Design and Analysis of Algorithms

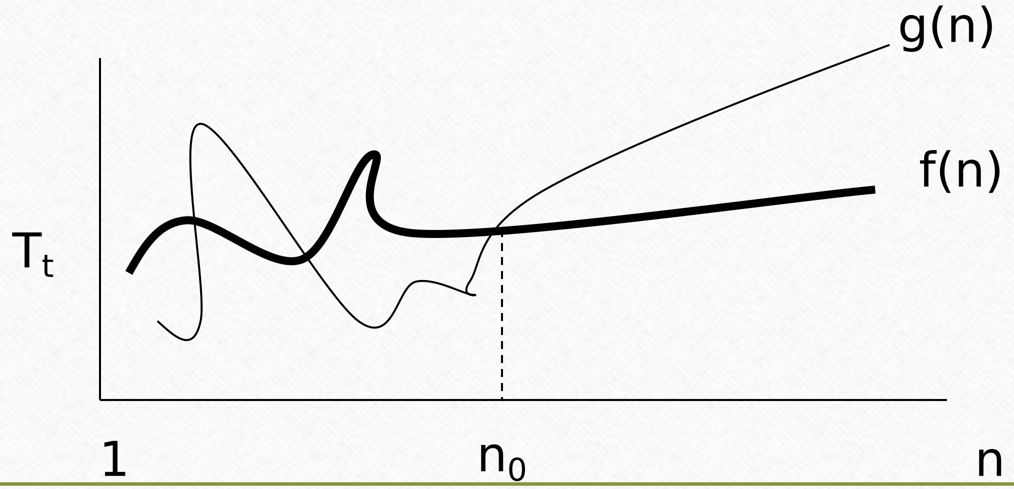
Asymptotic Notations

Asymptotic Upper Bound

$$f(n) \leq g(n) \text{ for all } n \geq n_0$$

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

$$f(n) = O(g(n)) \text{ iff } \exists\, c, n_0 : f(n) \leq cg(n)\ \forall n \geq n_0$$

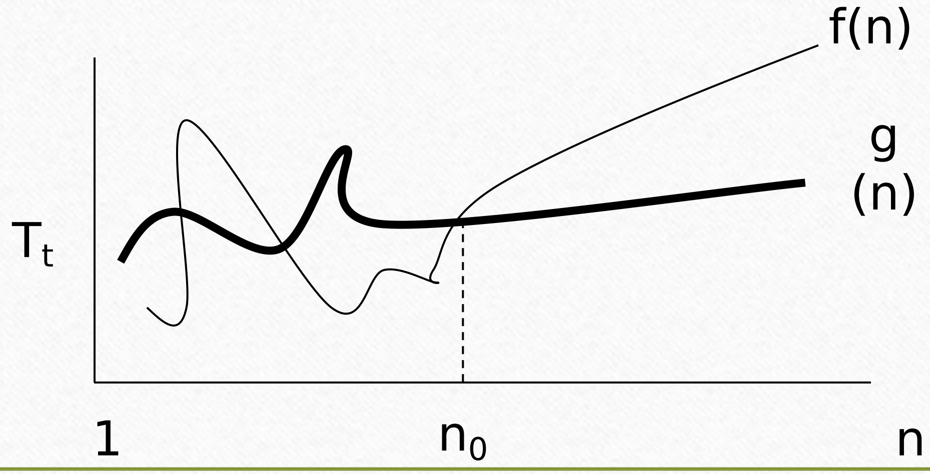


Asymptotic Lower Bound

$$f(n) \geq g(n) \;\; for\ all\ n \geq \ n_0$$

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

$$f(n) = \Omega(g(n)) \; iff \; \exists\ c, n_0 : f(n) \geq \ cg(n)\ \forall n \geq \ n_0$$

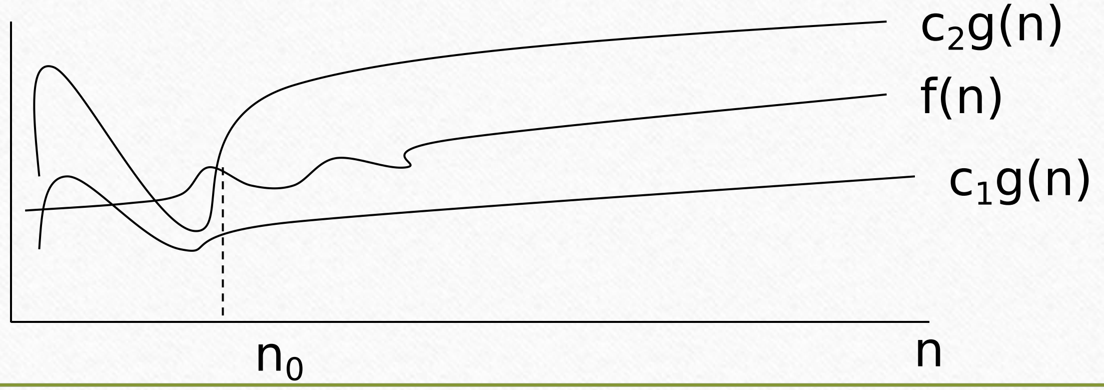


Asymptotic Tight Bound θ

$$c_1 g(n) \le f(n) \le c_2 g(n) \ for\ all\ n \ge \ n_0$$

$g(n)$ is an **asymptotic tight bound** on $f(n)$.

$$f(n) = \Theta(g(n))\ iff\ \exists\ c_1, c_2, n_0 : c_1 g(n) \le f(n) \le \ c_2 g(n)\ \forall n \ge \ n_0$$



Some Rules About Asymptotic Notation

- If $T1(n) = O(f(n))$ and $T2(n) = O(g(n))$
  - Then $T1(n) + T2(n) = Max\left(O(f(n)),\ O(g(n))\right)$
- $T1(n) * T2(n) = O(f(n) * g(n))$
- If T(x) is a polynomial of degree n
  - Then $T(x) = \Theta(x^n)$
- $\log_k n = O(n)$ for any constant k. This tells that logarithms grow very slowly.
- Do not include any constants or low order terms inside a big-Oh, e.g.,
  - $T(n) = O(2n^2)$ ----- wrong
  - $T(n) = O(n^2 + n)$----- wrong

Example: show that $(1/2)n^2 - 3n = \Theta(n^2)$

To do so we must determine positive constants c_1, c_2 and n_0 such that

$$c_1 n^2 \le \left(\frac{1}{2}\right) n^2 - 3n \le c_2 n^2, \; n \ge n_0$$

- Dividing by n^2 $c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$
- Right Hand Inequality $\frac{1}{2} \le c_2 + \frac{3}{n}$
- For positive n, if $c_2 \ge \frac{1}{2}$ then the inequality holds.
- Left Hand Inequality $c_1 + \frac{3}{n} \le \frac{1}{2}$
- For n $= 7$ and $c_1 \le \frac{1}{14}$, the inequality holds.
- $c_1 \le \frac{1}{14}$, $c_2 \ge \frac{1}{2}$ and n $= 7$

Example: show that $\left(\frac{1}{2}\right)n^3 - 3n^2 + n + 2 \quad = \Theta(n^3)$

To do so we must determine positive constants c_1, c_2 and n_0 such that $c_1 n^3 \leq \left(\frac{1}{2}\right)n^3 - 3n^2 + n + 2 \leq c_2 n^3,\ n \geq n_0$

- Dividing by n^3 $\qquad c_1 \leq \frac{1}{2} \ - \frac{3}{n} + \frac{1}{n^2} + \frac{2}{n^3} \leq c_2$
- For $n_0 = 1$
- $c_1 \leq \frac{1}{2} - \frac{3}{1} + \frac{1}{1} + \frac{2}{1} \leq c_2$
- $c_1 \leq \frac{1}{2} \leq c_2$

- For $n_0 = \infty$
- $c_1 \leq \frac{1}{2} - \frac{3}{\infty} + \frac{1}{\infty} + \frac{2}{\infty} \leq c_2$
- $c_1 \leq \frac{1}{2} \leq c_2$

Example: show that $\left(\frac{1}{2}\right) n^3 - 3n^2 + n + 2 \quad = \Theta(n^3)$

- $c_1 \leq \frac{1}{2} \leq c_2, \; c_1 \leq \frac{1}{2} \leq c_2$
- $c_1 \leq \frac{1}{2}$
- $c_1 \leq \frac{1}{2}$

- $c_1 \leq \frac{1}{2} \leq c_2, \; c_1 \leq \frac{1}{2} \leq c_2$
- $\frac{1}{2} \leq c_2$
- $\frac{1}{2} \leq c_2$

Standard Functions

n	lgn	nlgn	n^2	n^3	2^n
0			0	0	1
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536
32	5	160	1024	32768	4294967296
64	6	384	4096	262144	1.84467E+19
128	7	896	16384	2097152	3.40282E+38
256	8	2048	65536	16777216	1.15792E+77
512	9	4608	262144	134217728	1.3408E+154
1024	10	10240	1048576	1073741824	
2048	11	22528	4194304	8589934592	

Execution time (1 nano second/instruction)

n	f(n) = lgn	f(n) = n	f(n) = nlgn	f(n) = n^2	f(n) = n^3	f(n) = 2^n
10	0.003 micro sec	0.01 micro sec	0.033 micro sec	0.1 micro sec	1 micro sec	1 micro sec
20	0.004 micro sec	0.02 micro sec	0.086 micro sec	0.4micro sec	8 micro sec	1 milli sec
30	0.005 micro sec	0.03 micro sec	0.147 micro sec	0.9 micro sec	27micro sec	1 sec
40	0.005 micro sec	0.04 micro sec	0.213 micro sec	1.6 micro sec	64 micro sec	18.3 min
50	0.006 micro sec	0.05 micro sec	0.282 micro sec	2.5 micro sec	125 micro sec	13 days
10^2	0.007 micro sec	0.10 micro sec	0.664 micro sec	10 micro sec	1 milli sec	4 exp 13 years
10^3	0.010 micro sec	1.00 micro sec	9.966 micro sec	1 milli sec	1 sec	
10^4	0.013 micro sec	10 micro sec	130 micro sec	100 milli sec	16.7 min	
10^5	0.017 micro sec	0.10 milli sec	1.67 milli sec	10 s	11.6 days	
10^6	0.020 micro sec	1 milli sec	19.93 milli sec	16.7 min	31.7 years	
10^7	0.023 micro sec	0.01sec	0.23 sec	1.16 days	31709 years	
10^8	0.027 micro sec	0.10 sec	2.66 sec	115.7 days	3.17 exp 7 years	
10^9	0.030 micro sec	1 sec	29.90 sec	31.7 years		

Using Limits for Comparing Orders of Growth

- A much more convenient method
- computing the limit of the ratio of two functions

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \text{implies that } t(n) \text{ has a smaller order of growth than } g(n), \\ c & \text{implies that } t(n) \text{ has the same order of growth as } g(n), \\ \infty & \text{implies that } t(n) \text{ has a larger order of growth than } g(n).^3 \end{cases}$$

- the first two cases mean that $t(n) \in O(g(n))$,
- the last two mean that $t(n) \in \Omega(g(n))$,
- and the second case means that $t(n) \in \Theta(g(n))$.

- Compare the orders of growth of $\frac{1}{2}n(n-1)$ and n^2

$$\lim_{n\to\infty} \frac{\frac{1}{2}n(n-1)}{n^2} = \frac{1}{2}\lim_{n\to\infty} \frac{n^2-n}{n^2} = \frac{1}{2}\lim_{n\to\infty}\left(1-\frac{1}{n}\right) = \frac{1}{2}.$$

- Since the limit is equal to a positive constant, the functions have the same order of growth

Basic asymptotic efficiency classes

Class	Name	Comments
1	*constant*	Short of best-case efficiencies, very few reasonable examples can be given since an algorithm's running time typically goes to infinity when its input size grows infinitely large.
$\log n$	*logarithmic*	Typically, a result of cutting a problem's size by a constant factor on each iteration of the algorithm (see Section 4.4). Note that a logarithmic algorithm cannot take into account all its input or even a fixed fraction of it: any algorithm that does so will have at least linear running time.
n	*linear*	Algorithms that scan a list of size n (e.g., sequential search) belong to this class.

Basic asymptotic efficiency classes

$n \log n$	*linearithmic*	Many divide-and-conquer algorithms (see Chapter 5), including mergesort and quicksort in the average case, fall into this category.
n^2	*quadratic*	Typically, characterizes efficiency of algorithms with two embedded loops (see the next section). Elementary sorting algorithms and certain operations on $n \times n$ matrices are standard examples.
n^3	*cubic*	Typically, characterizes efficiency of algorithms with three embedded loops (see the next section). Several nontrivial algorithms from linear algebra fall into this class.

Basic asymptotic efficiency classes

2^n	*exponential*	Typical for algorithms that generate all subsets of an n-element set. Often, the term "exponential" is used in a broader sense to include this and larger orders of growth as well.
$n!$	*factorial*	Typical for algorithms that generate all permutations of an n-element set.