การวิเคราะห์เวลาการทำงาน

หัวข้อ

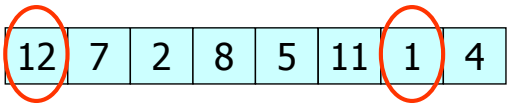
- การเปรียบเทียบเวลาการทำงาน
- การนับจำนวนครั้งที่คำสั่งทำงาน
- การวิเคราะห์ฟังก์ชันเวลาการทำงาน
- อัตราการเติบโตของฟังก์ชัน
- การวิเคราะห์เชิงเส้นกำกับ
- สัญกรณ์เชิงเส้นกำกับ
- การวิเคราะห์เวลาการทำงานของเมท็อดใน ArrayCollection

อยากรู้เวลาการทำงาน

- เขียนโปรแกรม
- สั่งทำงาน
- จับเวลา

ตัวอย่าง : มี int[] d

อยากทราบ ผลต่าง ของคู่ข้อมูลใน d ที่มีผลต่างมากสุด



คำตอบคือ 11

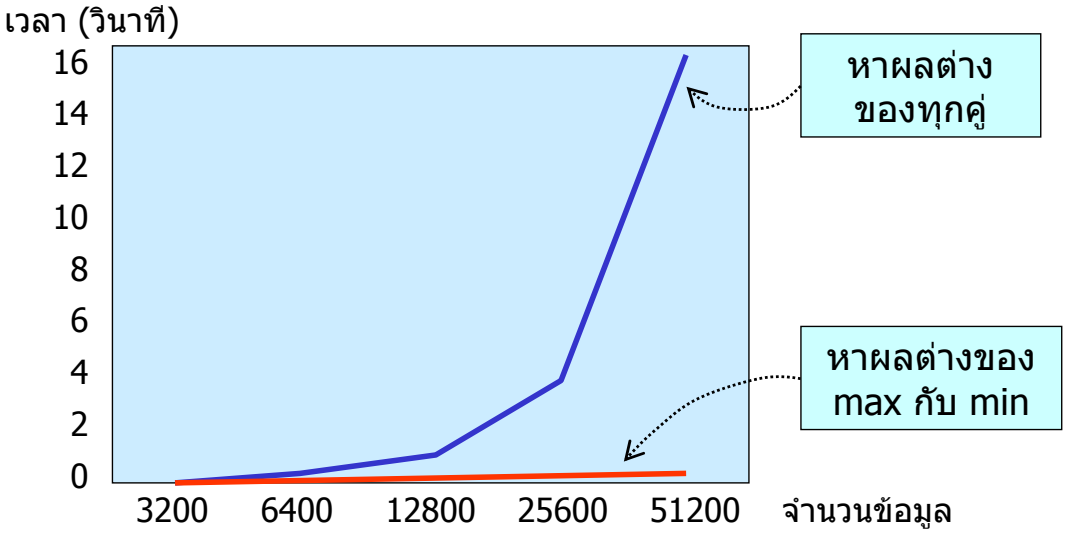
เขียนโปรแกรม : หาผลต่างของทุกคู่

```
public class Test1 {
  public static void main(String[] args) {
    for (int n = 3200; n < 100000; n *= 2) {
      int[] a = new int[n];
      long start = System.nanoTime();
      int diff = maxDiff(a);
      long d = (System.nanoTime() - start);
      System.out.println(n + " : " + d / 1E9);
    }
  }
  static int maxDiff(int[] d) {
    int maxDiff = 0;
    for (int i = 0; i < d.length; i++) {
      for (int j = i+1; j < d.length; j++) {
        int diff = Math.abs(d[i] - d[j]);
        if (diff > maxDiff) maxDiff = diff;
      }
    }
    return maxDiff;
  }
```

เขียนโปรแกรม : หาผลต่างของ max กับ min

```
public class Test2 {
  public static void main(String[] args) {
    for (int n = 3200; n < 100000; n *= 2) {
      int[] a = new int[n];
      long start = System.nanoTime();
      int diff = maxDiff(a);
      long d = (System.nanoTime() - start);
      System.out.println(n + " : " + d / 1E9);
    }
  }
  static int maxDiff(int[] d) {
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    for (int i = 0; i < d.length; i++) {
      if (d[i] > max) max = d[i];
      if (d[i] < min) min = d[i];
    }
    return max - min;
  }
```


เปรียบเทียบเวลาการทำงาน



ให้ n แทนจำนวนข้อมูล

$t(n)$ แทนเวลาในการหาผลต่างของคู่ข้อมูลที่มีผลต่างมากสุด

แบบที่ 1 : $t \propto n^2$

แบบที่ 2 : $t \propto n$

อยากรู้อัตราการเติบโตของเวลาการทำงาน

- เขียนอัลกอริทึม
- กำหนดตัวแปรที่แทนปริมาณข้อมูลขาเข้า
- หาคำสั่งตัวแทน
  - เวลาการทำงานแปรตามจำนวนครั้งที่คำสั่งตัวแทนทำงาน
- วิเคราะห์จำนวนครั้งที่คำสั่งตัวแทนทำงาน
- หาฟังก์ชันของจำนวนครั้งที่คำสั่งตัวแทนทำงานกับปริมาณข้อมูลขาเข้า

นับจำนวนครั้งที่คำสั่งทำงาน

```
public class Test1 {
  public static void main(String[] args) {
    int[] a = new int[5];
    int diff = maxDiff(a);
  }
  static int maxDiff(int[] d) {
    int maxDiff = 0;
    for (int i = 0; i < d.length; i++) {
      for (int j = i+1; j < d.length; j++) {
        int diff = Math.abs(d[i] - d[j]);
        if (diff > maxDiff) maxDiff = diff;
      }
    }
    return maxDiff;
  }
```

5, 10, 20

นับจำนวนครั้งที่คำสั่งทำงาน

```
public class Test2 {
  public static void main(String[] args) {
    int[] a = new int[5];
    int diff = maxDiff(a);
  }
  static int maxDiff(int[] d) {
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    for (int i = 0; i < d.length; i++) {
      if (d[i] > max) max = d[i];
      if (d[i] < min) min = d[i];
    }
    return max - min;
  }
```

5, 10, 20

ใช้เครื่องมือช่วยนับ

	n = 5	n = 10	n = 20	
แบบที่ 1	10	45	190	$n(n\text{-}1)/2$
แบบที่ 2	5	10	20	n

$(5\times4)/2$ $(10\times9)/2$ $(20\times19)/2$

วิเคราะห์ฟังก์ชันเวลาการทำงาน

```
public class Test1 {
  public static void main(String[] args) {
   int[] a = new int[5];
   int diff = maxDiff(a);
  }
  static int maxDiff(int[] d) {
    int maxDiff = 0;
    for (int i = 0; i < d.length; i++) {
      for (int j = i+1; j < d.length; j++) {
        int diff = Math.abs(d[i] - d[j]);
        if (diff > maxDiff) maxDiff = diff;
      }
    }
    return maxDiff;
  }
```

ให้ n คือขนาดของอาเรย์

เลือกบรรทัดนี้เป็นคำสั่งตัวแทน

$$\sum_{i=0}^{n-1}\sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-1}(n-1-i) = n^2 - n - \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

วิเคราะห์ฟังก์ชันเวลาการทำงาน

```
public class Test2 {
  public static void main(String[] args) {
   int[] a = new int[5];
   int diff = maxDiff(a);
  }
  static int maxDiff(int[] d) {
    int max = Integer.MIN_VALUE;
    int min = Integer.MAX_VALUE;
    for (int i = 0; i < d.length; i++) {
      if (d[i] > max) max = d[i];
      if (d[i] < min) min = d[i];
    }
    return max - min;
  }
```

ให้ n คือขนาดของอาเรย์

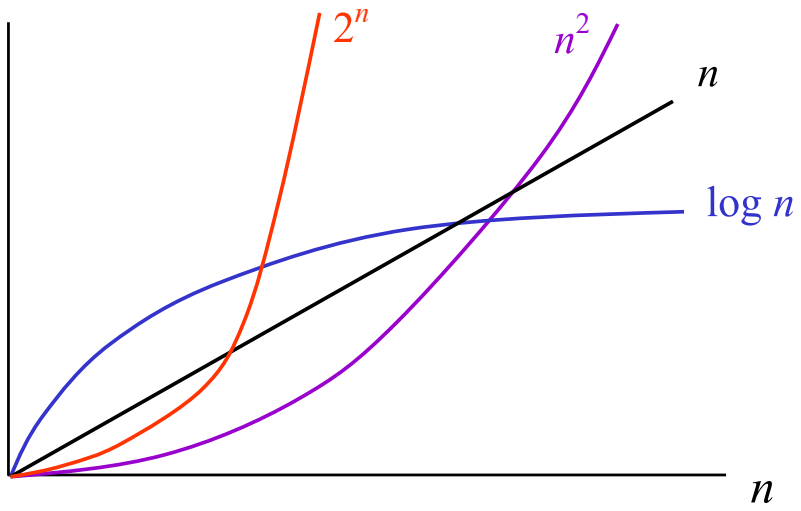
เลือกบรรทัดนี้เป็นคำสั่งตัวแทน

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

การวิเคราะห์ประสิทธิภาพเชิงเวลา

- เขียนโปรแกรมจริง แล้วจับเวลา
  - ต้องเขียนโปรแกรมให้สมบูรณ์
  - เวลาการทำงานขึ้นกับปัจจัยมากมาย
    ภาษา, เครื่อง, ผู้เขียน, ...
- วิเคราะห์อัตราการเติบโตของเวลาการทำงาน
  - เขียนอัลกอริทึม ไม่ต้องลงรายละเอียดมาก
  - เลือกนับเฉพาะคำสั่งตัวแทน
  - ผลที่ได้ใช้เปรียบเทียบได้ดี

อัตราการเติบโตแบบต่าง ๆ



$$\log n \prec n \prec n^2 \prec 2^n$$

ตัวอย่าง : อัตราการเติบโต

```
for (int i = 0; i < n; i++) {
  for (int j = 0; j < n; j++) {
    sum += j;
  }
}
```

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

```
for (int i = 1; i < n; i++) {
  for (int j = 3; j < n-1; j++) {
    sum += j;
  }
}
```

$$\sum_{i=1}^{n-1}\sum_{j=3}^{n-2} 1 = \sum_{j=1}^{n-1}(n-4)$$
$$= (n-1)(n-4)$$
$$= n^2 - 5n + 4$$

ตัวอย่าง : อัตราการเติบโต

n	n^2		$n^2 - 5n + 4$	
10	100		54	
20	400	4	304	5.63
40	1600	4	1404	4.62
80	6400	4	6004	4.28
160	25600	4	24804	4.13
320	102400	4	100804	4.06
640	409600	4	406404	4.03
1280	1638400	4	1632004	4.02
2560	6553600	4	6540804	4.01

เพิ่มขึ้นทีละ 2 เท่า

เพิ่มขึ้นทีละ 4 เท่า

เพิ่มขึ้นทีละ 4 เท่า

ทั้งคู่มีอัตราการเติบโตเท่ากัน
เป็นแบบ quadratic function

โตเร็ว โตช้า

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & f(n)\text{ โตช้ากว่า }g(n)\\ \infty & f(n)\text{ โตเร็วกว่า }g(n)\\ \text{ค่าคงตัว} & f(n)\text{ เติบโตในอัตราเท่ากับ }g(n)\end{cases}$$

$\log n,\quad n,\quad n\log n,\quad n^2,\quad n^3,\quad 2^n,\quad n^n$

โตช้า ⟶ โตเร็ว

$n^2,\quad 0.01n^2,\quad 2n^2-10n,\quad 5n^2+8$

⟵ มีอัตราการเติบโตเท่ากัน ⟶

polylogarithm โตช้ากว่า sublinear

$f(n)=\log n \quad g(n)=\sqrt{n}$

$$\lim_{n\to\infty}\frac{\log n}{\sqrt{n}}=\lim_{n\to\infty}\frac{\ln n}{\ln 10\sqrt{n}}$$

$$=\frac{1}{\ln 10}\lim_{n\to\infty}\frac{\ln n}{\sqrt{n}}$$

$$=\frac{1}{\ln 10}\lim_{n\to\infty}\frac{1/n}{1/\left(2\sqrt{n}\right)}$$

$$=\frac{1}{\ln 10}\lim_{n\to\infty}\frac{2}{\sqrt{n}}$$

$$=0$$

$\log n$ โตช้ากว่า $n^{0.5}$

สามารถแสดงได้ว่า
$(\log n)^c$ โตช้ากว่า n^k
$c, k > 0$

$(\log n)^{1000}$ โตช้ากว่า $n^{0.0001}$

การวิเคราะห์เชิงเส้นกำกับ

- ใช้วิเคราะห์พฤติกรรมการทำงานของอัลกอริทึม เมื่อข้อมูลขาเข้ามีขนาดใหญ่
- ช่วยให้วิเคราะห์ได้ง่ายขึ้น
- นำผลมาเปรียบเทียบได้ง่าย

$$\sum_{i=0}^{n-1}\sum_{j=0}^{n-1} 1 = \sum_{i=0}^{n-1} n \qquad \sum_{i=1}^{n-1}\sum_{j=3}^{n-2} 1 = \sum_{i=1}^{n-1} (n-4)$$

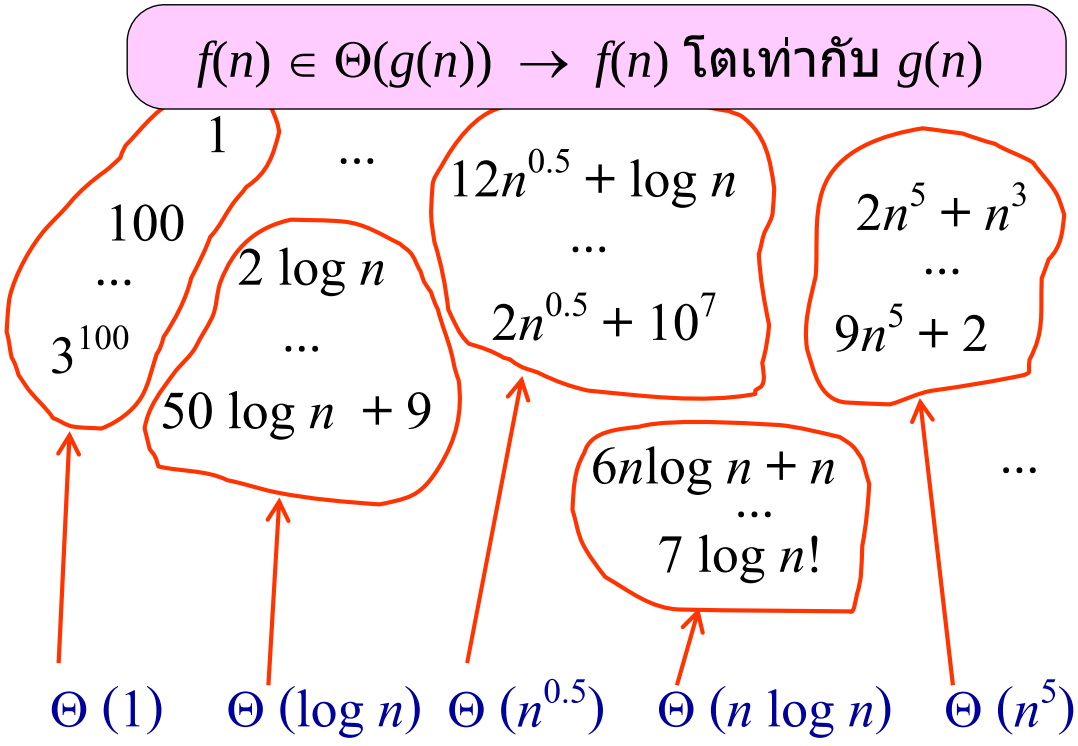
$$= n^2 \qquad = \sum_{i=1}^{n-1} \Theta(n) = \Theta\left(\sum_{j=1}^{n-1} n\right)$$

$$= \Theta(n^2) \quad \text{โตเร็วเท่ากัน} \quad = \Theta(n^2)$$

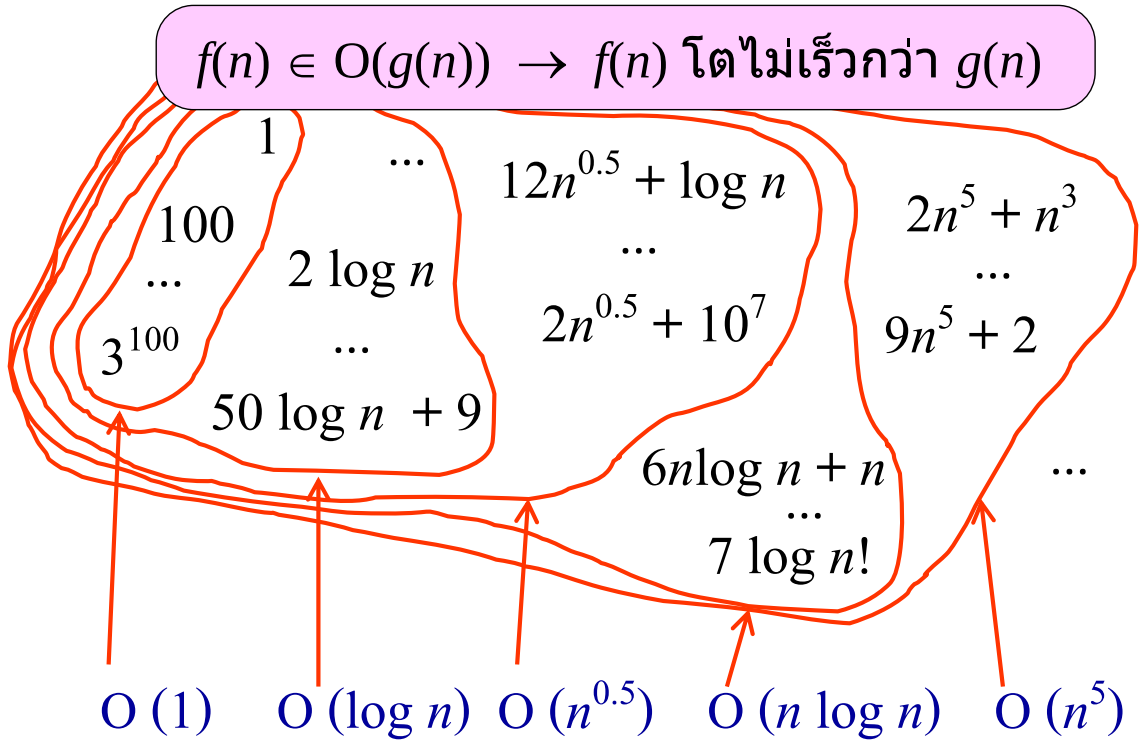
สัญกรณ์เชิงเส้นกำกับ

- $o(g(n))$ คือ เซตของฟังก์ชันที่โตช้ากว่า $g(n)$
- $\omega(g(n))$ คือ " " โตเร็วกว่า $g(n)$
- $\Theta(g(n))$ คือ " " โตเท่ากับ $g(n)$
  - $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$
- $O(g(n))$ คือ " " โตไม่เร็วกว่า $g(n)$
  - $O(g(n)) = o(g(n)) \cup \Theta(g(n))$
- $\Omega(g(n))$ คือ " " โตไม่ช้ากว่า $g(n)$
  - $\Omega(g(n)) = \omega(g(n)) \cup \Theta(g(n))$

$\Theta(g(n))$ คือเซตของฟังก์ชันที่โตเท่ากับ $g(n)$

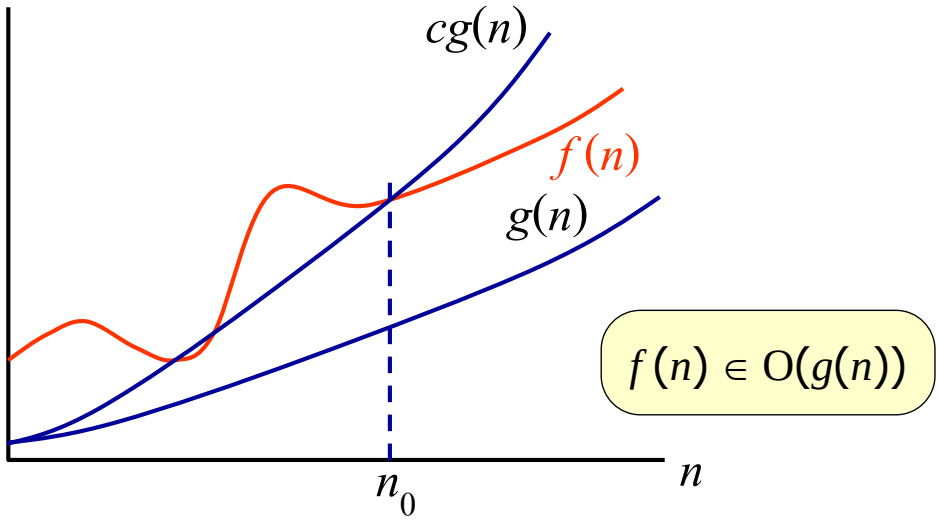


$O(g(n))$ คือเซตของฟังก์ชันที่โตไม่เร็วกว่า $g(n)$



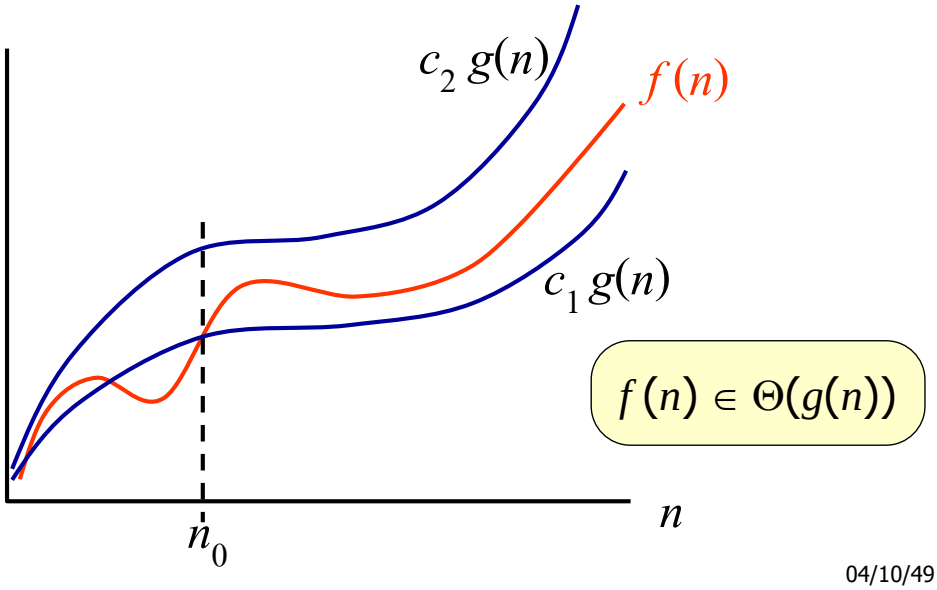
นิยาม $O(g(n))$ อีกแบบ

$O(g(n)) = \{ f(n) \mid$ มีจำนวน $c > 0$ และ $n_0 \geq 0$
ที่ทำให้ $f(n) \leq cg(n)$ เมื่อ $n \geq n_0 \}$



นิยาม $\Theta(g(n))$ อีกแบบ

$\Theta(g(n)) = \{ f(n) \mid$ มีจำนวน $c_1, c_2 > 0$ และ $n_0 \geq 0$
ที่ทำให้ $c_1 g(n) \leq f(n) \leq c_2 g(n)$ เมื่อ $n \geq n_0 \}$



ตัวอย่างที่ 1

$O(g(n)) = \{ f(n) \mid$ มีจำนวน $c > 0$ และ $n_0 \geq 0$
ที่ทำให้ $f(n) \leq cg(n)$ เมื่อ $n \geq n_0 \}$

จงแสดงว่า $15n^2 + 10n \in O(n^2)$

หา c และ n_0 ที่ทำให้ $15n^2 + 10n \leq cn^2$ เมื่อ $n \geq n_0$

หารด้วย n^2 ได้ $15 + 10/n \leq c$

ให้ $c = 16$ จะได้ $15 + 10/n \leq 16$ เมื่อ $n \geq 10$

สรุป $15n^2 + 10n \leq 16n^2$ เมื่อ $n \geq 10$

ดังนั้น $15n^2 + 10n \in O(n^2)$

ตัวอย่างที่ 2

$\Theta(g(n)) = \{ f(n) \mid$ มีจำนวน $c_1, c_2 > 0$ และ $n_0 \geq 0$
ที่ทำให้ $c_1 g(n) \leq f(n) \leq c_2 g(n)$ เมื่อ $n \geq n_0 \}$

จงแสดงว่า $15n^2 + 10n \in \Theta(n^2)$

ได้แสดงแล้วว่า $15n^2 + 10n \leq 16n^2$ เมื่อ $n \geq 10$

หา c_1 และ n_0 ที่ทำให้ $c_1 n^2 \leq 15n^2 + 10n$ เมื่อ $n \geq n_0$

ให้ $c_1 = 1$ จะได้ $n^2 \leq 15n^2 + 10n$ เมื่อ $n \geq 0$

สรุป $n^2 \leq 15n^2 + 10n \leq 16n^2$ เมื่อ $n \geq 10$

ดังนั้น $15n^2 + 10n \in \Theta(n^2)$

ตัวอย่างที่ 3

จงแสดงว่า $\sum_{i=1}^{n}\left(\frac{i}{2}\right)^k \in \mathrm{O}\left(n^{k+1}\right)$

$$\sum_{i=1}^{n}\left(\frac{i}{2}\right)^k < \sum_{i=1}^{n}\left(\frac{n}{2}\right)^k$$

$$< \sum_{i=1}^{n} n^k$$

$$= n^{k+1}$$

$$\in \mathrm{O}\left(n^{k+1}\right)$$

ตัวอย่างที่ 4

จงแสดงว่า $\sum_{h=0}^{\lfloor \log_2 n \rfloor}\left(\frac{n}{2^h}h\right) = \mathrm{O}(n)$

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor}\left(\frac{n}{2^h}h\right) = n\sum_{h=0}^{\lfloor \log_2 n \rfloor}\left(\frac{h}{2^h}\right)$$

$$< n\sum_{h=0}^{\infty}\left(\frac{h}{2^h}\right)$$

$$= 2n \quad \in \mathrm{O}(n)$$

ตัวอย่างที่ 5 : $\log n! \in \Theta(n \log n)$

ขอบเขตบน

$$\begin{aligned} n! &= n \times (n\text{-}1) \times \ldots \times 2 \times 1 \\ &\leq n \times n \times \ldots \times n \times n = n^n \\ \log n! &\leq \log n^n = n \log n \quad \text{เมื่อ } n \geq 1 \end{aligned}$$

ขอบเขตล่าง

$$\begin{aligned} n! &= n \times (n\text{-}1) \times \ldots \times (n/2) \times (n/2\text{-}1) \times \ldots \times 2 \times 1 \\ &\geq n/2 \times n/2 \times \ldots \times n/2 \times 1 \times \ldots \times 1 \times 1 \\ &\geq (n/2)^{n/2} \\ \log n! &\geq (n/2) \log (n/2) \\ &= (n/2) \log n - (n/2) \\ &\geq 0.4n \log n \quad \text{เมื่อ } n \geq 10^5 \end{aligned}$$

$$0.4n \log n \leq \log n! \leq n \log n \quad \text{เมื่อ } n \geq 10^5$$

$$\log n! \in \Theta(n \log n)$$

อัตราการเติบโตที่พบบ่อย

- 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 : $\Theta(c^n)$, $c > 1$

การเขียนฟังก์ชันในรูปของ O, Θ แบบง่าย ๆ

- ผลบวกของพจน์หลายพจน์ เลือกพจน์ที่โตเร็วสุด
- ข้อสังเกต
  - $cg(n) = \Theta(g(n))$ เมื่อ c เป็นค่าคงตัว
  - $\log_a n = \Theta(\log_b n)$ เพราะ $\log_a n = (\log_a b) \log_b n$
  - . $\sum \Theta(t(n)) = \Theta\left(\sum t(n)\right)$
- เช่น
  - $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0 = \Theta(n^k)$
  - $0.001n^3 + 7000n^2 - 11 = \Theta(n^3)$
  - $\log_2 n^{10} = 10(\log_2 n) = \Theta(\log n)$
  - $\cdot \sum_{i=1}^{n} \Theta(i) = \Theta\left(\sum_{i=1}^{n} i\right) = \Theta\left(n(n+1)/2\right) = \Theta(n^2)$

กลับมาวิเคราะห์ ArrayCollection กัน

- constructor
  - เป็นการจองอาเรย์จำนวนช่องตามที่ผู้ใช้กำหนด จึงใช้เวลาแปรตามขนาด
  - ใช้เวลา Θ(c)
- isEmpty และ size
  - ทำแค่ 1 คำสั่ง : ใช้เวลา Θ(1)

```
public class ArrayCollection implements Collection {
  private Object[] elementData;
  private int size;
  public ArrayCollection(int c) {
    elementData = new Object[c];
  }
  public boolean isEmpty() { return size == 0; }
  public int size() { return size; }
  ...
```

ArrayCollecton.add

- ถ้าไม่ต้องขยายอาเรย์ ก็ใช้เวลาแค่ $\Theta(1)$
- ให้ n แทนจำนวนข้อมูลในคอลเลกชัน
- ถ้าข้อมูลเต็มอาเรย์ เกิดการขยาย จึงใช้เวลา $\Theta(n)$

```
public void add(Object e) {
  if(e == null) throw new IllegalArgumentException();
  ensureCapacity(size+1);
  elementData[size++] = e;
}
private void ensureCapacity(int capacity) {
  if (capacity > elementData.length) {
    int s = Math.max(capacity, 2*elementData.length);
    Object[] arr = new Object[s];
    for(int i=0; i<elementData.length; i++)
      arr[i] = elementData[i];
    elementData = newA;
  }
}
```

$\Theta(n)$

ตอนที่เต็ม n = ขนาดของอาเรย์

ArrayCollection.contains

- contains เรียก indexOf เพื่อค้นหาข้อมูลแบบลำดับ
- ทำการค้นด้วยวงวน for หมุนไม่เกิน n รอบ
- ถ้าโชคดีก็พบที่ช่องที่ 0 โชคร้ายก็พบช่องท้าย ๆ
- จึงใช้เวลาการค้นเป็น $O(n)$

```
public boolean contains(Object e) {
  return indexOf(e) != -1;
}
private int indexOf(Object e) {
  for(int i=0; i<size; i++)
    if (elementData[i].equals(e)) return i;
  return -1;
}
```

$O(n)$

ArrayCollection.remove

- remove เรียก indexOf เพื่อค้นหาข้อมูลแบบลำดับ
- ทำการค้นด้วยวงวน for ใช้เวลา $O(n)$
- ทำการลบใช้เวลา $\Theta(1)$
- remove ใช้เวลาเป็น $O(n)$

```
public void remove(Object e) {
  int i = indexOf(e);
  if (i != -1) {
    elementData[i] = elementData[--size];
    elementData[size] = null;
  }
}
```

$O(n)$

$\Theta(1)$

สรุป

- ใช้จำนวนครั้งที่คำสั่งตัวแทนทำงานแทนเวลา
- หาความสัมพันธ์ของจำนวนครั้งที่คำสั่งตัวแทนทำงานกับจำนวนข้อมูล
- เขียนฟังก์ชันในรูปของสัญกรณ์เชิงเส้นกำกับ
- ใช้ O เพื่อแสดงขอบเขตบนของเวลา
- ใช้ Θ เมื่อการทำงานมีขอบเขตบนและล่างเท่ากัน