Review

$$X^a X^b = X^{a+b}$$

$$X^a / X^b = X^{a-b}$$

$$(X^a)^b = X^{ab}$$

definition $\log_x B = A$ iff $x^A = B$

$$\log (A^B) = B \log A$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^{n} a^i = (a^{n+1} - 1) / (a - 1) \quad a > 1$$

$$\leq 1 / (1 - a) \quad 0 < a < 1, n \to \infty$$

$$= 1 / (1 - a) \quad n = \infty$$

proof

$$S = 1 + a + a^2 + a^3 + \ldots$$

$$aS = a + a^2 + a^3 + a^4 + \ldots$$

$$S - aS = 1$$

$$S (1 - a) = 1$$

$$S = 1 / (1 - a)$$

$$\sum_{i=1}^{\infty} i / 2^i = 2$$

proof

$$S = 1/2 + 2/2^2 + 3/2^3 + 4/2^4 + 5/2^5 + \dots$$

$$2S = 1 + 2/2 + 3/2^2 + 4/2^3 + 5/2^4 + \dots$$

$$2S - S = 1 + 1/2 + 1/2^2 + 1/2^3 + 1/2^4 + \dots$$

$$S = 2$$

$$\sum_{i=1}^{n} i = n(n+1)/2 \approx n^2/2$$

$$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6 \approx n^3/3$$

Modular arithmetic

$A \equiv B \pmod{n}$ if n divides $A - B$

$81 \equiv 61 \equiv 1 \pmod{10}$

$A + C \equiv B + C \pmod{n}$

$AD \equiv BD \pmod{n}$

An example of proof by induction

$F_i < (5/3)^i$ for $F_0 = 1,\ F_1 = 1$

basic step: $F_1 < 5/3$

inductive step: $F_n < (5/3)^n$

from Fibonacci $F_{k+1} = F_k + F_{k-1}$

$$F_{k+1} < (5/3)^k + (5/3)^{k-1}$$

$$< (3/5)(5/3)(5/3)^k + (3/5)^2(5/3)^2(5/3)^{k-1}$$

$$< (3/5)(5/3)^{k+1} + (9/25)(5/3)^{k+1} = (3/5+9/25)(5/3)^{k+1}$$

$$= (24/25)(5/3)^{k+1}$$

$$< (5/3)^{k+1}$$

proof by counterexample

$F_k \leq k^2$ is false

we can find $F_{11} = 144 > 11^2$

proof by contradiction

theorem: there is an infinite number of primes

สมมุติจำนวนจำเพาะมีจำกัด (สิ้นสุด หรือไม่ infinite) กำหนดให้จำนวนจำเพาะตัวสุดท้าย (ที่ใหญ่ที่สุด) คือ P_k เมื่อนำจำนวนจำเพาะทั้งหมด ($P_1, P_2, \ldots, P_k$) มาคูณกันแล้วบวกหนึ่ง จะได้

$$n = P_1P_2 \ldots P_k + 1$$

แสดงว่า

$$n > P_k$$

แต่โดยสมมุติฐานข้างต้น n ไม่เป็นจำนวนจำเพาะ เพราะ P_k เป็นจำนวนจำเพาะตัวสุดท้าย แต่ปรากฏว่าไม่มีจำนวนจำเพาะใด ($P_1, P_2, \ldots, P_k$) ที่หาร n ลงตัว (จะมีเศษ 1 เสมอ) ดังนั้นข้อสมมุติที่ว่า P_k เป็นจำนวนจำเพาะตัวสุดท้าย (ใหญ่ที่สุด) นั้นไม่จริง เพราะ n มากกว่า P_k ซึ่งหมายความว่า ทฤษฎีเป็นจริง

Algorithm analysis

O(n):	T(n)	=	O(f(n)),	c, n_0 such that $T(n) \leq cf(n)$, $n \geq n_0$
				น้อยกว่าหรือเท่ากับ f(n)
Ω(n):	T(n)	=	Ω (g(n)),	c, n_0 such that $T(n) \geq cg(n)$, $n \geq n_0$
				มากกว่าหรือเท่ากับ g(n)
Θ(n):	T(n)	=	O(h(n))	
			Ω (h(n))	เท่ากับ h(n)
o(n):	T(n)	=	o(p(n)),	T(n) = O(p(n))
				$\neq \Theta$ (p(n))

rules:

1. $T_1(n) = O(f(n))$ $T_2(n) = O(g(n))$
   a) $T_1(n) + T_2(n) = \max(O(f(n)), O(g(n)))$
   b) $T_1(n) * T_2(n) = O(f(n) * g(n))$

2. if T(n) is a polynomial of degree k,
   $T(n) = \Theta (n^k)$

3. $\log^k n = O(n)$ for any constant k, logarithms grow very slowly.

4. $\lim_{n \to \infty} f(n) / g(n)$ by L'Hôpital's rule

- limit is 0: f(n) = o(g(n))
- limit is c: f(n) = Θ (g(n))
- limit is ∞: f(n) = o(f(n))
- limit oscillates: there is no relation

growth rates: c, log n, $\log^2$ n, n, nlog n, n^2, n^3, 2^n

Running time calculations

จากตัวอย่างหน้า 21

```
            int
            sum(int N)
            {
                  int   i, partialsum;

/* 1 */             partialsum = 0;
/* 2 */             for (i = 1; i <= N; i++)
/* 3 */                   partialsum += i * i * i;
/* 4 */             return partialsum;
            }
```

/* 1 */ ใช้เวลา 1

/* 4 */ ใช้เวลา 1

/* 3 */ ใช้เวลา 4n

/* 2 */ ใช้เวลา (1 + n+1 + n)

รวมเป็น 1 + 1 + 4n + (1 + n+1 + n) = 6n + 4 = O(n)

rule 1: เวลามากที่สุดที่ใช้ในการกระทำคำสั่งภายในลูป (รวมทั้งทดสอบ) คูณกับจำนวนครั้งของการทำซ้ำ (iteration)

rule 2: nested loops ให้ทำจากลูปในออกไปหาลูปนอก

rule 3: คำสั่งที่ต่อเนื่องกัน ให้นำมาบวกกัน

rule 4: คำสั่ง if-else ให้คิดตอนทดสอบ บวกกับ S1 หรือ S2 สุดแต่คำสั่งใดจะใช้เวลานานกว่ากัน

ตัวอย่างที่ไม่ดี

```
long  int
Fib(int  n)
{
      if (n <= 1)
            return 1;
      else
            return Fib(n-1) + Fib(n-2);
}
```

T(n) = T(n-1) + T(n-2) + 2 (มาจาก if บวกกับ +)

ถ้า n > 4

Fib(n) $\geq (3/2)^n$ ซึ่งเป็นการเพิ่มแบบ exponential

จะเห็นว่า recursion เป็นการออกแบบ algorithm ที่ทำให้โปรแกรมมีขนาดสั้น (เหมาะกับปัญหาบางประเภทเท่านั้น) แต่ใช้เวลาในการคำนวณนาน เพราะเป็นการเรียกซ้ำๆ กันในลักษณะของ recurrence ที่ลดจำนวนของข้อมูลจาก n ลงทีละหนึ่งเป็น n-1, n-2 จนถึง 1 ตามลำดับ หากมีวิธีอื่น เช่น iteration ก็ควรเลี่ยงไปใช้วิธีนี้ ซึ่งทำงานได้เร็วกว่า recursion มาก แม้ว่าขนาดของโปรแกรมอาจจะยาวกว่า (และในบางกรณีก็เข้าใจยากกว่าหรือเขียนยากกว่า) ดังตัวอย่างที่เป็นที่รู้จักกันดีข้างล่าง

```
int
recursion(int n)
{
      return n <= 1 ? 1 : n * recursion(n-1);
}
```

```
int
iteration(int n)
{
      int   i, fact = 1;

      for (i = 1; i <= n; i++)
            fact *= i;
      return fact;
}
```

(ดูตัวอย่างเพิ่มเติมในหน้า 27 และคำอธิบายในหน้า 28)

สรุป โดยทั่วไป การวิเคราะห์จะพิจารณา 2 กรณี คือ

Average case การวิเคราะห์จะยุ่งยากซับซ้อน

Worst case การวิเคราะห์จะง่ายกว่า แต่จะส่งผลให้เป็นการประมาณที่เกินความเป็นจริง

Note: เกร็ดความรู้ของการหาค่าสูงสุดจาก arrays

ตัวอย่าง รับค่า array ขนาด n เริ่มจากกำหนดให้ตัวแรกใน array มีค่ามากที่สุด โดยนำไปเก็บในตัวแปรชื่อ maxv (assign ให้กับ maxv) จากนั้น ใช้เทคนิคของลูป โดยเริ่มต้นจากตัวที่ 2 (subscript i = 1) จนถึง

ตัวสุดท้าย (i = n-1) หากปรากฏว่าตัวใดมีค่ามากกว่า maxv (ในคำสั่ง if) ก็ให้เก็บค่าของตัวนั้นไว้ใน maxv แทนที่ค่าเดิม ทำเช่นนี้จนครบทุกค่าใน arrays แล้วส่งค่า maxv กลับคืนสู่ที่มา (ฟังชันข้างบนที่เรียก func1)

```
int
func1(int value[N], int n)
{
      int   maxv;
      int   i;

      maxv   = value[0];
      for (i = 1; i < n; i++)
            if (value[i] > maxv)
                  maxv = value[i];
      return maxv;
}
```