Analysis of Algorithm Efficiency

The Analysis Framework

Efficiency of algorithms:

- **Time efficiency** (time complexity): indicates **how fast an algorithm runs**.
- **Space efficiency** (space complexity): refers to the **amount of memory units** required by the algorithm **in addition** to the space needed for the input and output.

The Analysis Framework

- **Input size**
- **Units for measuring** the running time
- **Order of growth**
- **Worst-case**, **best-case** and **average-case** efficiencies

Input Size

- Almost all algorithms **run longer** on **larger inputs**.
- Algorithm efficiency is a **function of parameter(s)** indicating the algorithm **input size**.
- **Input size:**
  - the size of a list (sorting, searching, etc.)
  - the degree of a polynomial
  - the order of a matrix
  - the number of characters

Units for Measuring Running Time

- **Standard unit** of time measurement: seconds, milliseconds, …

- **The running time of a program** implementing the algorithm

Drawbacks: dependence on

- the speed of a computer
- the quality of a program
- the quality of compiler
- the difficulty of clocking

Units for Measuring Running Time

- **A metric should not depend on the external factor!**

- We can count **the number of times each of the algorithm's operation** executed: **very difficult** and **usually unnecessary**.

- We **identify** the **most important operation** of the algorithm, called the **basic operation**, the operation contributing the most to the total running time.

- We **compute** the **number of times** the basic operation executed.

Algorithm Efficiency

Algorithm SequentialSearch($A[0..n-1], K$)

// **Input**: An array $A[0..n-1]$ & key K

// **Output**: The first i that $A[i] = K$ or -1 if no match

$i \leftarrow 0$

while $i < n$ **and** $A[i] \neq K$ **do**

$i \leftarrow i + 1$

if $i < n$ **return** i

else return -1

Algorithm Efficiency

Algorithm SequentialSearch($A[0..n-1], K$)

// **Input**: An array $A[0..n-1]$ & key K

// **Output**: The first i that $A[i] = K$ or -1 if no match

$i \leftarrow 0$

while $i < n$ **and** $A[i] \neq K$ **do**

$i \leftarrow i + 1$

if $i < n$ **return** i

else return -1

Units for Measuring Running Time

- **Basic operation:**

  the most time-consuming operation in the algorithm’s innermost loop.

- **Example:**

  Most sorting algorithms work by comparing elements (keys) of a list with each other.

  The **basic operation** is a **key comparison**.

Orders of Growth

- **Why to emphasize on the count's order of growth for large input sizes?**

- A difference in running times on small inputs is not what really distinguishes efficient algorithms from inefficient ones.

- Compare the algorithms for computing the **gcd** of two positive integers

Orders of Growth

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10	$3.3 \cdot 10$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.0 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Algorithm Efficiency

Algorithm SequentialSearch($A[0..n-1], K$)

// **Input**: An array $A[0..n-1]$ & key K

// **Output**: The first i that $A[i] = K$ or -1 if no match

$i \leftarrow 0$

while $i < n$ **and** $A[i] \neq K$ **do**

$i \leftarrow i + 1$

if $i < n$ **return** i

else return -1

Worst-Case Efficiency

Worse-case efficiency of an algorithm is the **longest running time** for **any** input size n.

$$C_{worst}(n) = n$$

- There are no matching elements.
- The first matching element is the last one in the list.

Note:

- The worst-case analysis provides an **upper bound** on the running time for any input – a **guarantee** that the algorithm will never take any longer.

Best-Case Efficiency

Best-case efficiency of an algorithm is the **smallest running time** for all possible inputs size n.

$$C_{best}(n) = 1$$

- The first element in the list is equal to the search key.

Note:

- The analysis of the best-case efficiency is not as important as that of the worst-case efficiency.
- Some algorithms run faster in specific inputs, for example, the insertion sort works very fast on almost sorted inputs.

Average-Case Efficiency

- **Worst-case and best-case analysis do not result in necessary information about an algorithm's behavior on a "typical" or "random" input.**

- **Average-case efficiency** of an algorithm is the **average running time** for all possible inputs size n.

- To analyze the algorithm's average case efficiency, we must make **assumption** about possible inputs of size n.

Average-Case Efficiency

Standard assumptions:

- the probability of successful search is equal to p:

$$0 \leq p \leq 1$$

- the probability of the first match occurring in the ith position in the list is same for each i, $1 \leq i \leq n$:

$$\frac{p}{n}$$

Average-Case Efficiency

$$C_{avg}(n) = 1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n} + n(1-p)$$

$$= \frac{p}{n} \cdot [1 + 2 + \cdots + n] + n(1-p)$$

$$= \frac{p}{n} \cdot \frac{n(n+1)}{2} + n(1-p) = \frac{p(n+1)}{2} + n(1-p)$$

- If $p = 0$,

$$C_{avg}(n) = n$$

- If $p = 1$,

$$C_{avg}(n) = (n+1)/2$$

Unsuccessful search:
# of comparisons n with probability of the search $(1-p)$

Asymptotic Notations

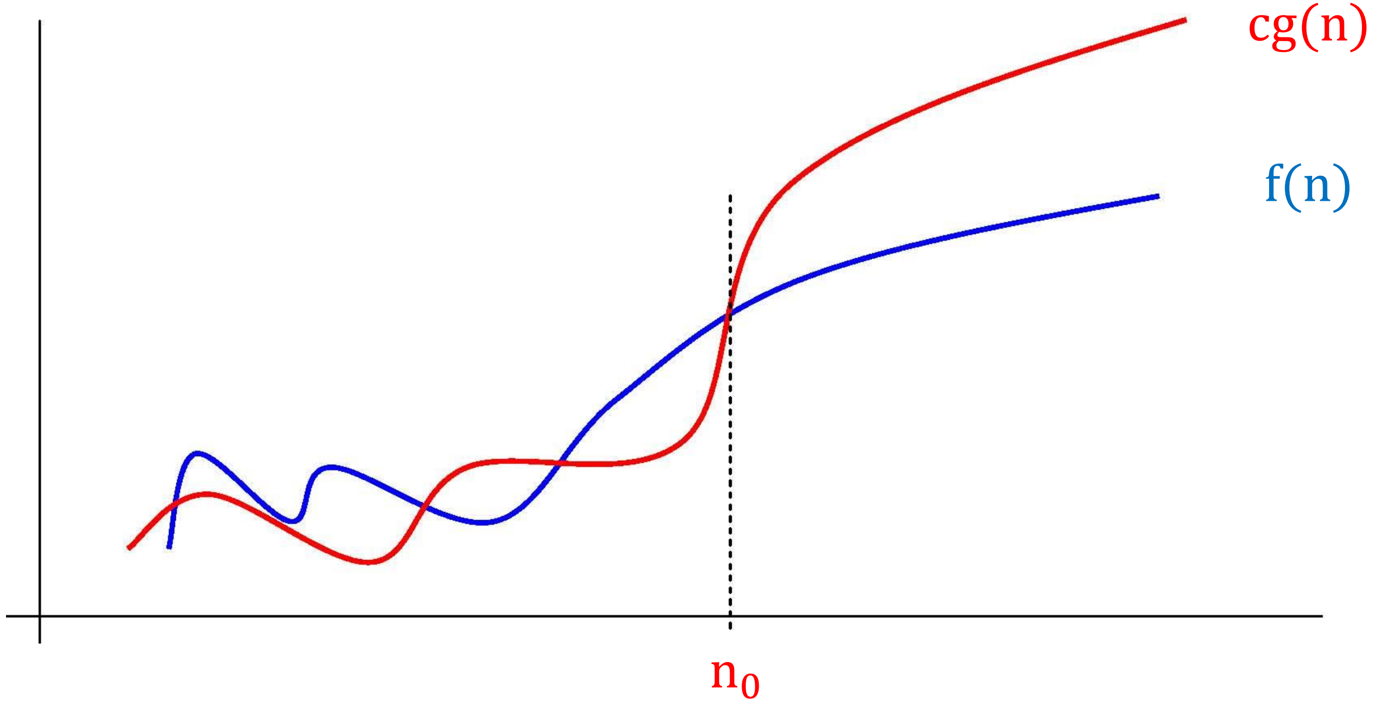
Asymptotic Notations

To compare and rank orders of growth, we use **three asymptotic notations:**

- O (Big oh) notation
- Ω (Big omega) notation
- Θ (Big theta) notation
- o (small oh), ω (small omega)

Asymptotic Notations

Definition: A function $f(n)$ is said to be in $O(g(n))$, denoted by $f(n) \in O\big(g(n)\big)$, if there exist some positive constant c and some nonnegative integer n_0 such that $f(n) \le c \cdot g(n)$, for all $n \ge n_0$.



Asymptotic Notations

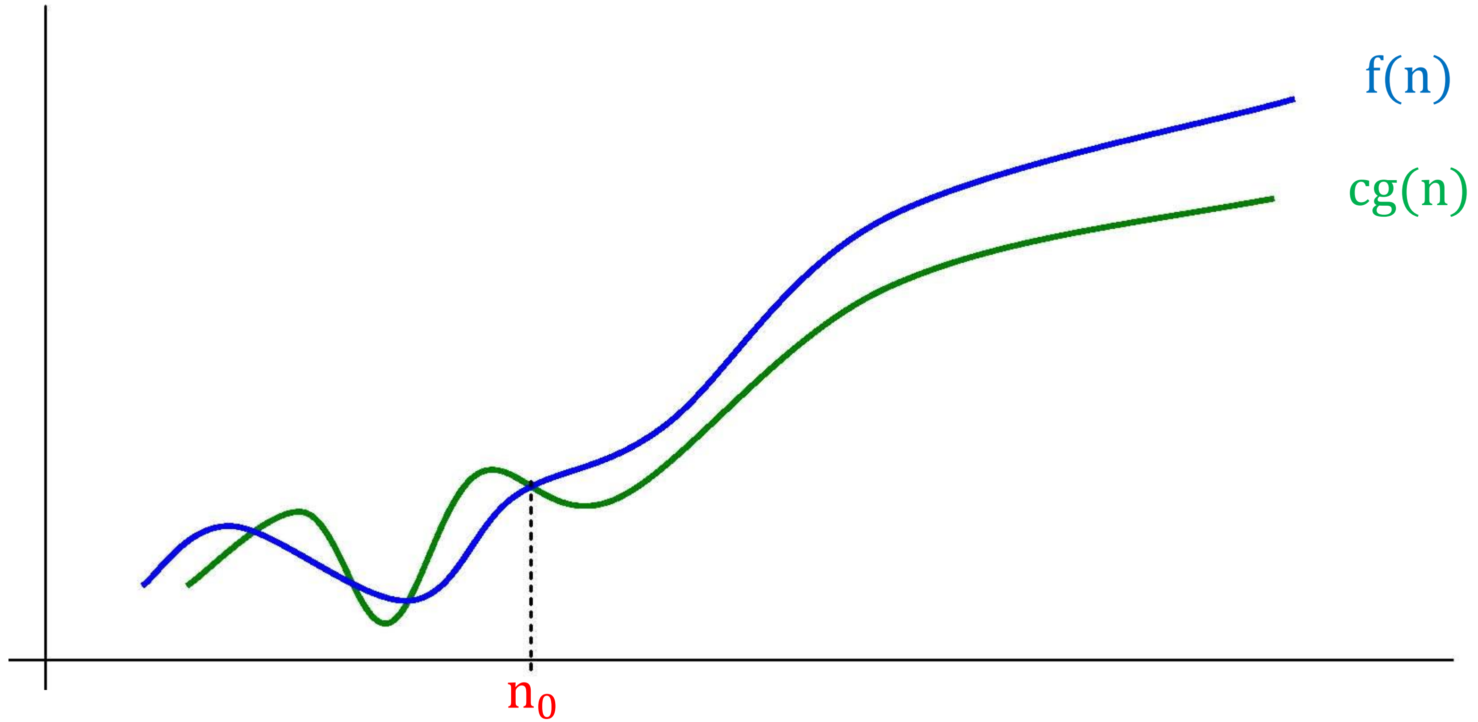
Definition: A function $f(n)$ is said to be in $O(g(n))$, denoted by $f(n) \in O(g(n))$, if there exist some positive constant c and some nonnegative integer n_0 such that $f(n) \leq c \cdot g(n)$, for all $n \geq n_0$.

Example:

- $100n + 5 \in O(n^2)$
- $1000n^2 - 6n + 5 \in O(n^2)$
- $n^3 \notin O(n^2)$

Asymptotic Notations

Definition: A function $f(n)$ is said to be in $\Omega(g(n))$, denoted by $f(n) \in \Omega\left(g(n)\right)$, if there exist some positive constant c and some nonnegative integer n_0 such that $f(n) \geq c \cdot g(n)$, for all $n \geq n_0$.



Asymptotic Notations

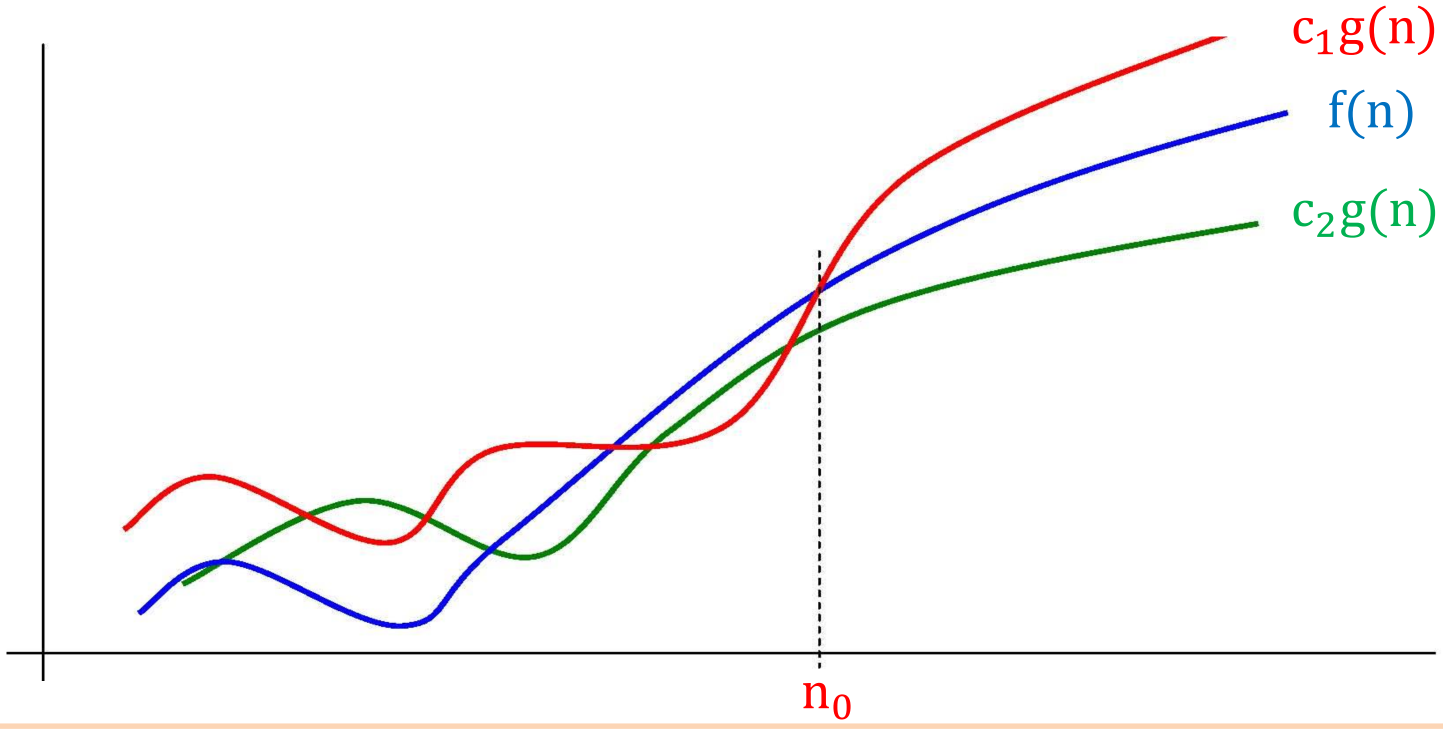
Definition: A function $f(n)$ is said to be in $\Omega(g(n))$, denoted by $f(n) \in \Omega\big(g(n)\big)$, if there exist some positive constant c and some nonnegative integer n_0 such that $f(n) \geq c \cdot g(n)$, for all $n \geq n_0$.

Example:

- $100n \notin \Omega(n^2)$
- $1000n^2 + 5n + 5 \in \Omega(n^2)$
- $n^3 \in \Omega(n^2)$

Asymptotic Notations

Definition: A function $f(n)$ is said to be in $\Theta(g(n))$, denoted by $f(n) \in \Theta(g(n))$, if there exist some positive constants c_1 and c_2, and some nonnegative integer n_0 such that $c_2 \cdot g(n) \le f(n) \le c_1 \cdot g(n)$, for all $n \ge n_0$.



Asymptotic Notations

Definition: A function $f(n)$ is said to be in $\Theta(g(n))$, denoted by $f(n) \in \Theta\big(g(n)\big)$, if there exist some positive constants c_1 and c_2, and some nonnegative integer n_0 such that $c_2 \cdot g(n) \leq f(n) \leq c_1 \cdot g(n)$, for all $n \geq n_0$.

Example:

- $\frac{1}{2}n(n-1) \in \Theta(n^2)$
- $1000n^2 + 5n + 5 \in \Theta(n^2)$
- $n^3 + \log n \in \Theta(n^3)$

Asymptotic Notations

Theorem: $f(n) \in O\big(f(n)\big)$.

Theorem: $f(n) \in O(g(n))$ iff $g(n) \in \Omega\big(f(n)\big)$.

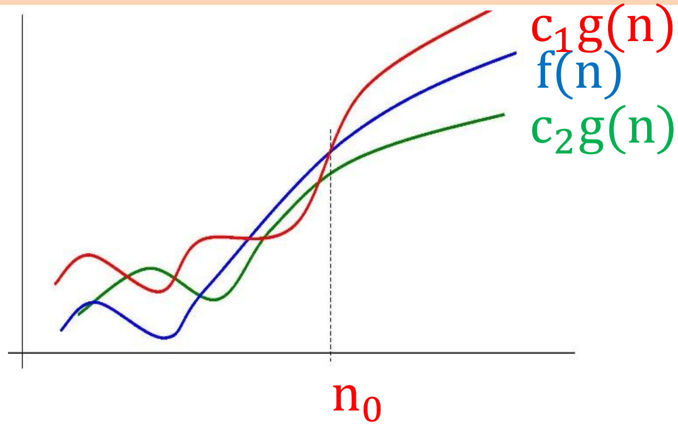
Theorem: If $f(n) \in O\big(g(n)\big)$ and $g(n) \in O\big(h(n)\big)$, then $f(n) \in O\big(h(n)\big)$.

Theorem: If $f_1(n) \in O\big(g_1(n)\big)$ and $f_2(n) \in O\big(g_2(n)\big)$, then

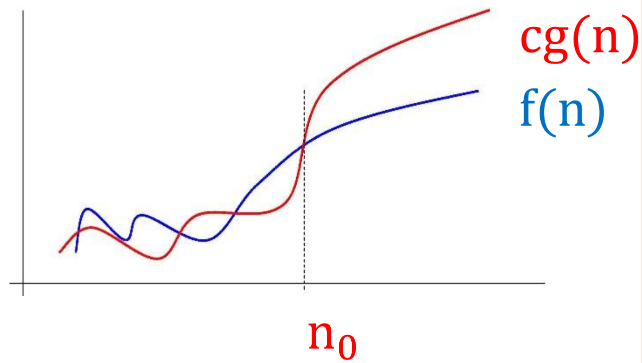
$$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\}).$$

Asymptotic Notations

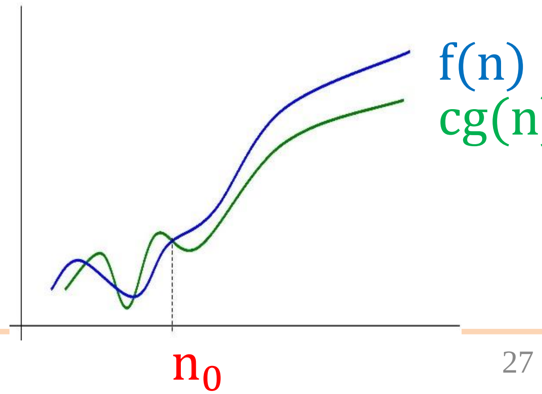
- If $f(n)$ same growth with $g(n)$
  - $f(n) \in \Theta(n)$ and also $O(n), \Omega(n)$



- If $f(n)$ smaller growth than $g(n)$
  - $f(n) \in O(n)$



- If $f(n)$ bigger growth than $g(n)$
  - $f(n) \in \Omega(n)$



Asymptotic Notations

Limits for comparing orders of growth: In order to compare the orders of growth of two specific functions, one can use the limit of the ratio of these functions

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \begin{cases} 0, & f(n) \in O\big(g(n)\big) \\ c, & f(n) \in \Theta\big(g(n)\big) \\ \infty, & f(n) \in \Omega\big(g(n)\big) \end{cases}$$

Example:

- $10n$ vs. n^2
- $n(n+1)/2$ vs. n^2

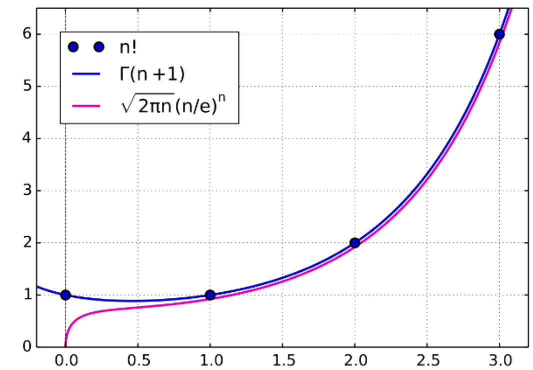
Asymptotic Notations

L'Hôpital's rule: If $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ and the derivatives f', g' exist, then

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

Stirling's formula: For large values of n,

$$n! \approx \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n.$$



Example: **(a)** $\log_2 n$ vs. $\sqrt{n}$; **(b)** 2^n vs. $n!$

Order of Growth

- All logarithmic functions $\log_a n$ belong to the same class $\Theta(\log_2 n)$ no matter what the logarithm's base $a > 1$ is.

- All polynomials of the same degree k belong to the same class:

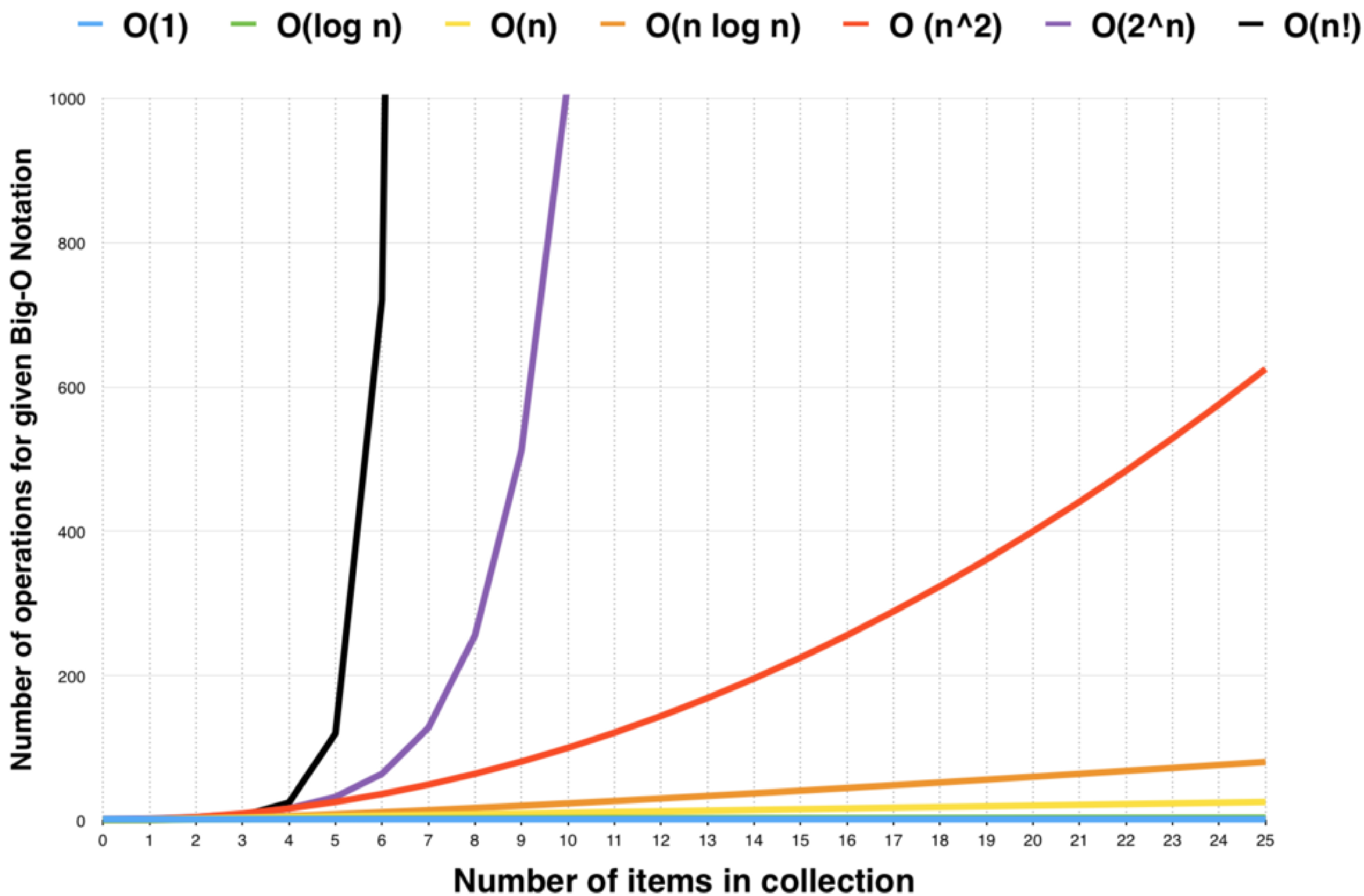
$$a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0 \in \Theta(n^k).$$

- Exponential functions a^n have different orders of growth for different a's.

- order $\log n$ < order n^α $(\alpha > 0)$ < order a^n < order $n!$ < order n^n

Basic Asymptotic Efficiency Classes

1	constant
$\log n$	logarithmic
n	linear
$n \log n$	$n - \log - n$ or linearithmic
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial



https://medium.freecodec

Exercise

- Using informal definitions of O, Θ, Ω,
  - Determine: True or False??

a. $\frac{n(n+1)}{2} \in O(n^3)$

b. $\frac{n(n+1)}{2} \in O(n^2)$

c. $\frac{n(n+1)}{2} \in \Theta(n^3)$

d. $\frac{n(n+1)}{2} \in \Omega(n)$

Exercise

- For each of the following functions, indicate the class $O(g(n))$ the function belongs to. (Use the simplest $g(n)$ possible in your answers.) Prove your assertions.

a. $(n^2 + 1)^{10}$

b. $\sqrt{10n^2 + 7n + 3}$

c. $2^{n+1} + 3^{n-1}$

d. $\lfloor \log n \rfloor$