BITS Pilani
Hyderabad Campus

Data Structures and Algorithms Design ZG519

Febin.A.Vahab
Asst.Professor(Offcampus)
BITS Pilani,Bangalore

SESSION 2 -PLAN

Online Sessions(#)	List of Topic Title	Text/Ref Book/external resource
2	Notion of best case, average case and worst case. Use of asymptotic notations- Big-Oh, Omega and Theta Notations. Correctness of Algorithms.	T1: 1.4, 2.1

Time Complexity- Why should we care?

for i =2 to n-1
 If i divides n
  n is not prime
----

1 ms for a division
In worst case (n-2) times.

n =11 ------?
n = 101 -------? 99

n

for i ← 2 to √n
 if i divides n
  n is not prime
----

1 ms for a division
In worst case (√n-1) times.

1/2
n

n=11,(3-1) = 2ms
n=101,(√101-1) times = 9ms

Notion of best case and worst case

- Best case: where algorithm takes the least time to execute.
  - In arrayMax ex, occurs when A[0] is the maximum element.
  - T(n)=5n
- Worst case :where algorithm takes maximum time.
  - Occurs when elements are sorted in increasing order so that variable *currentMax* is reassigned at each iteration of the loop.
  - T(n)=7n-2

Algorithm ***arrayMax*(*A*,*n*)**

currentMax ← *A*[0]
for (***i*** =1; i<n; i++)
if *A*[*i*] > *currentMax* **then**
currentMax ← *A*[***i***]
return *currentMax*

Use of asymptotic notation

- How the running time of an algorithm increases with the input size, as the size of the input increases without bound?

- Used to compare the algorithms based on the order of growth of their basic operations.

Informal Introduction

oh

- *O(g(n))* is the set of all functions with a lower or same order of growth as *g(n)* (to within a constant multiple, as *n* goes to infinity)

Omega

- Ω(*g(n)*), stands for the set of all functions with a higher or same order of growth as *g(n)* (to within a constant multiple, as *n* goes to infinity).

- *Θ(g(n))* is the set of all functions that have the same order of growth as *g(n)* (to within a constant multiple, as *n* goes to infinity).

n = size of the input

$g(n) = n^3$

$n \quad n^2 \quad n^3$

$n \in O(n^3)$ $\quad n^4 \notin O(n^3)$

$n^2 \in O(n^3)$ $\quad n^5 \notin O(n^3)$

$n^3 \in O(n^3)$

$n \notin \Omega(n^3)$ $\quad n^4 \in \Omega(n^3)$

$n^2 \notin \Omega(n^3)$ $\quad n^5 \in \Omega(n^3)$

$n^3 \in \Omega(n^3)$

$n^3 \in \Theta(n^3)$

$n^2 + \sin n \in \Theta(n^2)$

$n^2 + \log n \in \Theta(n^2)$

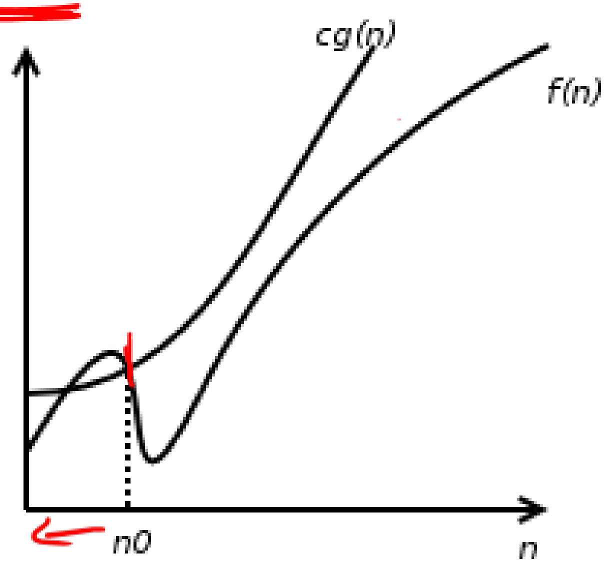
Big-Oh Notation

- Let f and g be functions from nonnegative numbers to nonnegative numbers. Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there is a real constant c>0 and an integer constant n_0 >=1 such that $\boxed{f(n) \le cg(n)}$ for every integer $\boxed{n \ge n_0}$

$n^2 \in O(n^3)$

$n^2 \le 5n^3$

$1/2$



Big-Oh Notation

- Big-Oh notation provides an upper bound on a function to within a constant factor.
- To prove big-Oh, find witnesses, specific values for C and n0, and prove n >= n0 implies $f(n) \leq C * g(n)$.

One Approach for Finding Witnesses

- *Generate a table for f(n) and g(n) using n = 1, n = 10 and n = 100. [Use values smaller than 10 and 100 if you wish.]*
- *Guess C = ⌈f(1)/g(1)⌉ (or C = ⌈f(10)/g(10)⌉).*
- *Check that f(10) ≤ C ∗ g(10) and f(100) ≤ C ∗ g(100).[If this is not true, f(n) might not be O(g(n)).]*
- *Choose n0 = 1 (or n0 = 10).*
- *Prove that ∀n(n >= n0 → f(n) ≤ C ∗ g(n)).*
- *[It's ok if you end up with a larger, but still constant, value for C.]*

One Approach for Finding Witnesses

- Assume n > 1 if you chose n0 = 1 (or n > 10 if you chose n0 = 10).
- To prove $f(n) \leq C * g(n)$, you need to find expressions larger than f(n) and smaller than $C * g(n)$.
- If the lowest-order term is negative, just eliminate it to obtain a larger expression.
- Repeatedly use n > k and 2n > 2k and 3n > 3k and so on to “convert” the lowest-order term into a higher-order term.
- Check that your expressions are less than $C * g(n)$ by using n = 100.

Big-Oh Notation Example 1

$f(n) \leq c\,g(n)$

$n \in O(n^2)$

$\frac{f(n)}{g(n)}$

Show that 3n + 7 is O(n).

– In this case, f(n) = 3n + 7 and g(n) = n.

n	f(n)	g(n)	Ceil(f(n)/g(n))
1	10	1	10
10	37	10	4
100	307	100	4

$3n+7 \in O(n^2)$

$3n+7 \in O(n^5)$

$3n+7 \in O(n^4)$

– This table suggests trying n0 = 1 and c = 10 or

– n0 = 10 and c = 4.

– Proving either one is good enough to prove big-Oh.

$n > 10 \quad 3n+7 \leq 4n$

Big-Oh Notation Example 1

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

$n_0 = 1$
$c = 10$

Try n0 = 1 and c = 10.

- Want to prove ***n > 1 implies 3n + 7 ≤ 10n.***
- Assume n > 1. Want to show ***3n + 7 ≤ 10n.***
- 7 is the lowest-order term, so work on that first.
- **n > 1 implies 7n > 7**, which implies
- 3n + 7 < 3n + 7n = 10n.
- This finishes the proof.

$7n > 7$

Big-Oh Notation Example 2

$f(n) \le c \cdot g(n)$

n_0

c

- Show that $n^2 + 2n + 1$ is $O(n^2)$.
  - In this case, f(n) = n^2 + 2n + 1 and g(n) = n^2.

n	f(n)	g(n)	Ceil(f(n)/g(n))
1	4	1	4
10	121	100	2
100	10201	10000	2

  - This table suggests trying n0 = 1 and C = 4
  - or n0 = 10 and C = 2.

Big-Oh Notation Example 2

- Try n0 = 1 and c = 4.
  - Want to prove ***n > 1 implies $n^2 + 2n + 1 \leq 4n^2$.***
  - Assume n > 1.
  - Want to show $n^2 + 2n + 1 \leq 4n^2$.
  - Work on the lowest-order term first.
  - n > 1 implies
  - $n^2 + 2n + 1 < n^2 + 2n + n = n^2 + 3n$
  - Now 3n is the lowest-order term.
  - n > 1 implies 3n > 3 and $3n^2 > 3n$, which implies
  - $n^2 + 3n < n^2 + (3n)n$

  $= n^2 + 3n^2 = 4n^2$. This finishes the proof.

$1n^0$

$1 \times n^1$

$n^2 + 2n + 1$

$< n^2 + 2n + n$

Big-Oh Notation Example 3

- Example
- 2n+10 is O(n)

  ie.

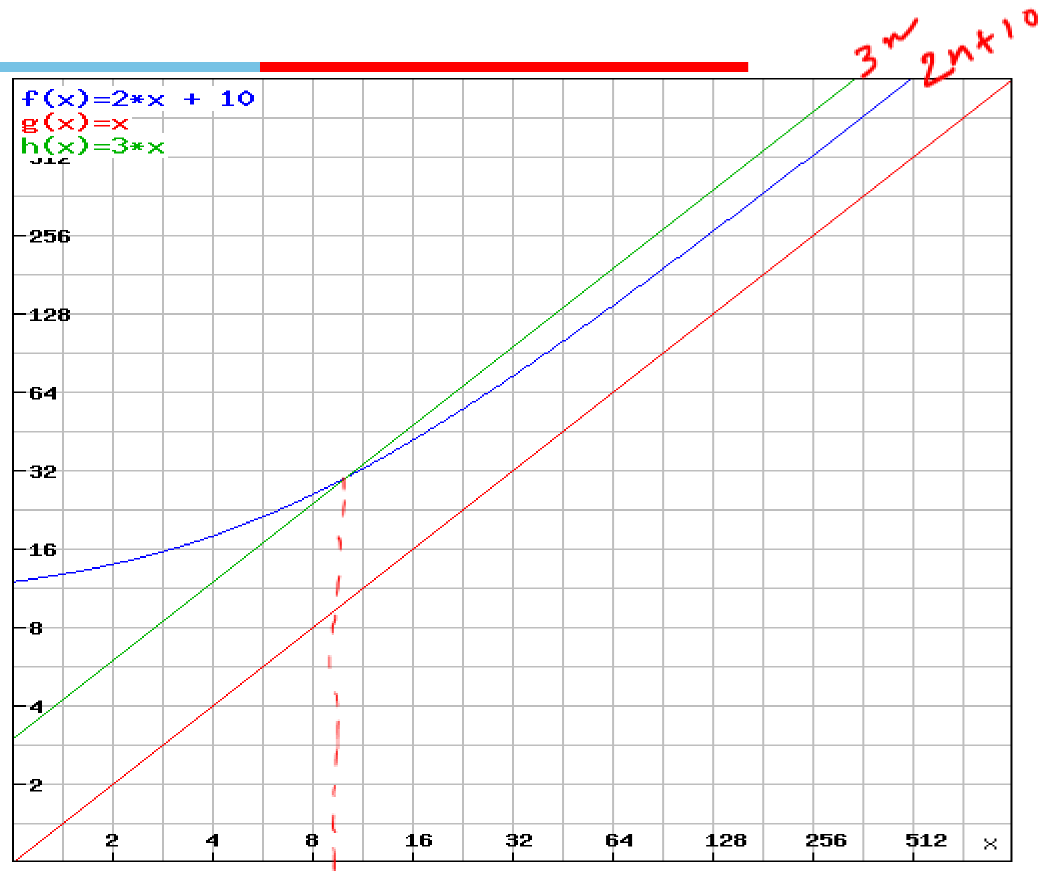
  2n+10 <=c*n

  10/n <=(c-2)

  10/c-2 <=n

  n >=10/(c-2)

  $\boxed{\text{ie .If c=3,n=10}}$



$f(n) \le 3\, g(n)$

$\boxed{2n+10} \le 3n$

$2n+10 = 14$ $n=2$

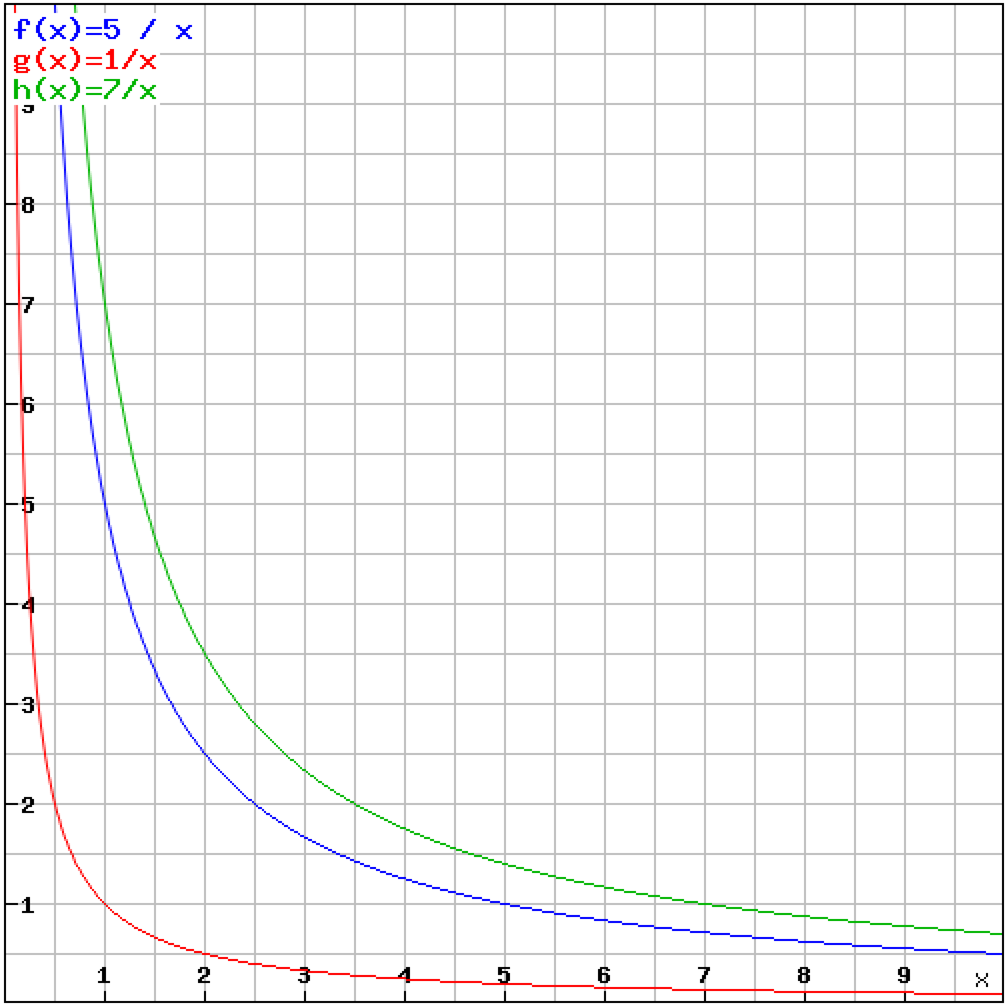
$3n = 6$

$O(n)$

$n \ge n_0$ 10

Big-Oh Notation Example 4

- Example
- 5/x is O(1/x)
- 5/x<=c*1/x
- c>=5 for x>=1

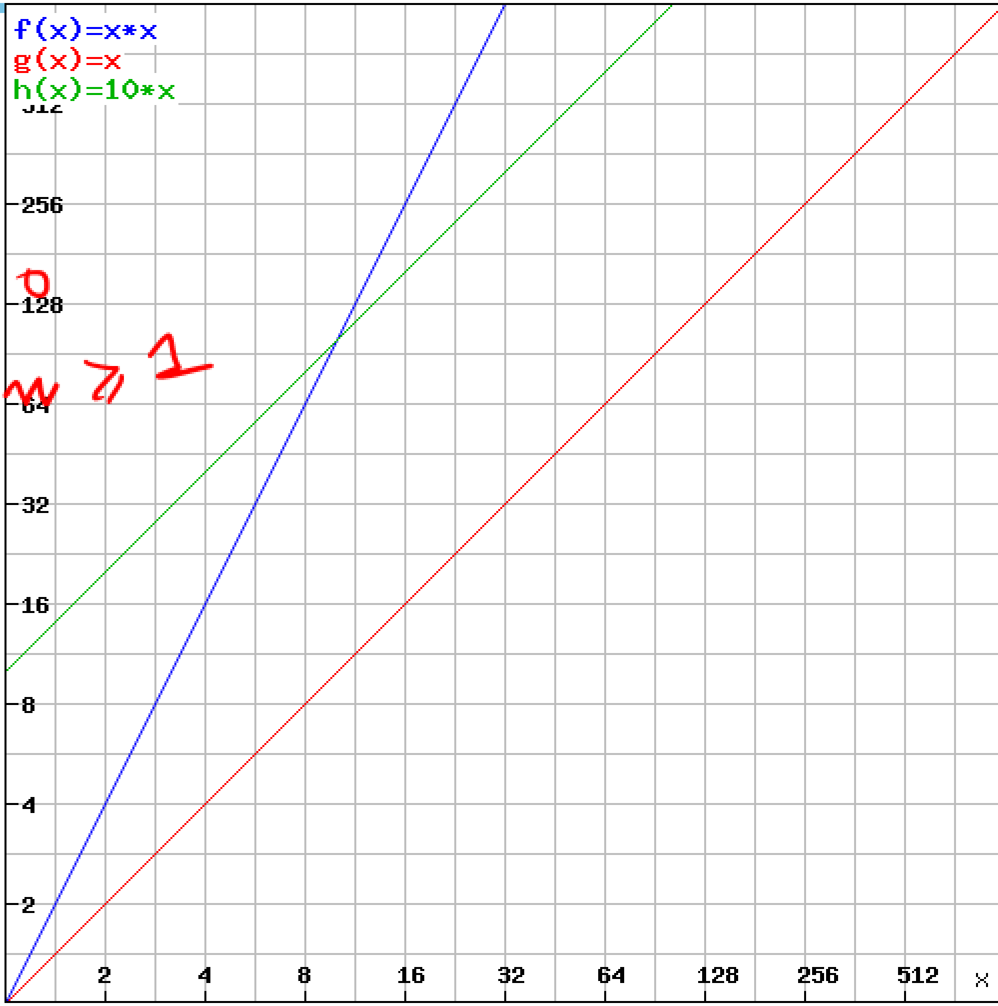


Big-Oh Notation Example 5

- Example:
- The function n^2 is not $O(n)$
  - $n^2 \le cn$
  - $\boxed{n \le c}$
  - The above inequality cannot be satisfied since c must be a positive constant

$c > 0$

$n \geq 1$



Big-Oh Notation Example 6

- Show that $8n^3 - 12n^2 + 6n - 1$ is $O(n^3)$.
  - In this case, $f(n) = 8n^3 - 12n^2 + 6n - 1$ and $g(n) = n^3$

n	f(n)	g(n)	Ceil(f(n)/g(n))
1	1	1	1
10	6859	1000	7
100	7880599	1000000	8

  - This table suggests trying n0 = 100 and C = 8.

Big-Oh Notation Example 6

- Try n0 = 100 and c = 8.
  - Want to prove $n > 100$ implies $8n^3 - 12n^2 + 6n - 1 \leq 8n^3$
  - Assume $n > 100$. Want to show $f(n) \leq 8n^3$.
  - The lowest-order term is negative, so eliminate it.
  - $8n^3 - 12n^2 + 6n - 1 < 8n^3 - 12n^2 + 6n$.
  - $n > 100$ implies $n > 6$, $n^2 > 6n$ which implies
  - $8n^3 - 12n^2 + 6n < 8n^3 - 12n^2 + n^2 = 8n^3 - 11n^2$.
  - Now lowest-order term is negative, so eliminate.
  - $n > 100$ implies $8n^3 - 11n^2 \leq 8n^3$.
  - This finishes the proof.

Big-Oh Notation
-More examples

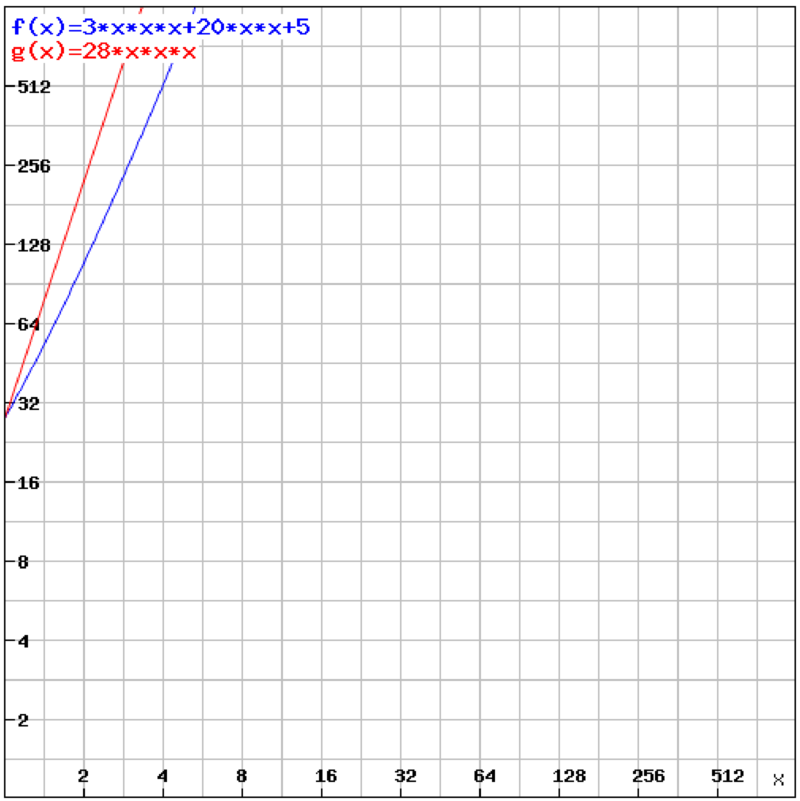
- $7n-2$ is $O(n)$
- $3n^3 + 20n^2 + 5$ is $O(n^3)$
- $3 \log n + \log \log n$ is $O(\log n)$

- Solution using one method is given below. Try other one.

Big-Oh Notation
-More examples

- 7n-2 is O(n)
  - 7n-2<=cn
  - 7-2/n<=c
  - c>=7-2/n
  - **n0=1 and c=7** is true .
- $3n^3 + 20n^2 + 5$ is $O(n^3)$
  - $3n^3+20n^2+5<=c.n^3$
  - $3+20/n+5/n^3<=c$
  - $c>=3+20/n+5/n^3$

    c>=28 and n0>=1 is true



Big-Oh Notation -More examples

- 3 log n+ log log n is O(log n)
  - 3log n+ log logn< c.log n
  - Let n=8,
  - 3*3+log 3<= 3c
  - 9+1.58 <=3c
  - c >=4

OR

- 3 log n + log log n <=4 logn , for n >=2 .
  - Note that log log n i s not even defined for n = 1 . That is why we use n >= 2.

Big-Oh Notation

$f(n) = 2n^2 + 5n + 3$ $f(n) \in O(n^2)$

- If $f(n)$ a polynomial of degree d, then $f(n)$ is $O(n^d)$, i.e.,
  1. Drop lower-order terms
  2. Drop constant factors
- Use the smallest possible class of functions
  - Say "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$"
- Use the simplest expression of the class
  - Say "$3n + 5$ is $O(n)$" instead of "$3n + 5$ is $O(3n)$"

Big-Oh Notation:Theorem

Cormen

Let $d(n)$, $e(n)$, $f(n)$, and $g(n)$ be functions mapping nonnegative integers to nonnegative reals. Then

1. If $d(n)$ is $O(f(n))$, then $ad(n)$ is $O(f(n))$, for any constant $a > 0$.
2. If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n) + e(n)$ is $O(f(n) + g(n))$.
3. If $d(n)$ is $O(f(n))$ and $e(n)$ is $O(g(n))$, then $d(n)e(n)$ is $O(f(n)g(n))$.
4. If $d(n)$ is $O(f(n))$ and $f(n)$ is $O(g(n))$, then $d(n)$ is $O(g(n))$.
5. n^x is $O(a^n)$ for any fixed $x > 0$ and $a > 1$.
6. $\log n^x$ is $O(\log n)$ for any fixed $x > 0$.

Big-Oh Notation:Proof of Theorem

1.If d(n) is O(f(n)), then a*d(n) is O(f(n)) for any constant a>0.

- d(n) ≤ C * f(n) where C is a constant
- a * d(n) ≤ a * C * f(n)
- a * d(n) ≤ C1 * f(n) where a* C = C1
- Therefore a*d(n) = O(f(n))

Big-Oh Notation:Proof of Theorem

2. If d(n) is O(f(n)) and e(n) is O(g(n)), then d(n)+e(n) is O(f(n)+g(n)). The proof will extend to orders of growth

- $d(n) \leq C1 * f(n)$ for all $n \geq n1$ where C1 is a constant
- $e(n) \leq C2 * g(n)$ all $n \geq n2$ where C2 is a constant
- $d(n) + e(n) \leq C1 * f(n) + C2 * g(n)$
- $\leq C3 (f(n) + g(n))$ where $C3=\max\{C1,C2\}$
- and $n \geq \max\{n1,n2\}$

Big-Oh Notation:Proof of Theorem

6. Log n^x is O(log n) for any fixed x > 0.

$\log n^x <= c.\log n$

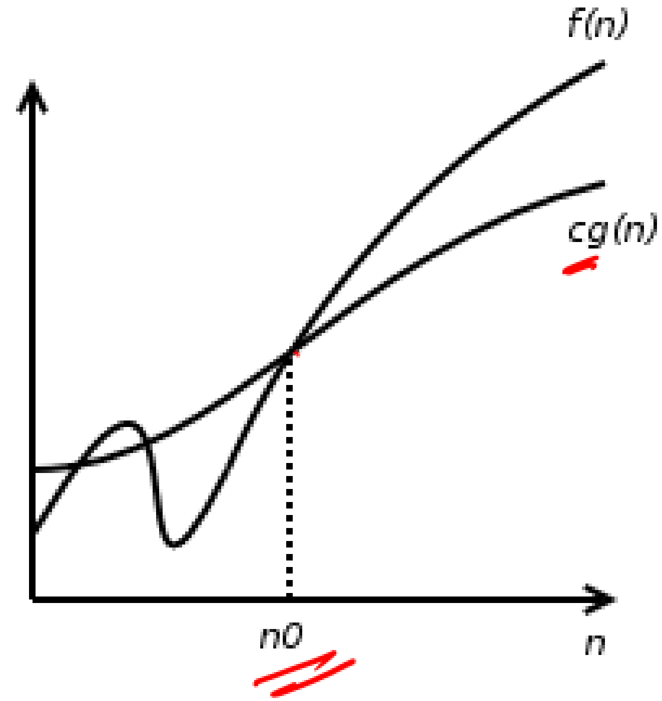
$x * \log n <= c. \log n$

$c >= x.$

Big-Omega Notation

- The function f(n) is said to be in Ω(g(n)) iff there exists a positive constant c and a positive integer n0 such that

f(n)>= c.g(n) for all n>=n0.



- Asymptotic lower bound
- $n^3 \in \Omega(n^2)$
- $n^5+n+3 \in \Omega(n^4)$

Big-Omega Notation

- Big-Omega notation provides a lower bound on a function to within a constant factor.
- To prove big-Omega, find witnesses, specific values for C and n0, and prove $n > n0$ implies $f(n) \geq C * g(n)$.

Tricks for Proving Big-Omega

- Assume $n > 1$ if you chose $n0 = 1$ (or $n > 10$ if you chose $n0 = 10$).
- To prove $f(n) \geq C * g(n)$, you need to find expressions smaller than $f(n)$ and larger than $C * g(n)$.
- If the lowest-order term is positive, just eliminate it to obtain a larger expression.
- Repeatedly use $-n0 > -n$ and $-0.1n0 > -0.1n$ and so on to "convert" the lowest-order term into a higher-order term.
- Check that your expressions are greater than $C * g(n)$ by using $n = 100$.

Tricks for Proving Big-Omega

- Generate a table for f(n) and g(n). using n = 1, n = 10 and n = 100.[Use values smaller than 10 and 100 if you wish.]
- Guess $1/C = [g(1)/f(1)]$ (or more likely $1/C = [g(10)/f(10)]$).
- Check that $f(10) \geq C * g(10)$ and $f(100) \geq C * g(100)$.[If this is not true, f(n) might not be (g(n)).]
- Choose n0 = 1 (or n0 = 10).
- Prove that $\forall n(n > n0 \rightarrow f(n) \geq C * g(n))$.[It’s ok if you end up with a smaller, but still positive, value for C.]

Big-Omega Example 1

f(n) g(n)

$f(n) \geq c \cdot g(n)$

- Show that 3n + 7 is Ω(n).
  - In this case, f(n) = 3n + 7 and g(n) = n.

n	f(n)	g(n)	Ceil(g(n)/f(n))	C
1	10	1	1	1
10	37	10	1	1
100	307	100	1	1

  - This table suggests trying n0 = 1 and C = 1.
  - Want to prove $\boxed{\textbf{n > 1}}$ **implies 3n + 7 ≥ n.**
  - n > 1 implies 3n + 7 > 3n > n.

3n+7 > 3n

Big-Omega Example 3

- Show that $n^2 - 2n + 1$ is $\Omega(n^2)$.
- In this case, $f(n) = n^2 - 2n + 1$ and $g(n) = n^2$.

$f(n) \geq c \cdot g(n)$

n	f(n)	g(n)	Ceil(g(n)/f(n))	C
1	0	1	-	-
10	81	100	2	1/2
100	9801	10000	1	1/2

- This table suggests trying n0 = 10 and C = 1/2.

Big-Omega Example 2

$n^2 - 2n + 1 \geq \frac{n^2}{2}$

- Try n0 = 10 and C = 1/2.
  - Want to prove n > 10 implies n^2 – 2n + 1 ≥ n^2/2.
  - Assume n > 10. Want to show f(n) ≥ n^2/2.
  - The lowest-order term is positive, so eliminate.
  - n^2 – 2n + 1 > n^2 $\boxed{-\ 2n}$
  - n > 10 implies −10 > −n, implies −2 > −0.2n.
  - −2 > −0.2n implies n^2 – 2n > n^2 – 0.2n^2 = 0.8n^2.
  - n > 10 implies 0.8n^2 > n^2/2.
  - This finishes the proof.

Big-Omega Example 3

- Show that $n^3/8 - n^2/12 - n/6 - 1$ is $O(n^3)$.
- In this case, $f(n) = n^3/8 - n^2/12 - n/6 - 1$ and $g(n) = n^3$.

n	f(n)	g(n)	Ceil(g(n)/f(n))	C
1	-8	1	-1	-1
10	117.3	1000	9	1/9
100	124,182.3	1000000	9	1/9

- $C = -1$ is useless, so try $n0 = 10$ and $C = 1/9$

Big-Omega Example 3

- Try n0 = 10 and C = 1/9.
  - Want to prove $n > 10$ implies $n^3/8 - n^2/12 - n/6 - 1 \geq n^3/9$
  - Assume $n > 10$, which implies the following:
  - $n^3/8 - n^2/12 - n/6 - 1$
  - $= (3n^3 - 2n^2 - 4n - 24)/24$
  - $> (3n^3 - 2n^2 - 4n - 2.4n)/24$
  - $> (3n^3 - 2n^2 - 7n)/24$
  - $> (3n^3 - 2n^2 - 0.7n^2)/24$
  - $> (3n^3 - 3n^2)/24$
  - $> (3n^3 - 0.3\, n^3)/24$
  - $> (3n^3 - n^3)/24$
  - $= (2n^3)/24 = n^3/12$
  - Ended up with n0 = 10 and C = 1/12, proving
  - $n > 10$ implies $n^3/8 - n^2/12 - n/6 - 1 \geq n^3/12$

Big-Theta Notation

- The function f(n) is said to be in $\Theta(g(n))$ iff there exists some positive constants c1 and c2 and a non negative integer n0 such that

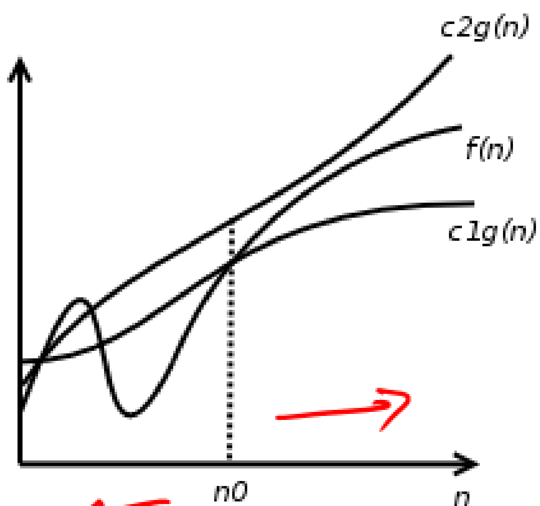
  $\boxed{\mathbf{c1.g(n)<=f(n)<=c2.g(n)\ for\ all\ n>=n0}}$
- **Asymptotic tight bound**
- $an^2+bn+c \in \Theta(n^2)$
- $n^2 \in \Theta(n^2)$

$3n^2$ → $6n^2$

$1n^2$ ← $3n^2$

$\Theta(n^2)$

$c_1(n^2)$ ⟷ $c_2(n^2)$



not sure

Examples – Ω and Θ

- **f(n)=5n^2 .Prove that f(n) is Ω(n)**
  - 5n^2>=c.n
  - c.n<=5n^2
  - c<=5n
  - If n=1,c<=5
  - 5*1<=4*1 hence the proof.

Examples – Ω and Θ

- **f(n)=5n^2 .Prove that f(n) is Ω(n)**
  - 5n^2>=c.n
  - c.n<=5n^2
  - c<=5n
  - If n=1,c<=5
  - 5*1<=4*1 hence the proof.
- Prove that f(n) is Θ(n^2)

Little-Oh and little omega Notation

- f(n) is o(g(n)) (or f(n) ∈ o(g(n))) if for any real constant c > 0, there exists an integer constant n0 ≥ 1 such that
  - ***f(n) < c ∗ g(n)*** for every integer n ≥ n0.

- f(n) is ω(g(n)) (or f(n) ∈ ω(g(n))) if for any real constant c > 0, there exists an integer constant n0 ≥ 1 such that
  - ***f(n) > c * g(n)*** for every integer n ≥ n0.

$n^2 \notin o(n^2)$

$n^2 \notin \omega(n^2)$

Little-Oh and Little omega Notation

- $12n^2 + 6n$ is $o(n^3)$
- $4n+6$ is $o(n^2)$
- $4n+6$ is $\omega(1)$
- $2n^9 + 1$ is $o(n^{10})$
- n^2 is $\omega(\log n)$

USING LIMITS

Little Oh- $= \lim_{n\to\infty} \frac{f(n)}{g(n)} = \mathbf{0}$

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

Correctness of algorithm

- An algorithm is said to be correct if, for every input instance, it halts with the correct output.
- When it can be incorrect?
  - Might not halt on all input instances
  - Might halt with an incorrect answer
- Does it makes sense to think of incorrect algorithm?
  - Might be useful if we can control the error rate and can be implemented very fast

$A[i] = x$ return i

.0000001

THANK YOU!

BITS Pilani
Hyderabad Campus