อัลกอริทึม

ในศาสตร์แห่งวิทยาการคอมพิวเตอร์จะเกี่ยวข้องกับขั้นตอนวิธีการแก้ปัญหา หรือ **“Algorithm”**

5.1 อัลกอริทึม

คำว่า อัลกอริทึม มาจากชื่อของนักคณิตศาสตร์ชาวเปอร์เซีย ที่ชื่อว่า **Abu Ja ’far Mohammed Ibin Musa Al-Khowarizmi** (ค.ศ **780-850**)

นิยาม *อัลกอริทึม* (Algorithm) หมายถึง เซตของชุดคำสั่งที่ถูกต้องและมีขนาดจำกัด เพื่อการคำนวณหรือแก้ปัญหาใดปัญหาหนึ่งโดยเฉพาะ

ตัวอย่าง

ขั้นตอนวิธีการหาค่ามากที่สุดของชุดตัวเลขจำนวนจำกัดชุดใดชุดหนึ่ง เรียกลำดับขั้นตอนอัลกอริทึมนี้ โดย

- เรียงลำดับขั้นตอนการแก้ปัญหาอย่างถูกต้อง
- กำหนดการดำเนินการพื้นฐาน เช่น $+, -, \times, \div$, เปรียบเทียบ
- วิเคราะห์เวลาที่ใช้ในการแก้ปัญหาดังกล่าว

คุณสมบัติของอัลกอริทึม

Input : อัลกอลิทึมจะประกอบด้วยเซตของข้อมูลเข้า **1** ชุด

Output : จากเซตของข้อมูลเข้า หากดำเนินการตามลำดับขั้นตอนในอัลกอลิทึมด้วยชุดข้อมูลนำเข้าจะได้เซตคำตอบ **1** ชุดที่เป็นผลลัพธ์

Correctness : อัลกอลิทึมควรให้ผลลัพธ์ที่ถูกต้องสำหรับทุกชุดของข้อมูลเข้าที่สอดคล้องกัน

Finiteness : อัลกอลิทึมควรให้ผลลัพธ์หลังจากดำเนินการตามลำดับขั้นตอนที่จำกัดในอัลกอลิทึม

Effectiveness: การดำเนินงานในแต่ละลำดับขั้นของอัลกอลิทึมเป็นไปอย่างถูกต้องและทำงานภายในระยะเวลาจำกัด

Generality: อัลกอลิทึมควรดำเนินงานอย่างถูกต้องกับทุกชุดข้อมูลเข้าของปัญหาตามรูปแบบที่ออกแบบ

ตัวอย่าง

การค้นหาแบบลำดับ (**Sequential Search** หรือ **Linear Search**)

ค้นหาตำแหน่งของข้อมูลที่ต้องการใน **list**ข้อมูลทีละตัวตามลำดับจากต้น

- ค้นหาข้อมูล x ใน list ข้อมูล $[a_1, a_2, a_3, \ldots, a_n]$
- เปรียบเทียบ x กับ a_1 หาก x เท่ากับ a_1 ตำแหน่งที่พบคือตำแหน่งที่ 1
- หาก x ไม่เท่ากับ a_1 ให้เปรียบเทียบ x เท่ากับ a_2 หาก x เท่ากับ a_2 ตำแหน่งที่พบคือตำแหน่งที่ 2
- หาก x ไม่เท่ากับ a_2 ทำเช่นนี้ต่อไปเรื่อยๆ จนครบสมาชิกทุกตัวใน list หาก x ไม่เท่ากับสมาชิกตัวใดหนึ่งใน list ตำแหน่งที่พบคือตำแหน่งที่ 0

Pseudocode:

Algorithm 1: Linear Search

Input: x : *integer*, $[a_1, \ldots, a_n]$: list of distinct integers
Output: Index i s.t. $x = a_i$ or 0 if x is not in the list.

```
i := 1;
while i ≤ n and x ≠ a_i do
    i := i + 1;
if i ≤ n then result := i else result := 0;
return result;
```

ตัวอย่าง

การค้นหาแบบ **Binary**

การค้นแบบ **Binary**

- สมมติ list ของข้อมูลเข้าได้รับการจัดเรียงลำดับข้อมูลจากน้อยไปมาก และข้อมูลที่ต้องการค้นหาจะต้องมีอยู่ใน list
- ขั้นตอนวิธีนี้เริ่มจากทำการเปรียบเทียบข้อมูลที่ต้องการค้นหากับข้อมูลกึ่งกลาง list
    - หากข้อมูลที่ต้องการค้นหาน้อยกว่าข้อมูลกึ่งกลาง list ช่วงการค้นหาจะอยู่ที่ครึ่งแรกของ list (ไม่รวมข้อมูลตรงกลาง)
    - มิฉะนั้น ช่วงการค้นหาจะอยู่ที่ครึ่งหลังของ list (รวมข้อมูลกึ่งกลาง)
- ดำเนินการตามลำดับขั้นตอนนี้ซ้ำจนกว่าจำนวนข้อมูลใน list เหลือเพียงตัวเดียว
    - ถ้าข้อมูลที่ต้องการค้นหาเท่ากับข้อมูลใน list แล้ว แสดงตำแหน่งที่ตั้งของข้อมูลใน list
    - มิฉะนั้น แสดงตำแหน่งที่ตั้งเป็น 0 (ไม่พบข้อมูล)

Pseudocode:

Algorithm 2: Binary Search

Input: x : *integer*, $[a_1, \ldots, a_n]$: strictly increasing list of integers
Output: Index i s.t. $x = a_i$ or 0 if x is not in the list.
$i := 1$; // i is the left endpoint of the interval
$j := n$; // j is the right endpoint of the interval
while $i < j$ **do**
    $m := \lfloor (i + j)/2 \rfloor$;
    if $x > a_m$ **then** $i := m + 1$ **else** $j := m$;
if $x = a_i$ **then** *result* $:= i$ **else** *result* $:= 0$;
return *result*;

ตัวอย่าง

จงหาค่า **19** จาก **list** ของข้อมูลต่อไปนี้ **1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22**

5.2 การเติบโตของฟังก์ชัน

กำหนดให้ฟังก์ชัน $\mathbf{f:}\aleph \rightarrow \Re$ หรือ $\mathbf{f:}\Re \rightarrow \Re$

จะทำการวิเคราะห์การเติบโตของฟังก์ชัน โดย

- เปรียบเทียบฟังก์ชัน 2 ฟังก์ชัน
- เปรียบเทียบประสิทธิภาพการแก้ปัญหาเดียวกันจาก 2 ขั้นตอนวิธี

นิยาม กำหนดให้ ฟังก์ชัน **f** และ **g** แทนด้วย $\mathbf{f, g :}\ \Re \rightarrow \Re$ กล่าวว่า **f** มีอัตราการเติบโตเป็นสัญกรณ์ Big-O O(g) ถ้า มีค่าคงที่ **C** และ **k** ซึ่งทำให้เกิดความสัมพันธ์ $\forall\ x > k, |f(x)| \leq C|g(x)|$

หมายเหตุ f เป็น Big-O ของ g หรือ f(x) = O(g(x))

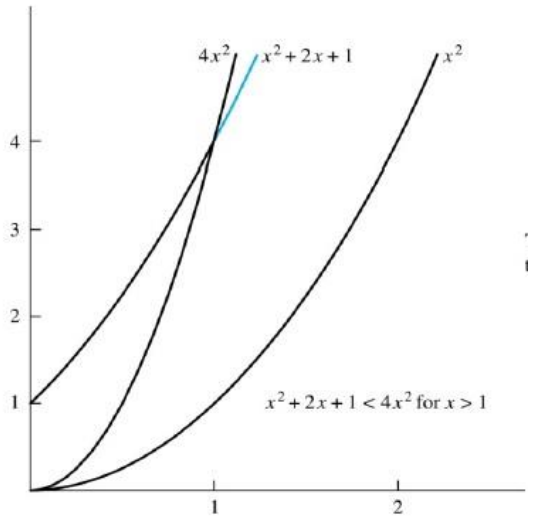
O(g(x)) เป็น **class** ที่ประกอบขึ้นจาก ทุกฟังก์ชัน **f** ที่ทำให้ความสัมพันธ์ $|f(x)| \leq C|g(x)|$ สำหรับทุกค่า $x > k$ เป็นจริง หรือ กล่าวว่า $f(x) \in O(g(x))$

ตัวอย่าง

กำหนดให้ $f(x) = x^2+2x+1$ และ $g(x) = x^2$

จงแสดงว่า ฟังก์ชัน $f(x) = x^2+2x+1$ มีอัตราการเติบโตเป็น $O(x^2)$

วิธีทำ



คุณสมบัติของสัญกรณ์ Big-O

1. ถ้า f(x) = O(g(x)) และ g(x) = O(f(x)) แล้ว ฟังก์ชัน f และ g อยู่ในลำดับเดียวกัน
2. ถ้า f(x) = O(g(x)) และ $h(x) \geq g(x)$ สำหรับทุกจำนวนจริง $x > 0$ แล้ว f(x) = O(h(x))

3. Big-O แทนขอบเขตบนที่ครอบคลุมการเติบโตของฟังก์ชัน เช่น $f(x) = x^2+3x$ อัตราการเติบโตเป็นสัญกรณ์ $O(x^2)$ หรือ $O(x^3)$
4. Big-O ที่ถูกเลือกเป็นคำตอบจะเป็นฟังก์ชันที่ง่ายกระชับและยังคงครอบคลุมการเติบโตของฟังก์ชัน f อยู่ เช่น $f(x) = x^2+3x$ อัตราการเติบโตเป็นสัญกรณ์ $O(x^2)$ หรือ $O(x^3)$ คำตอบที่เลือกตอบ คือ $f(x) = O(x^2)$

ตัวอย่าง

จงแสดงว่า

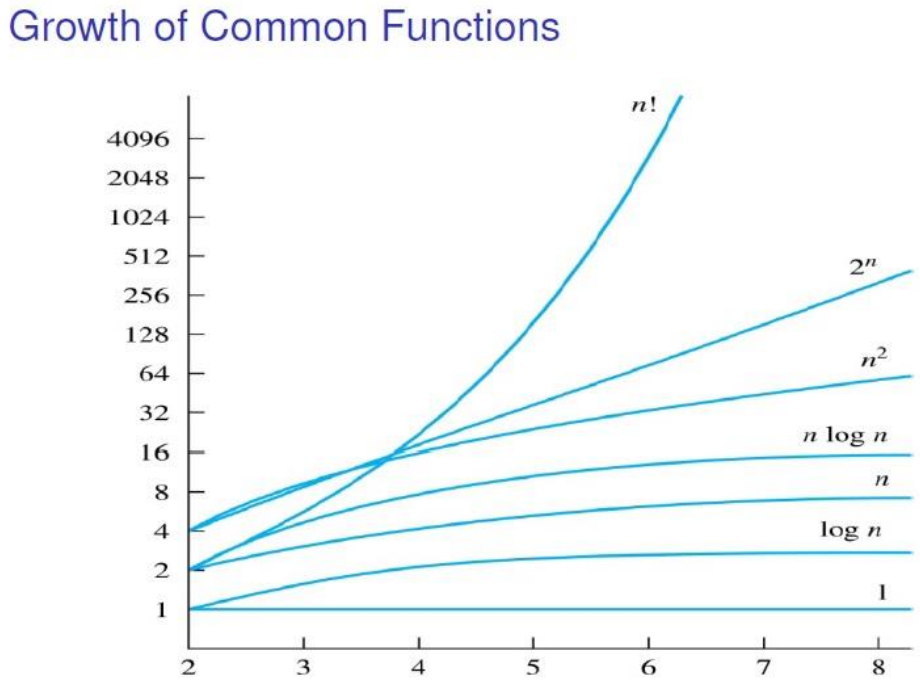
$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0 = O(x^n)$

$1+2+3+ \ldots + n = O(n^2)$

$n! = 1 \times 2 \times 3 \times \ldots \times n = O(n^n)$

$\log(n!) = O(n \log n)$

วิธีทำ



5.3 ความซับซ้อนของอัลกอลิทึม

หากกำหนดอัลกอลิทึมใดหนึ่งและเซตข้อมูลเข้าขนาด **n** ค่าหนึ่งค่าใด พบว่าอัลกอลิทึมนั้นมีประสิทธิภาพในการแก้ปัญหามากน้อยเพียงใด ขึ้นกับ

- ระยะเวลาที่อัลกอลิทึมใช้ในการแก้ปัญหา
- ขนาดหน่วยความจำที่เครื่องคอมพิวเตอร์ใช้ในการแก้ปัญหา

ระยะเวลาการทำงานของอัลกอลิทึมวัดจากจำนวนการดำเนินการตามขั้นตอนที่กำหนดในอัลกอลิทึม และใช้สัญกรณ์ **Big-O** ประมาณค่าเวลาซึ่งไม่เกินขอบเขตบนของฟังก์ชันจำนวนการดำเนินการ

การเปรียบเทียบประสิทธิภาพของอัลกอลิทึม กระทำโดยเปรียบเทียบอัลกอลิทึมที่ต่างกันเชิงเวลาในการแก้ปัญหาเดียวกัน

การประสิทธิภาพเชิงเวลาของอัลกอลิทึม กระทำได้ 2 รูปแบบ คือ

- ประสิทธิภาพเชิงเวลาแบบเลวสุด (worst-case time complexity)
- ประสิทธิภาพเชิงเวลาแบบถัวเฉลี่ย (average-case time complexity)

ตัวอย่าง

การหาประสิทธิภาพเชิงเวลา

- ***แบบเลวสุดของอัลกอลิทึมการค้นหาแบบลำดับ***

Pseudocode:

Algorithm 1: Linear Search

Input: x : *integer*, $[a_1, \ldots, a_n]$: list of distinct integers
Output: Index i s.t. $x = a_i$ or 0 if x is not in the list.

```
i := 1;
while i ≤ n and x ≠ a_i do
    i := i + 1;
if i ≤ n then result := i else result := 0;
return result;
```

จำนวนการดำเนินการเปรียบเทียบของอัลกอลิทึมข้างต้น มีค่าเป็น **2n+2** ครั้ง

ดังนั้น สัญกรณ์ **Big-O** คือ **O(n)**

- ***แบบถัวเฉลี่ย ของอัลกอลิทึมการค้นหาแบบลำดับ***

โอกาสที่จะพบข้อมูลที่ต้องการค้นหาใน **list** ข้อมูลจำนวน **n** ตัว มีค่าเท่ากับ **1/n**

การพบข้อมูลที่ต้องการเป็นลำดับที่ **1, 2, 3, …, n** ใน **list** จะต้องทำการดำเนินการเปรียบเทียบเท่ากับ

อัลกอลิทึมการค้นหาแบบลำดับมีประสิทธิภาพเชิงเวลาประสิทธิภาพเชิงเวลา เท่ากับ

ดังนั้น สัญกรณ์ **Big-O** คือ **O(n)**

ตัวอย่าง

การหาประสิทธิภาพเชิงเวลา

- แบบเลวสุดของอัลกอลิทึมการค้นหาแบบ Binary

Pseudocode:

Algorithm 2: Binary Search

Input: x : *integer*, $[a_1, \ldots, a_n]$: strictly increasing list of integers
Output: Index i s.t. $x = a_i$ or 0 if x is not in the list.
$i := 1$; // i is the left endpoint of the interval
$j := n$; // j is the right endpoint of the interval
while $i < j$ **do**
  $m := \lfloor (i + j)/2 \rfloor$;
  if $x > a_m$ **then** $i := m + 1$ **else** $j := m$;
if $x = a_i$ **then** *result* $:= i$ **else** *result* $:= 0$;
return *result*;

สมมติให้ จำนวนข้อมูลใน **list** มีขนาด $\mathbf{n = 2^k}$

จำนวนการดำเนินการเปรียบเทียบของอัลกอลิทึมข้างต้น มีค่าเป็น ครั้ง

ดังนั้น สัญกรณ์ **Big-O** คือ