DATA STRUCTURES AND ALGORITHMS

Asymptotic Notations

The Process of Algorithm Development

- Design
  - divide&conquer, greedy, dynamic programming
- Validation
  - check whether it is correct
- Analysis
  - determine the properties of algorithm
- Implementation
- Testing
  - check whether it works for all possible cases

Analysis of Algorithm

- Analysis investigates
  - What are the properties of the algorithm?
    - in terms of time and space
  - How good is the algorithm ?
    - according to its properties
  - How it compares with others?
    - not always exact
  - Is it the best that can be done?
    - difficult !

Mathematical Background

- Assume the running times of two algorithms are calculated:

  For input size N

  Running time of Algorithm A = $T_A(N) = 1000\ N$

  Running time of Algorithm B = $T_B(N) = N^2$

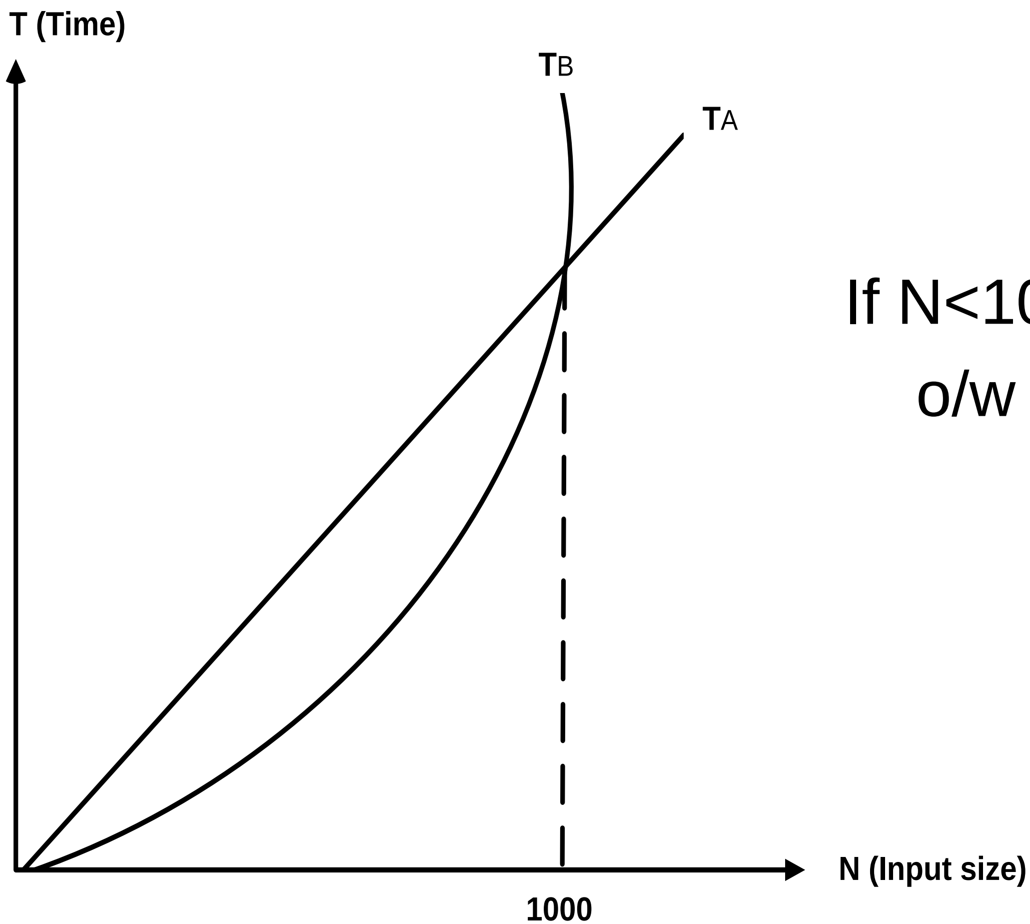
  Which one is faster ?

Mathematical Background

If the unit of running time of algorithms A and B is µsec

N	T_A	T_B
10	10^{-2} sec	10^{-4} sec
100	10^{-1} sec	10^{-2} sec
1000	1 sec	1 sec
10000	10 sec	100 sec
100000	100 sec	10000 sec

So which algorithm is faster ?



If N<1000 $T_A(N) > T_B(N)$

o/w $T_B(N) > T_A(N)$

Compare their relative growth ?

Mathematical Background

- Is it always possible to have definite results?

NO !

The running times of algorithms can change because of the platform, the properties of the computer, etc.

We use asymptotic notations (O, Ω, θ, o)

- compare relative growth
  - No constants
  - No lower order terms
- compare only algorithms

Big Oh Notation (O)

Provides an "upper bound" for the function *f*

- **Definition :**

T(N) = O (f(N)) if there are positive constants *c* and n_0 such that

$$T(N) \le cf(N) \text{ when } N \ge n_0$$

- T(N) grows no faster than f(N)
- growth rate of T(N) is less than or equal to growth rate of f(N) for large N
- f(N) is an upper bound on T(N)
  - not fully correct !

Big Oh Notation (O)

- **Analysis of Algorithm A**

$$T_A(N) = 1000\,N = O(N)$$

1000 N ≤ cN $\forall N \geq n_0$

if c= 2000 and n_0 = 1 for all N

$T_A(N) = 1000\,N = O(N)$ is right

Examples

- $7n+5 = O(n)$
  for $c=8$ and $n_0 = 5$
  $7n+5 \leq 8n \qquad n>5 = n_0$

- $7n+5 = O(n^2)$
  for $c=7$ and $n_0=2$
  $7n+5 \leq 7n^2 \qquad n \geq n_0$

- $7n^2+3n = O(n)$?

Advantages of O Notation

- It is possible to compare of two algorithms with running times
- Constants can be ignored.
  - Units are not important

$$O(7n^2) = O(n^2)$$

- Lower order terms are ignored
  - Compare relative growth only

$$O(n^3+7n^2+3) = O(n^3)$$

Big Oh Notation (O)

Running Times of Algorithm A and B

$T_A(N) = 1000\,N = O(N)$
$T_B(N) = N^2 = O(N^2)$

A is asymptotically faster than B !

Omega Notation (Ω)

- **Definition :**

  T(N) = Ω (f(N)) if there are positive constants c and n_0 such that T(N) ≥ c f(N) when N≥ n_0

  – T(N) grows no slower than f(N)

  – growth rate of T(N) is greater than or equal to growth rate of f(N) for large N

  – f(N) is a lower bound on T(N)

    • not fully correct !

Omega Notation

Example:

- $n^{1/2} = \Omega(\lg n)$.

  for $c = 1$ and $n_0 = 16$

  Let $n > 16$

  $c*(\lg n) \le n^{1/2}$

Omega Notation

- **Theorem:**

  $f(N) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(N))$

  Proof:

  $f(N) \leq c_1 g(n) \Leftrightarrow g(n) \geq c_2 f(N)$

  divide the left side with c_1

  $1/c_1 f(N) \leq g(n) \Leftrightarrow g(n) \geq c_2 f(N)$

  if we choose c_2 as $1/c_1$ then theorem is right.

Omega Notation

- $7n^2 + 3n + 5 = O(n^4)$
- $7n^2 + 3n + 5 = O(n^3)$
- $7n^2 + 3n + 5 = O(n^2)$
- $7n^2 + 3n + 5 = \Omega(n^2)$
- $7n^2 + 3n + 5 = \Omega(n)$
- $7n^2 + 3n + 5 = \Omega(1)$

n^2 and $7n^2 + 3n + 5$ grows at the same rate

$7n^2 + 3n + 5 = O(n^2) = \Omega(n^2) = \theta\ (n^2)$

Theta Notation (θ)

- **Definition :**

  $T(N) = \theta\ (h(N))$ if and only if

  $$T(N) = O(h(N)) \text{ and } T(N) = \Omega(h(N))$$

  - T(N) grows as fast as h(N)
  - growth rate of T(N) and h(N) are equal for large N
  - h(N) is a tight bound on T(N)
    - not fully correct !

Theta Notation (θ)

- **Example :**

  $T(N) = 3N^2$

  $T(N) = O(N^4)$

  $T(N) = O(N^3)$

  $T(N) = \theta(N^2) \rightarrow$ best

Little O Notation (o)

- **Definition :**

  $T(N) = o(p(N))$ if

  $T(N) = O(p(N))$ and $T(N) \neq \theta(p(N))$

  - p(N) grows strictly faster than T(N)
  - growth rate of T(N) is less than the growth rate of p(N) for large N
  - p(N) is an upperbound on T(N) (but not tight)
    - not fully correct !

Little O Notation (o)

- **Example :**

$T(N) = 3N^2$

$T(N) = o(N^4)$

$T(N) = o(N^3)$

$T(N) = \theta(N^2)$

RULES

- **RULE 1:**

if $T_1(N) = O(f(N))$ and $T_2(N) = O(g(N))$ then

a) $T_1(N) + T_2(N) = \max(O(f(N)), O(g(N)))$
b) $T_1(N) * T_2(N) = O(f(N) * g(N))$

You can prove these ?
Is it true for θ notation ?
What about Ω notation?

RULES

- **RULE 2:**

if T(N) is a polynomial of degree k

$$T(N) = a_k N^k + a_{k-1} N^{k-1} + \ldots + a_1 N + a_0$$

then

$$T(N) = \theta(N^k)$$

RULES

- **RULE 3:**

$\log^k N = o(N)$ for any constant k

logarithm grows very slowly !

Some Common Functions

- $c = o(\log N)$ => $c=O(\log N)$ but $c \neq \Omega(\log N)$
- $\log N = o(\log^2 N)$
- $\log^2 N = o(N)$
- $N = o(N \log N)$
- $N = o(N^2)$
- $N^2 = o(N^3)$
- $N^3 = o(2^N)$

Example

- $T(N) = 4N^2$
  - $T(N) = O(2N^2)$

    correct but bad style

    $T(N) = O(N^2)$

    drop the constants
  - $T(N) = O(N^2+N)$

    correct but bad style

    $T(N) = O(N^2)$

    ignore low order terms

Another Way to Compute Growth Rates

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = 0 \qquad \Rightarrow f(N) = o(g(N))$$

$$= c \neq 0 \qquad \Rightarrow f(N) = \theta(g(N))$$

$$= \infty \qquad \Rightarrow g(N) = o(f(N))$$

$$= \text{oscilate} \Rightarrow \text{there} \quad \text{is} \quad \text{no} \quad \text{relation}$$

Example

- $f(N) = 7N^2 \qquad g(N) = N^2 + N$

Example

- $f(N) = N \log N$ $g(N) = N^{1.5}$

  compare $\log N$ with $N^{0.5}$

  compare $\log^2 N$ with N

  compare $\log^2 N$ with $o(N)$

  $N \log N = o(N^{1.5})$

RULE 1 : For Loops

The running time of a for loop is at most the running time of the statements in the for loop times the number of iterations

Example :

```
int i, a = 0;
for (i=0; i<n; i++)
{
  print i;
  a=a+i;
}
return i;
```

$T(n) = \theta(n)$

- **RULE 2 : Nested Loops**

  Analyze nested loops inside out

Example :

```
for (int i=1;i<=q;i++)
{
   for (int j=1;j<=r;j++)
       k++;
}
```

$$\Theta(r) \qquad \Theta(q)$$

$T(n) = \theta(r*q)$

GENERAL RULES

- **RULE 3 : Consequtive Statements**

Add the running times

$$
\left.\begin{array}{l}
\left.\begin{array}{l}
\text{for} \ldots \\
\qquad \ldots;
\end{array}\right\} \theta(N) \\
\left.\begin{array}{l}
\text{for} \ldots \\
\qquad \text{for} \ldots \\
\qquad\qquad \ldots;
\end{array}\right\} \theta(N^2)
\end{array}\right\} \theta(N^2)
$$

GENERAL RULES

- **RULE 4 : If / Else**

$$\left.\begin{array}{lll} \text{if (condition)} & \} & T_3(n) \\ \quad\text{S1;} & \} & T_1(n) \\ \text{else} & & \\ \quad\text{S2;} & \} & T_2(n) \end{array}\right\} T(n)$$

Running time is never more than the running time of the test plus larger of the running times of S1 and S2

(may overestimate but never underestimates)

$$\mathbf{T(n) \le T_3(n) + max\ (T_1(n), T_2(n))}$$

Types of complexition

$$T_{worst}(N) = \max_{|I|=N}\{T(I)\} \quad \rightarrow \text{usually used}$$

$$T_{av}(N) = \sum_{|I|=N} T(I).\Pr(I)$$

$$T_{best}(N) = \min_{|I|=N}\{T(I)\}$$

$$T_{worst}(N) \geq T_{av}(N) \geq T_{best}(N)$$

$$T(n) = O(T_{worst}(n)) = \Omega(T_{best}(n))$$

GENERAL RULES

- **RULE 4 : If / Else**

if (condition)	$T_3(n)$	
S1;	$T_1(n)$	$T(n)$
else		
S2;	$T_2(n)$	

$T_w\ (n) = T_3(n) + \max(T_1(n), T_2(n))$
$T_b\ (n) = T_3(n) + \min(T_1(n), T_2(n))$
$T_{av}(n) = p(T)T_1(n) + p(F)T_2(n) + T_3(n)$

p(T) → p (condition = True)
p(F) → p (condition = False)

GENERAL RULES

- **Example :**

```
if (condition)
        S1;
else
        S2;
```

$$\left.\begin{array}{l} T_3(n) = \theta(n) \\ T_1(n) = \theta(n^2) \\ \\ T_2(n) = \theta(n) \end{array}\right\} T(n)$$

$T_w(n) = T_3(n) + \max(T_1(n), T_2(n)) = \theta(n^2)$

$T_b(n) = T_3(n) + \min(T_1(n), T_2(n)) = \theta(n)$

if $p(T) = p(F) = ½$

$T_{av}(n) = p(T)T_1(n) + p(F)T_2(n) + T_3(n) = \theta(n^2)$

$T(n) = O(n^2)$

$\quad = \Omega(n)$

Example:

Algorithm for computing factorial

```
int factorial (int n)
{
  if (n<=1)
      return 1;
  else
      return n*factorial(n-1);
}
```

1

1 for multiplication
+ 1 for substraction
+ cost of evaluation of factorial(n-1)

T(n) = cost of evaluation of factorial of n
T(n) = 2 + T(n-1)
T(1) = 1

RECURSIVE CALLS

$$T(n) = 2 + T(n-1)$$
$$T(n) = 2 + 2 + T(n-2)$$
$$T(n) = 2 + 2 + 2 + T(n-3)$$
$$\vdots$$
$$T(n) = k*2 + T(n-k) \qquad k = n-1 \Rightarrow$$
$$T(n) = (n-1)*2 + T(n-(n-1))$$
$$T(n) = (n-1)*2 + T(1)$$
$$T(n) = (n-1)*2 + 1$$
$$T(n) = \theta(n)$$