COMP 550
Algorithm and Analysis

Asymptotic Complexity

Based on CLRS Sec 3

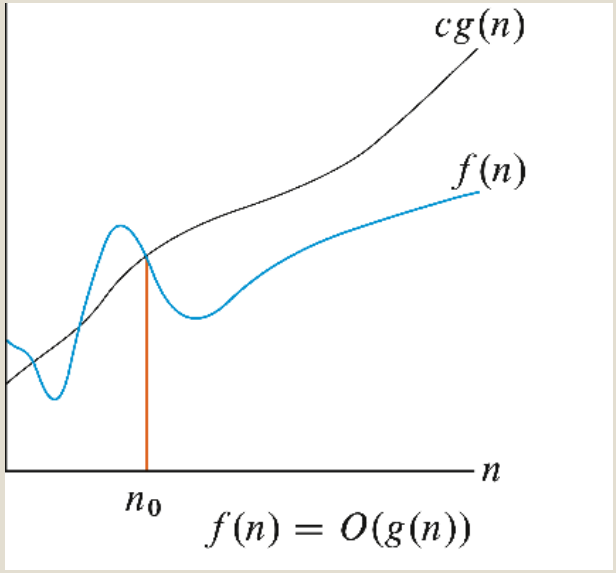
Some slides are adapted from ones by prior instructors Prof. Plaisted and Prof. Osborne

Asymptotic Complexity

- Order of growth: how an algorithm's running time increases w.r.t input size
  - Asymptotic behavior (**Asymptotic complexity**) of the algorithm
  - Running time increase as **input size increases without bound**
- Expressed using **Asymptotic Notations**
  - Example: O (Big-Oh), Ω (Big-Omega), Θ (Big-Theta)

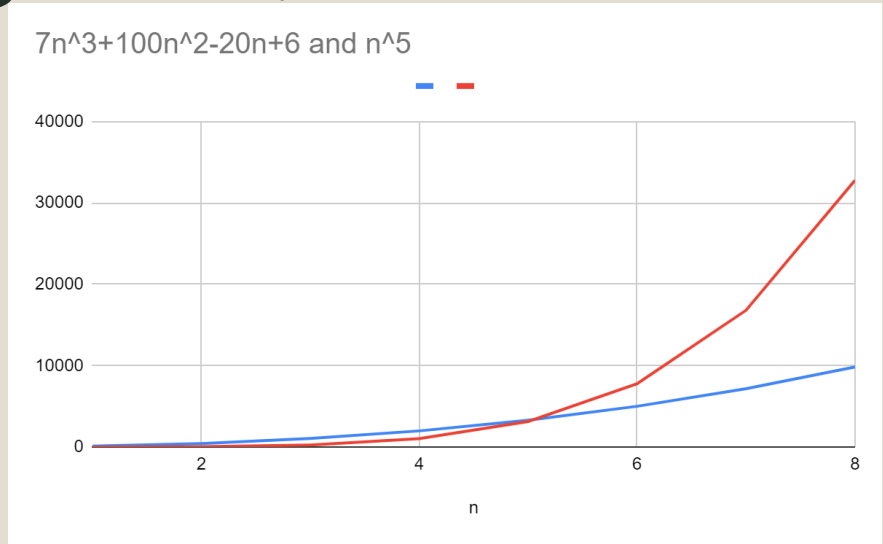
O-Notation (Big-Oh)

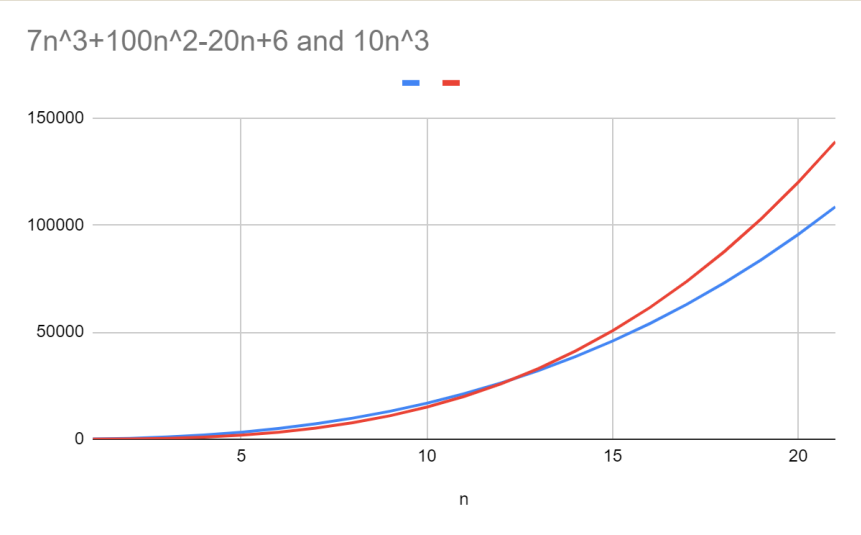
- O-notation characterizes an **upper bound** on the asymptotic behavior of a function
  - A function grows **no faster** than a certain rate (based on higher order terms)
  - A function is $O(g(n))$ if it grows no faster than $c \cdot g(n)$ for constant c

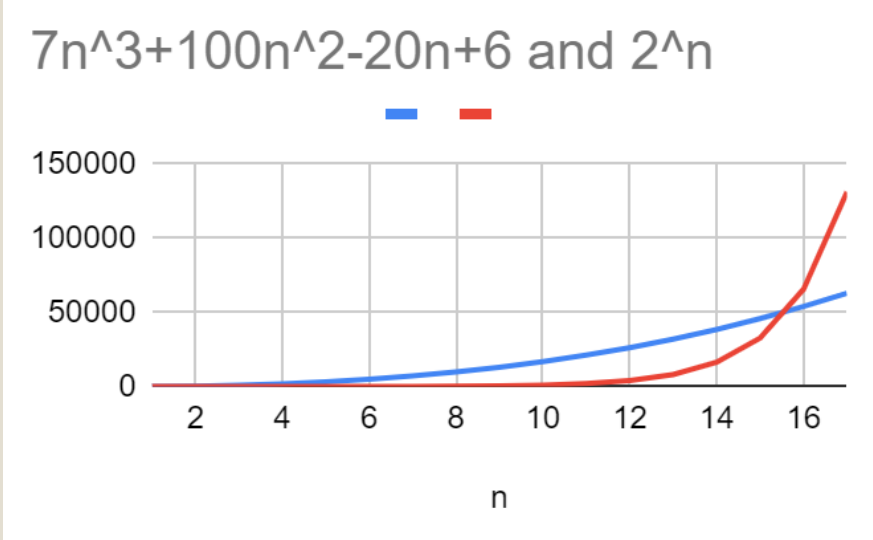


O-Notation (Big-Oh)

- A function is $O(g(n))$ if it grows no faster than $c \cdot g(n)$ for constant c
- **Example:** $f(n) = 7n^3 + 100n^2 - 20n + 6$
  - $f(n)$ is $O(n^3)$
  - $f(n)$ is also $O(n^5)$, $O(2^n)$, and $O(n!)$. **Why?**

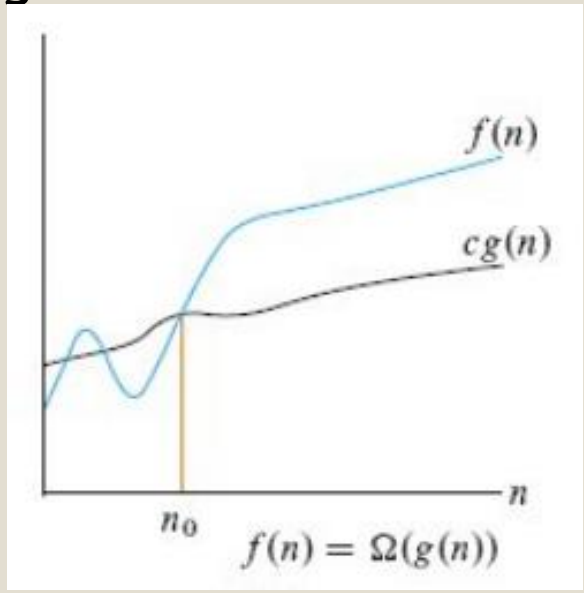






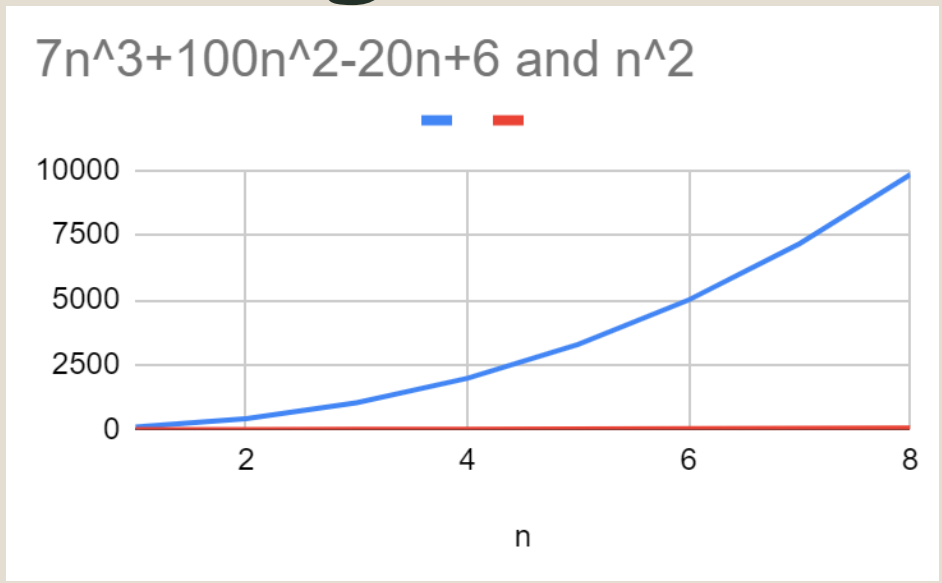
Ω-Notation (Big-Omega)

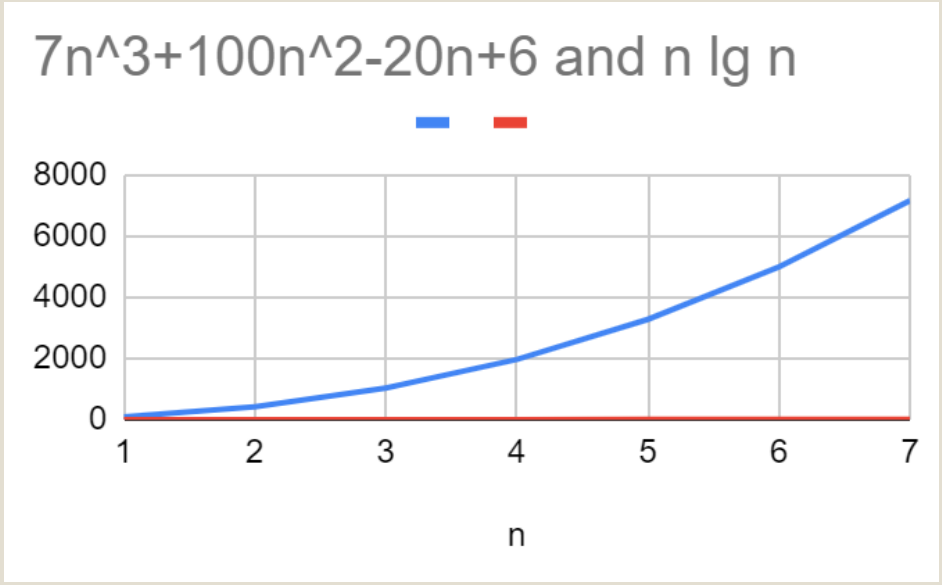
- Ω-notation characterizes a **lower bound** on the asymptotic behavior of a function
  - A function grows **no slower** than a certain rate (based on higher order terms)
  - A function is $\Omega(g(n))$ if it grows no slower than $c \cdot g(n)$ for constant c

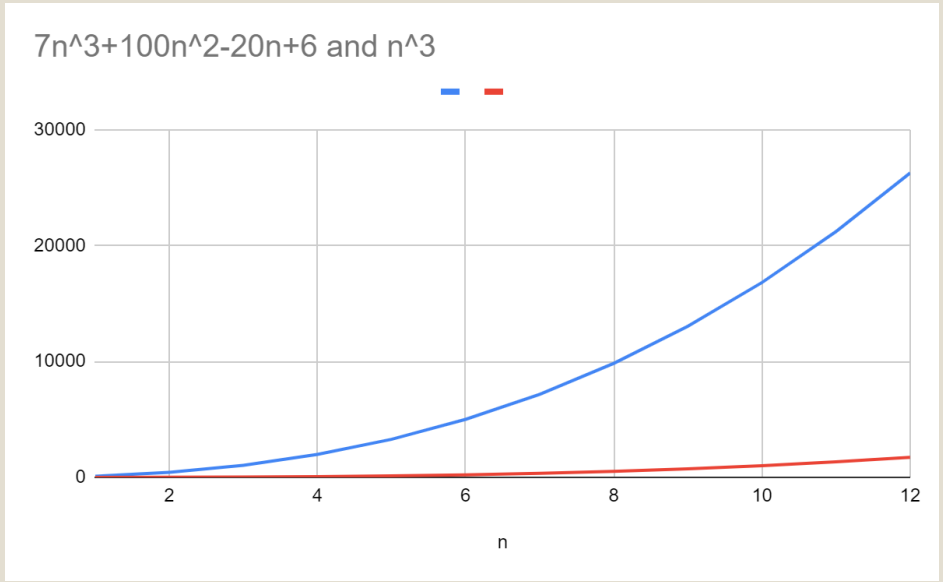


Ω-Notation (Big-Omega)

- A function is $\Omega(g(n))$ if it grows no slower than $c \cdot g(n)$ for constant c
- **Example:** $f(n) = 7n^3 + 100n^2 - 20n + 6$
  - $f(n)$ is $\Omega(n^3)$
  - $f(n)$ is also $\Omega(n^2)$, $\Omega(n)$, and $\Omega(\log n)$. **Why?**

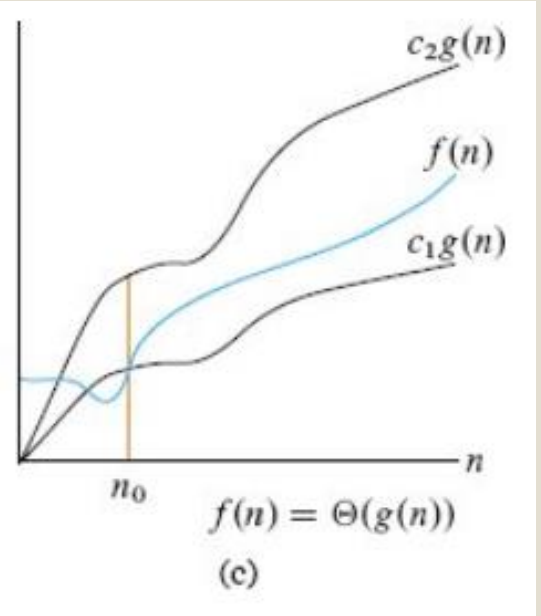






Θ-Notation (Big-Theta)

- Ω-notation characterizes a **tight bound** on the asymptotic behavior of a function
  - A function grows **precisely** a certain rate (based on higher order terms)
  - A function is $\Theta(g(n))$ if it is both $O(g(n))$ and $\Omega(g(n))$
- **Example:** $f(n) = 7n^3 + 100n^2 - 20n + 6$
  - $f(n)$ is $\Theta(n^3)$
  - $f(n)$ is **NOT** $\Theta(n^2)$, $\Theta(n^4)$, and $\Theta(\log n)$. **Why?**



Why Functions?

- Recall that our derived running times for FindMax, Linear Search, and Insertion Sort are functions of input size n
- We will represent these running times in asymptotic notations
  - Simplicity
  - Enable comparison

Insertion Sort Asymptotic Complexity

- Worst-case running time, $T(n)$ = Longest running time for any input of size n.
- Worst-case running time of Insertion Sort is $\Theta(n^2)$. **How?**

Insertion Sort: Time Complexity

In the worst case,

$$T(n) = a \cdot n + c_5 \cdot \sum_{i=2}^{n} t_i + (c_6 + c_7) \sum_{i=2}^{n} (t_i - 1) + b$$

$$= a \cdot n + c_5 \left(\frac{n(n+1)}{2} - 1\right) + (c_6 + c_7)\left(\frac{n(n-1)}{2}\right) + b$$

$$= \left(\frac{c_5 + c_6 + c_7}{2}\right) n^2 + \left(a + \frac{c_5 - c_6 - c_7}{2}\right) n + b - c_5$$

$$= a' n^2 + b' n + c'$$

Insertion Sort Asymptotic Complexity

- Can we determine this asymptotic complexity w/o evaluating sums?
- Show that the worst-case running time is both $O(n^2)$ and $\Omega(n^2)$.
- Worst-case running time of Insertion Sort is $O(n^2)$.
  - Outer loop runs $n-1$ times
  - For each iteration of outer loop
    - Inner loop runs at most $i-1$ times
    - $i \leq n \implies i-1 \leq n-1$
  - Total inner loop iteration $\leq (n-1)(n-1)$
  - Each line takes constant time
  - Worst-case running time $= O(n^2)$

```
INSERTION-SORT(A, n)
1  for i = 2 to n
2      key = A[i]
3      // Insert A[i] into the sorted subarray A[1 : i − 1].
4      j = i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] = A[j]
7          j = j − 1
8      A[j + 1] = key
```

Insertion Sort Asymptotic Complexity

- Worst-case running time of Insertion Sort is $\Omega(n^2)$,
  - For every n, there exists an input that takes at least $c \cdot n^2$ time for some constant c
  - For a reverse-sorted array,
    - A[1] moves through at least $n-1$ positions
    - A[2] moves through at least $n-2$ positions
    - …
    - Total steps $\geq (n-1)+(n-2)+\cdots+0 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$
  - Worst-case running time is at least $\frac{(n-1)n}{2}$, meaning $\Omega(n^2)$

* The textbook has a different example

Insertion Sort Asymptotic Complexity

- **The worst-case running time of Insertion Sort** is $\Theta(n^2)$
- Can we say that running time of Insertion Sort is $\Theta(n^2)$?
  - No. **Why?**
  - No "worst-case" in the statement $\Longrightarrow \Theta(n^2)$ for all cases!
  - Running time is $an + b$ for some input (best-case running time)
  - Running time of Insertion Sort is $O(n^2)$ and $\Omega(n)$

Insertion Sort Asymptotic Complexity

- Worst-case running time is both $O(n^2)$ and $\Omega(n^2)$

$$\Rightarrow \Theta(n^2)$$

- Best-case running time is both $O(n)$ and $\Omega(n)$

$$\Rightarrow \Theta(n)$$

- Running time is $O(n^2)$ and $\Omega(n)$

Insertion Sort Asymptotic Complexity

- Running time is $O(g(n)) \Rightarrow$ Worst-case running time is $O(g(n))$
- $O(g(n))$ bound on the worst-case running time $\Rightarrow$ $O(g(n))$ bound on the running time of every input.
- $\Theta(g(n))$ bound on the worst-case running time $\not\Rightarrow$ $\Theta(g(n))$ bound on the running time of every input.
- Running time is $\Omega(g(n)) \Rightarrow$ Best case is $\Omega(g(n))$
- Can still say "Worst-case running time is $\Omega(g(n))$"
  - The worst-case running time is given by some unspecified function $f(n) \in \Omega(g(n))$.

Insertion Sort Asymptotic Complexity

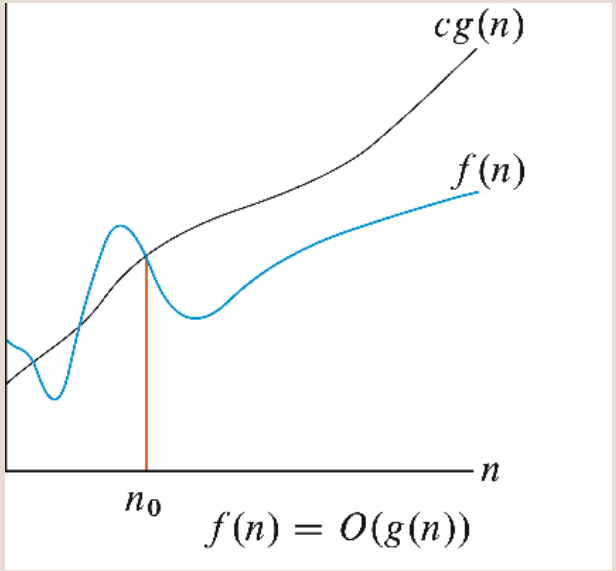
- **Insertion sort** takes $\Theta(n^2)$ in the worst case, so sorting (as a *problem*) is $O(n^2)$. **Why?**

- Any sorting algorithm must look at each item, so sorting is $\Omega(n)$.

O-Notation Definition

- For a given function $g(n)$, $O(g(n))$ is the set of functions

  $O\big(g(n)\big) = \{f(n) : \exists$ positive constants c, n_0 such that $0 \leq f(n) \leq cg(n)$ holds for all $n \geq n_0\}$



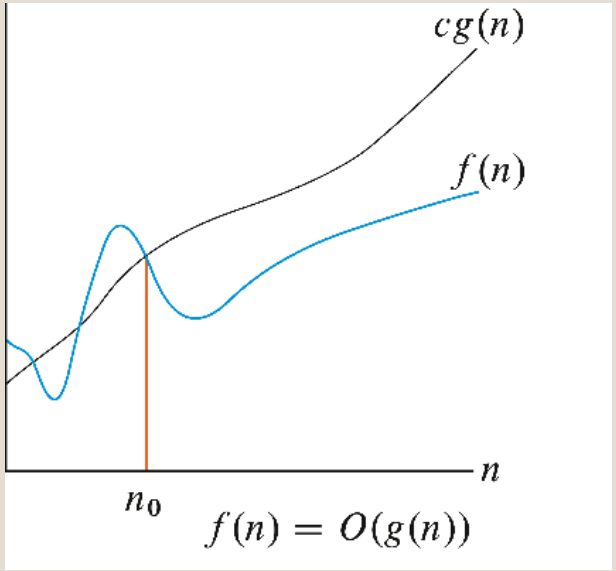
- Pronounced as big-oh of g of n

- $O\big(g(n)\big)$ is the set of all functions whose *rate of growth* is the same as or lower than that of *g*(*n*).

O-Notation Definition

- For a given function $g(n)$, $O(g(n))$ is the set of functions

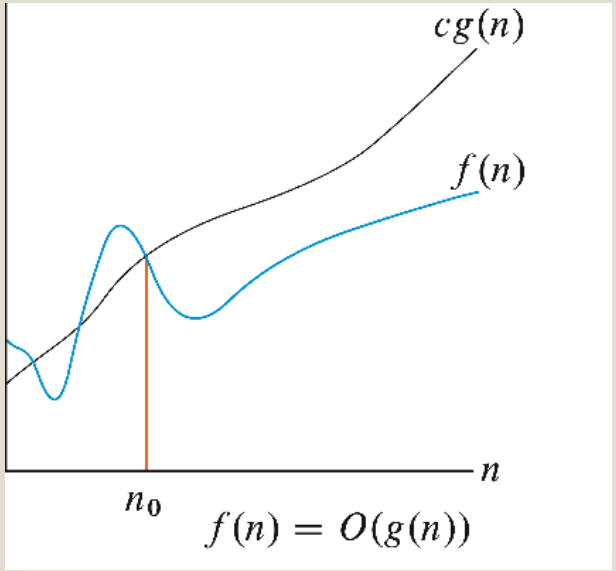
  $O(g(n)) = \{f(n) : \exists$ positive constants c, n_0 such that $0 \leq f(n) \leq cg(n)$ holds for all $n \geq n_0\}$



- We should write $f(n) \in O(g(n))$
- Common notational abuse, $f(n) = O(g(n))$

O-Notation Definition

$O(g(n)) = \{f(n) : \exists$ positive constants c, n_0 such that $0 \leq f(n) \leq cg(n)$ holds for all $n \geq n_0\}$



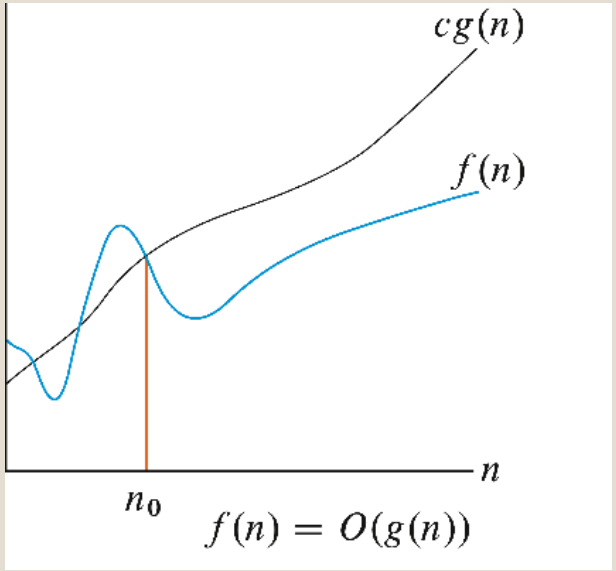
- Show that $4n^2 + 100n + 500 \in O(n^2)$
- Need to find appropriate c and n_0 such that

$$4n^2 + 100n + 500 \leq cn^2$$

- So, $c \geq 4 + \frac{100}{n} + \frac{500}{n^2}$
- One choice $n_0 = 1$ and any $c \geq 604$

O-Notation Definition

$O(g(n)) = \{f(n) : \exists$ positive constants c, n_0 such that $0 \leq f(n) \leq cg(n)$ holds for all $n \geq n_0\}$



- Show that $n^3 - 100n^2 \notin O(n^2)$
- Assume that

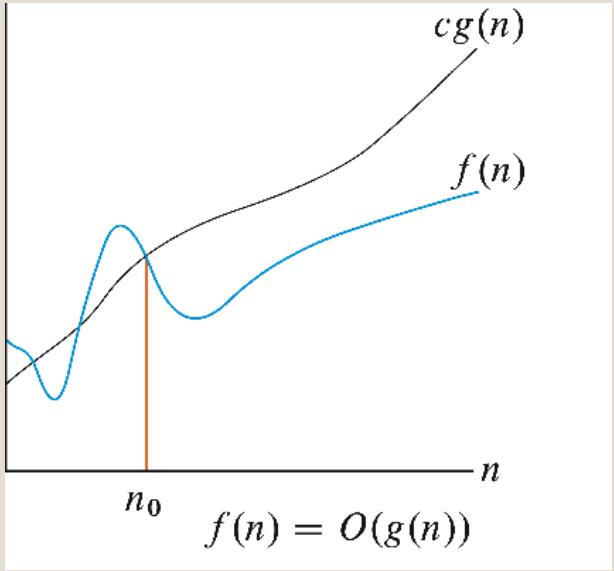
$$n^3 - 100n^2 \leq cn^2$$

- So, $c \geq n - 100$, which is impossible

O-Notation Definition

$O(g(n)) = \{f(n)$: ∃ positive constants c, n_0 such that $0 \leq f(n) \leq cg(n)$ holds for all $n \geq n_0\}$

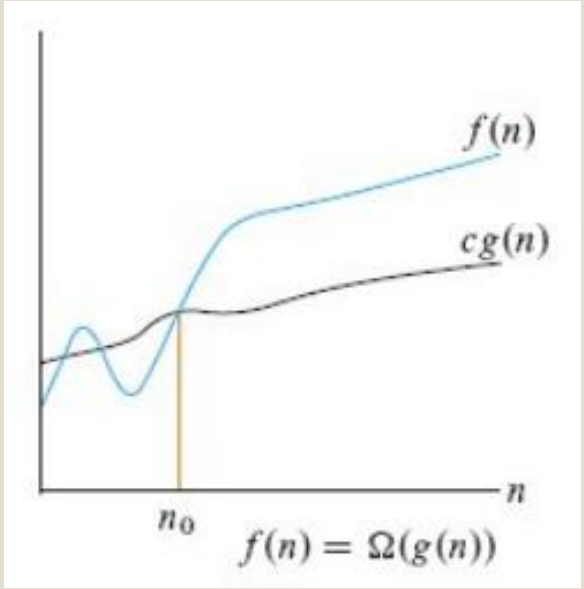
- Note that this definition requires $f(n)$ and $g(n)$ to be asymptotically non-negative
  - $f(n) \geq 0$, as $n \to \infty$



Ω-Notation Definition

- For a given function $g(n)$, $\Omega(g(n))$ is the set of functions

  $\Omega\big(g(n)\big) = \{f(n) : \exists$ positive constants c, n_0 such that $0 \leq cg(n) \leq f(n)$ holds for all $n \geq n_0\}$

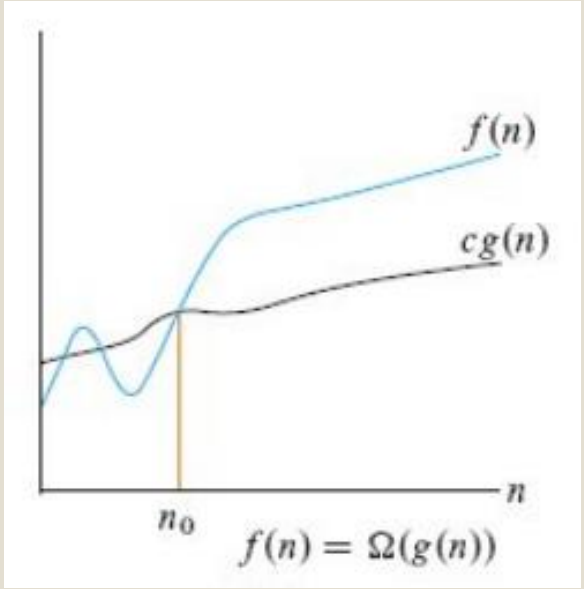


- Pronounced as big-omega of g of n

- $\Omega\big(g(n)\big)$ is the set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.

Ω-Notation Definition

$\Omega(g(n)) = \{f(n) : \exists$ positive constants c, n_0 such that $0 \leq cg(n) \leq f(n)$ holds for all $n \geq n_0\}$



- Show that $4n^2 + 100n + 500 \in \Omega(n^2)$
- Need to find appropriate c and n_0 such that

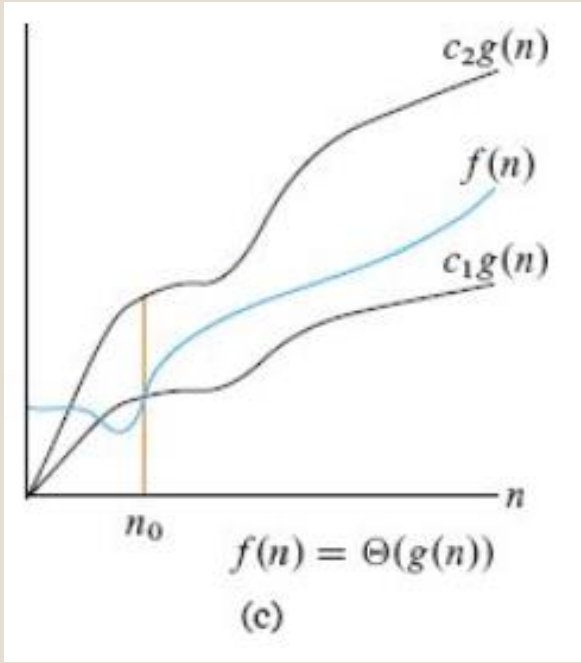
$$4n^2 + 100n + 500 \geq cn^2$$

- So, $c \leq 4 + \frac{100}{n} + \frac{500}{n^2}$
- One choice $n_0 = 1$ and any $c \leq 604$

Θ-Notation Definition

- For a given function $g(n)$, $\Theta(g(n))$ is the set of functions

$\Theta(g(n)) = \{f(n) :$ ∃ positive constants c_1, c_2, n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ holds for all $n \geq n_0\}$



- Pronounced as big-theta of g of n
- $\Theta(g(n))$ is the set of all functions whose *rate of growth* is the same as $g(n)$.

Θ-Notation Definition

Thm. 3.1. $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

- Question: How to prove statement "P **if and only if** Q"?
  - If P, then Q (Q is a necessary condition for P)
  - If Q, then P (Q is a sufficient condition for P)

Proof Obligation:

- If $f(n) = \Theta(g(n))$, then $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$
- If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then$f(n) = \Theta(g(n))$

Θ-Notation Definition

Thm. 3.1. $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

If $f(n) = \Theta(g(n))$, then $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$:

$$f(n) = \Theta(g(n)) \Longrightarrow \exists c_1, c_2, n_0 > 0 : \forall n \geq n_o, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

So, $\exists c_1, n_0 > 0 : \forall n \geq n_o, 0 \leq c_1 g(n) \leq f(n)$

$$\Rightarrow \boldsymbol{f(n) = \Omega(g(n))}$$

Also, $\exists c_2, n_0 > 0 : \forall n \geq n_o, 0 \leq f(n) \leq c_2 g(n)$

$$\Rightarrow \boldsymbol{f(n) = O(g(n))}$$

Θ-Notation Definition

Thm. 3.1. $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$

If $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$, then $f(n) = \Theta(g(n))$:

$$f(n) = \Omega(g(n)) \Longrightarrow \exists c_1, n_0' > 0 : \forall n \geq n_0', 0 \leq c_1 g(n) \leq f(n)$$

$$f(n) = O(g(n)) \Longrightarrow \exists c_2, n_0'' > 0 : \forall n \geq n_0'', 0 \leq f(n) \leq c_2 g(n)$$

Let $n_0 = \max\{n_0', n_0''\}$

So, $\exists c_1, c_2, n_0 > 0 : \forall n \geq n_o, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$

$$\Rightarrow \boldsymbol{f(n) = \Theta(g(n))}$$

Asymptotic Notations

- Be as precise as possible
  - If $f(n) = \Theta(n)$, then try to write so (not $f(n) = O(n)$)
- Algorithm A runs in $3n^2 + 20n$ time in all cases
  - Running time (or Time Complexity) is $\Theta(n^2)$
  - Running time is $O(n^3)$, this is imprecise
  - Running time is $\Theta(3n^2 + 20n)$, this has unnecessary details

Asymptotic Notations

- Common mistakes (but kind of accepted)
  - An $O(n \log n)$ time algorithm is faster than $O(n^2)$ time algorithm
- O-notation is the earliest (Bachman 1892)
- Θ- and Ω-notations were advocated by Knuth

Asymptotic Notations in Expressions

- $2n^2 + 3n + 1 = \Theta(n^2)$
  - Equality means set membership
- $2n^2 + \Theta(n)$
  - $\Theta(n)$ is standing for an anonymous function $f(n) \in \Theta(n)$
- $2n^2 + \Theta(n) = \Theta(n^2)$
  - No matter what anonymous function $\Theta(n)$ represents, there is always a way to choose an anonymous function for $\Theta(n^2)$ so that the equation is valid.

Asymptotic Notations in Expressions

- O(g(n)) means order of growth as n goes to ∞
- What about $O(1)$?
  - Infer from context
  - If $f(n) = O(1)$, n goes to ∞
- $T(n) = O(1)$ for $n < 3$
  - $\exists c > 0 : \ T(n) \leq c$ for $n < 3$

o-Notation

- Non-tight upper bound

$o(g(n)) = \{f(n) : \forall$ positive constants $c, \exists n_0 > 0$ such that $0 \leq f(n) < cg(n)$ holds for all $n \geq n_0\}$

- Compare the definition with O-notation
- $2n^2 + n$ is $O(n^2)$, but not $o(n^2)$
- $2n^{2-\epsilon} + n$ for any $\epsilon > 0$ is $o(n^2)$

o-Notation

- Non-tight upper bound

$o(g(n)) = \{f(n) : \forall$ positive constants $c, \exists n_0 > 0$ such that $0 \leq f(n) < cg(n)$ holds for all $n \geq n_0\}$

- Intuitively, $f(n)$ becomes insignificant w.r.t $g(n)$ as $n \to \infty$

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

ω-Notation

- Non-tight upper bound

$\omega(g(n)) = \{f(n)$: $\forall$ positive constants $c, \exists n_0 > 0$ such that $0 \leq cg(n) < f(n)$ holds for all $n \geq n_0\}$

- Compare the definition with Ω-notation
- $2n^2 + n$ is $\Omega(n^2)$, but not $\omega(n^2)$
- $2n^{2+\epsilon} + n$ for any $\epsilon > 0$ is $\omega(n^2)$

ω-Notation

- Non-tight upper bound

$o(g(n)) = \{f(n) : \forall$ positive constants $c, \exists n_0 > 0$ such that $0 \leq f(n) < cg(n)$ holds for all $n \geq n_0\}$

- Intuitively, $f(n)$ becomes arbitrarily large w.r.t $g(n)$ as $n \to \infty$

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

Asymptotic Notations: Properties

- Transitivity:
  - $f(n) = X\big(g(n)\big) \wedge g(n) = X\big(h(n)\big) \Rightarrow f(n) = X\big(h(n)\big)$
  - X can be any one of $O, \Omega, \Theta, o, \omega$
- Reflexivity:
  - $f(n) = X(f(n))$, where X is one of O, Ω, Θ
  - Why not applicable to o and ω?

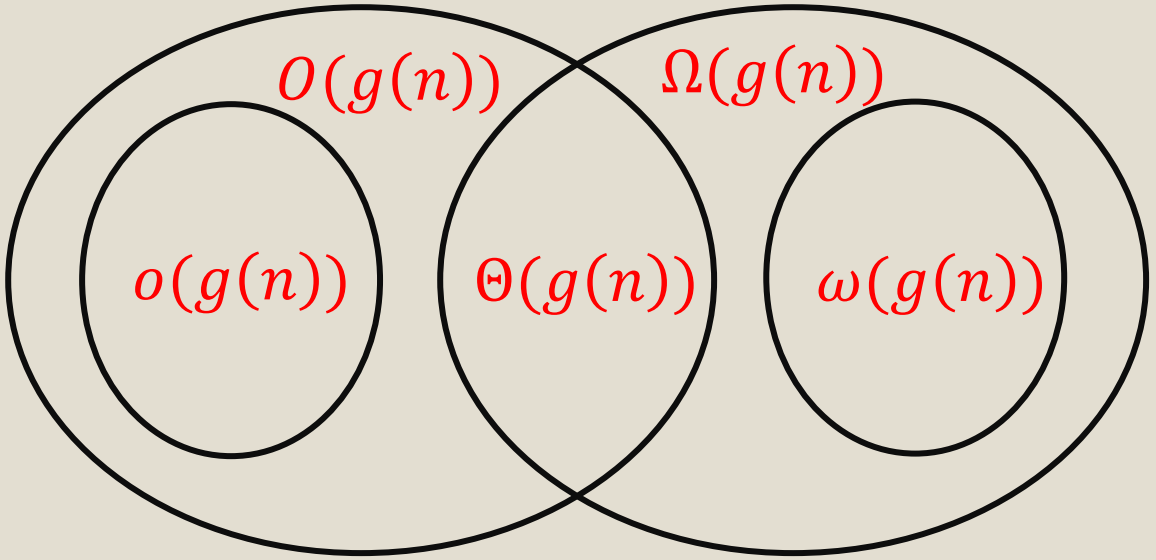
Asymptotic Notations: Properties

- Symmetry:
  - $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$
- Transpose symmetry:
  - $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$
  - $f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$

Asymptotic Notations: Properties

- Prove that $o(g(n)) \cap \omega(g(n)) = \emptyset$.
  - Assume that $f(n) \in o(g(n)) \cap \omega(g(n))$
  - From $f(n) \in o(g(n))$, $\forall c > 0, \exists n_0' > 0 : \forall n \geq n_0', 0 \leq f(n) < cg(n)$
  - From $f(n) \in \omega(g(n))$, $\forall c > 0, \exists n_0'' > 0 : \forall n \geq n_0'', 0 \leq cg(n) < f(n)$
  - Let $n_0 = \max\{n_0', n_0''\}$
  - Then, $\forall n \geq n_0 : f(n) < cg(n) \wedge cg(n) < f(n)$
    - This is impossible, a contradiction

Asymptotic Notations: Properties



Useful but Incorrect Analogies

Asymptotic Notation	Is Like
$f(n) = O(g(n))$	$f(n) \leq g(n)$
$f(n) = \Theta(g(n))$	$f(n) = g(n)$
$f(n) = \Omega(g(n))$	$f(n) \geq g(n)$
$f(n) = o(g(n))$	$f(n) < g(n)$
$f(n) = \omega(g(n))$	$f(n) > g(n)$

Warning: DO NOT formally use them in HWs/Exams/Interviews

Exercise

Express functions in A in asymptotic notation using functions in B.

A	B	
$5n^2 + 100n$	$3n^2 + 2$	$A \in \Theta(B)$

$A \in \Theta(n^2)$, $n^2 \in \Theta(B) \Rightarrow A \in \Theta(B)$

$\log_3(n^2)$	$\log_2(n^3)$	$A \in \Theta(B)$

$\log_b a = \log_c a / \log_c b$; $A = 2\lg n / \lg 3$, $B = 3\lg n$, $A/B = 2/(3\lg 3)$

$\lg^2 n$	$n^{1/2}$	$A \in o(B)$

$\lim_{n\to\infty} (\lg^a n / n^b) = 0$ (here $a = 2$ and $b = 1/2$) $\Rightarrow A \in o(B)$

Common Functions

Motonocity

- A function $f(n)$ is
  - Monotonically increasing if $m \leq n \implies f(m) \leq f(n)$
  - Monotonically decreasing if $m \leq n \implies f(m) \geq f(n)$
  - Strictly increasing if $m < n \implies f(m) < f(n)$
  - Strictly decreasing if $m < n \implies f(m) > f(n)$

- Question: Which of the above is true for $f(n) = n^2$ where $n \geq 0$?

Polynomials

- $p(n)$ is a polynomial in n of degree d if
  - $p(n) = a_d p^d + a_{d-1} p^{d-1} + \cdots + a_0$ AND $a_d \neq 0$ ($d \geq 0$ is an integer constant)
  - $a_d, a_{d-1,}$ … are co-efficients
  - $p(n) = \Theta(n^d)$
- A function $f(n) = O(n^k)$ for a constant k is called polynomially bounded
- Common polynomials:
  - $O(n)$: Linear running time, $O(n^2)$: Quadratic running time, $O(n^3)$: Cubic running time

Exponentials

- Functions that have a^n terms (a is a constant)
  - Polynomials: variable base, constant exponent
  - Exponentials: constant base, variable exponent
- For any real constants a, b with $a > 1$
  - $n^b = o(a^b)$, because $\lim_{n \to \infty} \frac{n^b}{a^b} = 0$
- See book for some (in)equalities regarding e^x

Identities

$$
\begin{aligned}
a^0 &= 1 , \\
a^1 &= a , \\
a^{-1} &= 1/a , \\
(a^m)^n &= a^{mn} , \\
(a^m)^n &= (a^n)^m , \\
a^m a^n &= a^{m+n} .
\end{aligned}
$$

Logarithms

- $x = \log_b a \Leftrightarrow a = bx$

Identities

$$
\begin{aligned}
a &= b^{\log_b a}, \\
\log_c(ab) &= \log_c a + \log_c b, \\
\log_b a^n &= n \log_b a, \\
\log_b a &= \frac{\log_c a}{\log_c b}, \\
\log_b(1/a) &= -\log_b a, \\
\log_b a &= \frac{1}{\log_a b}, \\
a^{\log_b c} &= c^{\log_b a},
\end{aligned}
$$

CLRS Notations

$$
\begin{aligned}
\lg n &= \log_2 n && \text{(binary logarithm)}, \\
\ln n &= \log_e n && \text{(natural logarithm)}, \\
\lg^k n &= (\lg n)^k && \text{(exponentiation)}, \\
\lg \lg n &= \lg(\lg n) && \text{(composition)}.
\end{aligned}
$$

Logarithms

- Log bases do not matter in asymptotic notations
  - $\log_a n = \Theta(\log_b n)$. **Why?**
- A function $f(n) = O(\lg^k n)$ for a constant k is called polylogarithmically bounded
  - $\lg^b n = o(n^a)$ for any $a > 0$

Identities

$$
\begin{aligned}
a &= b^{\log_b a}, \\
\log_c(ab) &= \log_c a + \log_c b, \\
\log_b a^n &= n \log_b a, \\
\log_b a &= \frac{\log_c a}{\log_c b}, \\
\log_b(1/a) &= -\log_b a, \\
\log_b a &= \frac{1}{\log_a b}, \\
a^{\log_b c} &= c^{\log_b a},
\end{aligned}
$$

Factorials

- How many ways can you arrange n items?

$$n! = \begin{cases} 1 & \text{if } n = 0 , \\ n \cdot (n-1)! & \text{if } n > 0 . \end{cases}$$

- Stirling's approximation

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

- Using the above:

$$\begin{aligned} n! &= o(n^n) , \\ n! &= \omega(2^n) , \\ \lg(n!) &= \Theta(n \lg n) , \end{aligned}$$

Growth Rates

n	$\lg n$	$\lg^2 n$	$\sqrt{n}$	$n \lg n$	n^2	n^3	2^n
4	2	4	2	8	16	64	16
16	4	16	4	64	256	4096	65536
64	6	36	8	384	4096	2.62E+05	1.84E+19
256	8	64	16	2048	65536	1.68E+07	1.16E+77
1024	10	100	32	10240	1.05E+06	1.07E+09	1.79E+308
4096	12	144	64	4.92E+04	1.68E+07	6.87E+10	E+1233
16384	14	196	128	2.29E+05	2.68E+08	4.40E+12	E+4932
65536	16	256	256	1.05E+06	4.29E+09	2.81E+14	E+19728
262144	18	324	512	4.72E+06	6.87E+10	1.80E+16	E+78913

Growth Rates

Running times on a million instruction/sec machine

n	$\lg n$	$\lg^2 n$	$\sqrt{n}$	$n \lg n$	n^2	n^3	2^n
4	<1 sec	<1 sec	<1 sec	< 1 sec	<1 sec	<1 sec	<1 sec
16	<1 sec	<1 sec	<1 sec	< 1 sec	<1 sec	<1 sec	<1 sec
64	<1 sec	<1 sec	<1 sec	< 1 sec	<1 sec	<1 sec	2*10^8 years
256	<1 sec	<1 sec	<1 sec	< 1 sec	<1 sec	16 sec	very long
1024	<1 sec	<1 sec	<1 sec	< 1 sec	1 sec	18 min	very long
4096	<1 sec	<1 sec	<1 sec	< 1 sec	16 sec	19 hours	very long
16384	<1 sec	<1 sec	<1 sec	< 1 sec	5 min	51 years	very long
65536	<1 sec	<1 sec	<1 sec	1 sec	1 hour	3257 years	very long
262144	<1 sec	<1 sec	<1 sec	5 sec	19 hours	2*10^5 years	very long

Growth Rate	Name	Examples and Notes
$\Theta(1)$	Constant	Add to linked list (unsorted) Assign value to variable
$\Theta(\lg n)$	Logarithmic	Binary Search
$\Theta(\sqrt{n})$	Square Root	Primality Testing;
$\Theta(n)$	Linear	Find item in unsorted lists, String comparison, Special-purpose sorts
$\Theta(n \lg n)$	Log-Linear	Good sorting algorithms (merge sort, heapsort)
$\Theta(n^2)$	Quadratic	Naive sorting algorithms (Bubble, Insertion, Selection)
$\Theta(n^3)$	Cubic	Naive Matrix Multiplication Floyd-Warshall (All-pairs shortest paths)
$\Theta(n^p), p > 3$	High-Order Polynomial	Perfect graph recognition, $O(n^9)$. https://cstheory.stackexchange.com/questions/6660/polynomial-time-algorithms-with-huge-exponent-constant/
$\Theta(c^n), c > 1$	Exponential	Find all subsets. Algorithm is inefficient, if can't do better, then the problem is hard.
$\Theta(n!)$	Factorial	Find all permutations.

Thank You!