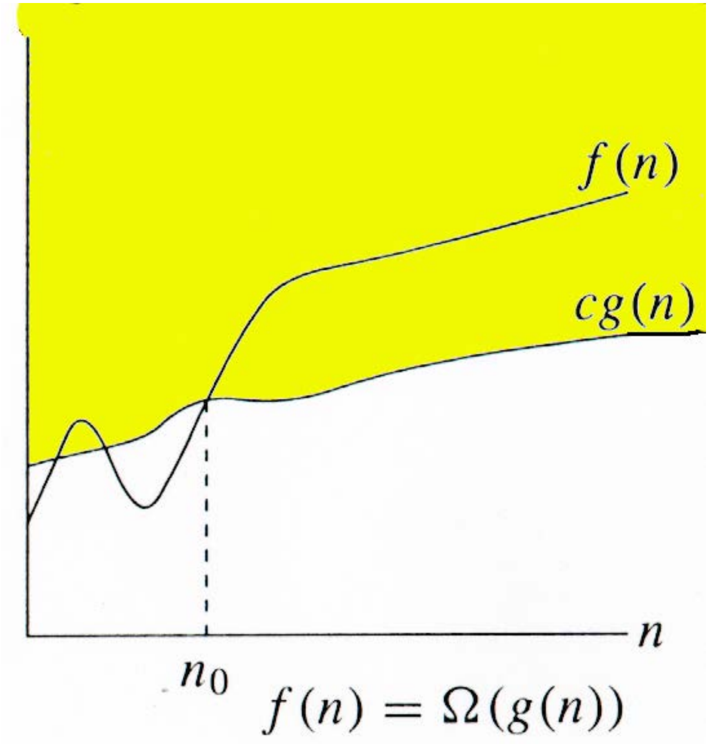
Just an upper bound. Factually true but practically meaningless.

- 3n is $O(n^2)$
- 3n is $O(n^4)$
- 3n is $O(n)$

Many times only Big-Oh is reported but it is assumed a “tight” upper bound.

Omega Ω -notation

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\} .$$



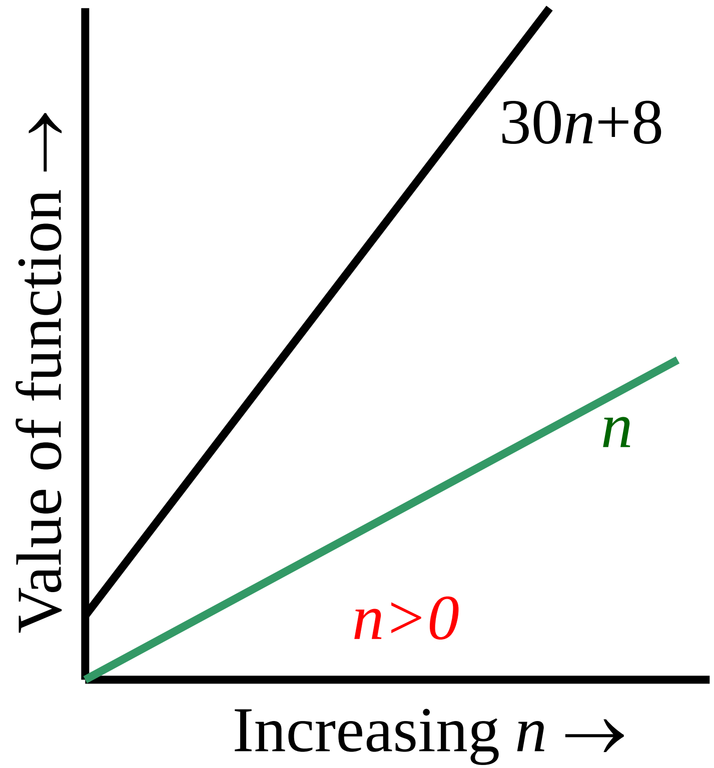
$\Omega(g(n))$ is the set of functions with larger or same order of growth as $g(n)$

$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Omega Graphically

- Note $30n+8$ isn't less than n *anywhere* ($n>0$).

$30n+8$ is $\Omega(n)$



Examples

$5n^2 = \Omega(n)$

$\exists$ c, n_0 such that: $0 \le cn \le 5n^2$

$\Rightarrow cn \le 5n^2$

$\Rightarrow c = 5$ and $n_0 = 1$

$5n^2+10= \Omega(n^2)$

$\exists$ c, n_0 such that: $0 \le cn^2 \le 5n^2+10$

$\Rightarrow 5n^2c \le 5n^2 +10$

$\Rightarrow c = 1$ and $n_0 = 1$

Property of Big-Oh and Omega

If $f(n) = O(g(n))$ then $g(n) = \Omega(f(n))$

By definition of Big-Oh

$$f(n) \leq cg(n) \text{ for all } n \geq n_0 \text{ for some } n_0, c > 0.$$

Dividing by c yields

$$\frac{1}{c} f(n) \leq g(n) \text{ or } c_2 f(n) \leq g(n) \text{ where } c_2 = \frac{1}{c} \geq 0$$

If we use the same $n_{0.}$ this implies that $g(n) = \Omega(f(n))$.

logn vs n

Show that $n = \Omega(\log n)$ which will imply that $\log n = O(n)$

Start with showing that $2^n = \Omega(n)$.

Find c and n_0 such that $cn \leq 2^n$ for $n \geq n_0$.

$2n \leq 2^n$ for $n \geq 2$. *(this could be proved by induction too)*

So $cn \leq 2^n$ take the log of both sides

$\lg(cn) \leq \lg(2^n)$

$\lg(c) + \lg(n) \leq n$ so

$1 * \lg(n) \leq n$ for $n \geq 2$

Therefore $n = \Omega(\log n)$ and $\log n = O(n)$

A non-negative polynomial of degree k is $\Omega(n^k)$

Proof:

Suppose $f(n) = b_k n^k + b_{k-1} n^{k-1} + \ldots + b_1 n + b_0$

$$f(n) = n^k \left(b_k + b_{k-1} \frac{n^{k-1}}{n^k} + \ldots + b_1 \frac{n^1}{n^k} + b_0 \frac{1}{n^k} \right)$$

$$f(n) = n^k \left(b_k + \frac{b_{k-1}}{n^1} + \ldots + \frac{b_1}{n^{k-1}} + \frac{b_0}{n^k} \right)$$

For large n's the fractions go to zero, so if we set n_0 large enough we can ignore all terms except b_k . Thus a value for n_0 must exist.

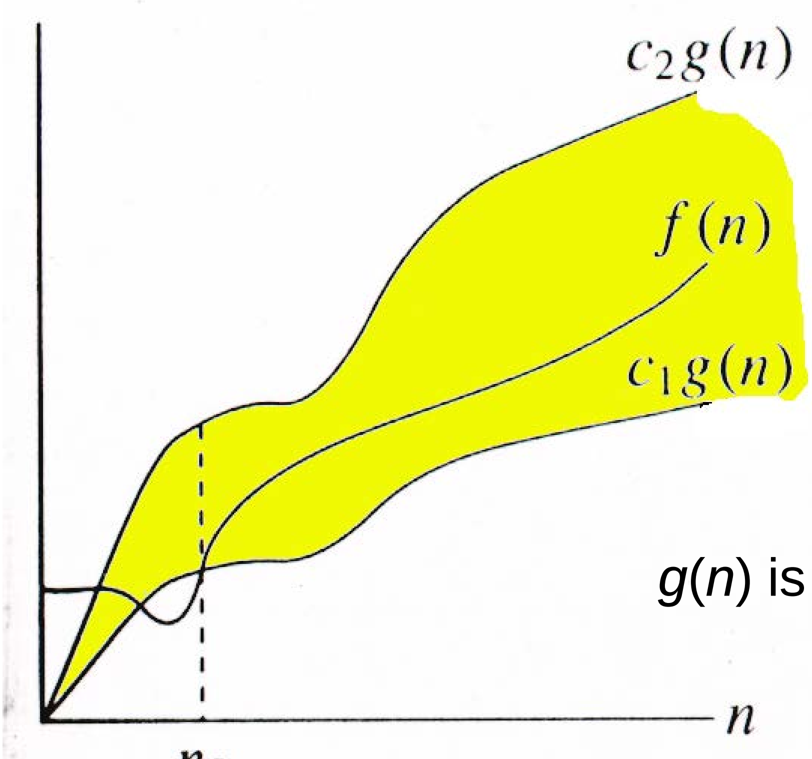
$$let\ \ c = \frac{b_k}{2} \qquad\qquad cn^k \le f(n) \text{ for } n \ge n_0$$

$$\frac{b_k}{2} n^k \le b_k n^k \text{ for } n \ge n_0$$

Therefore all polynomial functions $f(n)$ of degree k are $\Omega(n^k)$.

Θ-notation

$\Theta(g(n)) = \{f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.



$f(n) = \Theta(g(n))$

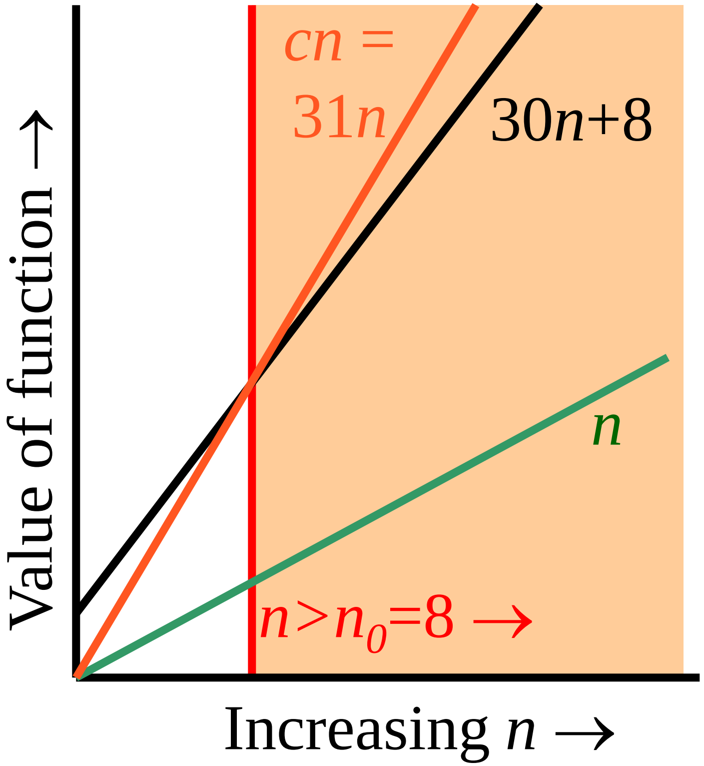
$\Theta(g(n))$ is the set of functions with the same order of growth as $g(n)$

$g(n)$ is an *asymptotically tight bound* for $f(n)$.

Big-Theta example, graphically

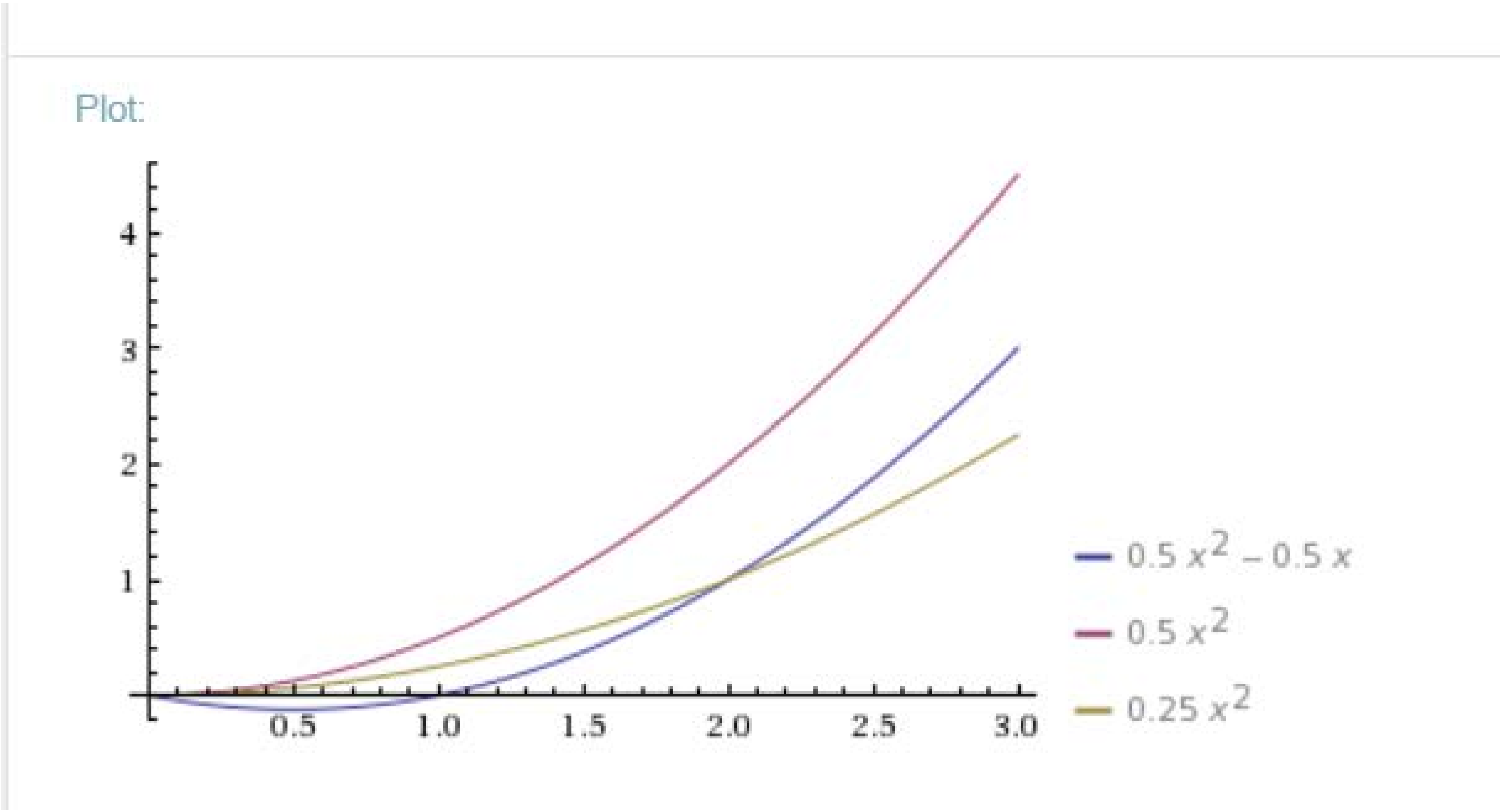
- Note $30n+8$ isn't less than n *anywhere* ($n>0$).
- It isn't even less than $31n$ *everywhere*.
- But it *is* less than $31n$ <u>everywhere to the right of $n=8$</u>.

$30n+8 \in \Theta(n)$



$f(n) = \frac{1}{2} n^2 - \frac{1}{2}n = \Theta(n^2)$

$\Theta(g(n)) = \{ f(n) :$ there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0 \}$



A non-negative polynomial of degree *k* is $\Theta(n^k)$

Proof: From previous Big-Oh and Omega

$f(n) = b_k n^k + b_{k-1} n^{k-1} + \ldots + b_1 n + b_0$ *is* $\Theta(n^k)$

Short-cut:

- Drop low-order terms; ignore leading constants.
- Example: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

For the functions f(n)=logn and g(n)=lgn. Which is true?

f(n) is O(g(n))

f(n) is Θ(g(n))

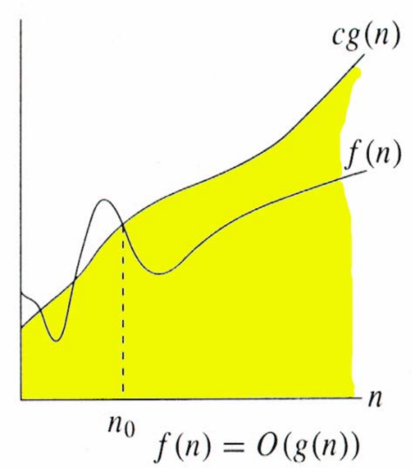
f(n) is Ω(g(n))

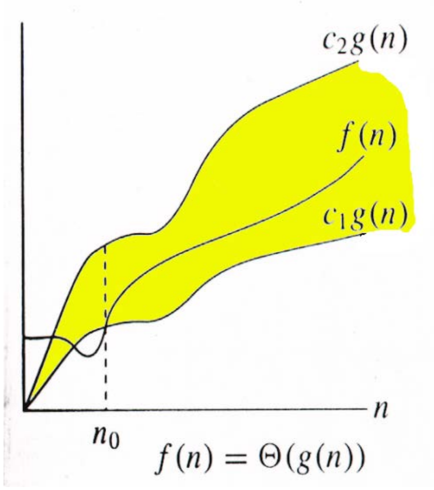
All of the above

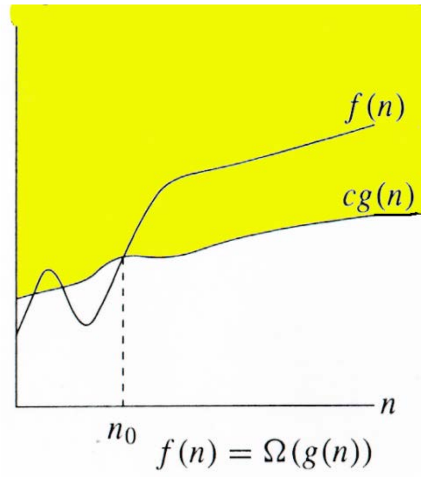
$$lgn = log_2 n = \frac{logn}{log2} = c_1\ logn$$

$$logn = log2(lgn) = c_2\ lgn$$

Relations Between Θ, *O*, Ω



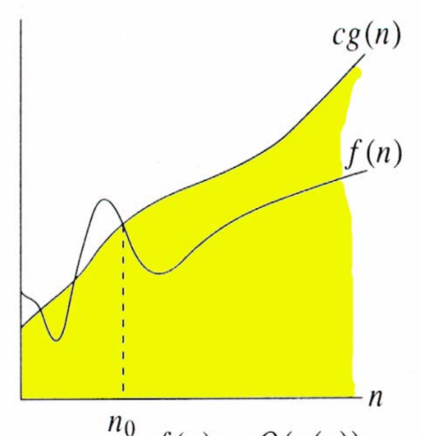




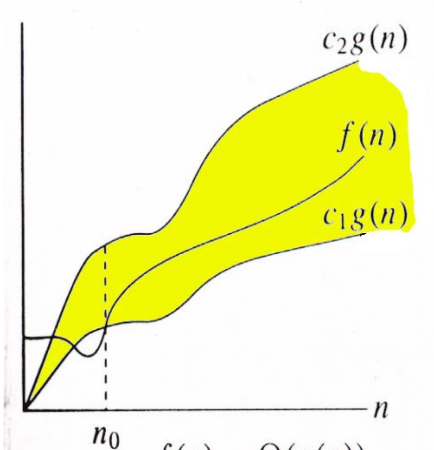
CS 325 – Asymptotic Analysis

Week 1 Part 2: Limits

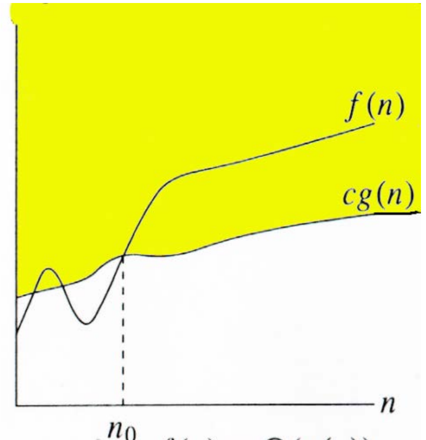
Relations Between Θ, *O*, Ω



$f(n) = O(g(n))$



$f(n) = \Theta(g(n))$



$f(n) = \Omega(g(n))$

Rate of Growth

The low order terms in a function are relatively insignificant for **large** *n*

$$n^4 + 100n^2 + 10n + 50 \quad \sim \quad n^4$$

$$\lim_{n\to\infty} \frac{n^4 + 100n^2 + 10n + 50}{n^4} = 1$$

That is we say that $n^4 + 100n^2 + 10n + 50$ and n^4 have the same **rate of growth**

Mathematics a **tilde symbol** (~) indicating equivalency or similarity between two values.

Limit Method: The Process

Say we have functions f(n) and g(n). We set up a limit quotient between f and g as follows

If lim_n

L’Hôpital Rule

Theorem (L’Hôpital Rule):

- Let f and g be two functions,
- if the limit between the quotient f(n)/g(n) exists,
- Then, it is equal to the limit of the derivative of the numerator and the denominator

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

Limit Method: Example 1

- Example: Let $f(n) = 2^n$, $g(n) = 3^n$. Determine a tight inclusion of the form $f(n) \in \Delta(g(n))$
- What is your intuition in this case? Which function grows quicker?

Limit Method: Example1

Using algebra

$$\lim_{n\to\infty} 2^n/3^n = \lim_{n\to\infty} (2/3)^n$$

Now we use the following Theorem

$$_{\to\infty}\ \alpha \quad \begin{cases} & \alpha \\ & \alpha \\ \infty & \alpha \end{cases}$$

$$\lim_{n\to\infty} 2^n/3^n = \lim_{n\to\infty} (2/3)^n$$

$$= 0$$

Using Wolfram Alpha

https://www.wolframalpha.com/

lim(x->inf)(2^x/3^x)

Limit Method: Example 2

Example: Let $f(n) = \log_2 n$, $g(n)=\log_3 n^2$. Determine a tight inclusion of the form

$$f(n) \in \Delta(g(n))$$

What is your intuition in this case?

Limit Method: Example 2

We set up our limit

$$\lim_{n\to\infty} f(n)/g(n) = \lim_{n\to\infty} \log_2 n/\log_3 n^2$$
$$= \lim_{n\to\infty} \log_2 n/(2\log_3 n)$$

$$\lim_{n\to\infty} \log_2 n/(2\log_3 n) = \lim_{n\to\infty} (\log_2 n \log_2 3) /(2\log_2 n)$$
$$= \lim_{n\to\infty} (\log_2 3)/2$$
$$= (\log_2 3) /2$$
$$\approx 0.7924, \text{ a positive constant}$$

So we conclude that $f(n) \in \Theta(g(n))$ that is
$\log_2 n \in \Theta(\log_3 n^2)$ which implies that $\log_3 n^2 \in \Theta(\log_2 n)$

Important Result

- All logs have the same asymptotic growth rate no what the base is.
- In many CS algorithms the base is 2.
- But we get sloppy since lg(n) is Θ(logn)

Properties

Properties

Theorem:

$$f(n) = \Theta(g(n)) \Leftrightarrow f = O(g(n)) \text{ and } f = \Omega(g(n))$$

- Transitivity
  - $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
  - Same for O and Ω
- Reflexivity:
  - $f(n) = \Theta(f(n))$
  - Same for O and Ω
- Symmetry:
  - $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
- Transpose symmetry:
  - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

Let f, g and h be asymptotically positive functions. Prove or disprove each of the following conjectures.

Transitivity f(n) = Θ(g(n)) and g(n) = Θ(h(n)) $\Rightarrow$ f(n) = Θ(h(n))

1. By definition f(n) = Θ(g(n)) implies there exist 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 $n \geq n_0$

2. By definition g(n) = Θ(h(n)) implies there exist positive constants c_3, c_4, and n_1 such that $0 \leq c_3 h(n) \leq g(n) \leq c_4 h(n)$ for all $n \geq n_1$

3. Show f(n) = Θ(h(n)) that is there exist positive constants c_5, c_6, and n_2 such that $0 \leq c_5 h(n) \leq f(n) \leq c_6 h(n)$ for all $n \geq n_2$

By combining 1 and 2: $c_1 c_3 h(n) \leq c_1 g(n) \leq f(n)$ let $c_5 = c_1 c_3$ so $c_5 h(n) \leq f(n)$

Again from 1 and 2: $f(n) \leq c_2 g(n) \leq c_2 c_4 h(n)$ let $c_6 = c_2 c_4$ so $c_6 h(n) \leq f(n)$

And let $n_2 = \max\{n_0, n_1\}$

Let f, g and h be asymptotically positive functions. Prove or disprove each of the following conjectures.

If f(n) = O(g(n)) and h(n) = O(g(n)), then f(n) = Θ(h(n)) ?

1. By definition f(n) = O(g(n)) implies there exist positive constants c_1 and n_0 such that $0 \leq f(n) \leq c_1 g(n)$ for all $n \geq n_0$

2. By definition h(n) = O(g(n)) implies there exist positive constants c_2 and n_1 such that $0 \leq h(n) \leq c_2 g(n)$ for all $n \geq n_1$

3. Show f(n) = Θ(h(n)) that is there exist positive constants c_3, c_3, and n_2 such that $0 \leq c_3 h(n) \leq f(n) \leq c_4 h(n)$ for all $n \geq n_2$

?????????

Let f(n) = n. g(n) = n^2 and h(n)= n^3