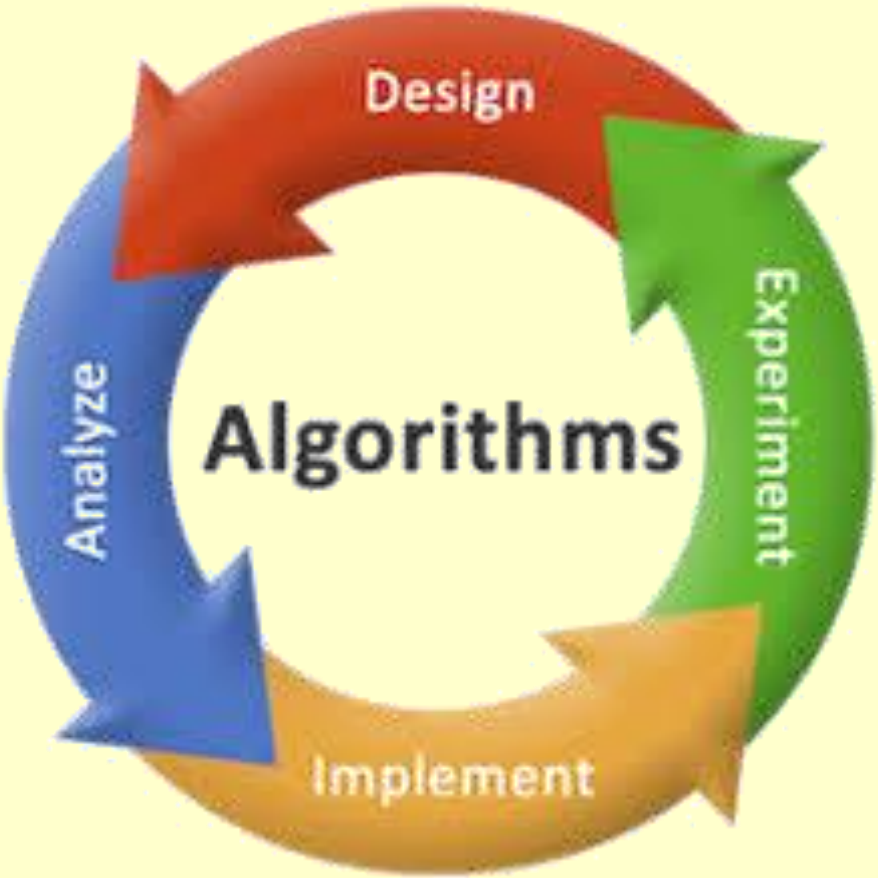


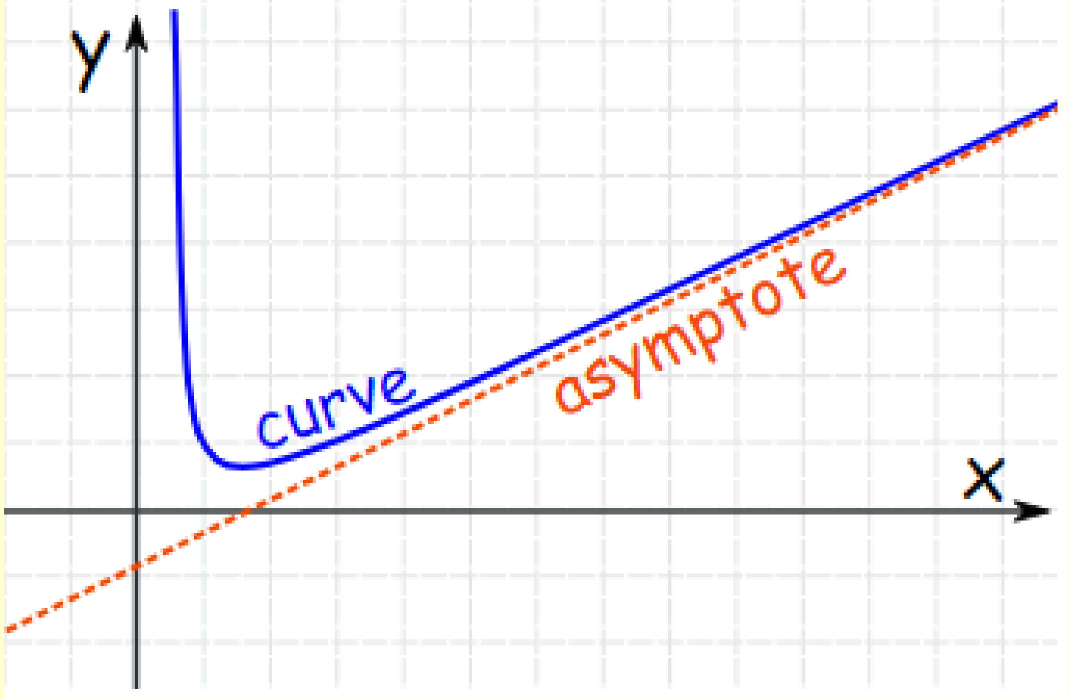
Design & Analysis of Algorithms

Lecture 6

Asymptotic Complexity & Notations

What is an asymptote?

- An asymptote is a **line** that a curve approaches, as it heads towards infinity:



For other examples refer the following website:

https://www.mathsisfun.com/algebra/asymptote.html

Asymptotic Complexity

- Running time of an algorithm as a function of input size n **for large *n***.
- Expressed using only the **highest-order term** in the expression for the exact running time.
  - Instead of exact running time, say $\Theta(n^2)$.
- Describes behavior of function in the limit.
- Written using ***Asymptotic Notations***.

Asymptotic Notations

- **Asymptotic notations: Θ, *O*, Ω**
- Defined for functions over the natural numbers.

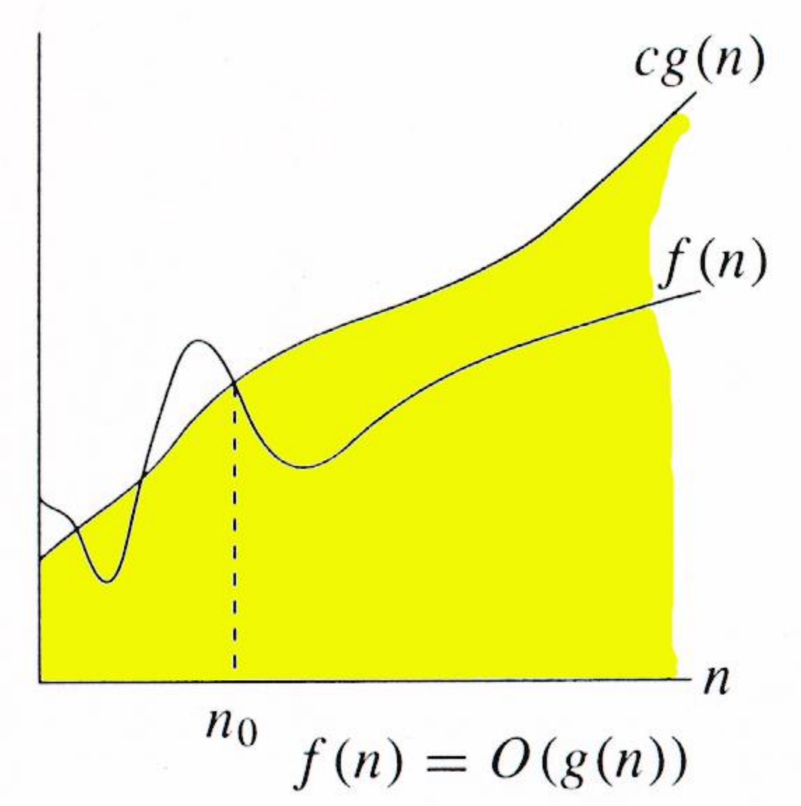
  Example: $f(n) = \Theta(n^2)$.

  Describes how $f(n)$ grows in comparison to n^2.
- Define a ***set*** of functions; in practice used to compare two function sizes.
- The notations describe different rate-of-growth relations between the defining function and the defined set of functions.

O-notation

For function $g(n)$, we define $O(g(n))$, big-O of n, as the set:

$O(g(n)) = \{f(n) :$
$\exists$ **positive constants** c **and** n_0,
such that $\forall n \geq n_0$,
we have $0 \leq f(n) \leq cg(n)$ $\}$



Intuitively: Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.

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

Examples

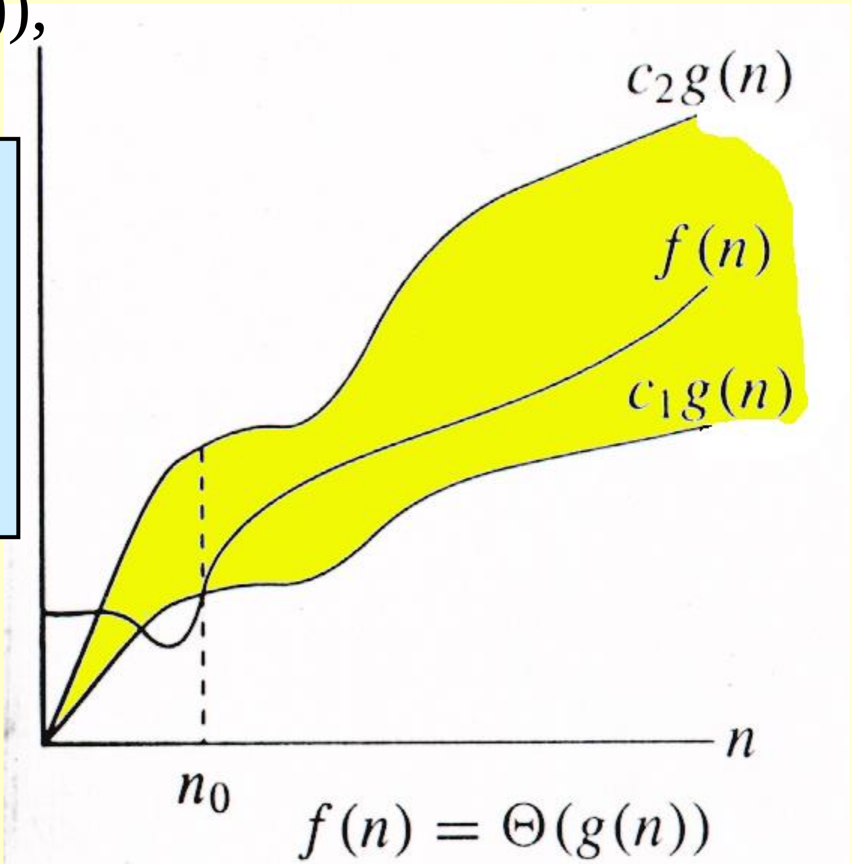
$O(g(n)) = \{f(n)$: $\exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq f(n) \leq cg(n)$ }

- $3n+2=O(n)$ /* $3n+2\leq 4n$ for $n\geq 2$ */
- $3n+3=O(n)$ /* $3n+3\leq 4n$ for $n\geq 3$ */
- $100n+6=O(n)$ /* $100n+6\leq 101n$ for $n\geq 10$ */
- $10n^2+4n+2=O(n^2)$ /* $10n^2+4n+2\leq 11n^2$ for $n\geq 5$ */
- $6*2^n+n^2=O(2^n)$ /* $6*2^n+n^2 \leq 7*2^n$ for $n\geq 4$ */

Θ-notation

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

$\Theta(g(n)) = \{f(n)$:
$\exists$ **positive constants** c_1, c_2, **and** n_0,
such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$



Intuitively: Set of all functions that have the same *rate of growth* as $g(n)$.

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

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n)).$$
$$\Theta(g(n)) \subset O(g(n)).$$

Examples

$\Theta(g(n)) = \{f(n)$: $\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$

1. 3n+2 = Θ(n)

For n>=2, c1=3 and c2=4

2.10$n^2 + 4n + 2 >= \Theta(n^2)$

For n=?, c1=10 and c2=11

10$n^2 + 4n + 2 >= 10n^2$

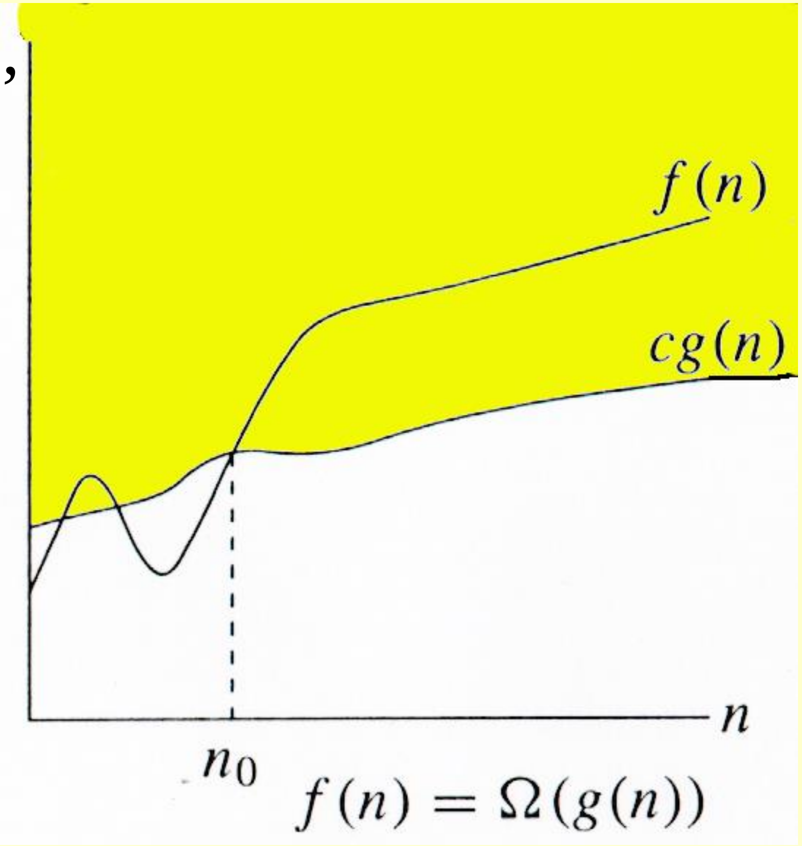
10$n^2 + 4n + 2 <= 11n^2$

Ω -notation

For function $g(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n) :$
$\exists$ positive constants c and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq cg(n) \leq f(n)\}$

Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.



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

$$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n)).$$
$$\Theta(g(n)) \subset \Omega(g(n))$$

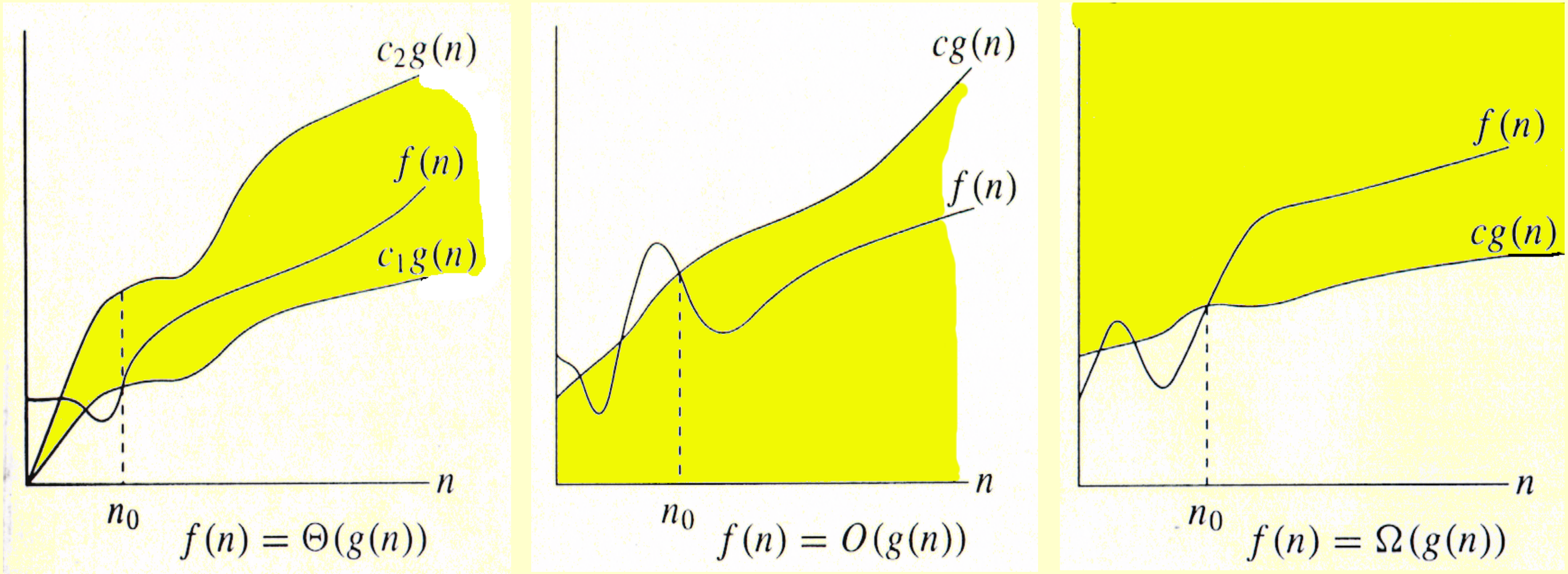
Example

$\Omega(g(n)) = \{f(n) :$ $\exists$ positive constants c and n_0, such that $\forall n \geq n_0$, we have $0 \leq cg(n) \leq f(n)\}$

- $\sqrt{n} = \Omega(\lg n)$.

Choose c and n_0.

Relations Between Θ, *O*, Ω



Relations Between Θ, O, Ω

Theorem : For any two functions $g(n)$ and $f(n)$,

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

$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

- i.e., $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- In practice, asymptotically tight bounds are obtained from asymptotic upper and lower bounds.

o-notation

For a given function $g(n)$, the set little-o:

$$o(g(n)) = \{f(n): \forall\, c > 0,\ \exists\, n_0 > 0 \text{ such that } \forall\, n \geq n_0, \text{ we have } 0 \leq f(n) < cg(n)\}.$$

$f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity:

$$\lim_{n\to\infty} [f(n) / g(n)] = 0$$

$g(n)$ is an ***upper bound*** for $f(n)$ that is not asymptotically tight.

Observe the difference in this definition from previous ones. **Why?**

Running Times

- "Running time is $O(f(n))$" $\Rightarrow$ Worst case is $O(f(n))$
- $O(f(n))$ bound on the worst-case running time $\Rightarrow$ $O(f(n))$ bound on the running time of every input.
- "Running time is $\Omega(f(n))$" $\Rightarrow$ Best case is $\Omega(f(n))$
- Can still say "Worst-case running time is $\Omega(f(n))$"
  - Means worst-case running time is given by some unspecified function $g(n) \in \Omega(f(n))$.

Common Time Complexities

BETTER

◆ $O(1)$	constant time
◆ $O(\log n)$	log time
◆ $O(n)$	linear time
◆ $O(n \log n)$	log linear time
◆ $O(n^2)$	quadratic time
◆ $O(n^2 \log n)$	log quadratic time
◆ $O(n^3)$	cubic time
◆ $O(2^n)$	exponential time
◆ $O(n^n)$	exponential time

WORSE

Useful Property

- Using the formal definitions of the asymptotic notations, we can prove their general properties

Example:

- **Theorem:** If $t_1(n) \in O(g_1(n))$ and $t_2(n) \in O(g_2(n))$, then $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$.

(The analogous assertions are true for the and notations as well.)

- The property, in particular, is useful in analyzing algorithms that comprise **two consecutively executed parts**.

Useful Property

Proof:

- The proof extends to orders of growth the following simple fact about four arbitrary real numbers a_1, b_1, a_2, b_2: if $a_1 \leq b_1$ and $a_2 \leq b_2$, then $a_1 + a_2 \leq 2 \max\{b1, b_2\}$.

- Since $t_1(n) \in O(g_1(n))$, there exist some positive constant c_1 and some nonnegative integer n_1 such that

$$\boldsymbol{t_1(n) \leq c_1 g_1(n) \text{ for all } n \geq n_1.}$$

Useful Property

- Similarly, since $t_2(n) \in O(g_2(n))$, there exist some positive constant c_2 and some nonnegative integer n_2 such that

$$\boldsymbol{t_2(n) \le c_2 g_2(n) \text{ for all } n \ge n_2.}$$

- Let $c_3 = \max\{c_1, c_2\}$ and consider $n \ge \max\{n_1, n_2\}$ so that we can use both inequalities. Adding them yields the following:

$$\begin{aligned} t_1(n) + t_2(n) &\le c_1 g_1(n) + c_2 g_2(n) \\ &\le c_3 g_1(n) + c_3 g_2(n) = c_3[g_1(n) + g_2(n)] \\ &\le c_3\, 2 \max\{g_1(n), g_2(n)\}. \end{aligned}$$

Useful Property

- Hence, $t_1(n) + t_2(n) \in O(\max\{g_1(n), g_2(n)\})$, with the constants c and n_0 required by the O definition being

  $2\ c_3 = 2\ \max\{c_1, c_2\}$ and $\max\{n_1, n_2\}$, respectively.

- **So what does this property imply for an algorithm that comprises two consecutively executed parts?**
- **It implies that the algorithm's overall efficiency is determined by the part with a higher order of growth, i.e., its least efficient part.**

References

- **Chapter 2**: Anany Levitin, “Introduction to the Design and Analysis of Algorithms”, Pearson Education, Third Edition, 2017

- **Chapter 2**: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, “Introduction to Algorithms”, MIT Press/PHI Learning Private Limited, Third Edition, 2012.

Homework

1. Sort in ascending order

   $\sqrt{n}$, log n, n, n^2, n log n

2. Sort in Descending order

   n^3, 2^n, n^n, n!, n^2 log n

3. Sort in Ascending/Descending order

$\sqrt{n}$, log n, n, n^2, n log n, n^3, 2^n, n^n, n!, n^2 log n