EXAMPLES

- Is 1,000,000 n in O(n)? Yes.
  $c = 2{,}000{,}000$, $n_0 = 0$

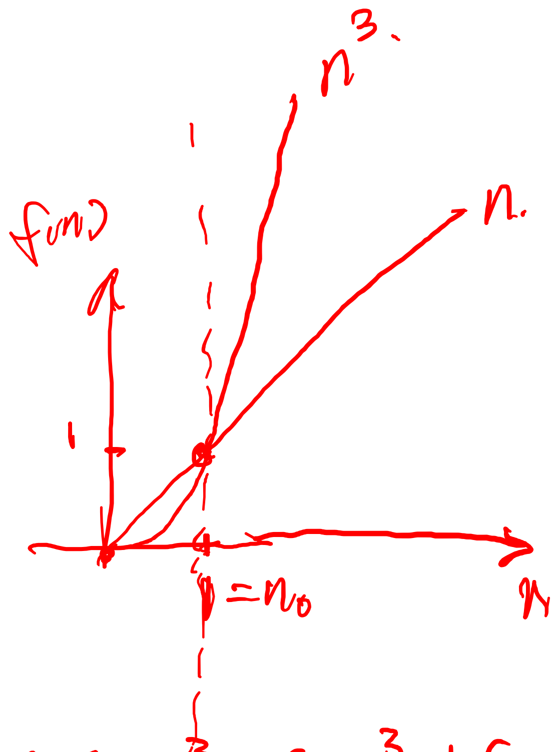
- Is n in O(n³)? Yes.
  c, n_0; $c = 1$
  $n \le 1 \cdot n^3$
  $1 \le n^2$
  $n \ge 1$ (n_0)

- Is n³ + n² + n in O(n³)?
  $\le O(n^3)$ dominant terms ✓
  $n^3 + n^2 + n \le c_1 n^3 + c_2 n^3 + c_3 n^3$
  $= (c_1 + c_2 + c_3) \cdot n^3$
  $= c \cdot n^3 = O(n^3)$

- Is n² in O(n)? No.
  $n^2 \not\le c \cdot n$

- Is e³ⁿ in O(eⁿ)? No.
  $n \le c$
  $e^{3n} = (e^n)^3 = e^n \cdot e^n \cdot e^n$, $e^n \not\le c$
  $e^{3n} \not\le O(e^n)$

- Is 10ⁿ in O(2ⁿ)? No.
  $10^n = (2 \cdot 5)^n = 2^n \cdot 5^n$, $5^n \not\le c$



Exponent Rules:

$$\sqrt[n]{a^m} = a^{\frac{m}{n}}$$

$$a^m a^n = a^{m+n}$$

$$\frac{a^m}{a^n} = a^{m-n}$$

$$(a^m)^n = a^{mn}$$

$$(ab)^n = a^n b^n$$

$$\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$$

$$a^{-n} = \frac{1}{a^n}$$

$$\frac{1}{a^{-n}} = a^n$$

Table of Important Big-O Sets

- Arranged from smallest to largest, happiest to saddest, in order of increasing domination

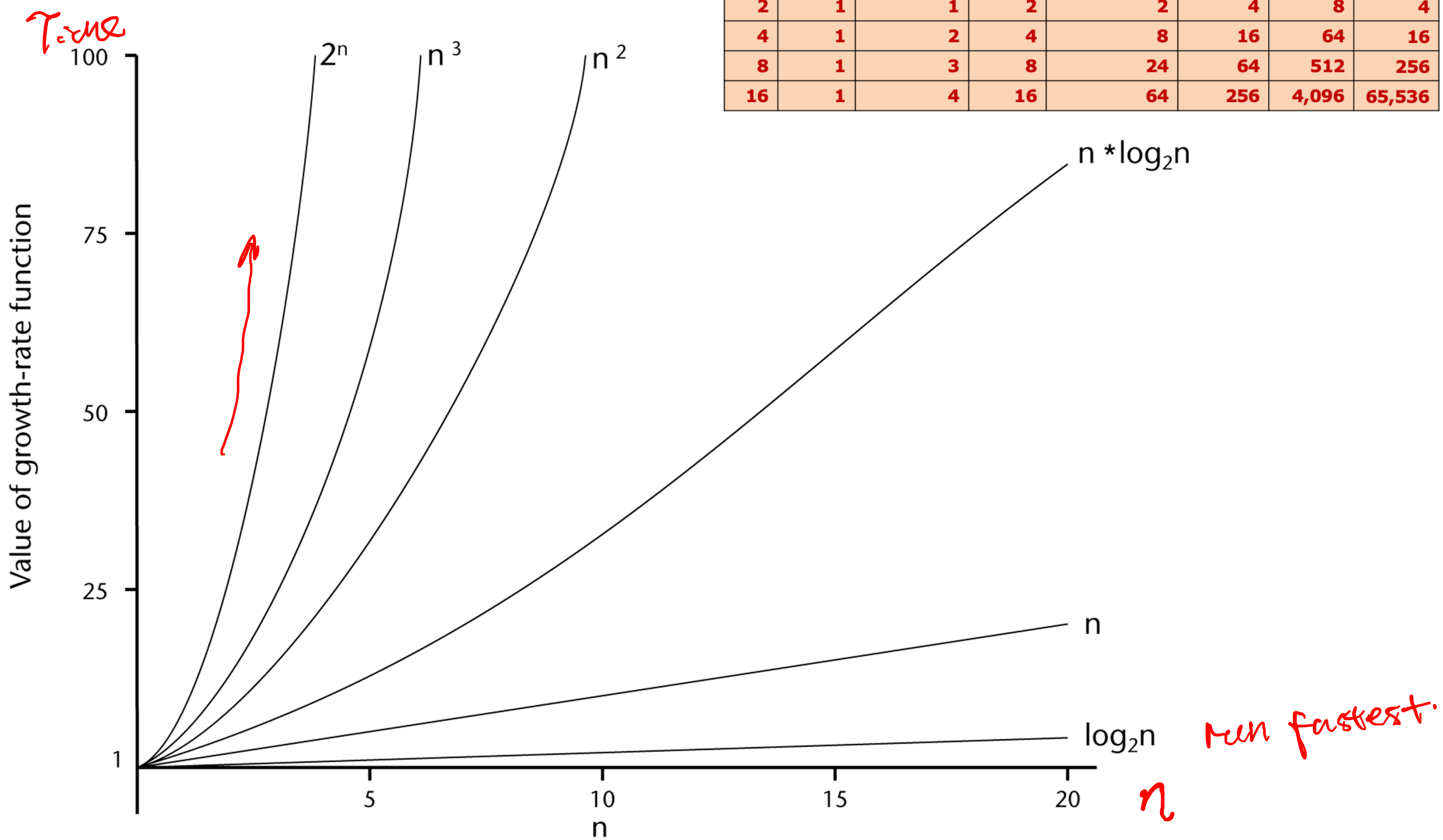
function	common name
men faster O(1)	constant
is a subset of O(log n)	logarithmic
is a subset of O($\log^2$ n)	log-squared [that's $(\log n)^2$]
is a subset of O(root(n))	root-n [that's the square root]
is a subset of O(n)	linear
is a subset of O(n log n)	n log n
is a subset of O(n^2)	quadratic
is a subset of O(n^3)	cubic
is a subset of O(n^4)	quartic
is a subset of O(2^n)	exponential
slowest is a subset of O(e^n)	exponential (but more so)

faster efficient ✓

- Algorithms that run in O(n log n) time or faster are considered efficient. Algorithms that take n^7 time or more are usually considered useless. In the region between n log n and n^7, the usefulness of an algorithm depends on the typical input sizes and the associated constants hidden by the Big-Oh notation.

A Comparison of Growth-Rate Functions

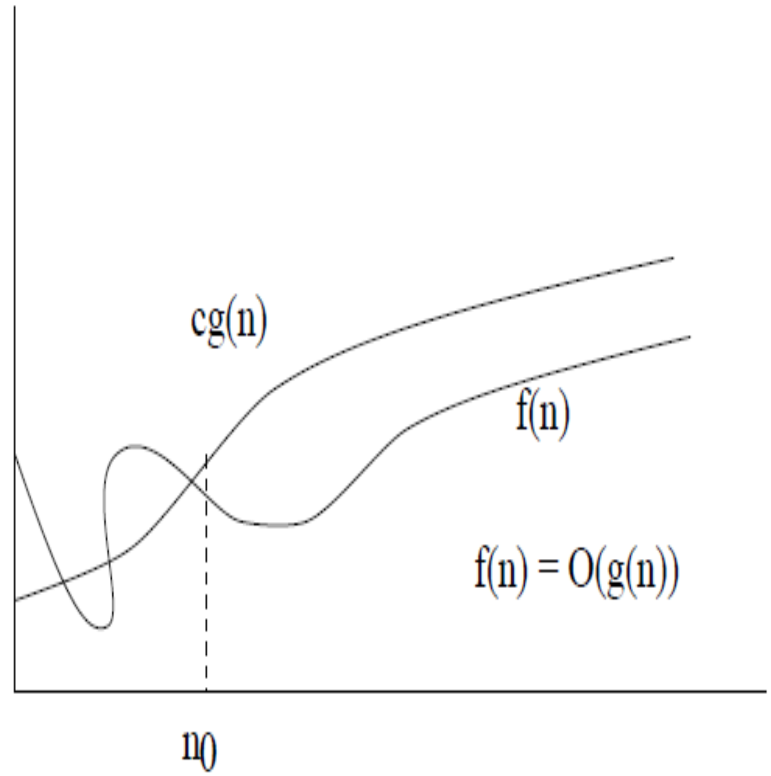
n	O(1)	O(log n)	O(n)	O(n log n)	O(n^2)	O(n^3)	O(2^n)
1	1	1	1	1	1	1	1
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65,536



EXAMPLES - Textbook

$\leq$ $\geq$

- Show that $4n^2$ is $O(n^3)$
- $0 <= 4n^2 <= cn^3$
- $0/n^3 <= 4n^2/n^3 <= cn^3/n^3$
- $0 <= 4/n <= c$
- $0 <= 4/1 <= c$, max at $n=1$
- $0 <= 4 <= c$, so $c=4$
- For n_0, $0 <= 4/n_0 <= 4$
- Multiply by $n_0/4$ for $0 <= 4/4 <= n_0$ yielding $0 <= 1 <= n_0$
- For $c = 4$ and $n >= n_0 = 1$, $4n^2$ is $O(n^3)$
- $4n^2 <= O(n^3)$



Calculating Algorithm Efficiency

get f(n)

Linear loops $i<n$, $\leq n$; i++

```
for(i=0;i<100;i++)
        statement block
```

= O(1)

f(n) = O(n)

O(1)·n

Logarithmic Loops

```
for(i=1;i<64;i*=2)
        statement block;
```

O(1) ← statement block;

f(n) = O(log n)

O(n)

Nested Loops

Linear logarithmic

```
for(i=0;i<10;i++)
        for(j=1; j<10;j*=2)
                statement block;
```

n, n, m

statement block; → O(1)

f(n)= O(n log n)

In the following for-loop: (with <)

```
for (int i = k; i < n; i = i * m){
        statement1;
        statement2;
 }
```

$=1$, 2

$\lceil \log_2 (n/1) \rceil = \lceil \log_2 n \rceil$

– The number of iterations is: $\lceil (\mathbf{Log_m\ (n / k)}) \rceil$

log n

In the following for-loop: (with <=)

```
for (int i = k; i <= n; i = i * m){
        statement1;
        statement2;
 }
```

– The number of iterations is: $\lfloor (\mathbf{Log_m\ (n / k) + 1}) \rfloor$

$O(n) \cdot O(\log n) \cdot O(1)$

$= O(n \cdot \log n)$

$O(n) \cdot O(\log m) \cdot O(1)$

$= O(n \cdot \log m)$

Calculating Algorithm Efficiency

Nested Loops

Quadratic Loop

```
for(i=0;i<10;i++)
        for(j=1; j<10;j++)
                statement block;
```

$f(n) = O(n^2)$

Dependent Quadratic Loop

```
for(i=0;i<10;i++)
        for(j=0; j<=i;j++)
                statement block;
```

$f(n) = 1 + 2 + \ldots + n = O(n(n+1)/2)$