Design and Analysis of Data Structures and Algorithms :: Algorithm Design and Analysis

Warin Wattanapornprom PhD.

Algorithm and Complexity

PART 1:
ALGORITHM DESIGN

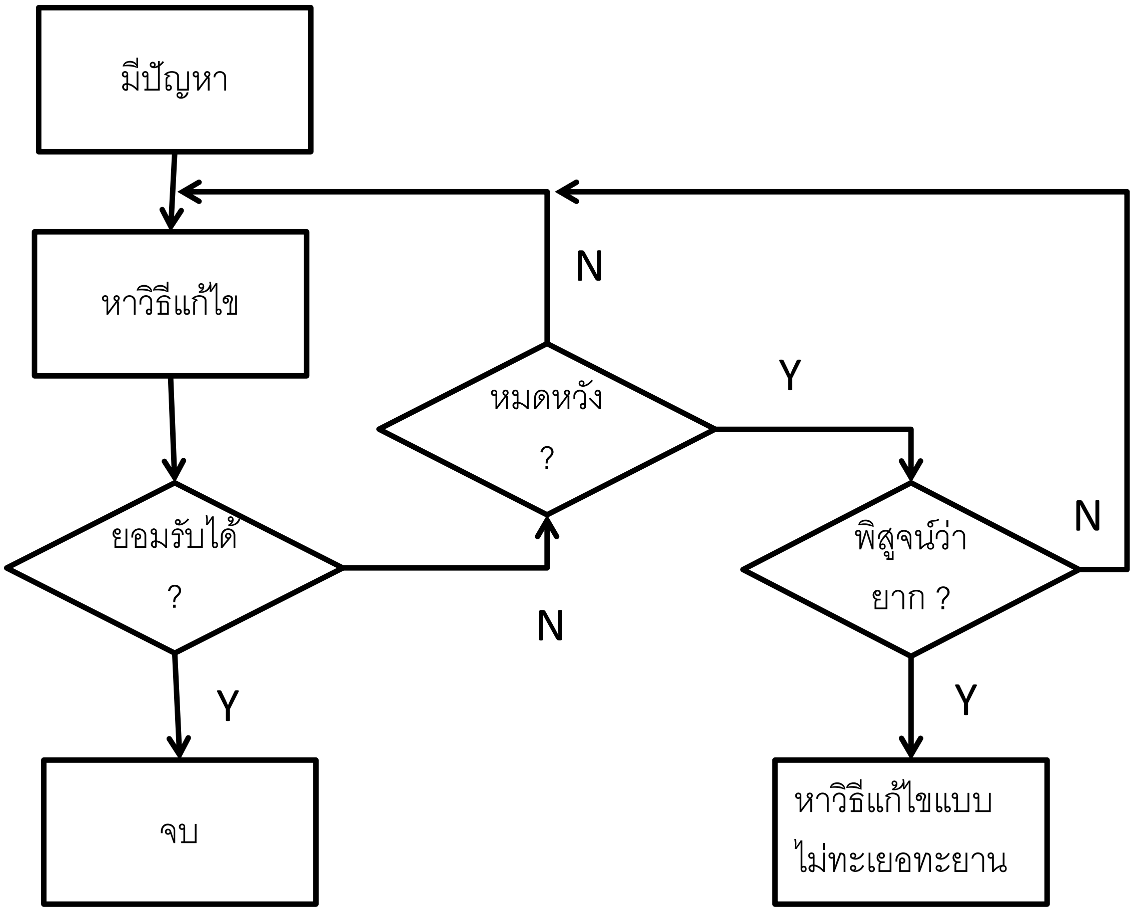
Agenda

- Introduction
- Computational problems
- Algorithms
- Algorithm Design Techniques
- Analysis of Algorithms
- Examples

Main Ideas

- Algorithm Design
- Algorithm Analysis
- Computation Complexity

Design and Analysis Steps



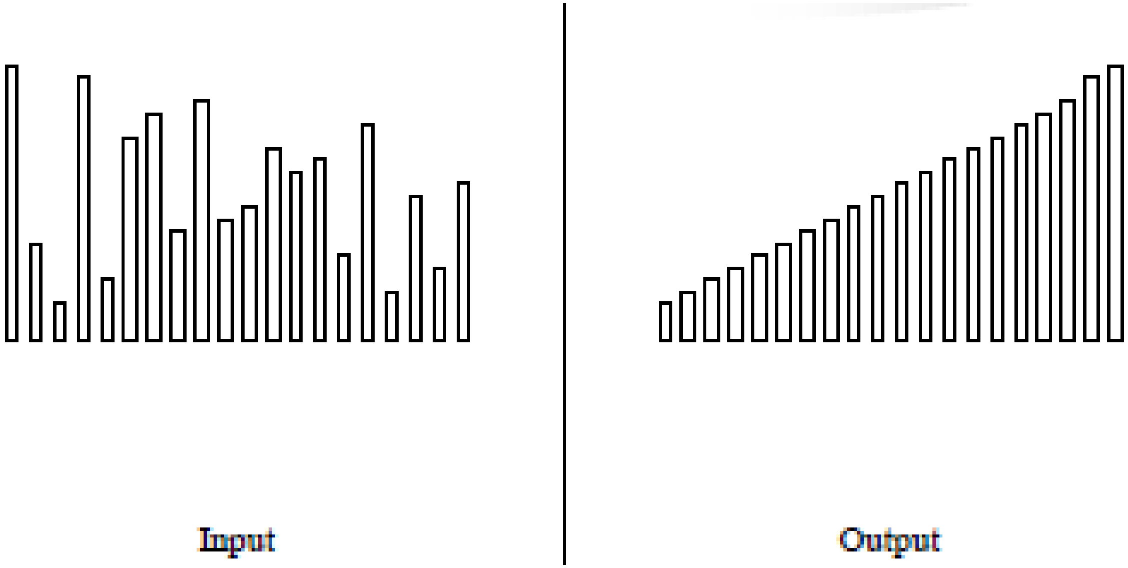
Computational Problems

- A computational (or algorithmic) problem is specified by a precise definition of
  - the legal inputs
  - the required outputs as a function of those inputs

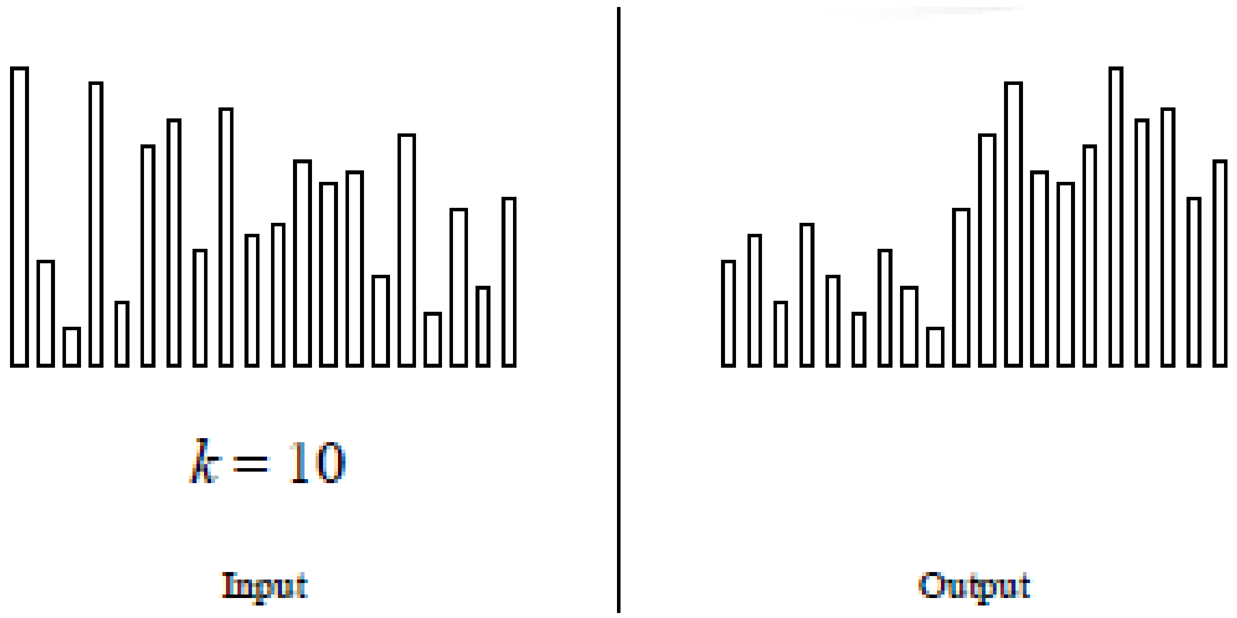
Sorting Problem

- Input : a sequence of n numbers
  < $a_1, a_2, \ldots, a_n$ >
- Output : a reordering
  < $a'_1, a'_2, \ldots, a'_n$ > of the input
  such that $a'_1 \leq a'_2 \leq \ldots \leq a'_n$
- Sorting problem instances :
  < 31, 41, 59, 26, 41 >
  <1, 2, 5, 7, 6, 9, 2>
  <2, 2, 2>

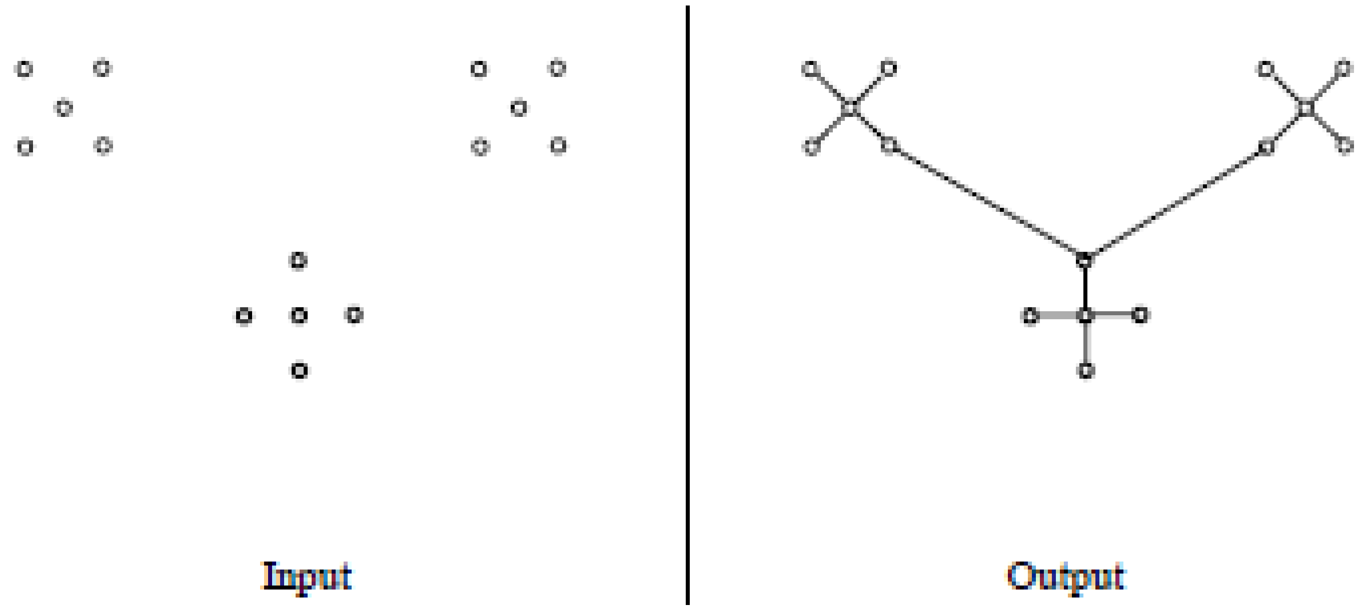
Sorting



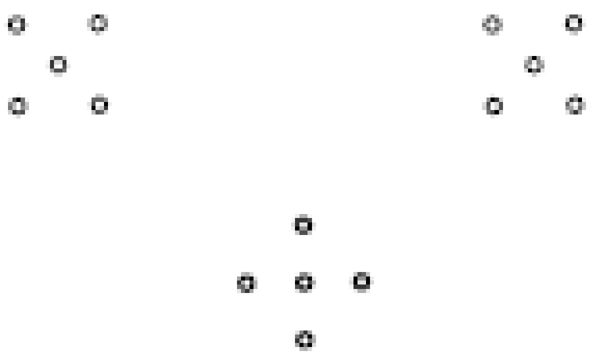
Median and Selection



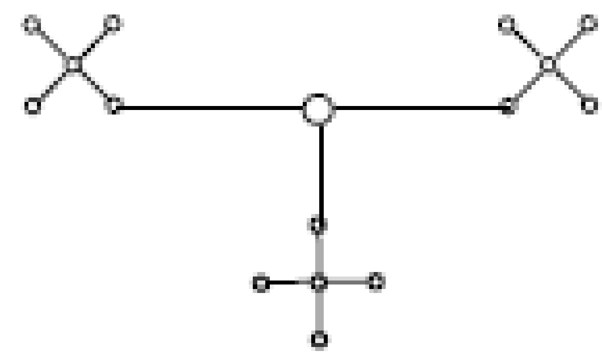
Minimum Spanning Tree



Steiner Tree



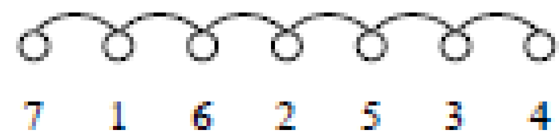
Input



Output

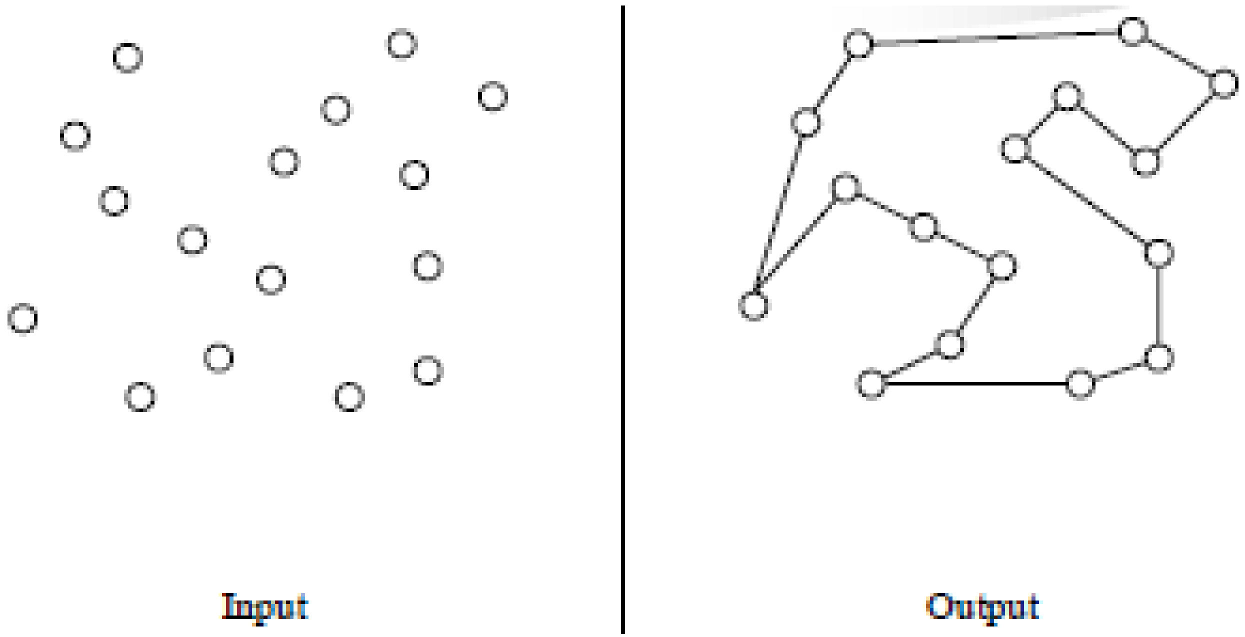
Bandwidth Minimization

Input

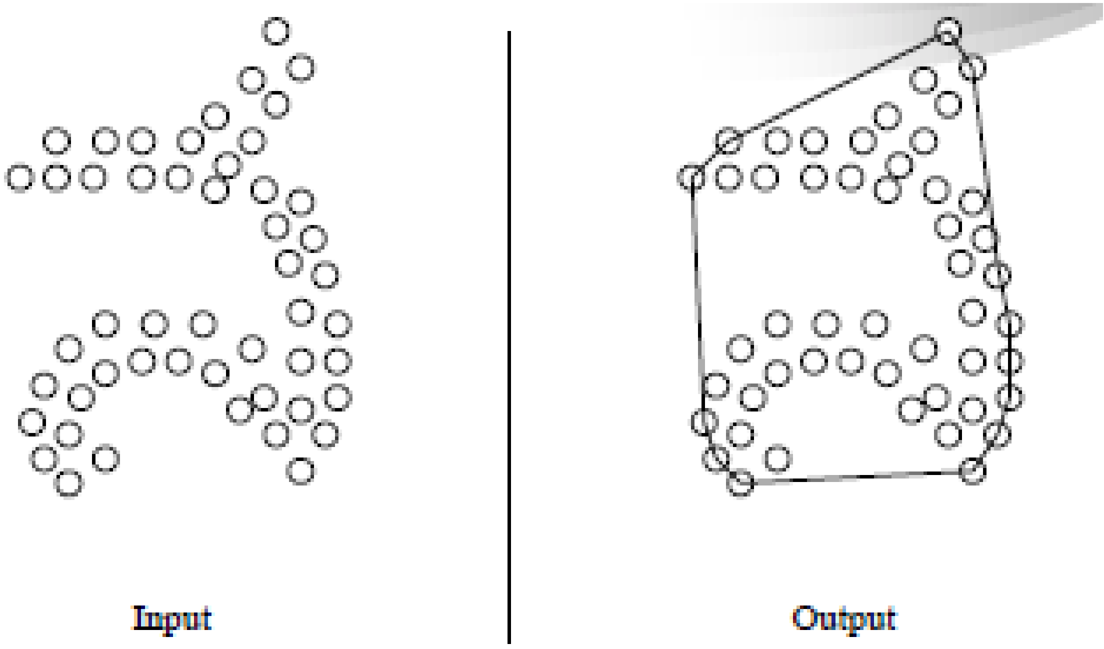


Output

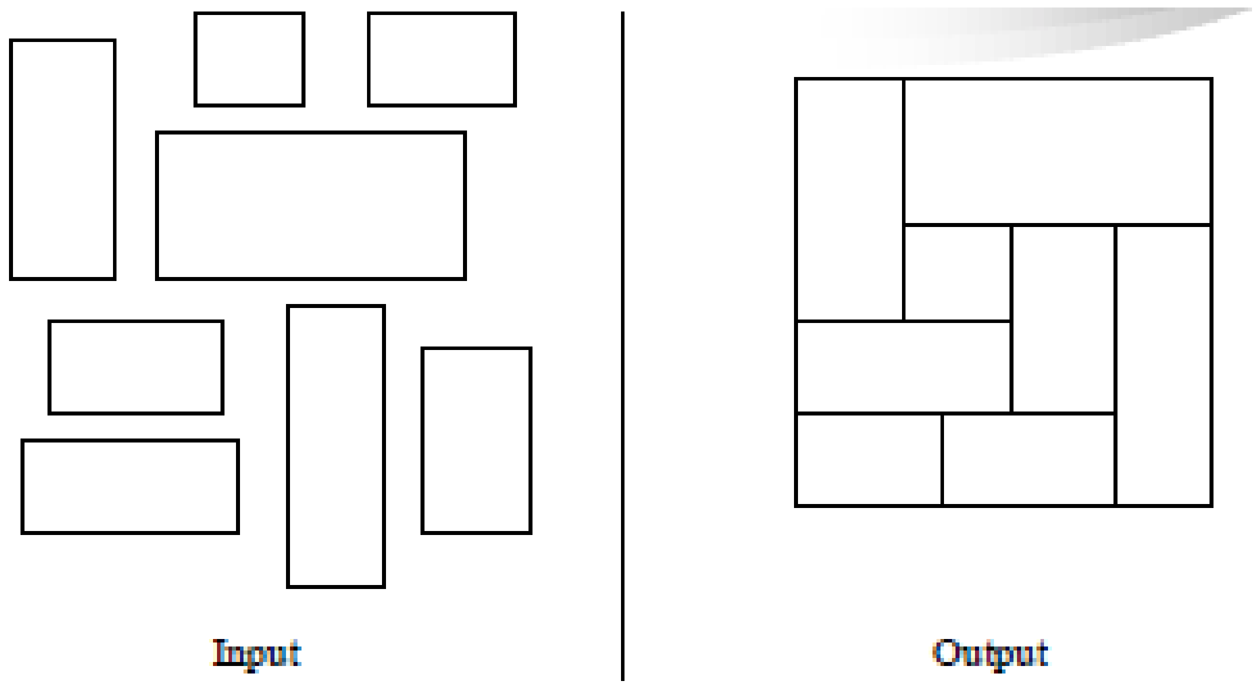
Traveling Salesman Problem



Convex Hull



Bin Packing



Primality Testing

Input	Output
833816926455584605284210207 1	NO

Factoring

Input	Output
8338169264555846052842102071	179424673
	x 2038074743
	x 22801763489
	= 8338169264555846052842102071

String Matching

You are the fairest of your sex,
Let me be your hero;
I love you as one over *x*,
As *x* approaches zero.
Positively.

you

Input

You are the fairest of your sex,
Let me be your hero;
I love you as one over *x*,
As *x* approaches zero.
Positively.

Output

Approximate String Matching

You are the fairest of your sex,
Let me be your hero;
I love you as one over *x*,
As *x* approaches zero.
Positively.

heero

Input

You are the fairest of your sex,
Let me be your hero;
I love you as one over *x*,
As *x* approaches zero.
Positively.

Output

Data Compression

You are the fairest of your sex,
Let me be your hero;
I love you as one over *x*,
As *x* approaches zero.
Positively.

Input

You are the fairest of your sex,
Let me be your hero;
I love you as one over *x*,
As *x* approaches zero.
Positively.

Output

Cryptography

You are the fairest of your sex,
Let me be your hero;
I love you as one over x,
As x approaches zero.
Positively.

Input

)*#$(*KJSNpsld09LKDF0osk
POS)s8sd,??<CNZisd(&sD(6%
(^%#&(dls28s8&AK)8dksF_8
OSD7slx.zz846(&%){l'ps
ska@

Output

Satisfiability

$(x+y)(x+\overline{y})\overline{x}$	NO
Input	Output

Halting

```
while x ≠ 1 do
  if x is even
    then x = x/2
    else x = 3x+1
```

x = 7

Input

?

Output

Algorithms

- A sequence of computational steps that transform any legal input into the desired output
- A tool for solving a well-specified computational problem
- Idea behinds a computer program

Sequential Search

```
SeqSearch( D[1..n], x )
{
      i = 1
      while ( i <= n && D[i] != x )
            i = i + 1
      if ( i > n ) return 0
      return i
}
```

Search forward

Sequential Search

```
SeqSearch( D[1..n], x )
{
     i = n
     while ( i >= 1 && D[i] != x )
          i = i - 1
     if ( i < 1 ) return 0
     return i
}
```

Search backward

Sequential Search

```
SeqSearch( D[1..n], x )
{
     i = n; D[0] = x
     while ( D[i] != x )
          i = i - 1
     return i
}
```

Search backward with sentinel

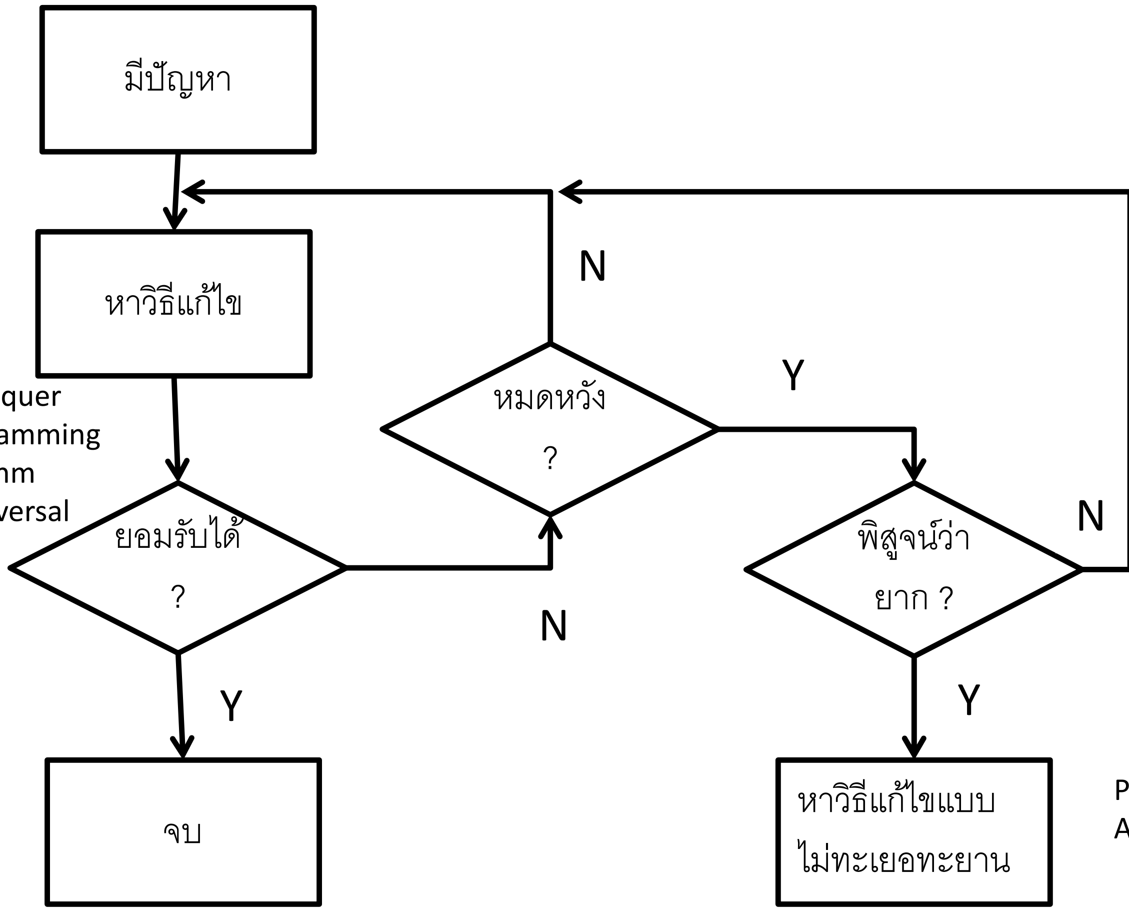
Algorithms

- An algorithms is correct if, for every input instance, it halts with the correct output.
- We seek algorithms which are
  - correct
  - efficient

Algorithm Design Techniques

- Brute Force
- Incremental
- Divide and Conquer
- Dynamic Programming
- Greedy Algorithm
- Search and Traversal
- Probabilistic Algorithm
- Approximation Algorithm

Design and Analysis Steps



Brute Force
Incremental
Divide and Conquer
Dynamic Programming
Greedy Algorithm
Search and Traversal

Probabilistic
Approximation

Abu Abd-Allah ibn Musa al'Khwarizmi

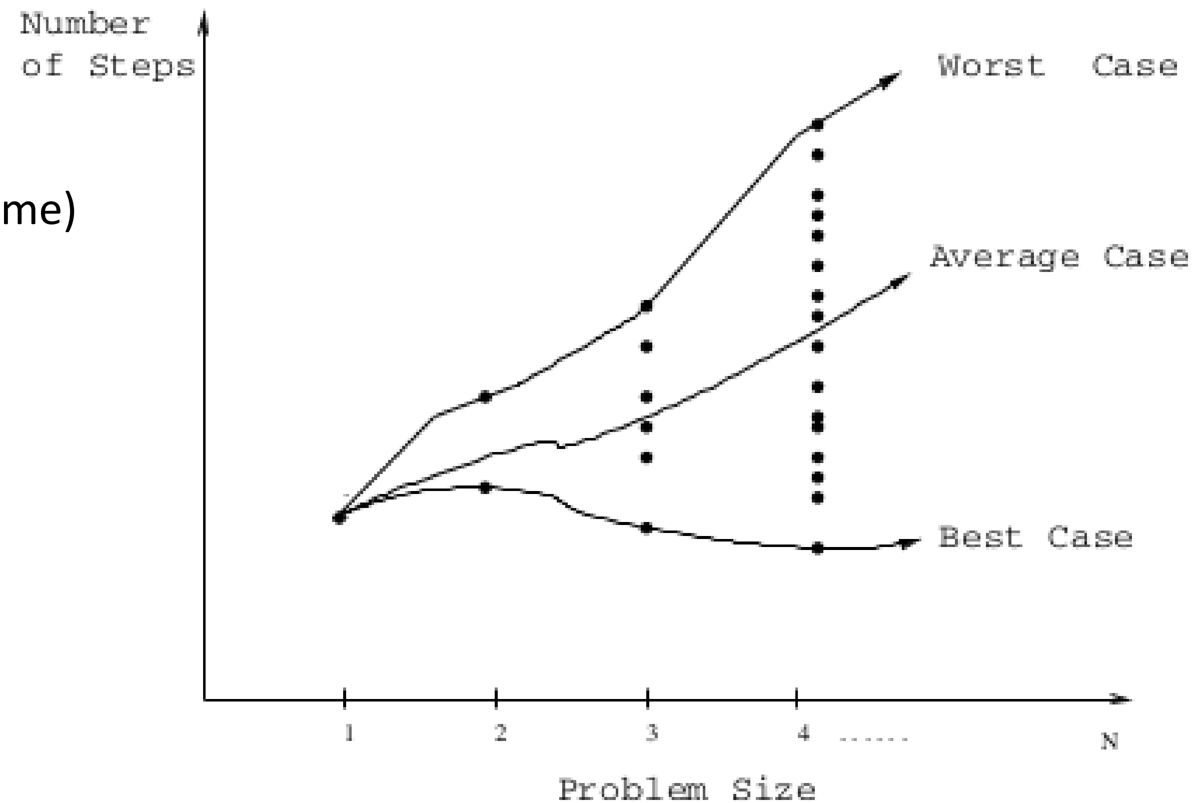
Al'Khwarizmi (lived from 790 to840) wrote on Hindu-Arabic numerals who wrote **Kitāb al-jabr wa’l-muqābala**, which evolved into today’s algebra text and was the first to use zero as a place holder in positional base notation. The word algorithm derives from his name.

Analysis of Algorithms

- วิเคราะห์ประสิทธิภาพของอัลกอริทึม
  - เวลาการทำงาน
  - จำนวน memory ที่ใช้ในการทำงาน
- Worst case analysis
- Average case analysis
- Amortized analysis

Algorithm Complexity

(Time)



(no. of instance)

Sorting Algorithms

- Bubble sort $t(n) \propto n^2$
- Insertion sort $t(n) \propto n^2$
- Shell sort $t(n) \propto n^{1.\mathrm{xx}}$
- Heap sort $t(n) \propto n \log n$
- Merge sort $t(n) \propto n \log n$
- Quick sort $t(n) \propto n \log n$ (average case)

Computational Complexity

- Problem reduction

$$\begin{bmatrix} 0 & A \\ B & 0 \end{bmatrix}^2 = \begin{bmatrix} AB & 0 \\ 0 & BA \end{bmatrix}$$

การยกกำลังสองเมตริกซ์ไม่ง่ายกว่าการคูณเมตริกซ์

A Small Problem

- Input : a number n, k
- Output : (the n^{th} Fibonacci number) $\mathrm{mod}\ k$
- Input instance : $10, 21$
- Output : $f_{10} \bmod 21 = 55 \bmod 21 = 13$

Fibonacci Number

n:	0	1	2	3	4	5	6	7
fn:	0	1	1	2	3	5	8	13

$$f_n = f_{n-1} + f_{n-2} \qquad \text{for } n > 1$$
$$f_0 = 0, f_1 = 1$$

$$f_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

Fib 1

```
Fib1( n, k )
{
      t1 = (1 + sqrt(5))/2
      t2 = (1 - sqrt(5))/2
      f = int( (t1^n + t2^n)/sqrt(5) )
      return f mod k
}
```

$$f_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n + \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

For a given prime number k, which Fibonacci numbers are divisible by k?

Fib 2

```
Fib2( n, k )
{
  if ( n < 2 ) return n mod k
  return (  Fib2(n-1,k) +
            Fib2(n-2,k)    ) mod k
}
```

$$f_n = f_{n-1} + f_{n-2} \qquad \text{for } n > 1$$
$$f_0 = 0,\ f_1 = 1$$

Fib 2

```
Fib2( n, k )
{
  if ( n < 2 ) return n mod k
  return (  Fib2(n-1,k) +
            Fib2(n-2,k)    ) mod k
}
```

$t(n) = t(n\text{-}1) + t(n\text{-}2) + 1$

$t_0 = t_1 = 1$

$t(n) \propto 1.618^n$

Moore’s Law

- Moore’s Law :อีก 75 ปีจะมี Pentium XXX 10^{14}GHz. (เร็วกว่าเครื่องที่ทดลอง1017 เท่า)
- ใช้เวลา 109 ปี เพื่อหาค่า $Fn \bmod k$ เมื่อ $n = 181$ $(\ t(n) \propto \varphi n \therefore \log_{1.618} 10^{17} \approx 81)$
- รอ Pentium XXXX 10^{1000} GHz !!!!!

Fib 3

```
Fib3( n, k )
 {
   fn2 = 0
   fn1 = 1
   for i = 2 to n
     fn  = ( fn1 + fn2 ) mod k
     fn2 = fn1
     fn1 = fn
   return fn
 }
```

$t(n) \propto n$

Conclusion

- รู้จักปัญหา
- รู้จักวิธีออกแบบ
- รู้จักวิธีวิเคราะห์

"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing."

The Joy of Algorithms
Computing in Science and Engineering
- Francis E. Sullivan

Algorithm and Complexity

PART 2: ALGORITHM ANALYSIS

Agenda

- Complexity Measures
- Size of Problem Instance
- Worst-, Best-, and Average-Case Analysis
- Exact Analysis
- Analysis Simplification

Complexity Measures

- ประสิทธิภาพของอัลกอริทึมวัดจาก
  - เวลาการทำงาน
  - จำนวนหน่วยความจำที่ใช้

Analysis of Algorithms

- **predict the behavior** of an algorithm without implementing it on a specific computer
  - impossible to predict “exact” behavior
  - analysis = “approximate” the main characteristics use for comparison

Instances and Sizes

- การทำงานของอัลกอริทึมขึ้นกับ
  - ขนาดของ problem instance
  - ลักษณะของ problem instance
- ขนาดของตัวอย่างปัญหา คือจำนวนข้อมูลที่ต้องใช้เพื่อเข้ารหัสตัวอย่างปัญหา
  - จำนวน bits, bytes, words
  - จำนวน vertices, edges, triangles, literals,....

Sorting

- Input : a list of n numbers
- Assumption : each number can be stored in a single computer word.
- Input size : n

Minimum Spanning Tree

- Input : a weighted graph $G=(V, E)$
- Assumption : edge weights are real numbers each can be stored in a computer word
- Input size : $|V|, |E|$

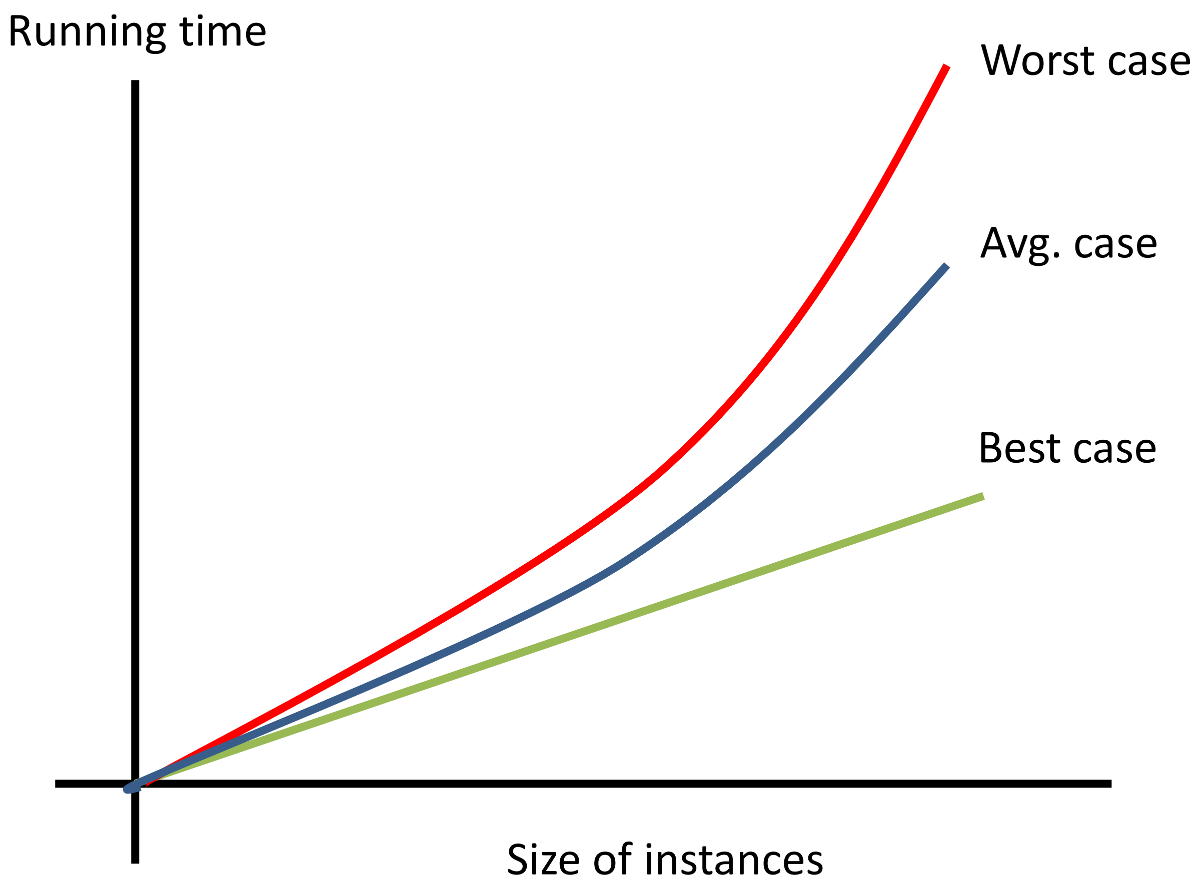
Primality Testing

- Input : a positive integer i
- Input size : $\log n$
- e.g., $1{,}000{,}000$ -> $1 + \log_2 10^6 = 20$ bits

Primality Testing

- Input : an positive integer i, $i < 2^{237}$
- Input size : constant

Cost vs. Instance Size



Average-Case Analysis

- Expected cost
- Depend on frequencies of instances of size n occur in practice

$$t_{avg}(n) = \sum_{I \in I_n} p(i) \cdot t(i)$$

Insertion Sort : Analysis

Insertion_Sort(A[1..n])	
`for j=2 to n`	$c_1 n$
`key = A[j]`	$c_2 (n-1)$
`i= j-1`	$c_3 (n-1)$
`while i>0 and A[i]>key`	$c_4 \sum_{j=2}^{n} (t_j)$
`A[i+1] = A[i]`	$c_5 \sum_{j=2}^{n} (t_j - 1)$
`i = i+1`	$c_6 \sum_{j=2}^{n} (t_j - 1)$
`A[i+1]=key`	$c_7 (n-1)$

Insertion Sort : Best-Case

$t(n)$
$= c_1\, n + c_2\,(n-1) + c_3\,(n-1) + c_4 \sum_{j=2}^{n}(t_j) + c_5 \sum_{j=2}^{n}(t_j - 1)$
$+ c_6 \sum_{j=2}^{n}(t_j - 1) + c_7\,(n-1)$

Best-case: t_j=1

$t_{best}(n)$
$= c_1\, n + c_2\,(n-1) + c_3\,(n-1) + c_4 \sum_{j=2}^{n}(1) + c_5 \sum_{j=2}^{n}(1)$
$+ c_6 \sum_{j=2}^{n}(1) + c_7\,(n-1)$
$= (c_1 + c_2 + c_3 + c_4 + c_7)n_1 + (c_2 + c_3 + c_4 + c_7)$
$= linear\ function$

Insertion Sort : Worst-Case

$t(n)$
$= c_1\, n + c_2\,(n-1) + c_3\,(n-1) + c_4 \sum_{j=2}^{n}(t_j) + c_5 \sum_{j=2}^{n}(t_j - 1)$
$+ c_6 \sum_{j=2}^{n}(t_j - 1) + c_7\,(n-1)$

Worst-case: $t_j = j$

$t_{worst}(n)$
$= c_1\, n + c_2\,(n-1) + c_3\,(n-1) + c_4 \sum_{j=2}^{n}(j) + c_5 \sum_{j=2}^{n}(j - 1)$
$+ c_6 \sum_{j=2}^{n}(j - 1) + c_7\,(n-1)$
$= \left(\frac{c_4+c_5+c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4+c_5+c_6}{2} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)$
$= quadratic\ function$

Insertion Sort : Average-Case

$t(n)$
$= c_1\, n + c_2\,(n-1) + c_3\,(n-1) + c_4 \sum_{j=2}^{n} (t_j) + c_5 \sum_{j=2}^{n} (t_j - 1)$
$+ c_6 \sum_{j=2}^{n} (t_j - 1) + c_7\,(n-1)$

Worst-case: $t_j = j/2$

$t_{worst}(n)$
$= c_1\, n + c_2\,(n-1) + c_3\,(n-1) + c_4 \sum_{j=2}^{n} (j/2) + c_5 \sum_{j=2}^{n} (j/2 - 1)$
$+ c_6 \sum_{j=2}^{n} (j/2 - 1) + c_7\,(n-1)$
$= \left(\frac{c_4+c_5+c_6}{4}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4+c_5+c_6}{4} + c_7\right) n - (c_2 + c_3 + c_4 + c_7)$
$= quadratic\ function$

Best, Average, Worst

- Best-case is usually ruled out
- Average-case is good,
  - but hard to measure effectively
  - not clear what an average input is
- Worst-case is usually fairly easy to analyze and often close to the average

Analysis : Simplification

- Time $\propto$ the number of “elementary” instructions get executed

Time vs. # Instructions

- จำนวนของ instruction ขึ้นอยู่กับ
  - สถาปัตยกรรม SISD vs. SIMD หรือ CISC vs. RISC
  - ความสามารถของ compiler
- ถ้าวัดได้ก็ดี ในมุมมองของ Computer Engineer
- ในมุมมองของ Software Engineer มองการคำนวณเป็น Layer
- ในมุมมองของนักคณิตศาสตร์มอง step ของอัลกอริทึม

Elementary Operation

- Operation whose execution time can be bounded above by a constant
- Examples
  - 128-bit multiplication
  - n-bit multiplication

Selection Sort

```
SelectionSort( A[1..n], n )
{
      for j = n downto 2
      {
            k = MaxIndex( A[1..j], j )
            Swap( A, k, j )
      }
}
```

Wilson’s Algorithm

```
Wilson( n )
     m = (n-1)!  + 1
     if m mod n = 0 then return TRUE
                         else return FALSE
```

Analysis : More Simplification

- To simplify running-time analysis
  - count only Barometer instructions
  - use asymptotic analysis
- Sufficient for obtaining growth rate of running time

Barometer

```
Insertion_Sort( A[1..n] )
    for j = 2 to n
        key = A[j]
        i = j-1
        while i>0 and A[i]>key
            A[i+1] = A[i]
            i = i-1
        A[i+1] = key
```

$$\sum_{j=2}^{n} (t_j)$$

$$t(n) \leq c \cdot \sum_{j=2}^{n} (t_j)$$

Asymptotic Analysis

$$t(n) \leq c \cdot \sum_{j=2}^{n} (t_j)$$

$$t_{worst}(n) \leq c \cdot \sum_{j=2}^{n} (t_j)$$

$$\leq c \cdot \left(\frac{n(n+1)}{2} - 1 \right)$$

$$\leq \frac{c}{2} \cdot (n^2 + n - 2)$$

$$t(n) \leq c \cdot \sum_{j=2}^{n} (t_j)$$

$$t_{worst}(n) \leq c \cdot \sum_{j=2}^{n} (t_j)$$

$$= O(n^2)$$

Conclusion

- เวลาการทำงานแปรตาม
  - ขนาดและลักษณะของ instance
- ถ้าหนึ่งขนาดมีหลาย instances
  - worst-case, best-case, average-case analysis
- การวิเคราะห์
  - นับจำนวนคำสั่ง “Barometer”
  - ใช้ asymptotic notation ช่วยในการจัดการ

Algorithm and Complexity

PART 3:

ALGORITHM ANALYSIS: ASYMPTOTIC ANALYSIS

Don Knuth

- Father of analysis of algorithm
- Author of The Art of Computer Programming
- Programmer of the TEX and METAFONT

Asymptotic Analysis

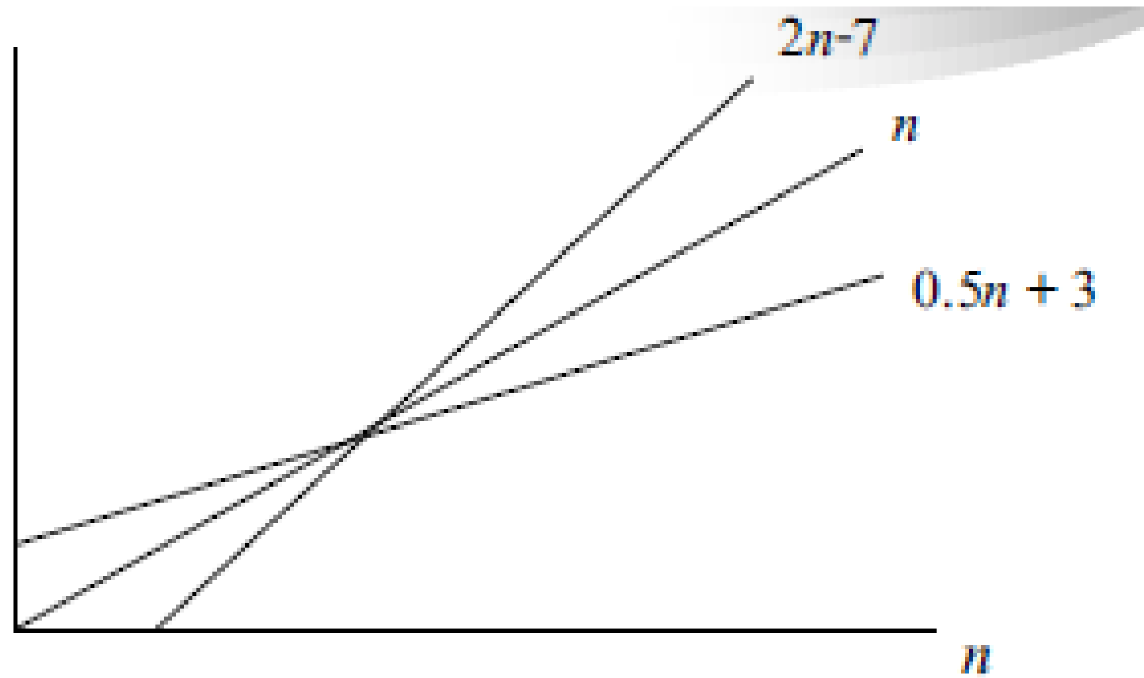
- To compare two algorithms with running times $f(n)$ and $g(n)$, we need a **rough measure** that characterizes **how fast each function grows.**
- <u>*Hint:*</u> use *rate of growth*
- Compare functions in the limit, that is, **asymptotically!**

  (i.e., for large values of n)

Growth of Functions

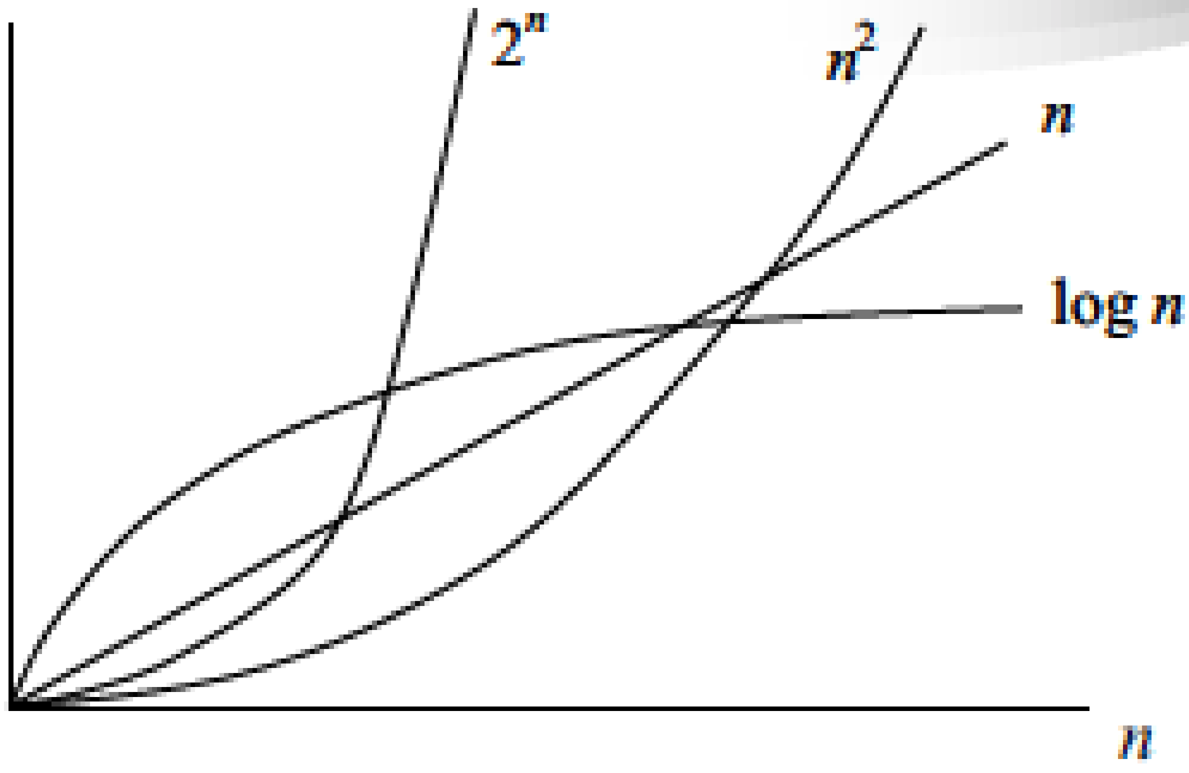
- Growth of Functions
- Asymptotic Notation : O, Ω, Θ, o, ω
- Asymptotic Notation Properties

Growth of Functions



Linear

Growth Rates



Growth Rates

$$\lim_{n\to\infty} f(n)/g(n) = \begin{cases} 0 & f(n) \text{ grows slower than } g(n) \\ \infty & f(n) \text{ grows faster than } g(n) \\ \text{otherwise} & f(n) \text{ and } g(n) \text{ have the same growth rate} \end{cases}$$

$f(n) \prec g(n)$

- $\prec$ (Karp reduction)
- $f(n) \prec g(n)$ iff $\lim_{n \to \infty} f(n)/g(n) = 0$
- $f(n) \prec g(n)$: $f(n)$ grows slower than $g(n)$
- $0.5^n \prec 1 \prec \log n \prec \log^6 n \prec n^{0.5} \prec n^3 \prec 2^n \prec n!$

$L'H\hat{o}pital'$s Rule

- If $f(n)$ and $g(n)$ are differentiable, $\lim_{n \to 0} f(n) = \infty$, $\lim_{n \to \infty} g(n) = \infty$,
- and $\lim_{n \to \infty} f'(n)/g'(n)$ exist, then
- $\lim_{n \to \infty} f(n)/g(n) = \lim_{n \to \infty} f'(n)/g'(n)$

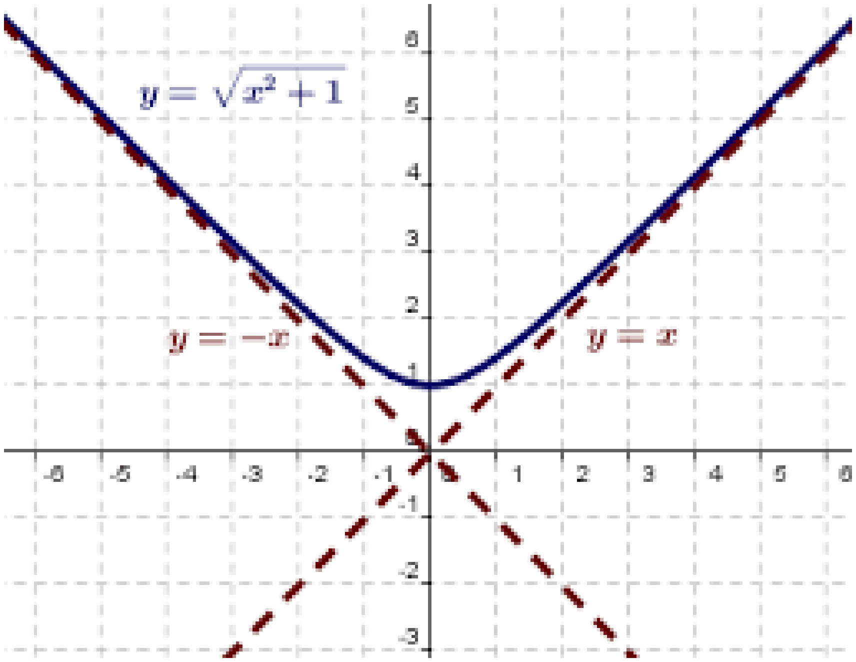
$\log n$ vs. n

- จากกฎของโลปิตาล
- $\lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \frac{(1/\ln 10)(1/n)}{1} = 0$
- ดังนั้น $\log n \prec n$

$\lg n$ vs. $\sqrt{n}$

- $\lim_{n\to\infty} \frac{\lg n}{\sqrt{n}} = \lim_{n\to\infty} \frac{\ln n}{\ln 2\sqrt{n}}$
- $\frac{1}{\ln 2} \lim_{n\to\infty} \frac{\ln n}{\sqrt{n}}$
- $\frac{1}{\ln 2} \lim_{n\to\infty} \frac{1/n}{1/(2\sqrt{n})}$
- $\frac{1}{\ln 2} \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$
- ดังนั้น $\lg n \prec \sqrt{n}$

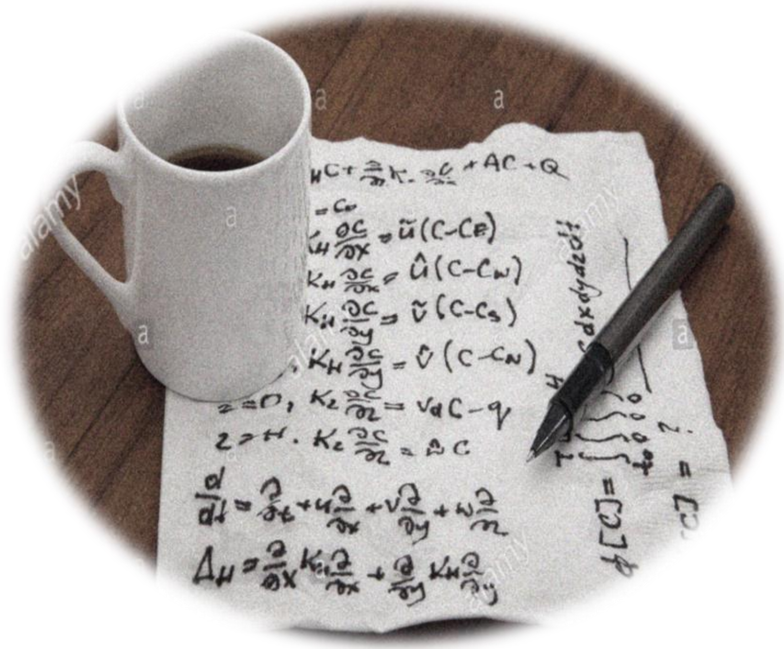
Asymptotic



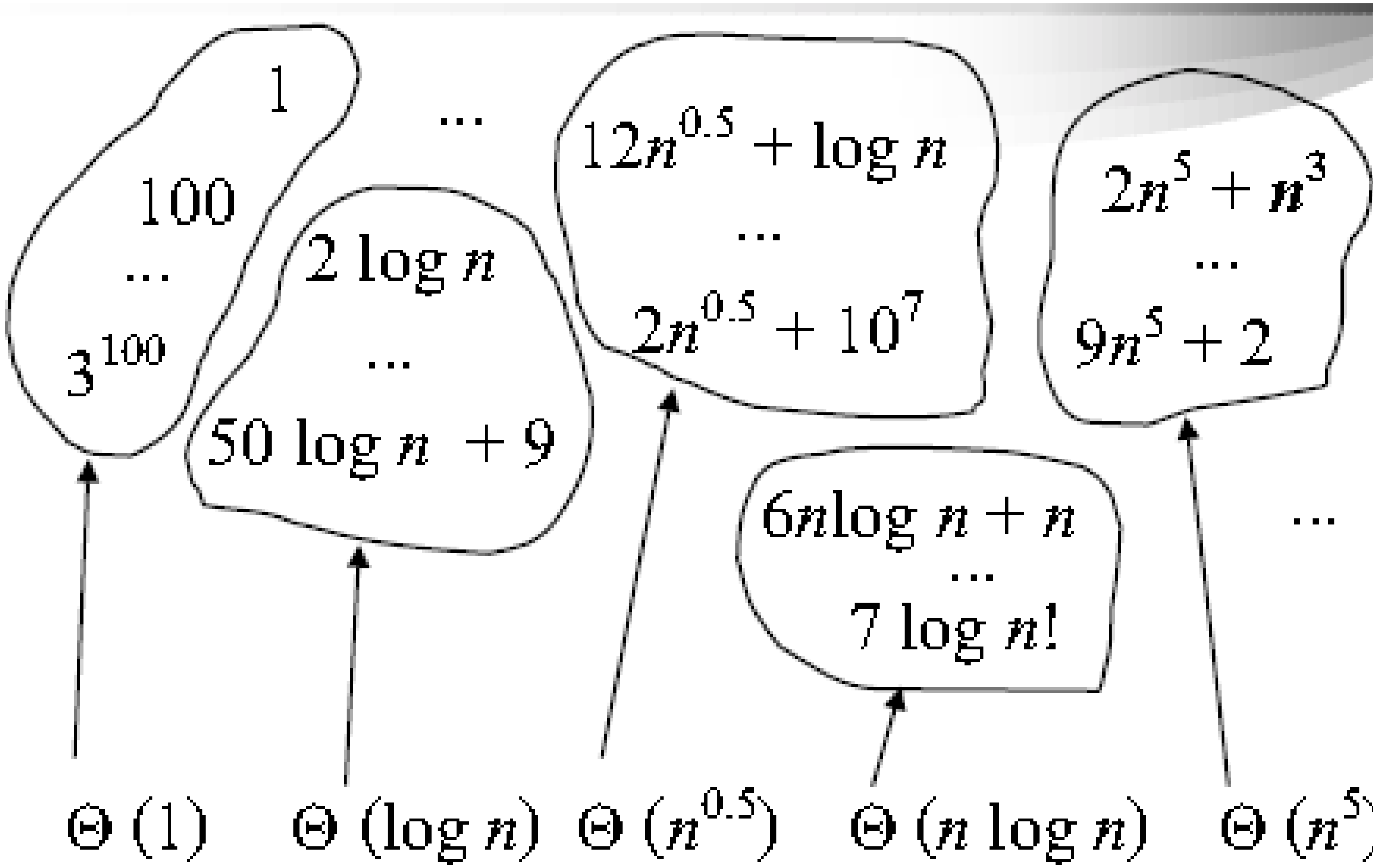
- Asymptotic : any approximation value that gets closer and closer to the truth, when some parameter approaches a limiting value.

Asymptotic Notations

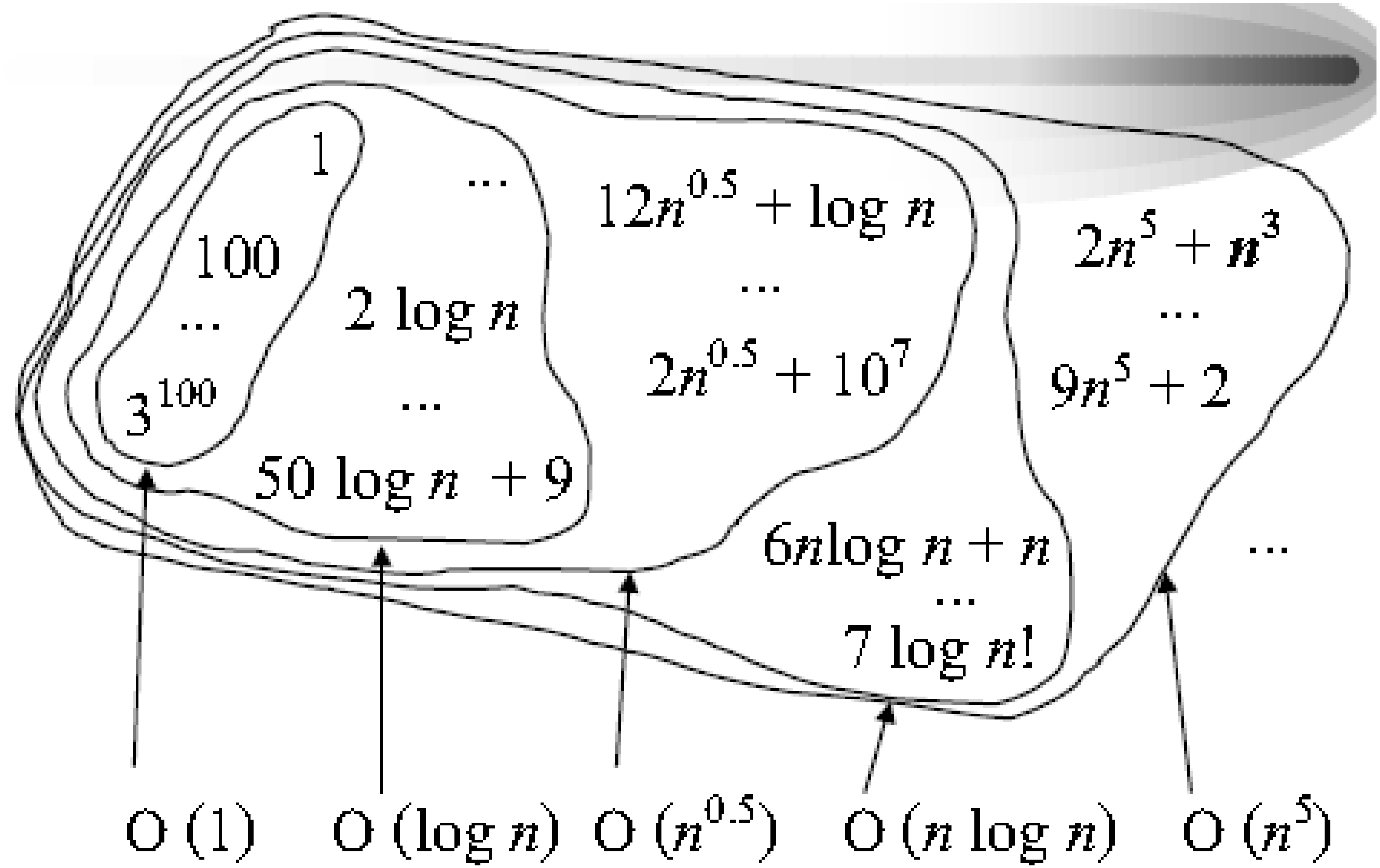
- Deal with the behaviour of functions in the limit (for sufficiently large value of its parameters)
- Permit substantial simplification (napkin mathematic, rough order of magnitude)
- Classify functions by their growth rates



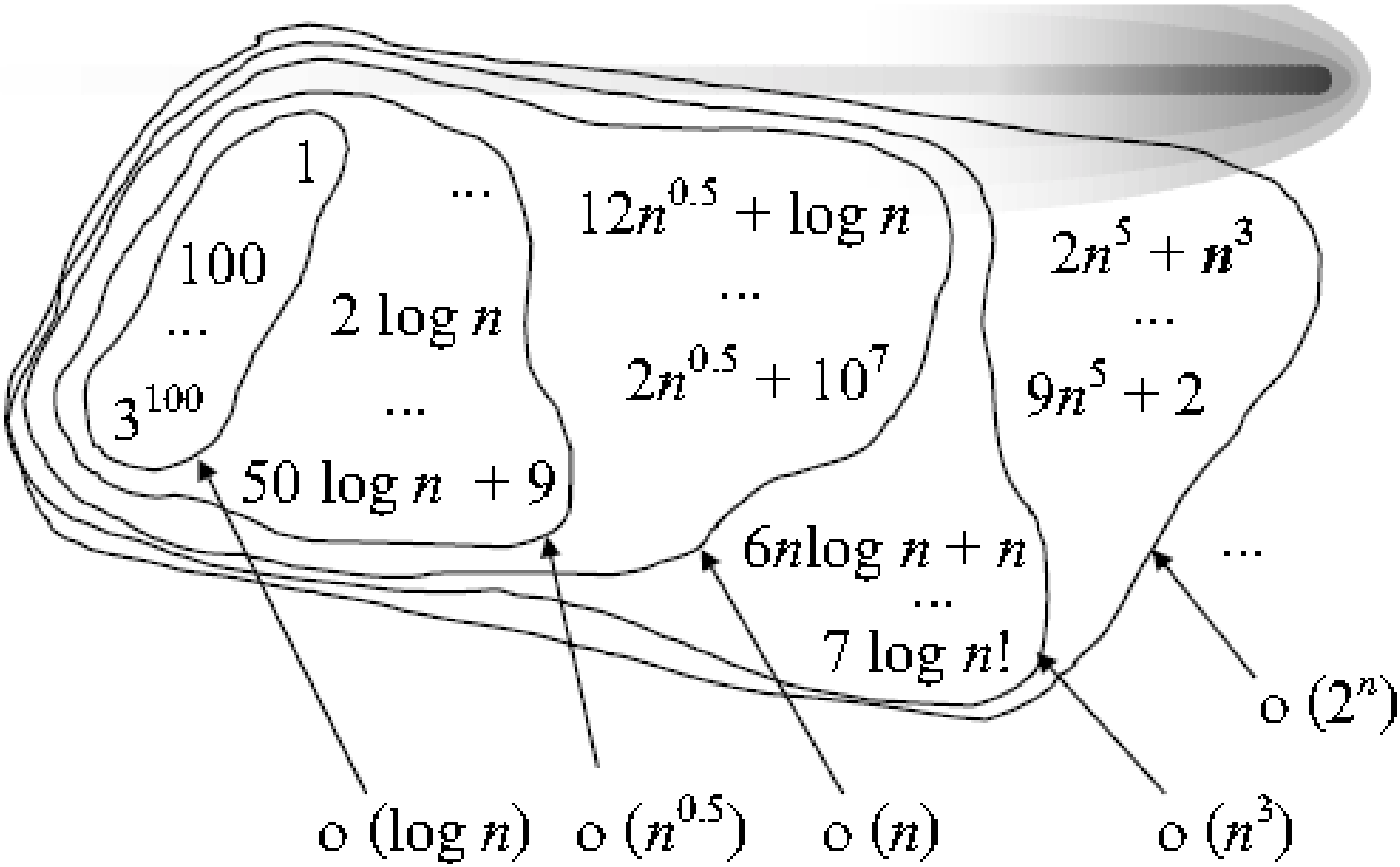
“Same” growth rate



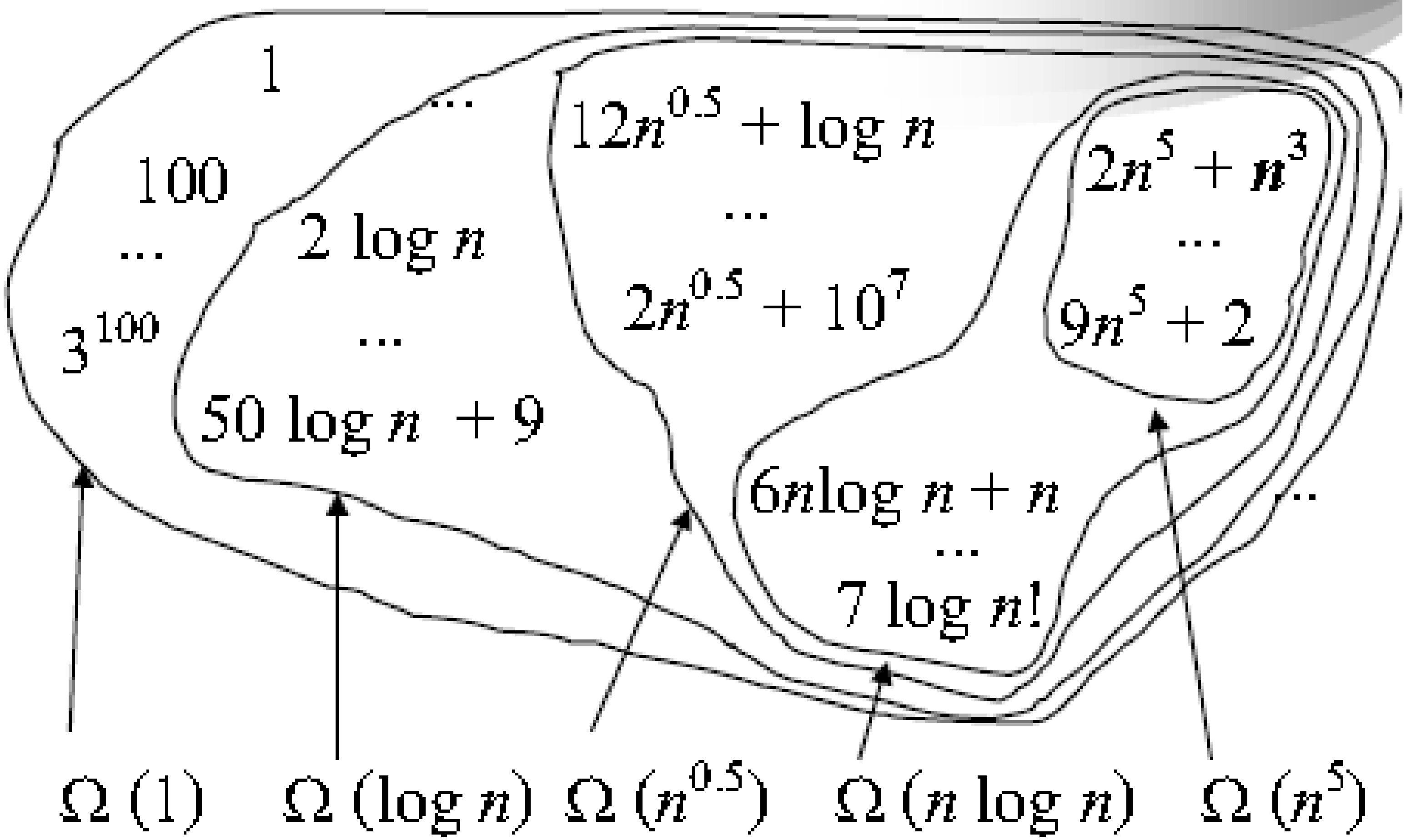
“No faster than” growth rate



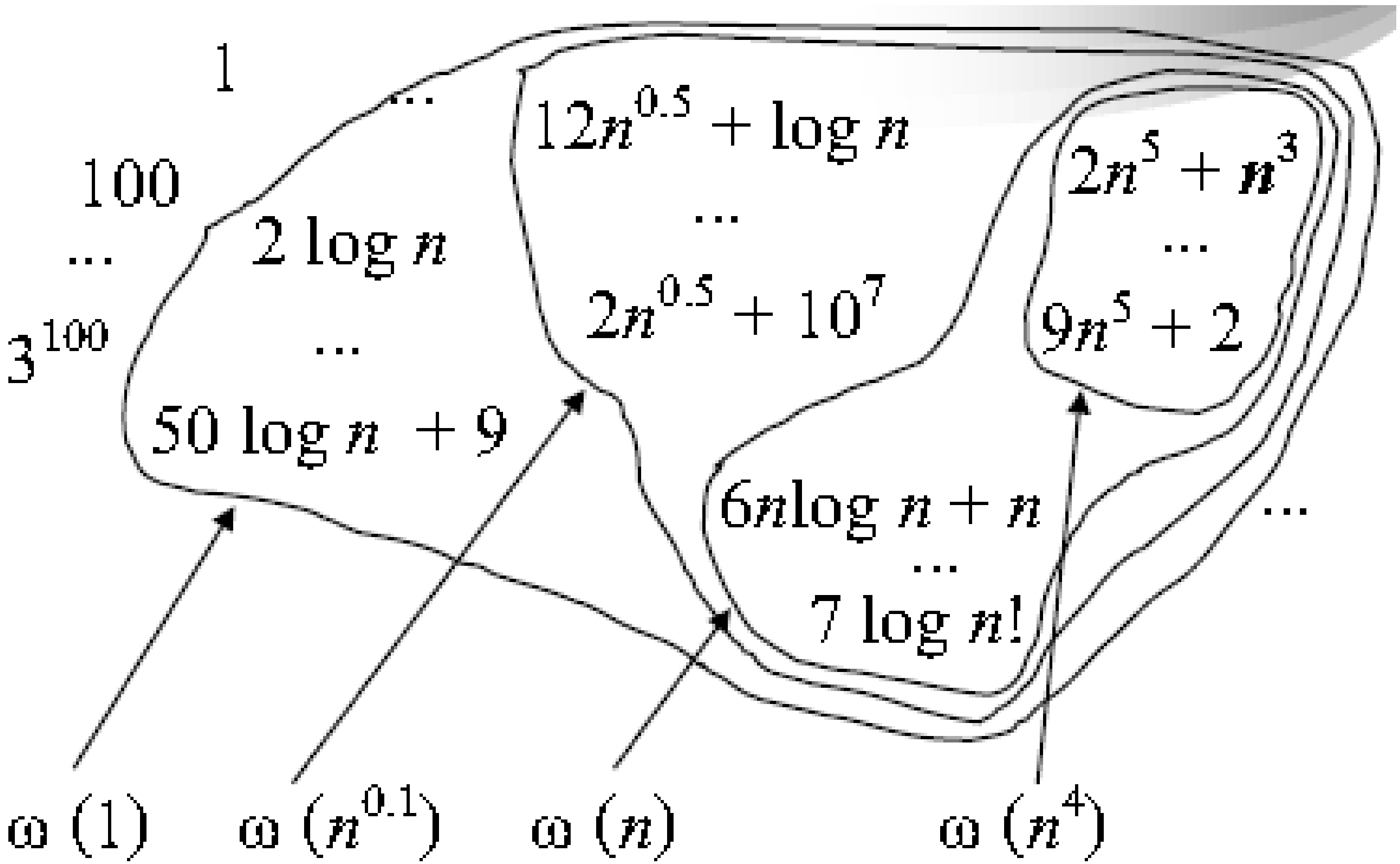
“Slower than” growth rate



“No Slower than” growth rate



"Faster than" growth rate



Special Orders of Growth

- constant : $\Theta(1)$
- logarithmic : $\Theta(\log n)$
- polylogarithmic : $\Theta(\log^c n)$, $c \geq 1$
- sublinear : $\Theta(n^a)$, $0 < a < 1$
- linear : $\Theta(n)$
- quadratic : $\Theta(n^2)$
- polynomial : $\Theta(n^c)$, $c \geq 1$
- exponential : $\Theta(c^n)$, $c > 1$

Analogy

• $f(n)$ and $g(n)$		f and g (real numbers)
• $f(n) = \Theta(g(n))$	$\approx$	$f = g$
• $f(n) = O(g(n))$	$\approx$	$f \leq g$
• $f(n) = o(g(n))$	$\approx$	$f < g$
• $f(n) = \Omega(g(n))$	$\approx$	$f \geq g$
• $f(n) = \omega(g(n))$	$\approx$	$f > g$

Not all functions are asymptotically comparable

n vs. $n^{1+\sin n}$

O-notation

- **นิยาม** : ความหมายของ $\mathrm{O}(n)$ คือ

  ฟังก์ชั่นนั้น ๆ ใช้เวลาทำงานช้าที่สุด $\leq n$

- $\mathrm{O}(g(n)) = \{ f(n)$: there exist positive constants
  c and $n0$ such that
  $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0 \}$

<u>ตัวอย่าง</u> เช่น อัลกอริทึม $a1$ มีประสิทธิภาพเป็น $\mathrm{O}(\mathrm{n}^2)$

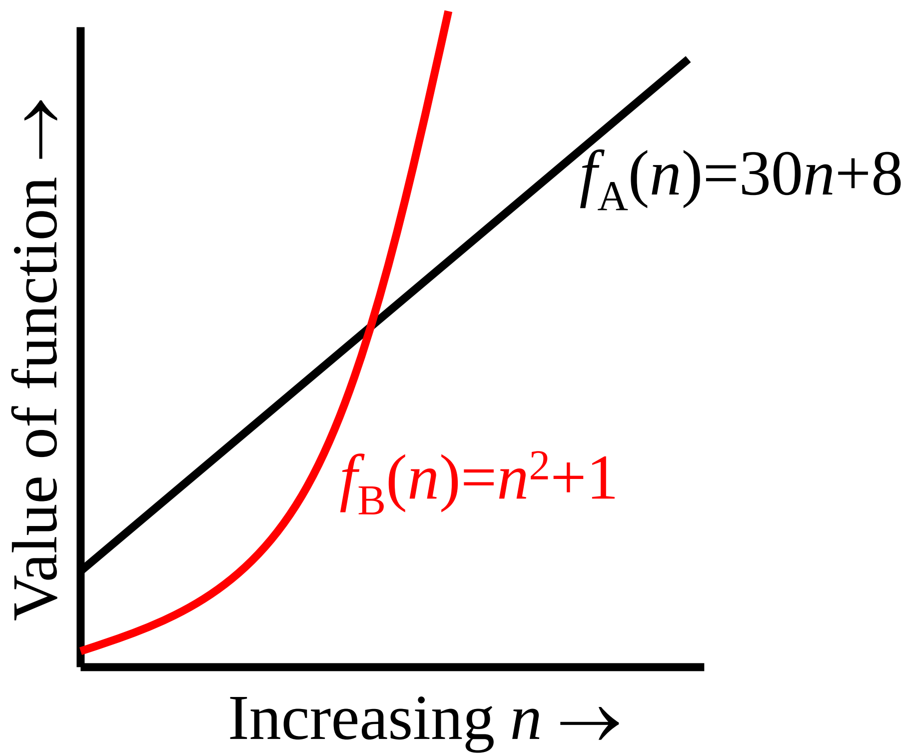
- ถ้า $n = 10$ แล้ว $a1$ จะใช้เวลาทำงานช้าที่สุด 100 หน่วยเวลา (รับประกันว่าไม่ช้าไปกว่านี้ - แต่อาจจะเร็วกว่านี้ได้)

O-notation

- We say $f_A(n)=30n+8$ is *order n*, or $O(n)$ It is, at most, roughly *proportional* to n.
- $f_B(n)=n^2+1$ is *order* n^2, or $O(n^2)$. It is, at most, roughly proportional to n^2.
- In general, any $O(n^2)$ function is faster-growing than any $O(n)$ function.

Visualizing Orders of Growth

- On a graph, as you go to the right, a faster growing function eventually becomes larger...



More Examples …

- $n^4 + 100n^2 + 10n + 50$ is $O(n^4)$
- $10n^3 + 2n^2$ is $O(n^3)$
- $n^3 - n^2$ is $O(n^3)$
- constants
  - 10 is O(1)
  - 1273 is O(1)

Examples

- $2n^3 = O(n^3)$: $2n^3 \le cn^3 \Rightarrow 2 \le cn \Rightarrow c = 1$ and $n_0 = 2$

- $n^2 = O(n^2)$: $n^2 \le cn^2 \Rightarrow c \ge 1 \Rightarrow c = 1$ and $n_0 = 1$

- $1000n^2+1000n = O(n^2)$:

$1000n^2+1000n \le 1000n^2+ n^2 = 1001n^2 \Rightarrow c = 1001$ and $n_0 = 1000$

- $n = O(n^2)$: $n \le cn^2 \Rightarrow cn \ge 1 \Rightarrow c = 1$ and $n_0 = 1$

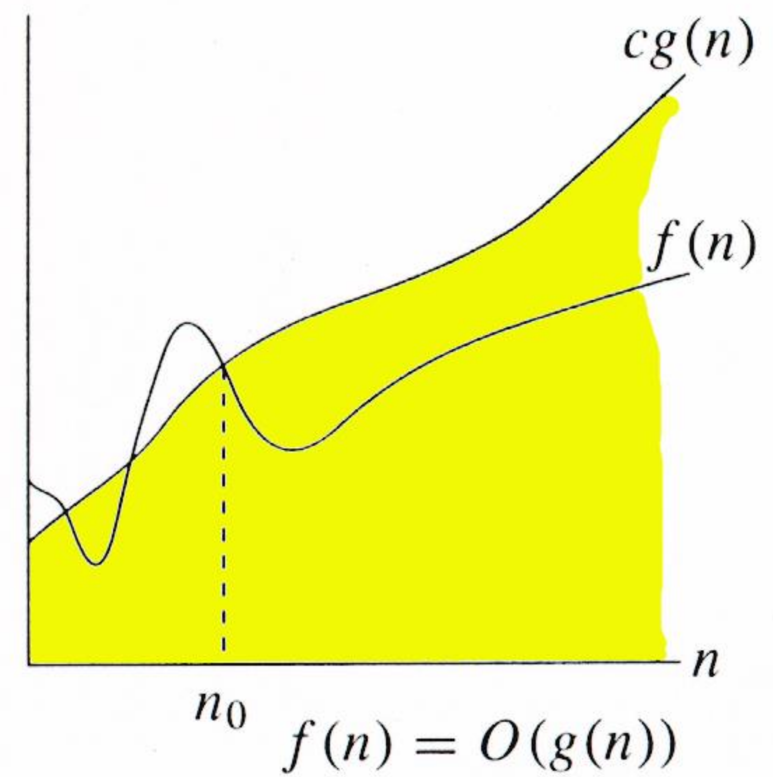
More Examples

- Show that $30n+8$ is $\mathrm{O}(n)$.
  - Show $\exists c,n_0$: $30n+8 \le cn$, $\forall n>\mathrm{n}_0$.
    - Let $c=31$, $n_0=8$. Assume $n>n_0=8$. Then $cn = 31n = 30n + n > 30n+8$, so $30n+8 < cn$.

Asymptotic Upper Bound

For function $g(n)$, we define $O(g(n))$, big-O of n, as the set:

$O(g(n)) = \{f(n)$:
$\exists$ positive constants c and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq f(n) \leq cg(n)$ \}



Intuitively: Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.

$g(n)$ is an *asymptotic upper bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$.
$\Theta(g(n)) \subset O(g(n))$.

Ω-notation

- **นิยาม** : ความหมายของ $\Omega\ (n)$ คือ

  ฟังก์ชั่นนั้น ๆ ใช้เวลาทำงานเร็วที่สุด $\geq n$

- $\Omega(g(n)) = \{ f(n)$: there exist positive constants
  c and $n0$ such that
  $0 \leq c\ g(n) \leq f(n)$ for all $n \geq n_0 \}$

- <u>ตัวอย่าง</u> เช่น อัลกอริทึม $a1$ มีประสิทธิภาพเป็น $\Omega(n)$
  - ถ้า $n = 10$ แล้ว $a1$ จะใช้เวลาทำงานเร็วที่สุด 10 หน่วยเวลา (รับประกันว่าไม่เร็วไปกว่านี้ - แต่อาจจะช้ากว่านี้ได้)

Examples

- $5n^2 = \Omega(n)$

  $\exists\ c, n_0$ such that: $0 \le cn \le 5n^2 \Rightarrow cn \le 5n^2 \Rightarrow c = 1$ and $n_0 = 1$

- $100n + 5 \ne \Omega(n^2)$

  $\exists\ c, n_0$ such that: $0 \le cn^2 \le 100n + 5$

  $100n + 5 \le 100n + 5n\ (\forall\ n \ge 1) = 105n$

  $cn^2 \le 105n \Rightarrow n(cn - 105) \le 0$

  Since n is positive $\Rightarrow cn - 105 \le 0 \qquad \Rightarrow n \le 105/c$

  $\Rightarrow$ contradiction: n cannot be smaller than a constant

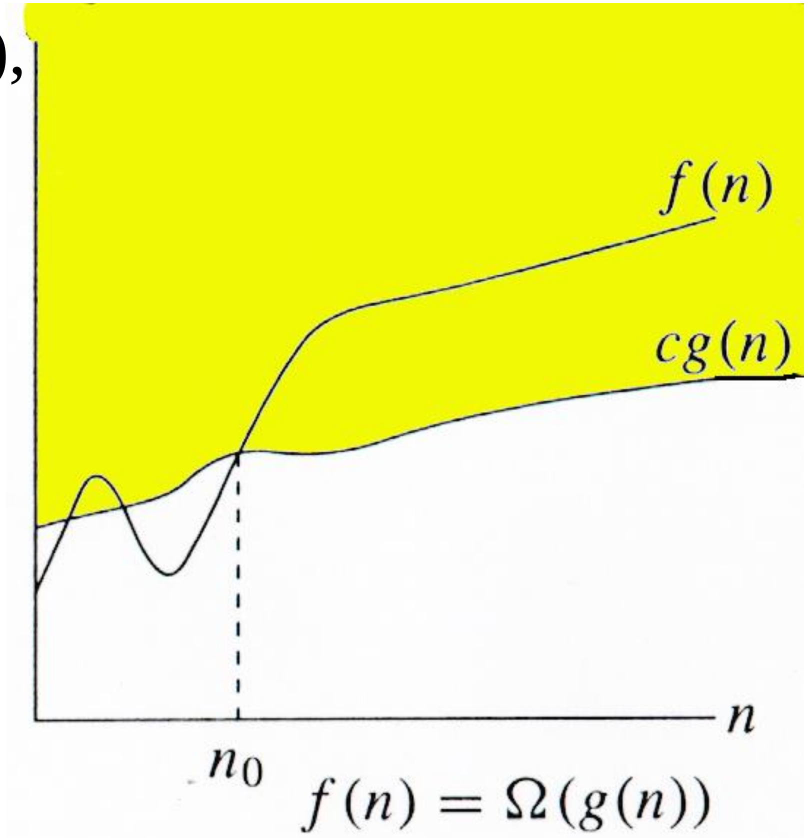
- $n = \Omega(2n)$, $n^3 = \Omega(n^2)$, $n = \Omega(\log n)$

Asymptotic Lower Bound

For function $g(n)$, we define $\Omega(g(n))$, big-Omega of n, as the set:

$\Omega(g(n)) = \{f(n)$:
$\exists$ **positive constants** c **and** n_0,
such that $\forall n \geq n_0$,
we have $0 \leq cg(n) \leq f(n)\}$

Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of $g(n)$.



$g(n)$ is an *asymptotic lower bound* for $f(n)$.

$f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$.
$\Theta(g(n)) \subset \Omega(g(n))$.

Θ-notation

- นิยาม : $f(n) = \Theta(g(n))$ ก็ต่อเมื่อ $f(n) = O(g(n))$

  และ $f(n) = \Omega(g(n))$

- $\Theta(g(n)) = \{ f(n)$: there exist positive constants c and $n0$ such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0 \}$

Examples

- $n^2/2 - n/2 = \Theta(n^2)$
  - $\frac{1}{2} n^2 - \frac{1}{2} n \leq \frac{1}{2} n^2 \ \forall n \geq 0 \quad \Rightarrow \quad c_2 = \frac{1}{2}$
  - $\frac{1}{2} n^2 - \frac{1}{2} n \geq \frac{1}{2} n^2 - \frac{1}{2} n * \frac{1}{2} n \ (\forall n \geq 2) = \frac{1}{4} n^2 \quad \Rightarrow$

    $c_1 = \frac{1}{4}$

- $n \neq \Theta(n^2)$: $c_1 n^2 \leq n \leq c_2 n^2$

$\Rightarrow$ only holds for: $n \leq 1/c_1$

Examples

– $6n^3 \neq \Theta(n^2)$: $c_1 n^2 \leq 6n^3 \leq \mathrm{c}_2 n^2$

$\Rightarrow$ only holds for: $n \leq c_2 /6$

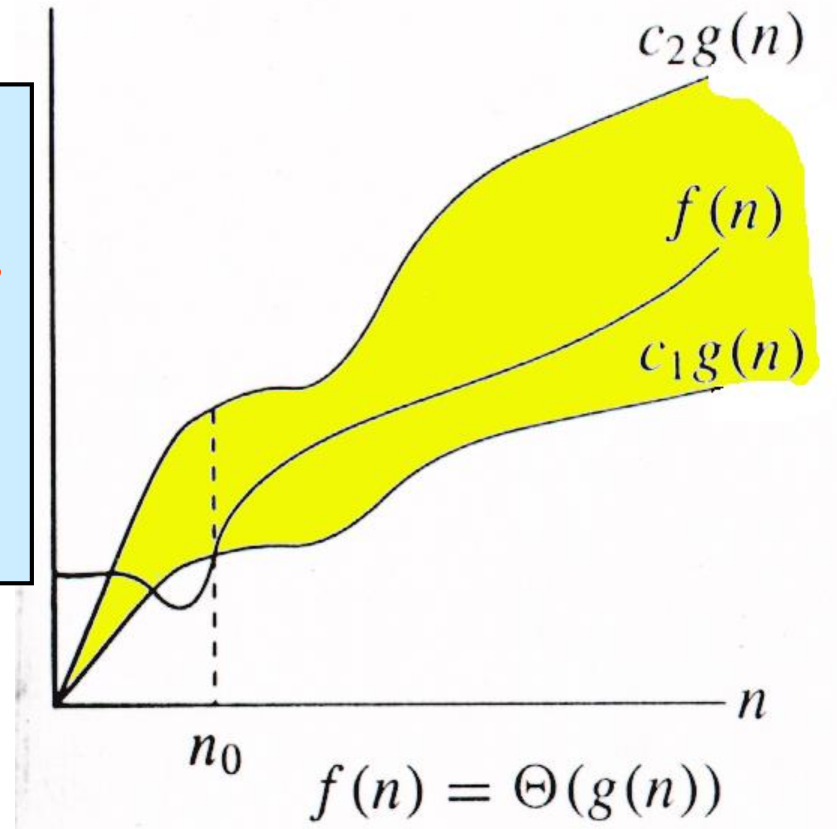
– $n \neq \Theta(\log n)$: $c_1 \log n \leq n \leq c_2 \log n$

$\Rightarrow c_2 \geq n/\log n, \forall n \geq n_0$--- impossible

Asymptotic Tight Bound

For function $g(n)$, we define $\Theta(g(n))$, big-Theta of n, as the set:

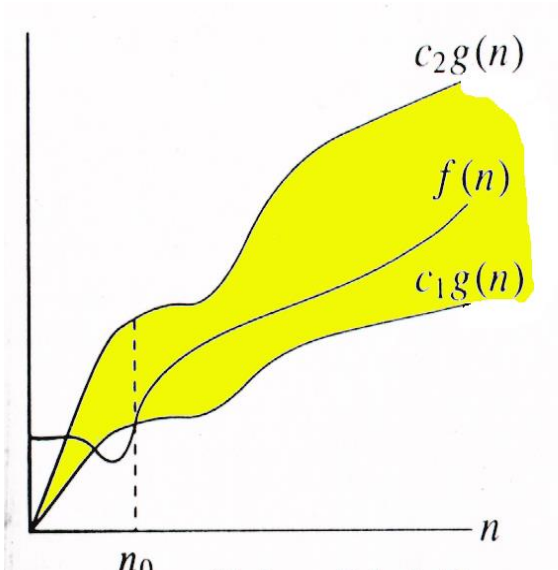
$\Theta(g(n)) = \{f(n)$:
$\exists$ positive constants c_1, c_2, and n_0,
such that $\forall n \geq n_0$,
we have $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$
$\}$



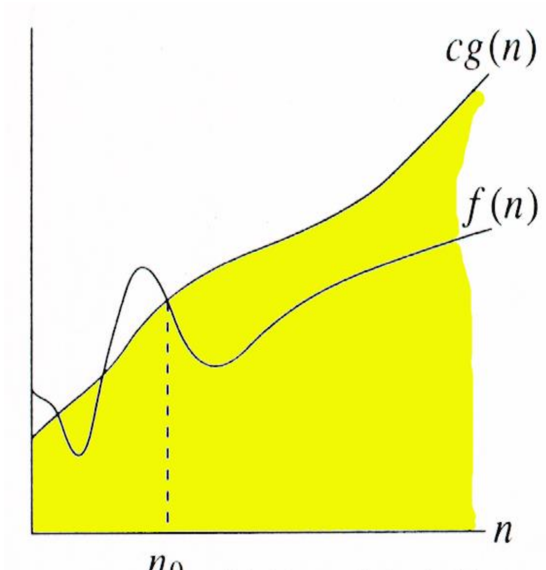
Intuitively: Set of all functions that have the same *rate of growth* as $g(n)$.

$g(n)$ is an *asymptotically tight bound* for $f(n)$.

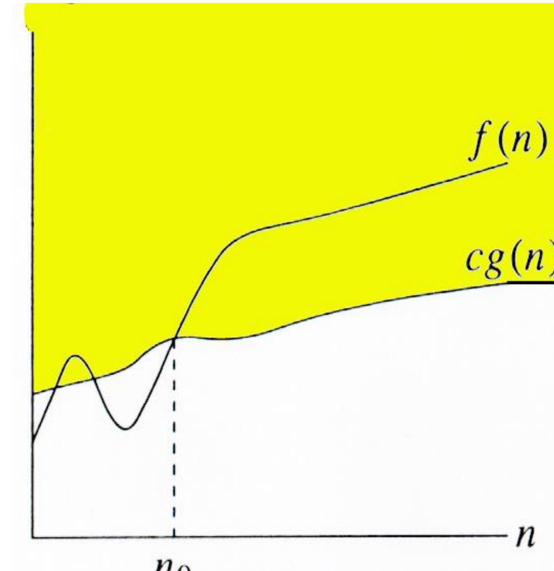
Relations Between Θ,O,Ω



$f(n) = \Theta(g(n))$



$f(n) = O(g(n))$



$f(n) = \Omega(g(n))$

o-notation

- **นิยาม** : ความหมายของ $o(n)$ คือ ฟังก์ชั่นนั้น ๆ **ใช้เวลาทำงานช้าที่สุด** $< n$
- ต่างจาก Big-O ตรงที่ Little-o จะไม่แตะขอบบน นั่นคือ ฟังก์ชั่นนี้ทำงานช้าที่สุดไม่ถึง n

$$\mathbf{o(g(n))} = \{f(n): \forall\ c > \mathbf{0},\ \exists\ n_0 > \mathbf{0} \text{ such that } \forall\ n \geq n_0, \text{ we have } 0 \leq f(n) < cg(n)\}.$$

– หากเรามี $t(n) = n0.98 + 0.05\sqrt{n}$ เราสามารถเขียนได้เป็น $O(n)$ หรือ $o(n)$ แต่หากระบุเป็น Little-o จะเน้นให้เห็นชัดว่าไม่ถึง n (เพราะค่ากำลังของ n คือ 1 แต่ในฟังก์ชั่น $t(n)$ ค่ากำลังของ n คือ 0.98)

ω-notation

- **นิยาม** : ความหมายของ $\omega\ (n)$ คือ ฟังก์ชั่นนั้น ๆ **ใช้เวลาทำงานเร็วที่สุด** $> n$
- ต่างจาก Big-omega ตรงที่ Little-omega จะไม่แตะขอบล่าง นั่นคือ ฟังก์ชั่นนี้ทำงานเร็วที่สุดมากกว่า n

$$\omega(g(n)) = \{f(n): \forall\ c > 0, \exists\ n_0 > 0 \text{ such that } \forall\ n \geq n_0, \text{ we have } 0 \leq cg(n) < f(n)\}.$$

Comparison of Functions

$$f \leftrightarrow g \;\approx\; a \leftrightarrow b$$

$$f(n) = \mathrm{O}(g(n)) \;\approx\; a \le b$$

$$f(n) = \Omega(g(n)) \;\approx\; a \ge b$$

$$f(n) = \Theta(g(n)) \;\approx\; a = b$$

$$f(n) = \mathrm{o}(g(n)) \;\approx\; a < b$$

$$f(n) = \omega(g(n)) \;\approx\; a > b$$

Asymptotic Notation Properties

- Transitivity
- Reflexivity
- Symmetry
- Transpose symmetry

Transitivity

- $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$ imply $f(n) = \Theta(h(n))$
- $f(n) = O(g(n))$ and $g(n) = O(h(n))$ imply $f(n) = O(h(n))$
- $f(n) = \Omega(g(n))$ and $g(n) = \Omega(h(n))$ imply $f(n) = \Omega(h(n))$
- $f(n) = o(g(n))$ and $g(n) = o(h(n))$ imply $f(n) = o(h(n))$
- $f(n) = \omega(g(n))$ and $g(n) = \omega(h(n))$ imply $f(n) = \omega(h(n))$

Symmetry and Transpose Symmetry

- $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
- $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$
- $f(n) = \Omega(g(n))$ if and only if $g(n) = O(f(n))$

Logarithms and properties

- In algorithm analysis we often use the notation "$\log n$" without specifying the base

Binary logarithm $\lg n = \log_2 n$

Natural logarithm $\ln n = \log_e n$

$$\lg^k n = (\lg n)^k$$

$$\lg \lg n = \lg(\lg n)$$

$$\log x^y = y \log x$$

$$\log xy = \log x + \log y$$

$$\log \frac{x}{y} = \log x - \log y$$

$$a^{\log_b x} = x^{\log_b a}$$

$$\log_b x = \frac{\log_a x}{\log_a b}$$

Manipulating Asymptotic Notations

- $c\ \mathrm{O}(f(n)) = \mathrm{O}(f(n))$
- $\mathrm{O}(\mathrm{O}\ f(n)) = \mathrm{O}\ (f(n))$
- $\mathrm{O}\ (f(n))\mathrm{O}(g(n)) = \mathrm{O}(f(n)\ g(n))$
- $\mathrm{O}\ (f(n)\ g(n)) = f(n)\mathrm{O}(g(n))$
- $\mathrm{O}\ (f(n) + g(n)) = \mathrm{O}(\ \max(f(n), g(n))\ )$

Conclusion

- การเติบโตของฟังก์ชัน
  - แสดงลักษณะการทำงานของฟังก์ชันอย่างง่าย
  - ช่วยให้เราสามารถเปรียบเทียบอัตราการเติบโตสัมพัทธ์ของฟังก์ชัน
- เราใช้ asymptotic notation เพื่อจำแนกฟังก์ชันตามอัตราการเติบโต
- Asymptotic เป็นศิลปะของการรู้หนักรู้เบาว่าตรงไหนควรพิจารณาเป็นพิเศษหรือตรงไหนควรละเว้น (knowing where to be sloppy and where to be precise)