資料結構
DATA STRUCTURE

Chap.01 Algorithm & Asymptotic notation

Ming-Han Tsai

2022 Fall

演算法(Algorithm)

- 演算法是指一連串有限數量的指令，為了完成某個特定任務。
- 基本特性
  - 輸入input: 支援0個、1個或多個輸入
  - 輸出output: 產生至少1個輸出
  - 定義definiteness: 每個步驟必須精準的定義：嚴格且明確的(unambiguously)指定每一個狀況該做的事
  - 有限finiteness: 經過有限步驟後必會終止
  - 有效effectiveness: 每個操作必須夠基本並且能被清楚完成
- 和程式不同：一個程式不一定要滿足 finiteness criteria.
  - Ex: 作業系統

演算法(Algorithm)

- 表示方式
  - 文字敘述.
  - 圖像化，例如流程圖.
  - 程式語言.
- 演算法 + 資料結構 = 程式 [Niklus Wirth, 1976]
- 範例：插入排序法(Insertion Sort)

排序(Sorting)

- 重新排列a[0]、a[1]、...a[n-1] ，使得排列之後遵守某個順序(Ex: 由小到大)
- 5,1,8,6,0 => 0,1,5,6,8
- 想想，該怎麼做？

各種排序方法

- 插入排序法(Insertion Sort)
- 氣泡排序法(Bubble Sort)
- 選擇排序法(Selection Sort)
- 計數排序法(Count Sort)
  - Bucket Sort，Radix Sort
- 堆積排序法(Heap Sort)
- 合併排序法(Merge Sort)
- 快速排序法(Quick Sort)

5,1,8,6,0 =>
0,1,5,6,8

想想，該怎麼做？

插入(Insert)

- 在一串已經排序好的數字中，加入新的數字

- Given 3, 6, 9, 14
- Insert 5
- Result 3, 5, 6, 9, 14

插入一個數字

- 3, 6, 9, 14　　插入 5
- 比較新插入的元素(5) 和 最後一個元素(14)
- 將14往右移：3, 6, 9, , 14
- 將9往右移：3, 6, , 9, 14
- 將6往右移：3, , 6, 9, 14
- (找到正確的位子了！) 插入 5 ：3, 5, 6, 9, 14

插入一個數字

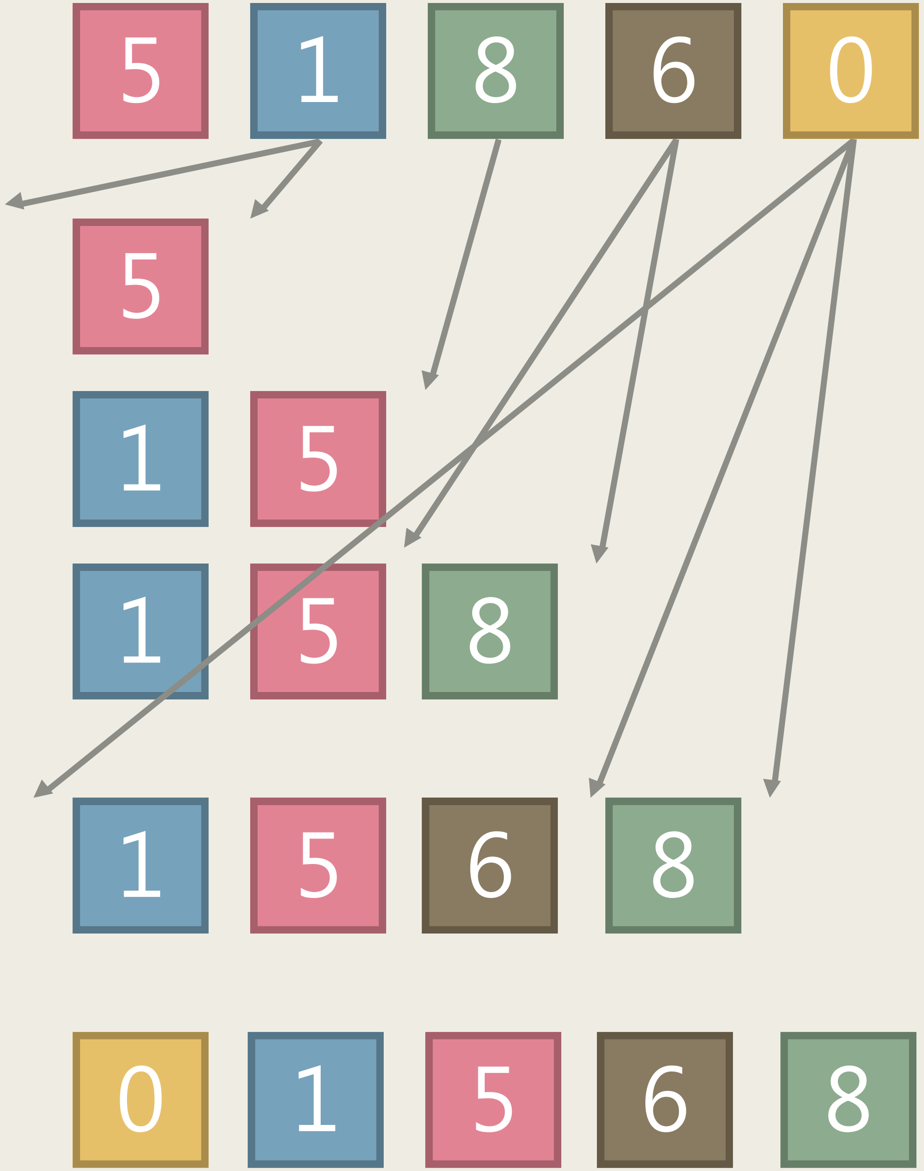
```
/* insert t into a[0:i-1] */
int j;
for (j = i - 1; j >= 0 && t < a[j]; j--)
    a[j + 1] = a[j];
a[j + 1] = t;
```

插入排序法(Insertion Sort)

- 一開始陣列長度為[]
- “看”第一個數字a，加入空陣列變成[a]
- 將元素重複一個一個加入

Insertion Sort

- Sort 5, 1, 8, 6, 0
- Start with 5 and insert 1 => 1, 5
- Insert 8 => 1, 5, 8
- Insert 6 => 1, 5, 6, 8
- Insert 0 => 0, 1, 5, 6, 8

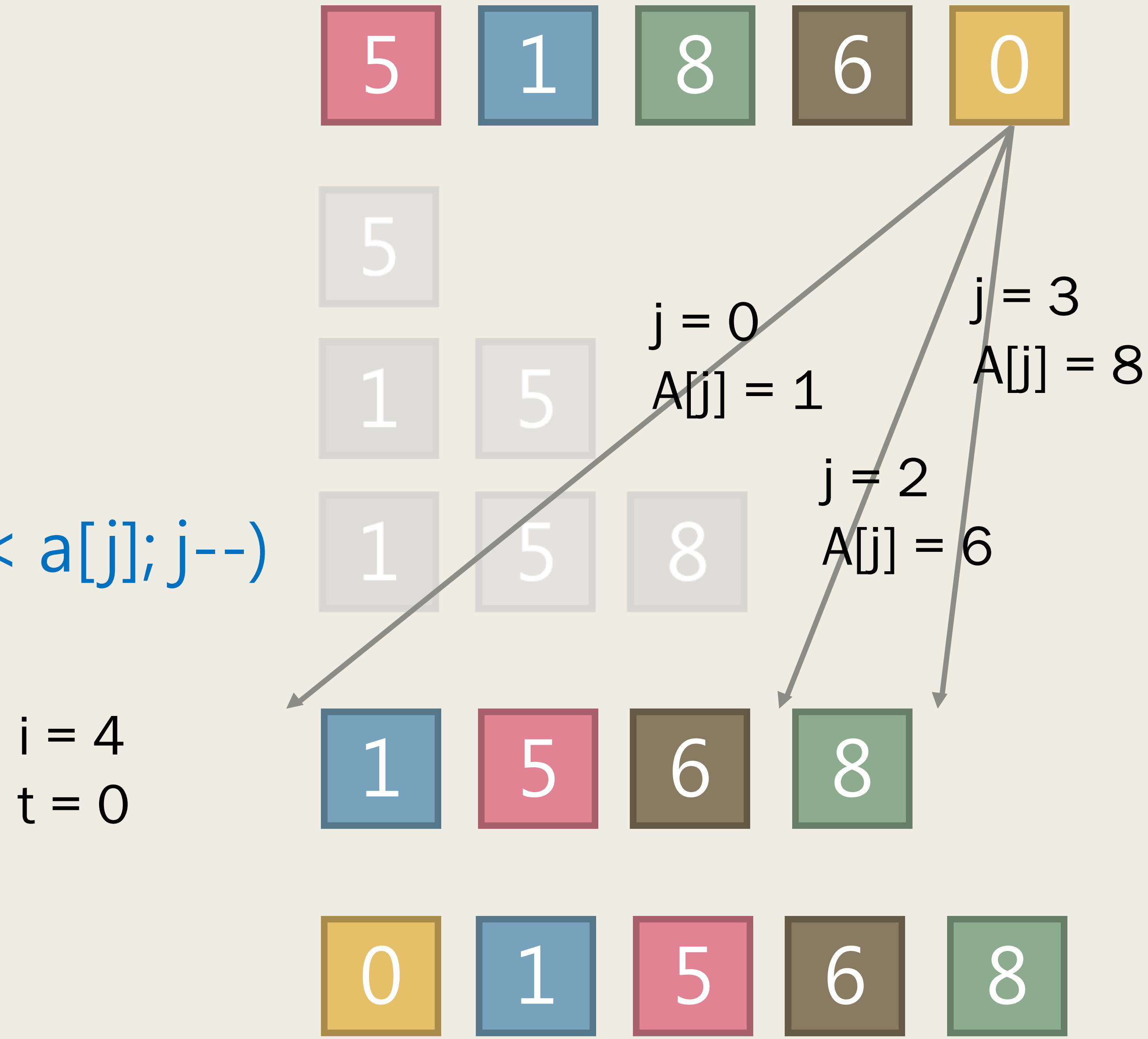


Insertion Sort

```
for (i = 1; i < n; i++)
{/* insert a[i] into a[0:i-1] */
   /* code to insert comes here */
}
```

Insertion Sort

```
for (i = 1; i < n; i++)
{/* insert a[i] into a[0:i-1] */
   int t = a[i];
   int j;
   for (j = i - 1; j >= 0 && t < a[j]; j--)
      a[j + 1] = a[j];
   a[j + 1] = t;
}
```



效能分析(Performance Analysis)

- 正確性
- 可讀性
- 效能分析(和機器無關)
  - 空間複雜度space complexity: 所需要的儲存空間
  - 時間複雜度time complexity: 所花費的計算時間
- 效能計算 (和機器有關)

效能分析(Performance Analysis)

- 空間複雜度 Space Complexity: 說穿了就是所佔的空間啦

$$S(P)=C+S_P(I)$$

  - 固定的空間需求 C
    和輸入和輸出無關
    - 指令所佔的空間
    - 必要的變數、資料結構所需的空間、常數等
  - 變動的空間需求 $S_P(I)$
    和使用方法和輸入I的**特性**有關
    - 數量、大小、值等輸入資料的特性
    - 迴圈或遞迴所需的空間、參數、區域變數、回傳值等隨著執行過程中可能需要的空間

效能分析(Performance Analysis)

- 計算a+b+b*c+(a+b-c)/(a+b)+4.00：$S_{abc}(I)=0$.

```
float abc(float a, float b, float c)
{
  return a+b+b*c +(a+b-c)/(a+b)+4.00;
}
```

Program 1.9: Simple arithmetic function

- 計算1+2+…+n-1：$S_{sum}(I)=S_{sum}(n)=0$.

```
float sum(float list[], int n)
{
  float tempsum = 0;
  int i;
  for (i = 0; i < n; i++)
    tempsum += list[i];
  return tempsum;
}
```

Recall: pass the address of the first element of the array & pass by value

Program 1.10: Iterative function for summing a list of numbers

效能分析(Performance Analysis)

– *利用遞迴(recursive)計算1+2+…+n-1*

```
float rsum(float list[], int n)
{
   if (n) return rsum(list,n-1) + list[n-1];
   return 0;
}
```

Program 1.11: Recursive function for summing a list of numbers

Type	Name	Number of bytes
parameter: float parameter: integer return address: (used internally)	*list*[] *n*	2 2 2 (unless a far address)
TOTAL per recursive call		6

$S_{sum}(I)=S_{sum}(n)=6n$

Figure 1.1: Space needed for one recursive call of Program 1.11

效能分析(Performance Analysis)

- 時間複雜度 Time Complexity: 說穿了就是所需要的時間啦

$$T(P)=C+T_P(I)$$

  - 固定的時間需求(C)
    - 編譯(Compile)所花的時間
  - 可變的時間需求$T_P(I)$
    - 執行(execution) 所需的時間 T_P
    - 通常會和輸入資訊的特徵有關

效能分析(Performance Analysis)

- 程式的步驟可來用分析執行時間
    - abc = a + b + b * c + (a + b - c) / (a + b) + 4.0
    - abc = a + b + c
- 計算程式所需的步驟數
    - 根據變數/計算次數
    - 表格
        - 計算每一段程式所需要的步數，以及每一步所真正需要的CPU time：execution × frequency
        - 加總每一段程式所需的時間

Performance of Insertion Sort

```
for (i = 1; i < n; i++)
{/* insert a[i] into a[0:i-1] */
   int t = a[i];
   int j;
   for (j = i - 1; j >= 0 && t < a[j]; j--)
      a[j + 1] = a[j];
   a[j + 1] = t;
}
```

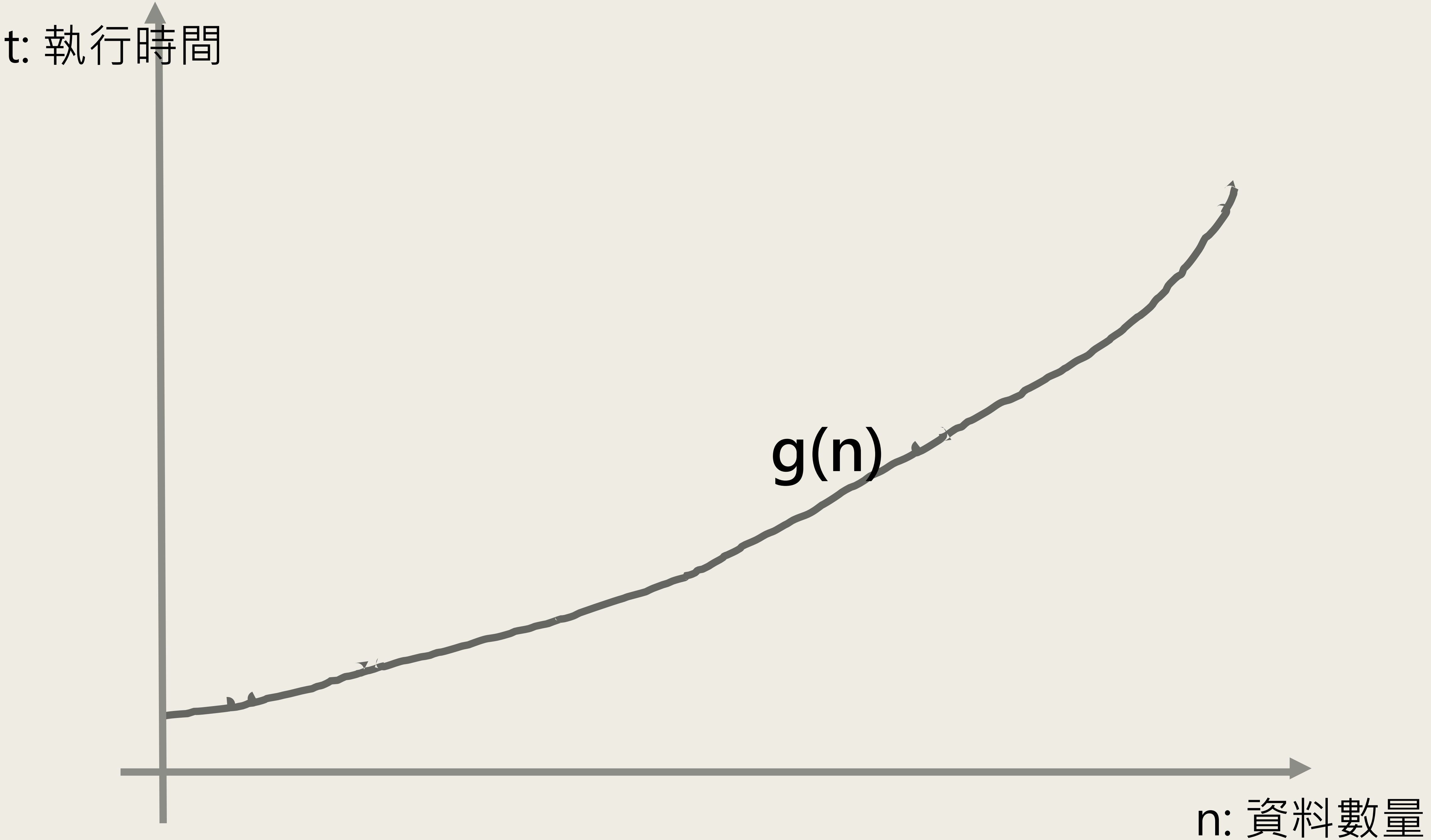
Best Case = ?
Worst Case = ?

Asymptotic notation(漸進表示符號)

- Asymptotic notation (O, Ω, Θ)
  - 考慮一下：$c_1n^2+c_2n$ 和 c_3n 的複雜度
    - for sufficiently large of value, c_3n is faster than $c_1n^2+c_2n$
    - for small values of n, either could be faster
      - $c_1=1$, $c_2=2$, $c_3=100$ --> $c_1n^2+c_2n \leq c_3n$ for $n \leq 98$
      - $c_1=1$, $c_2=2$, $c_3=1000$ --> $c_1n^2+c_2n \leq c_3n$ for $n \leq 998$
    - break even point
      - no matter what the values of c1, c2, and c3, the n beyond which c_3n is always faster than $c_1n^2+c_2n$

Asymptotic notation

- **Definition:** [Big "oh' ']
  - $f(n) = O(g(n))$ iff there exist positive constants c and n_0 such that $f(n) \leq cg(n)$ for all n, $n \geq n_0$.
  - 存在某個n_0和正數c，使得當$n > n_0$時, $f(n) <= cg(n)$
- **Definition:** [Omega]
  - $f(n) = \Omega(g(n))$ (read as "f of n is omega of g of n") iff there exist positive constants c and n_0 such that $f(n) \geq cg(n)$ for all n, $n \geq n_0$.
  - 存在某個n_0和正數c，使得當$n > n_0$時, $f(n) >= cg(n)$
- **Definition:** [Theta]
  - $f(n) = \Theta(g(n))$ (read as "f of n is theta of g of n") iff there exist positive constants c_1, c_2, and n_0 such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all n, $n \geq n_0$.
  - 存在某個n_0和正數c_1, c_2 ，使得當$n > n_0$時, $c_1 g(n) \leq f(n) \leq c_2 g(n)$

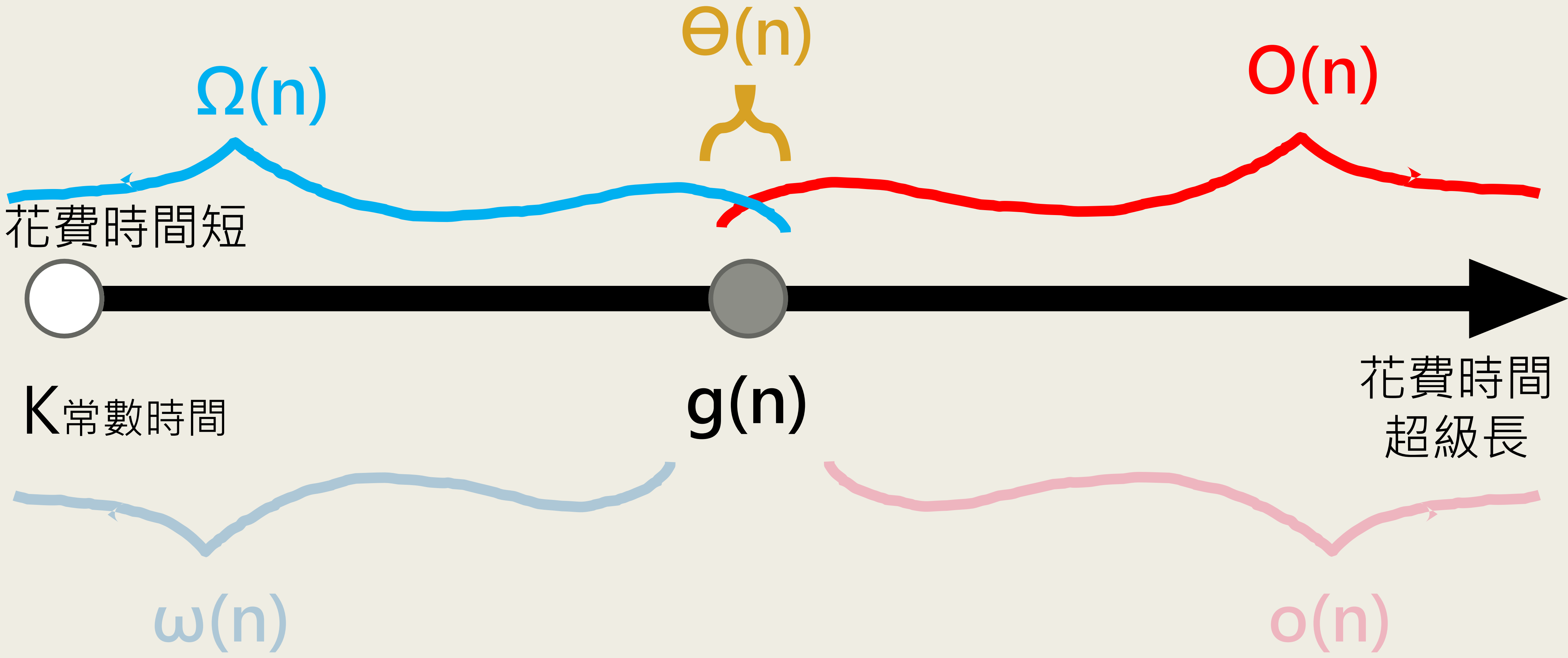

t: 執行時間
g(n)
n: 資料數量

Asymptotic notation

- **Theorem 1.2**:
  - *If $f(n) = a_m n^m + \ldots + a_1 n + a_0$, then $f(n) = O(n^m)$.*
- **Theorem 1.3:**
  - *If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Omega(n^m)$.*
- **Theorem 1.4:**
  - *If $f(n) = a_m n^m + \ldots + a_1 n + a_0$ and $a_m > 0$, then $f(n) = \Theta(n^m)$.*

Asymptotic notation

- **Examples**
  - *$f(n) = 3n+2$*
    - $3n + 2 <= 4n$, for all $n >= 2$, $\therefore 3n + 2 = O(n)$
      $3n + 2 >= 3n$, for all $n >= 1$, $\therefore 3n + 2 = \Omega(n)$
      $3n <= 3n + 2 <= 4n$, for all $n >= 2$, $\therefore 3n + 2 = \Theta(n)$
  - *$f(n) = 10n^2+4n+2$*
    - $10n^2+4n+2 <= 11n^2$, for all $n >= 5$, $\therefore 10n^2+4n+2 = O(n^2)$
      $10n^2+4n+2 >= n^2$, for all $n >= 1$, $\therefore 10n^2+4n+2 = \Omega(n^2)$
      $n^2 <= 10n^2+4n+2 <= 11n^2$, for all $n >= 5$, $\therefore 10n^2+4n+2 = \Theta(n^2)$
  - *$100n+6=O(n)$ /* $100n+6 \le 101n$ for $n \ge 10$ */*
  - *$10n^2+4n+2=O(n^2)$ /* $10n^2+4n+2 \le 11n^2$ for $n \ge 5$ */*
  - *$6*2^n+n^2=O(2^n)$ /* $6*2^n+n^2 \le 7*2^n$ for $n \ge 4$ */*



Q: Why we always use O(n) rather than Θ(n)?

Practice

- 猜密碼

假設小明有一個n位數
$(a_1,a_2,\ldots a_n,\ a_k = A\sim Z)$的密碼，
設計一個演算法能夠猜到
該密碼是什麼

1. 設計一個演算法，可以在有限次數內猜到小明的密碼是什麼
2. 該演算法的空間複雜度(Space Complexity)為？
3. 該演算法的時間複雜度(Time Complexity)為？

Asymptotic notation

		Instance characteristic n					
Time	Name	1	2	4	8	16	32
1	Constant	1	1	1	1	1	1
$\log n$	Logarithmic	0	1	2	3	4	5
n	Linear	1	2	4	8	16	32
$n \log n$	Log linear	0	2	8	24	64	160
n^2	Quadratic	1	4	16	64	256	1024
n^3	Cubic	1	8	64	512	4096	32768
2^n	Exponential	2	4	16	256	65536	4294967296
$n!$	Factorial	1	2	24	40326	20922789888000	26313×10^{33}

Figure 1.7 Function values

Asymptotic notation

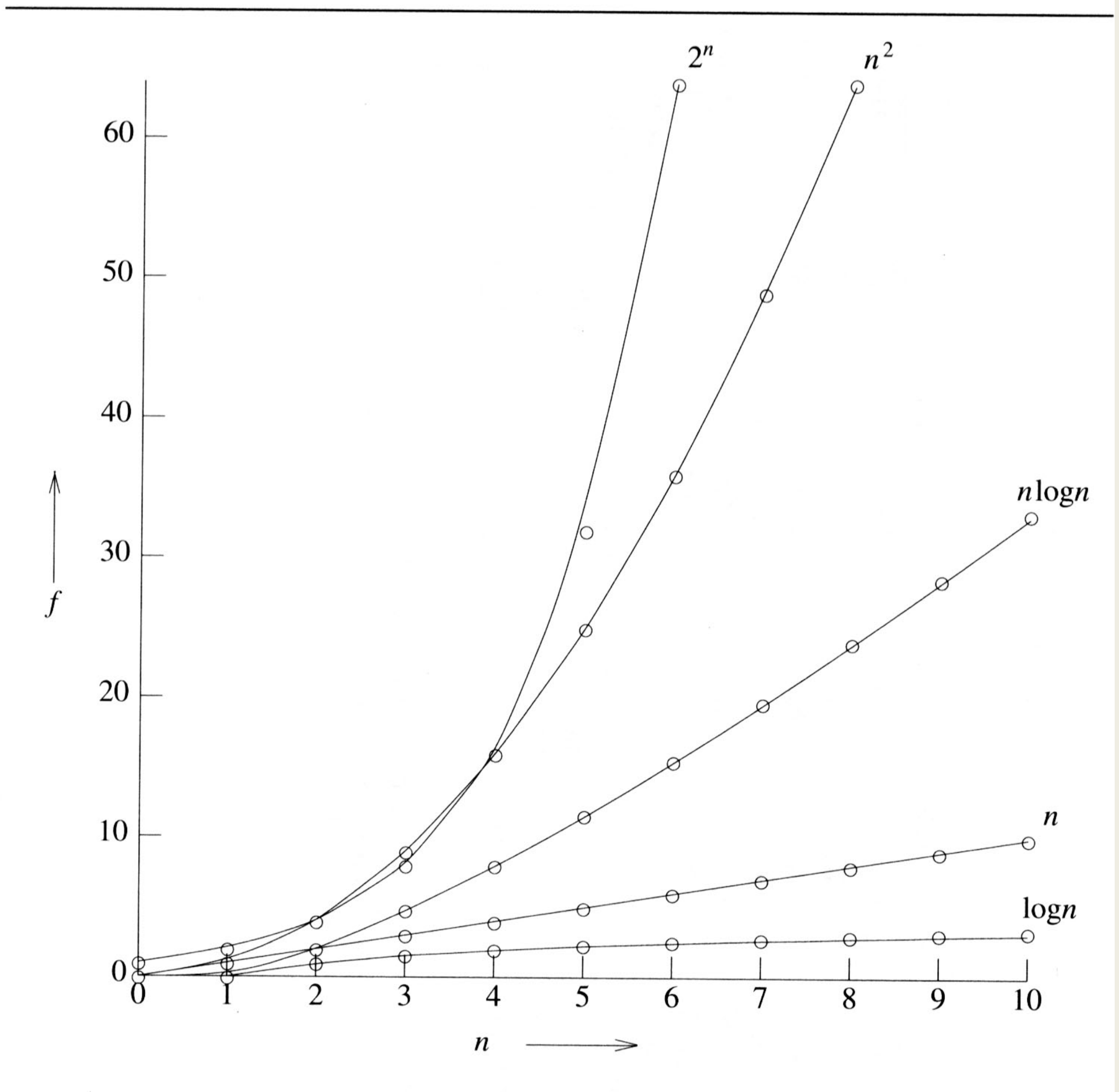


Figure 1.8 Plot of function values

Asymptotic notation

- 轉化成執行時間 by a 1 billion instructions per second computer

1.9 Times on a 1 billion instruction per second computer

	Time for $f(n)$ instructions on a 10^9 instr/sec computer						
n	$f(n)=n$	$f(n)=\log_2 n$	$f(n)=n^2$	$f(n)=n^3$	$f(n)=n^4$	$f(n)=n^{10}$	$f(n)=2^n$
10	.01µs	.03µs	.1µs	1µs	10µs	10sec	1µs
20	.02µs	.09µs	.4µs	8µs	160µs	2.84hr	1ms
30	.03µs	.15µs	.9µs	27µs	810µs	6.83d	1sec
40	.04µs	.21µs	1.6µs	64µs	2.56ms	121.36d	18.3min
50	.05µs	.28µs	2.5µs	125µs	6.25ms	3.1yr	13d
100	.10µs	.66µs	10µs	1ms	100ms	3171yr	$4*10^{13}$yr
1,000	1.00µs	9.96µs	1ms	1sec	16.67min	$3.17*10^{13}$yr	$32*10^{283}$yr
10,000	10.00µs	130.03µs	100ms	16.67min	115.7d	$3.17*10^{23}$yr	
100,000	100.00µs	1.66ms	10sec	11.57d	3171yr	$3.17*10^{33}$yr	
1,000,000	1.00ms	19.92ms	16.67min	31.71yr	$3.17*10^7$yr	$3.17*10^{43}$yr	

µs = microsecond = 10^{-6} seconds
ms = millisecond = 10^{-3} seconds
sec = seconds
min = minutes
hr = hours
d = days
yr = years

Performance Measurement

- [***Worst case performance of the selection function***]:
  - *The tests were conducted on an IBM compatible PC with an 80386 cpu, an 80387 numeric coprocessor, and a turbo accelerator. We use Broland's Turbo C compiler.*

n	Time	n	Time
30 · · · 100	.00	900	1.86
200	.11	1000	2.31
300	.22	1100	2.80
400	.38	1200	3.35
500	.60	1300	3.90
600	.82	1400	4.54
700	1.15	1500	5.22
800	1.48	1600	5.93

Figure 1.11: Worst case performance of selection sort (in seconds)

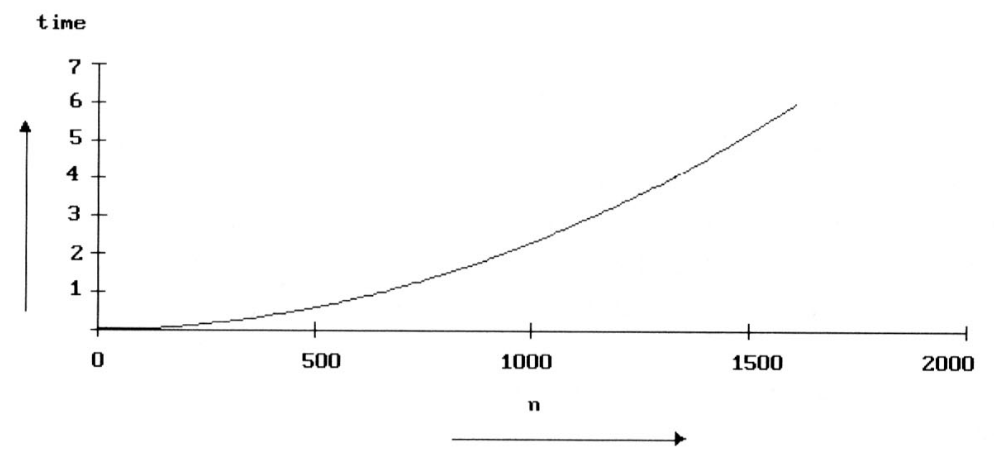
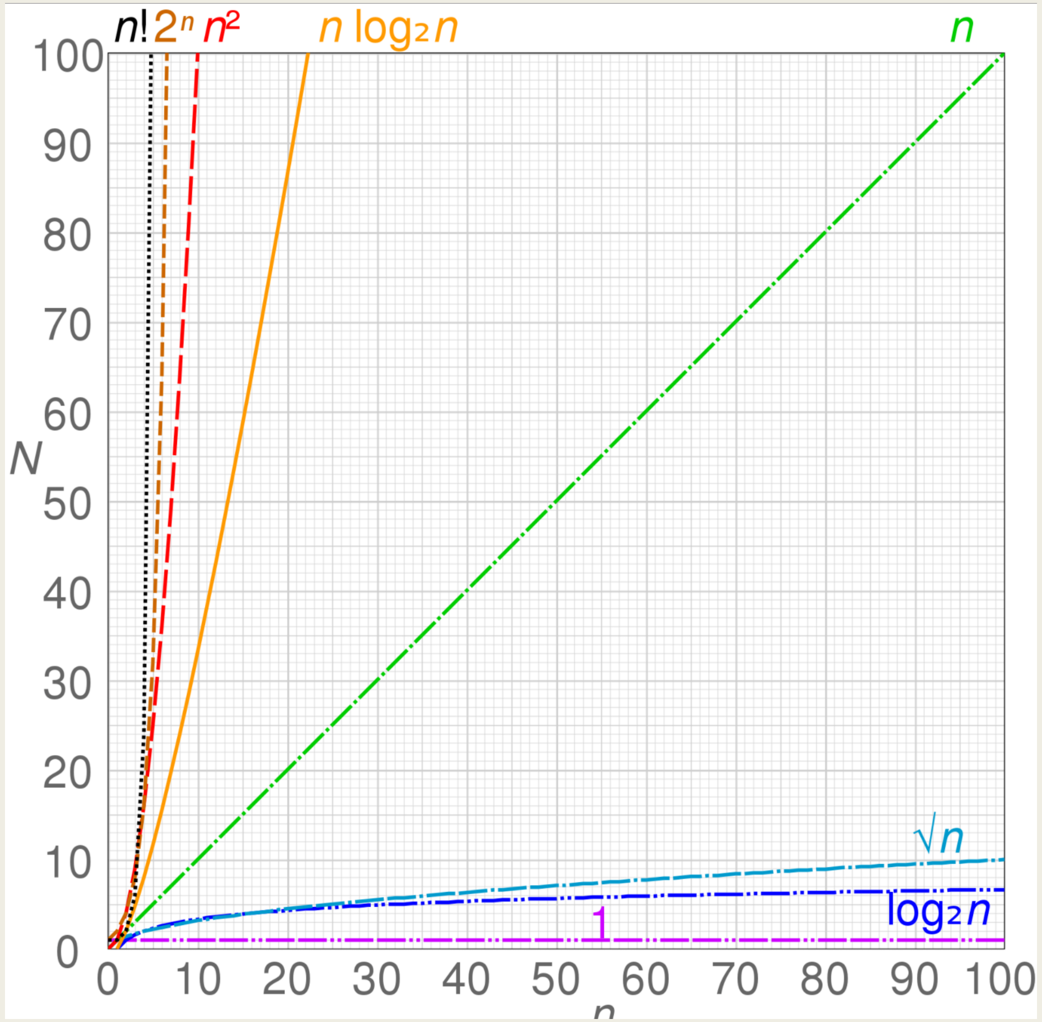


Figure 1.12: Graph of worst case performance for selection sort

Practice

- **Big O 的效率排序**

1. O(1)
2. O(log n)
3. O(n)
4. O(n log n)
5. O(n^2)
6. O($\sqrt{n}$)
7. O(n!)
8. O(2^n)



Ans: O(1) < O(log n) < O($\sqrt{n}$)<O(n)<O(n log n)<O(n^2)<O(2^n)<O(n!)

Practice

- 請計算下列程式中f(x): x=x+1 的執行次數

(a)
```
for i = 1 to n
    for j = 1 to n
        x = x + 1
    end
end
```

(b)
```
i=1
while i<=n
    x = x+1
    i = i+1
end
```

(c)
```
for i = 1 to n
    for j = 1 to i
        for k = 1 to j
            x = x+1
        end
    end
end
```

(d)
```
for i = 1 to n
    j = i
    for k = j+1 to n
        x = x + 1
    end
end
```

(e)
```
for i = 1 to n
    j = i
    while j>=2
        j = j / 5
        x = x + 1
    end
end
```

(f)
```
k = 100000
While k > 5
    k = k / 10
    x = x +1
end
```