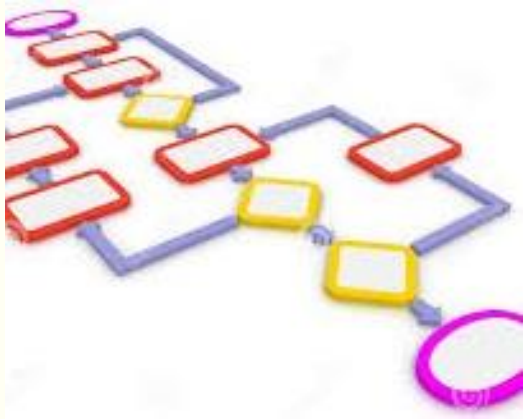
การออกแบบและวิเคราะห์ขั้นตอนวิธี

DESIGN AND ANALYSIS OF ALGORITHMS

02-212-212



อ.ธิดาวรรณ คล้ายศรี

บทที่1

Fundamental of Design and Analysis Algorithms

1.1 การแก้ปัญหาและขั้นตอนวิธี (Problems Solving and Algorithms)

1.2 An Introduction to Algorithm Design (บทนำการออกแบบขั้นตอนวิธี)

1.3 คณิตศาสตร์พื้นฐานเพื่อการวิเคราะห์ (Mathematic Preliminaries for Analysis)

1.4 Analysis of Algorithms (การวิเคราะห์อัลกอริธึ่ม)

1.5 Growth of Functions (แนวโน้มการเพิ่มขึ้นของฟังก์ชั่น)

ทบทวนก่อนเรียน

1.3 **คณิตศาสตร์พื้นฐานเพื่อการวิเคราะห์** (Mathematic Preliminaries for Analysis)

1.3.1 **อนุกรม (Series)** นำมาใช้งานบ่อย:

- อนุกรมเลขคณิต $\sum_{i=1}^{n} i$

  → ฟังก์ชั่นความซับซ้อนด้านเวลา = n(n+1)/2 หรือประมาณ n^2

- อนุกรมเรขาคณิต $\sum_{i=0}^{n} x^i$

  → ฟังก์ชั่นความซับซ้อนด้านเวลา = $(x^{n+1} - 1)/ (x-1)$ หรือประมาณ x^n

1.3 คณิตศาสตร์พื้นฐานเพื่อการวิเคราะห์ (Mathematic Preliminaries for Analysis)

1.3.2 ฟังก์ชั่นความซับซ้อนด้านเวลาของอัลกอริธึ่ม:

- ฟังก์ชั่น Exponential
- ฟังก์ชั่น Logarithm
- ฟังก์ชั่น Factorial

บทที่1

Fundamental of Design and Analysis Algorithms

1.1 การแก้ปัญหาและขั้นตอนวิธี (Problems Solving and Algorithms)

1.2 An Introduction to Algorithm Design (บทนำการออกแบบขั้นตอนวิธี)

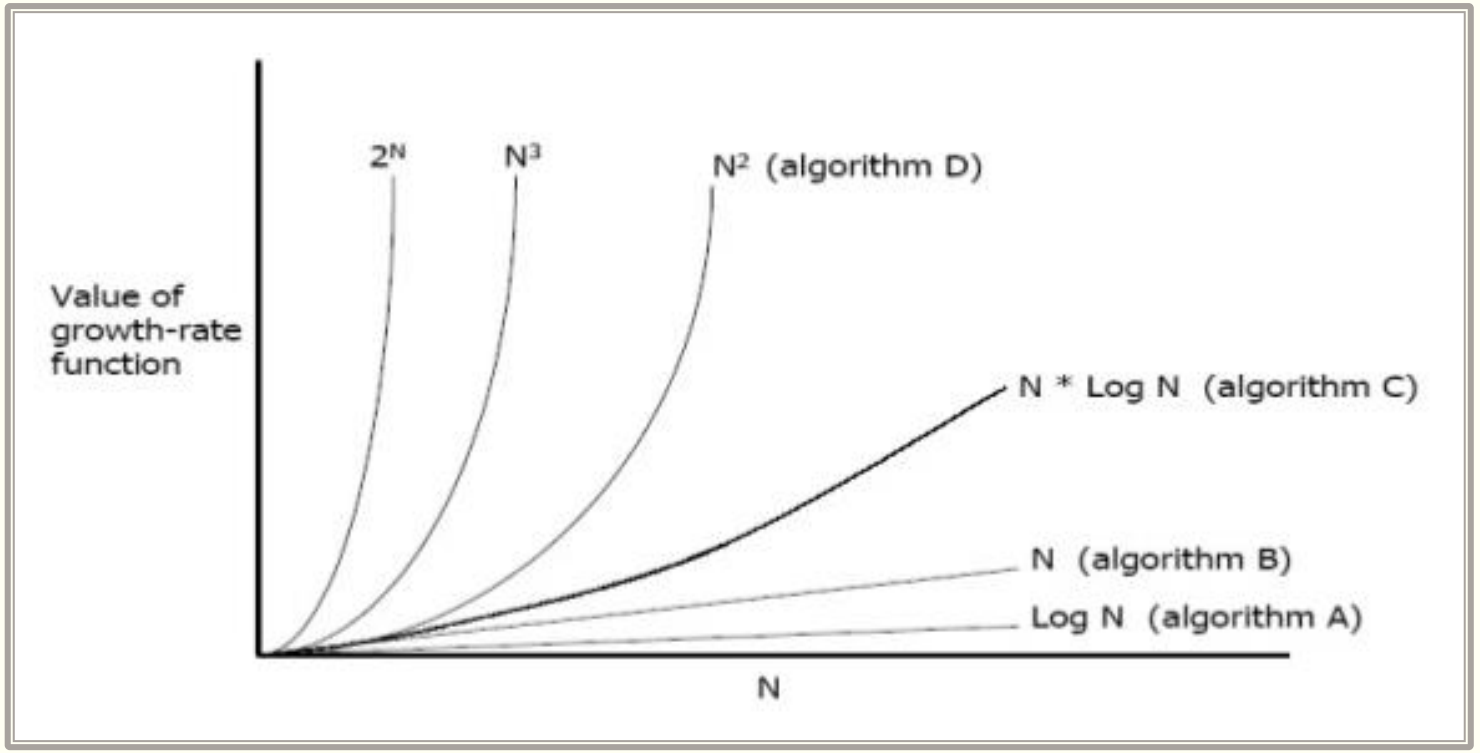
1.3 คณิตศาสตร์พื้นฐานเพื่อการวิเคราะห์ (Mathematic Preliminaries for Analysis)

1.4 Analysis of Algorithms (การวิเคราะห์อัลกอริธึ่ม)

1.5 Growth of Functions (แนวโน้มการเพิ่มขึ้นของฟังก์ชั่น)

1.4 Analysis of Algorithms (การวิเคราะห์อัลกอริธึ่ม)

ในการแก้ปัญหาทางคอมพิวเตอร์ เราอาจออกแบบอัลกอริธึ่มไว้หลายอัลกอริธึ่ม เช่น 4 อัลกอริธึ่ม (A-D) ข้างล่างนี้



[ภาพ: internet 2017]

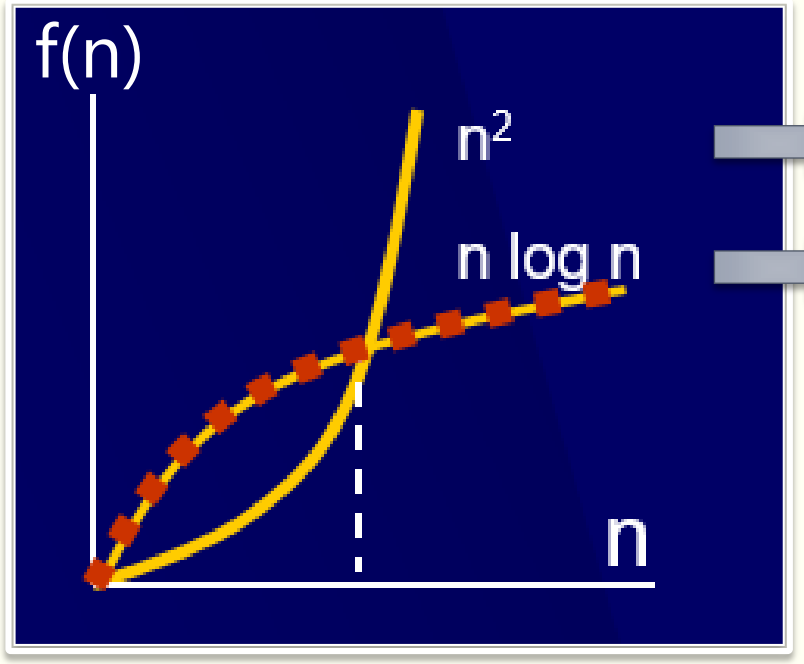
1.4 Analysis of Algorithms (**การวิเคราะห์อัลกอริธึ่ม**)

ในการวิเคราะห์ประสิทธิภาพของอัลกอริธึ่มทางคอมพิวเตอร์ที่ได้ออกแบบไว้นั้น
มักพิจารณาจากความซับซ้อนของอัลกอริธึ่มใน 2 ด้านคือ

- **Time Complexity** (ความซับซ้อนด้านเวลา)
  วัดจากจำนวน n รอบที่ใช้ในการคำนวณ (+ - * /) เปรียบเทียบทางตรรกศาสตร์ (> = <) ใน CPU → loops

- **Space Complexity** (ความซับซ้อนด้านเนื้อที่)
  วัดจากเนื้อที่ทั้งหมดที่ใช้ในการเก็บข้อมูล ตัวแปรต่างๆ ไว้ใน main memory แต่จะมีความสำคัญน้อยกว่าความซับซ้อนด้านเวลา

Time Complexity (ความซับซ้อนด้านเวลา)



[ภาพโดย อ. จีรพร วีระพันธุ์]

ปัญหา-การจัดเรียงข้อมูล

- Insertion Sort; $f(n) = n^2$
- Merge Sort; $f(n) = n \log n$

e.g. n = 100;

$n^2 = 10000$; $n \log n = 664.39$

คำถาม:

แล้วถ้า $n \rightarrow \infty$

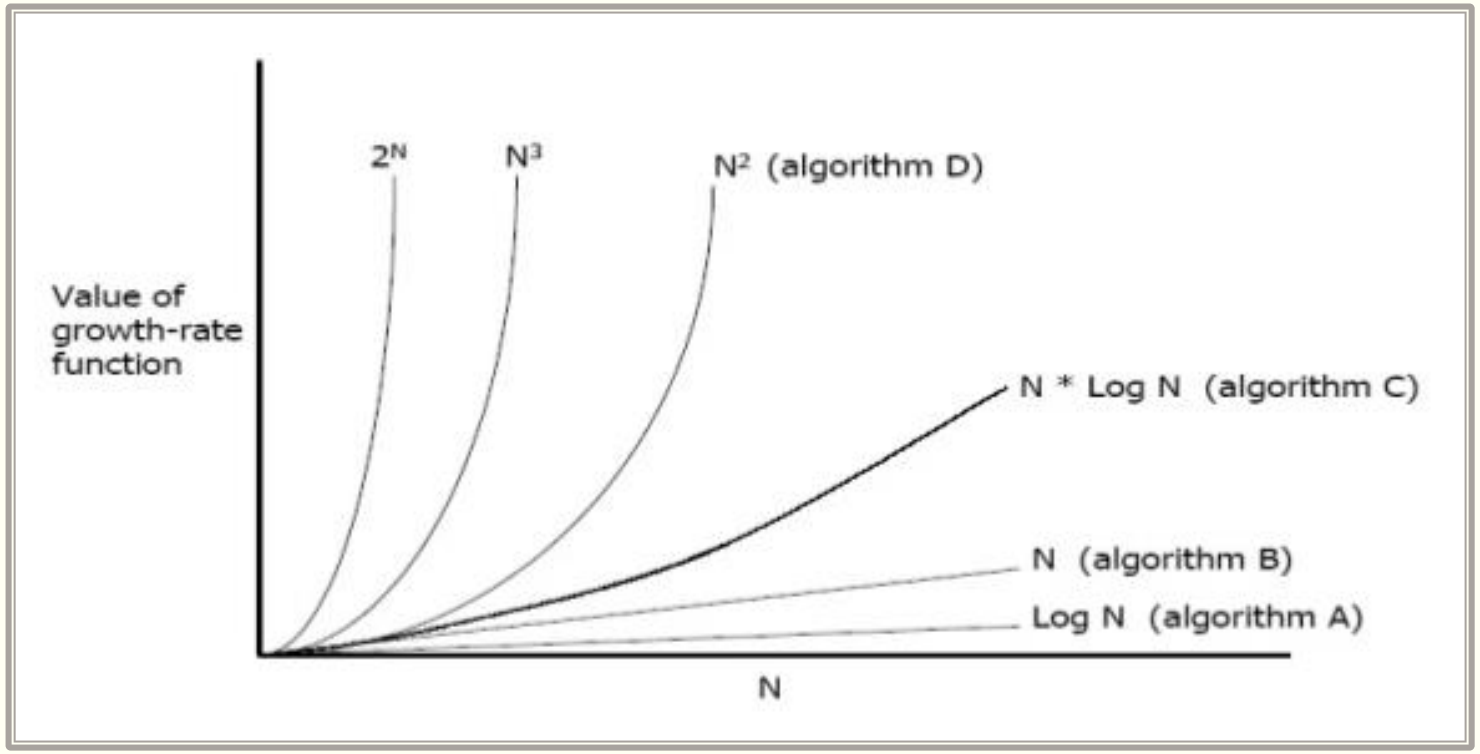
จะนศ. จะเลือกอัลกอริธึ่มใด

1.5 Growth of Functions (แนวโน้มการเพิ่มขึ้นของฟังก์ชั่น)

หลังจากที่เราได้ออกแบบอัลกอริธึ่มไว้หลายอัลกอริธึ่ม หรือเมื่อเราต้องเลือกอัลกอริธึ่มที่มีประสิทธิภาพที่สุดจาก 4 อัลกอริธึ่ม (A-D)

คำถาม เราจะพิจารณาจาก?

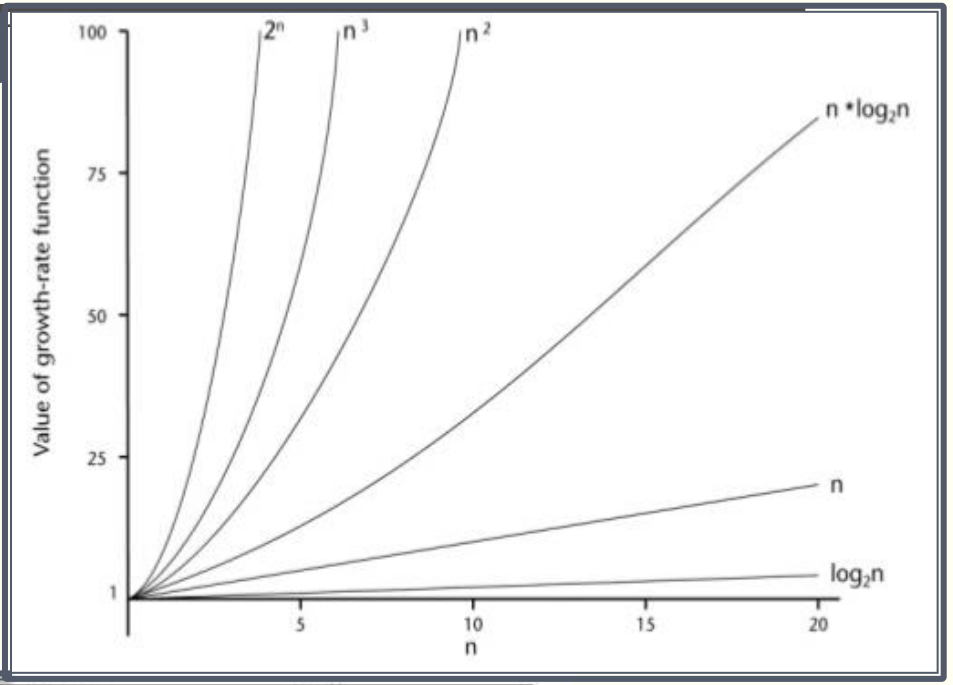
คำตอบ แนวโน้มการเพิ่มขึ้นของฟังก์ชั่น



[ภาพโดย อ. จีรพร วีระพันธุ์]

Functions of Growth Rate (อัตราการเติบโตของฟังก์ชั่น) เรียงจากน้อยไปมาก

Function	Name	Algorithms
c	Constant	Array index lookup
$\log N$	Logarithmic	Binary search
$\log^2 N$	Log-squared	
N	Linear	Graph traversal
$N \log N$	N log N	Merge, Quick sort
N^2	Quadratic	Shortest path
N^3	Cubic	Dynamic programming
2^N	Exponential	Traveling salesman



g(n) / n	log n	n	n log$_2$ n	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	100	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	10^3	10^4	10^6	10^9	10^{300}

[ภาพ: internet 2017]

การวิเคราะห์อัลกอริธึ่ม

โดยปกติประสิทธิภาพของอัลกอริธึ่มจะพิจารณาเป็น 3 cases ได้แก่

- Worst case -เมื่อข้อมูลที่เข้ามาประมวลผลส่งผลให้อัลกอริธึ่มมีประสิทธิภาพต่ำที่สุดและใช้เวลาในการประมวลผลนานที่สุด-อัตราการเติบโตของฟังก์ชั่นสูง
- Average case –เมื่อข้อมูลที่เข้ามาประมวลผลส่งผลให้อัลกอริธึ่มโดยเฉลี่ยที่สามารถระบุ/ประมาณการได้
- Best case -เมื่อข้อมูลที่เข้ามาประมวลผลส่งผลให้อัลกอริธึ่มมีประสิทธิภาพสูงที่สุดและใช้เวลาในการประมวลผลสั้นที่สุด-อัตราการเติบโตของฟังก์ชั่นต่ำ

Asymptotic Notations (สัญลักษณ์อะซิมพ์ทอทิค)

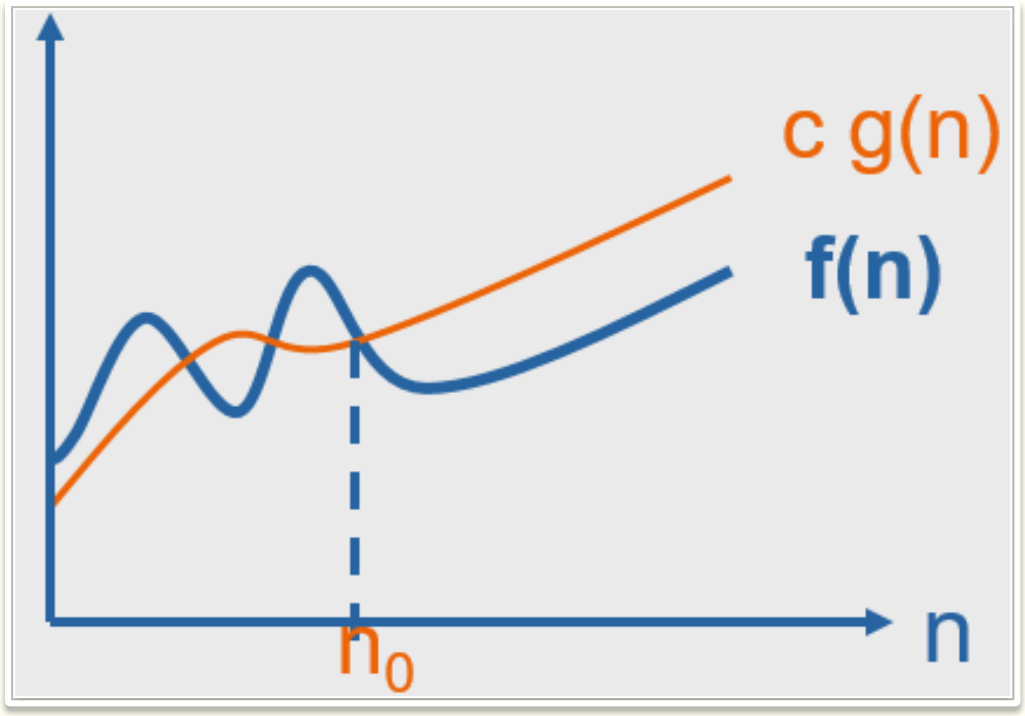
เป็นสัญลักษณ์ที่ใช้นำเสนอความซับซ้อนด้านเวลาของอัลกอริธึ่ม มีหลายแบบสัญลักษณ์ ได้แก่

- **Big-O** notation = Asymptotic upper bound
  ที่บอกขอบเขตบนของฟังก์ชั่น → Worst-case
- **Big-Ω** (Big-Omega) notation = Asymptotic lower bound
  ที่บอกขอบเขตล่างของฟังก์ชั่น → best case
- **Big-θ** (Big-Theta) notation = Asymptotic tight bound
  ที่บอกขอบเขตในช่วงกลางของฟังก์ชั่น → average case
- **Little-o** notation → Asymptotic bound ของฟังก์ชั่น

Big-O Notation

- การใช้สัญลักษณ์ Big-O มีวัตถุประสงค์เพื่ออธิบายขอบเขต(บน) ของฟังก์ชันการเติบโตทางด้านเวลา
- ใช้พิจารณาความซับซ้อนด้านเวลาของฟังก์ชั่น เพื่อเลือกอัลกอริธึ่มที่มีประสิทธิภาพสูงสุด (ไม่ได้มีวัตถุประสงค์เพื่อวัดเวลา)
- โดยจะพิจารณาปริมาณอินพุทข้อมูลมากๆ → Worst-case

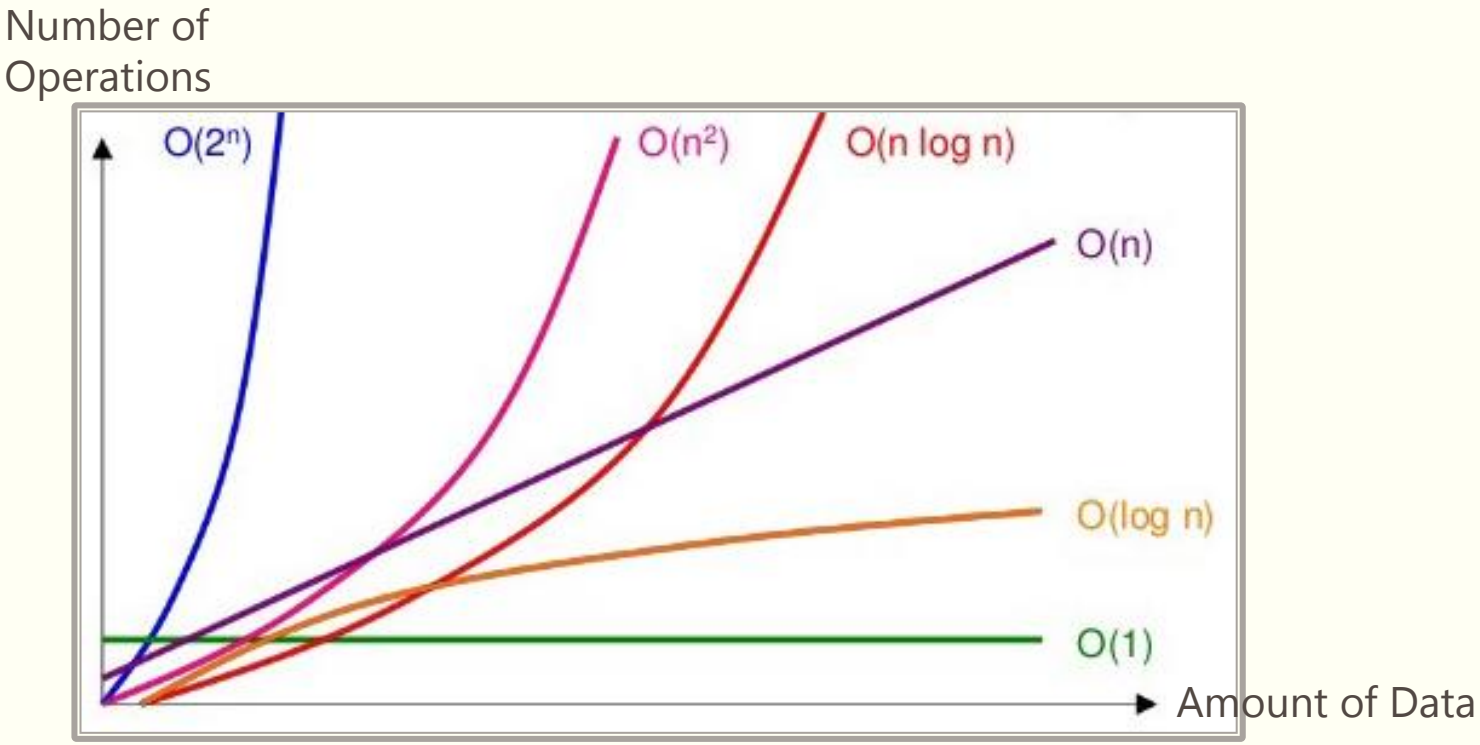
Big-O Notation



f(n) = O(g(n)) ก็ต่อเมื่อ มีค่าคงที่ c และ n_0

ที่ทำให้ f(n) <= c g(n) สำหรับทุกค่า n >= n_0

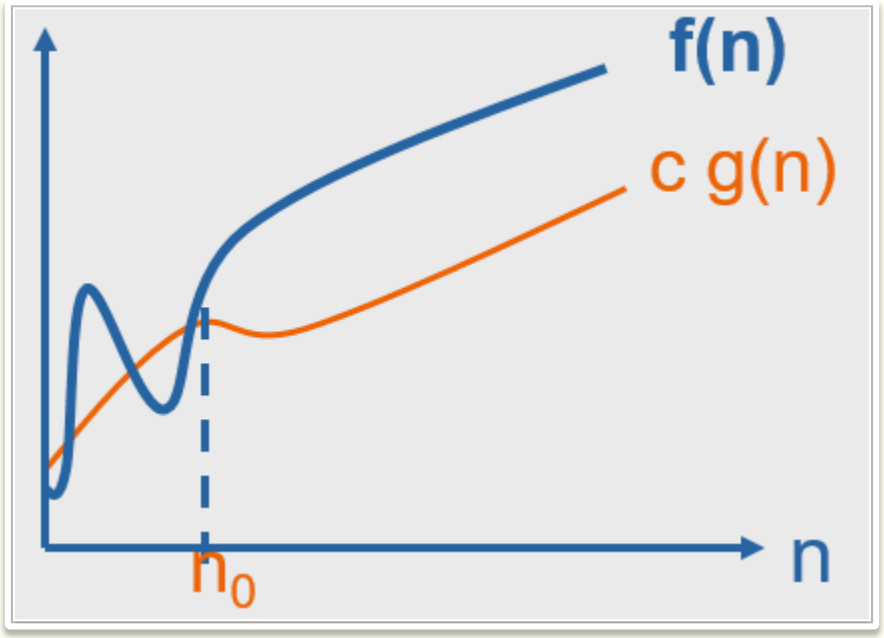
Big-O Notation



เปรียบเทียบ Big-O

[ภาพ: internet 2017]

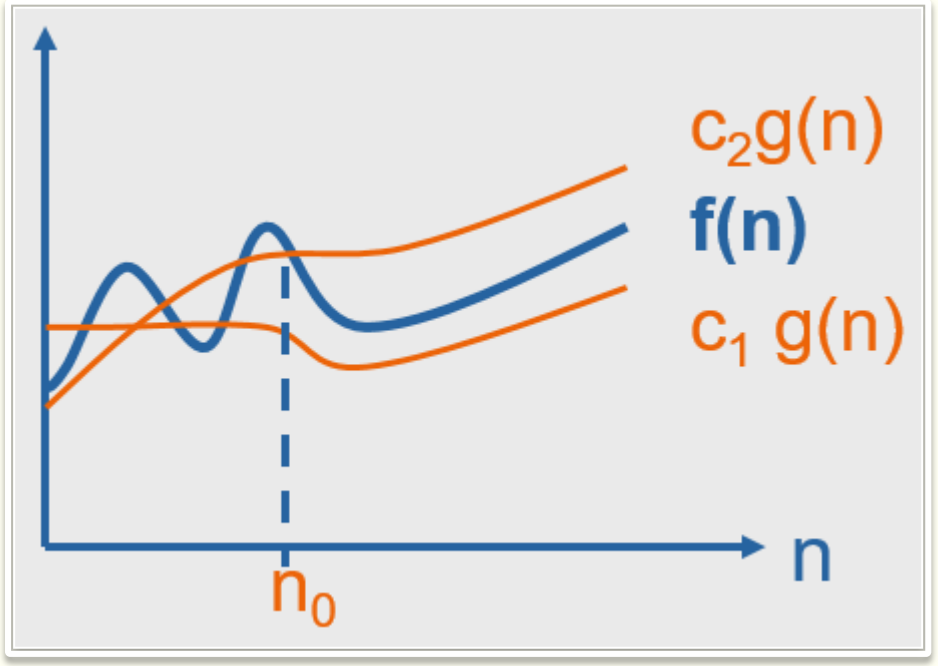
Big-Ω (Big-Omega) Notation



$f(n) = \Omega(g(n))$ ก็ต่อเมื่อ มีค่าคงที่ c และ n_0

ที่ทำให้ $f(n) >= c\ g(n)$ สำหรับทุกค่า $n >= n_0$

Big-θ (Big-Theta) Notation

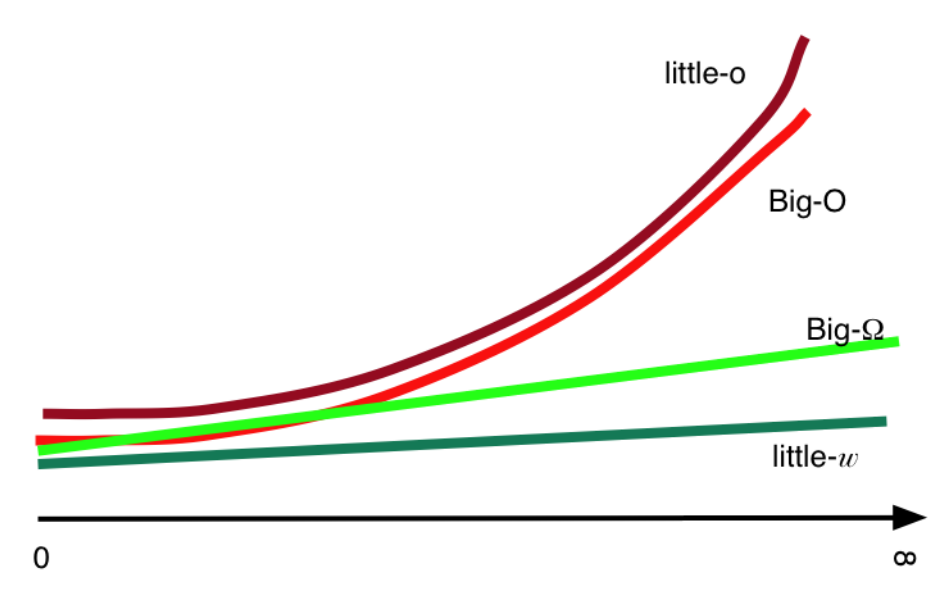


$f(n) = \theta(g(n))$ ก็ต่อเมื่อ มีค่าคงที่ c_1, c_2 และ n_0

ที่ทำให้ c_1 g(n) <= f(n) <= c_2 g(n)

สำหรับทุกค่า n >= n_0

Little-o Notation



$f(n) \sim o(g(n))$

ก็ต่อเมื่อ $\lim_{n \to \infty} f(n)/g(n) \to 1$

จะวัดประสิทธิภาพอัลกอริธึ่มได้ดีกว่า Big-O