Algorithms

Asymptotic Notation

ปรับปรุงจาก slide รายวิชา 204451 ของอ.เบญจมาศ ปัญญางาม

Algorithm คืออะไร

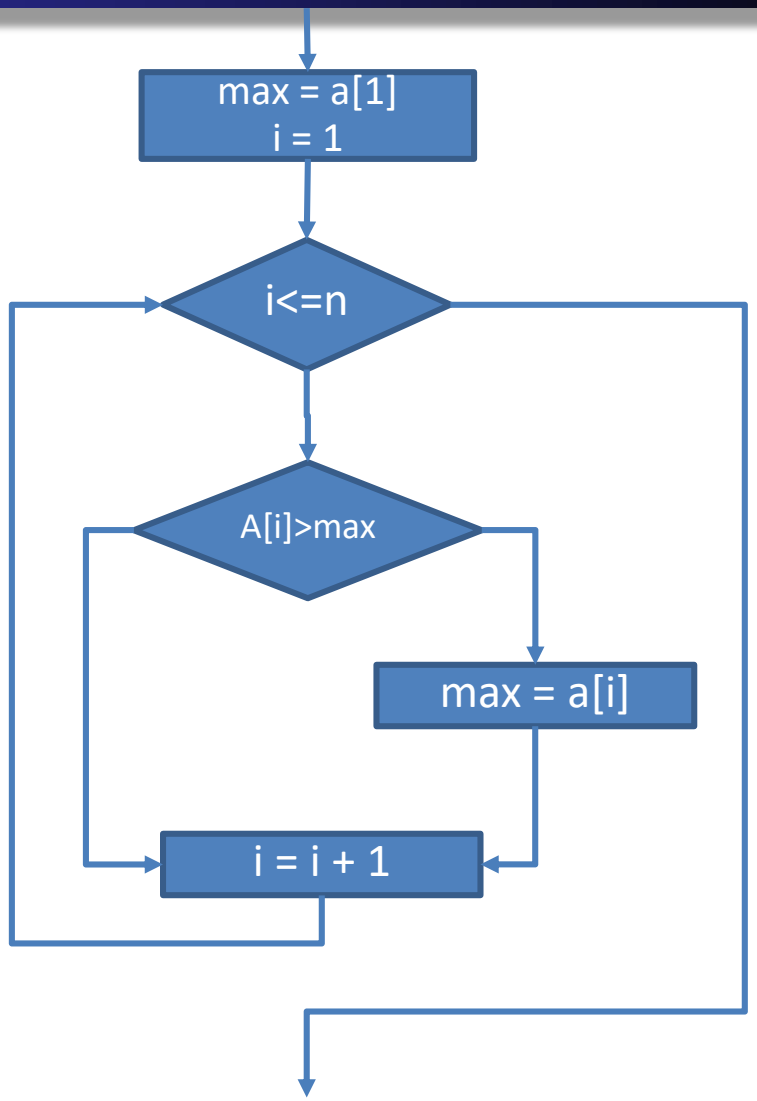
- Algorithm คือ ขั้นตอนวิธีในการแก้ปัญหา
- ปัญหาที่เราสนใจคืออะไร
  - ปัญหาเชิงคำนวณที่หาคำตอบได้ทางคณิตศาสตร์ (Computational problem)

ตัวอย่าง

- **ปัญหา** การหาจำนวนที่มีค่ามากที่สุดจากข้อมูลชุดที่ไม่ได้เรียงลำดับ
  - เราต้องตรวจสอบข้อมูลทุกจำนวน
- ขั้นตอนในการแก้ปัญหา
  - ให้ข้อมูลตัวแรกเป็นข้อมูลตัวที่มีค่ามากที่สุด
  - ดูข้อมูลทีละตัวจากข้อมูลชุด หากข้อมูลตัวใหม่มีค่ามากกว่าข้อมูลที่เราเคยเก็บไว้ให้ทำการเปลี่ยนข้อมูลที่เราเก็บเป็นข้อมูลตัวใหม่
  - สุดท้ายจะได้ข้อมูลตัวที่มากที่สุด

ตัวอย่าง

```
findMax(int A[1, .., n])
{
        max = A[1]
        for(i=2 to n)
                if(A[i]>max) max=A[i]
        return max
}
```



Efficient Algorithm

- คุณสมบัติที่ต้องการของอัลกอริทึม
  - **ความถูกต้อง** (Correctness)
    - Correct: สำหรับทุกๆ ตัวอย่างของปัญหา (instance) ที่เป็นไปได้ จะหยุดด้วยคำตอบที่ถูกต้อง
    - Incorrect: ไม่หยุดการทำงานในบาง instance หรือหยุดด้วยคำตอบที่ผิด
  - **มีประสิทธิภาพ** (Efficient Algorithm)
    - Memory: Space Complexity
    - Running Time: Time Complexity

Time Complexity

- พิจารณา Running time
  - นับจำนวน Operation พื้นฐานที่ถูกใช้ในการแก้ปัญหา ซึ่งขึ้นกับ**ขนาด**ของข้อมูล
- กำหนดให้ T(n) เป็นฟังก์ชันแทนเวลาในการทำงานของอัลกอริทึมเมื่อรับ input ขนาด n
- n คือขนาดของ ตัวอย่างปัญหา (Instance)
  - เช่นรายการ {100,5,3,8,4,34,9,454} มีขนาด 8 ตัว

Time Complexity

<u>การนับเวลาการทำงาน</u>

Primitive Operations (+ - * / = การอ้างอิงอาร์เรย์)

นับว่าใช้เวลาการทำงานเป็นค่าคงที่ค่าหนึ่ง

If ..(t_1).. then ..(t_2).. Else ..(t_3).. : t_1, t_2, t_3

นับเป็น t โดย $t_1 + \min(t_2,t_3) \leq t \leq t_1 + \max(t_2,t_3)$

Loop : for , while

ขึ้นกับขนาด input หรือ จำนวนรอบที่วนทำงาน

Three cases of Analysis

- Best Case : เป็นการวิเคราะห์กรณีที่ให้ผลลัพธ์โดยใช้เวลาในการทำงานเร็วที่สุด
- Worst Case : เป็นการวิเคราะห์กรณีที่ให้ผลลัพธ์โดยใช้เวลาในการทำงานช้าที่สุด
- Average Case : เป็นการวิเคราะห์เวลาการทำงานโดยเฉลี่ยสำหรับผลลัพธ์ที่เป็นไปได้ทุกแบบ

Math Review

For integer a and b, a $\leq$ b,

Constant series: $\sum_{i=a}^{b} 1 = b - a + 1$

For n$\geq$0,

Linear Series: $\sum_{i=1}^{n} i = 1 + 2 + , , , + n = \frac{n(n+1)}{2}$

Quadratic Series: $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$

Cubic Series: $\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$

Math Review

Logarithm

$x = \log_b a \rightarrow a = b^x$

$\ln a = \log_e a$

$\log_2 a = \lg a$

$\log^2 a = (\log a)^2$

$\log \log a = \log (\log a)$

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

$$\log_c (ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b (1/a) = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

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

Analyzing the time complexity

```
findMax(int A[1 .., n])
{
   max = A[1]                          c1
   for(i=2 to n)                       c2
      if(A[i]>max) max=A[i]            c3
   return max                          c4
}
```

กำหนดให้ c_k เป็นเวลาในการทำงานของคำสั่งในบรรทัดที่ k

T(n) เป็นเวลาในการทำงานของโปรแกรมเมื่อมีข้อมูลเข้าขนาด n

โปรแกรม findMax มีเวลาในการทำงานทั้งหมดเป็น

$$T(n)=c_1+c_2(n)+c_3(n-1)+c_4$$

Analyzing the time complexity

Insertion_Sort_Algorithm(A)	Constant	Times
For j = 2 to length(A)	c_1	n
key = A[j]	c_2	n-1
//Insert A[j] into the sorted sequence A[1..j-1]		
i = j -1	c_4	n-1
while i > 0 and A[i] > key	c_5	$\sum_{j=2}^{n} t_j$
A[i +1] =A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
i = i -1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
A[i+1] = key	c_8	n-1

t_j เป็นจำนวนครั้งที่ While loop ถูกเรียกในรอบที่ j

Analyzing the time complexity

Analyzing algorithm : Best Case

- Best Case : data is already sorted $\therefore t_j = 1, \ \forall \ j$

$$T(n)=c_1 n+ (c_2+c_4) (n-1) + c_5 \sum_{j=2}^{n} t_j + (c_6+c_7) \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$$

$$T(n) = c_1 n+ (c_2+c_4) (n-1)+c_5(n-1)+ c_8(n-1)$$

$$= (c_1+c_2+c_4+c_5+c_8)n - (c_2+c_4+c_5+ c_8)$$

Running time -> can be express as an+b

-> a linear function of n -> O(n)

Analyzing the time complexity

Analyzing algorithm : Worst Case

- Worst Case : reverse sorted order $\therefore t_j = j, \ \forall \ j$

$$T(n) = c_1 n + (c_2 + c_4)(n-1) + c_5 \sum_{j=2}^{n} t_j + (c_6 + c_7) \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$$

$$T(n) = C_1 n + (C_2 + C_4)(n-1) + C_5 \left[\frac{n(n+1)}{2} - 1 \right] + (C_6 + C_7) \left[\frac{n(n-1)}{2} \right] + C_8 (n-1)$$

$$= \left[\frac{C_5}{2} + \frac{C_6}{2} + \frac{C_7}{2} \right] n^2 + \left[C_1 + C_2 + C_4 + \frac{C_5}{2} - \frac{C_6}{2} - \frac{C_7}{2} + C_8 \right] n - (C_2 + C_4 + C_5 + C_8)$$

Running time -> can be express as $an^2 + bn + c \approx O(n^2)$

$$\sum_{j=2}^{n} t_j = \frac{n(n+1)}{2} - 1 \quad , \quad \sum_{j=2}^{n} (t_j - 1) = \frac{(n-1)n}{2}$$

⇩

a quadratic function of n

Analyzing the time complexity

ตัวอย่าง while loop : A linear (sequential) search

Input: Array A[1..n] and key. x

Output: Position of key (0 , if not found)

linear_search(x, A)

```
1.  i = n
2.  while ( i > 0 and x ≠ A[ i ] )
3.      i = i - 1
4.  return(i)
```

$\sum_{i=n}^{1} 1$ =n รอบ

T(n) = ?

Best Case: O(1)
Worst Case : O(n)

Analyzing the time complexity

Case : for loop

```
1.  for i = 1 to n
2.      A[ i ]=0
3.  for i = 1 to n
4.      for j = 1 to n
5.          A[ i ] = A[ i ] + A[ j ]
```

$\sum_{i=1}^{n} 1 = n$ รอบ

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

$T(n) = c_1(n+1) + c_2 n + c_3(n+1) + c_4 \sum_{i=1}^{n}(n+1) + c_5 \sum_{i=1}^{n}(n)$

$= c_1 n + c_1 + c_2 n + c_3 n + c_3 + c_4 n^2 + c_4 n + c_5 n^2$

$= (c_4 + c_5)n^2 + (c_1 + c_2 + c_3 + c_4)n + (c_1 + c_3)$

$T(n) = an^2 + bn + c \approx O(n^2)$

ตัวอย่าง Binary Search

While loop : Binary search

Input: Sorted array A[1..n] and a searched key, x.
Output: Position of key (0 , if not found).

Binary_search(x, A)

```
i = 1, j = n
while (i < j)
      m = ⌊(i + j)/2⌋
      if x = A[m] then return (m)
      else if x > A[m] then  i = m + 1
                  else j = m -1
return(0)
```

จำนวนข้อมูลทั้งหมดใน array มีค่าเท่ากับ n และ k แทนความสูงของ binary tree

$2^k-1 = n$

$\log_2 2^k = \log (n+1)$

$k = \log_2 (n+1)$

จำนวนการเปรียบเทียบทั้งหมดมีค่าไม่เกิน k+1 ครั้ง

T(n)= ?

Worst Case : O(log n)

Analyzing the time complexity

แบบฝึกหัด Selection sort algorithm to solve the sorting problem

Input: data array A[1..n]

Output: sorted data array A

```
Selection_Sort(A)
for i = n downto 2
   j=1
   for (k = 2; k <= i; k++)
       if (A[j] < A[k]  ) then j = k;
  swap(A,i,j)
```

T(n) = ?

Analyzing the time complexity

แบบฝึกหัด Function1_1

```
int Function1_1(int n)
{   int  sum,i,j,k;
/*1*/   sum=0;
/*2*/   for (i = n; i>0; i = i/2)
/*3*/     for (j=1; j<=n; j++)
/*4*/           sum++;
/*5*/   return sum;
}
```

T(n) = ?

Analyzing the time complexity

$$\sum_{k=a}^{b} 1 = b - a + 1$$

แบบฝึกหัด Function1_2

```
int Function1_2( const int A[ ], int N )
 { int sum, MaxSum, i, j, k;
/* 1*/ MaxSum = 0;
/* 2*/ for( i = 1; i <= N; i++ )
/* 3*/       sum = 0;
/* 4*/       for( j = i; j <= N; j++ ) {
/* 5*/           sum += A[ j ];
/* 6*/           if (sum>MaxSum ) then MaxSum = sum; }
/* 7*/ return MaxSum;
}
```

T(n) = ?