Algorithm analysis & design

Asymptotic Notations

Presented By:

T.A. Asmaa Hamad El-saied

E-mail: eng.asmaa134@gmail.com

Agenda

- **Asymptotic Notations**
- **Examples**

Running time vs. Asymptotic performance

- Exact **running time** can be computed for small problem size (small n)
- **Asymptotic performance** can be found for large n where multiplication constants and lower-order terms of the exact running time are dominated.

- **Asymptotic performance of algorithm**

  How the running time of an algorithm increases with the size of the input in the limit, as the size of the input increases without bound.

Growth of Functions and Asymptotic Notation

- When we study algorithms, we are interested in characterizing them according to their efficiency.

- We are usually interesting in the order of growth of the running time of an algorithm, not in the exact running time. This is also referred to as the asymptotic running time.

- We need to develop a way to talk about rate of growth of functions so that we can compare algorithms.

- **Asymptotic notation** gives us a method for classifying functions according to their rate of growth.

Classifying Functions by Their Asymptotic Growth

- **Asymptotic growth:** The rate of growth of a function

- Given a particular differentiable function **f(n)**, all other differentiable functions fall into three classes:
  - growing with the same rate
  - growing faster
  - growing slower

Bounds (asymptotic notations)

$$O \quad \Theta \quad \Omega$$

- Bounds describe the **limiting behavior** of algorithm complexity at large (n). They are:

  - Upper Bound (**Big O** complexity)
  - Lower Bound (**Big Ω** complexity)
  - Exact (**Big Θ** complexity)

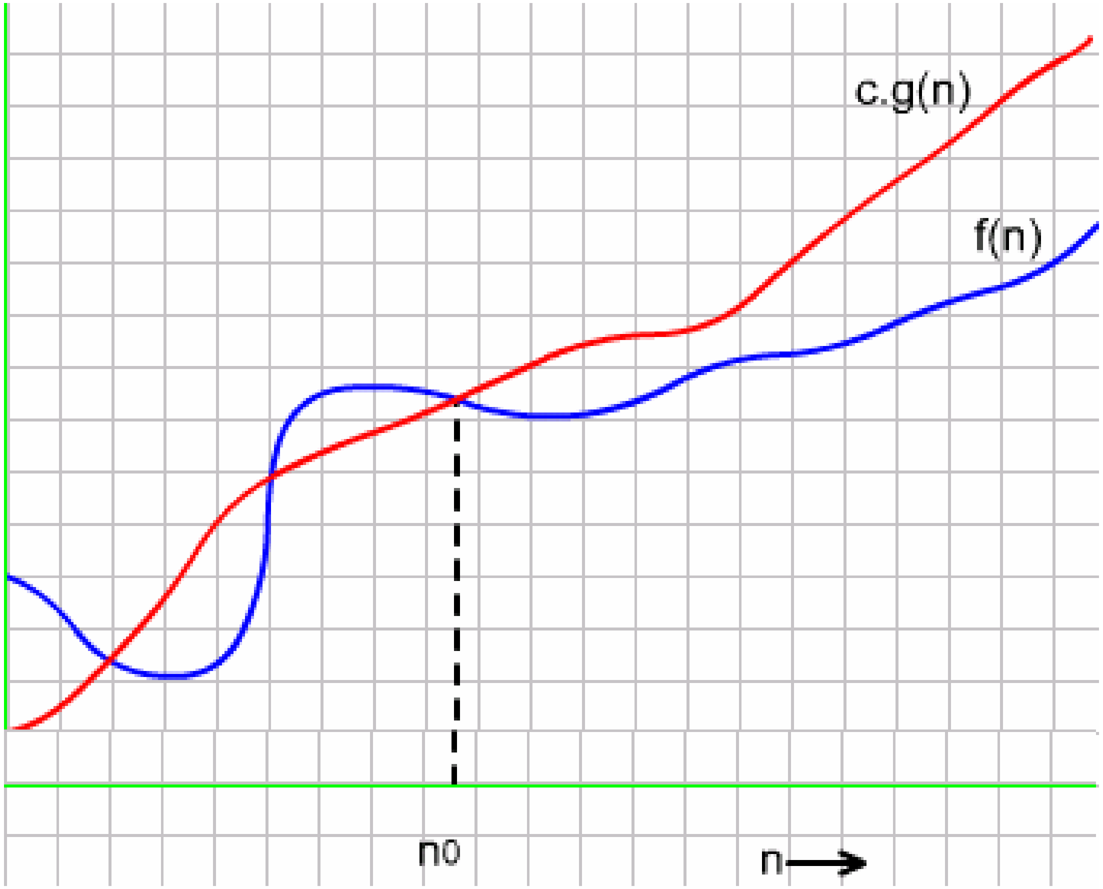
O-NOTATION: DEFINITION

- Big-O:- defines an upper bound of an algorithm. It bounds a function only from above.
- For function F(n), we define O (g(n)), big-Oh of n, as:

> ***f(N) is O(g(N)) If***
>
> ***∃ positive constants c and n_0, such that***
>
> $f(n) \leq c \times g(n) \quad \forall n \geq n_0$

- For example
  - $T(n) = O(n^{100})$
    - T(n) will never grow asymptotically faster than n^{100}

O-NOTATION: DEFINITION



O-NOTATION : Example

Prove that

- $5n^2$ is $O(n^3)$

Proof:

- According to the definition of O*()*, we should find a constant $\boldsymbol{c}$ s.t.

$$\boldsymbol{f}(\boldsymbol{n}) \leq \boldsymbol{c} \times \boldsymbol{g}(\boldsymbol{n}) \;\; \forall\, \boldsymbol{n} \geq \; \boldsymbol{n_0}$$

$$5 \times n^2 \leq c \times n^3 \;\forall\, n > \; n_0 \text{ , divide by } n^3$$

$$\frac{5}{n} \leq c \qquad \forall\, n \geq \; n_0$$

- Substitute n=5 (for example), then any **c≥1** will satisfy the inequality ∀ n≥ 5

➔it's true

Ω-NOTATION: DEFINITION

- Ω notation provides an asymptotic lower bound of an algorithm.
- For function F(n), we define Ω (g(n)), big-Omega of n, as:

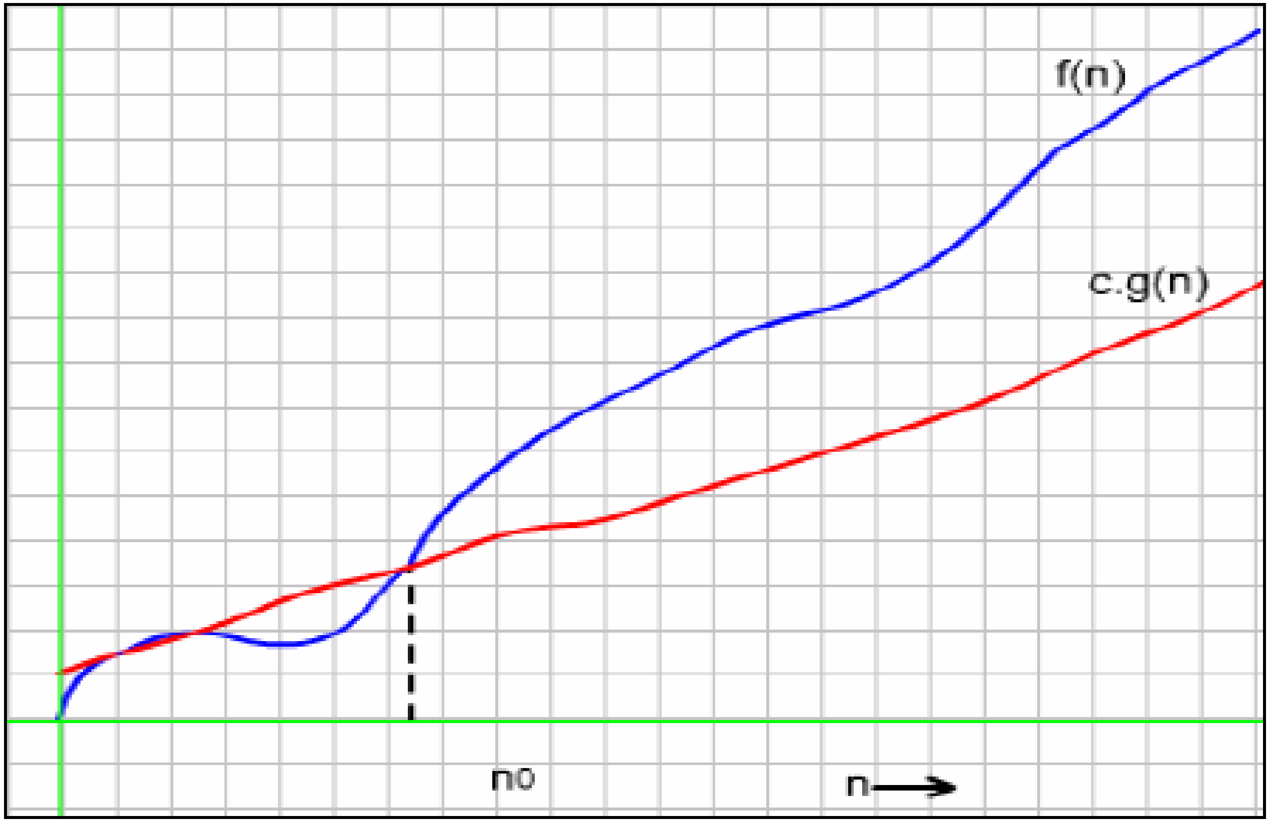
f(n) is Ω(g(n)) If

∃ positive constants c and n_0, such that

$$c \times g(n) \leq f(n) \quad \forall n \geq n_0$$

- For example:
  - $T(n) = \Omega(n^3)$
    - T(n) will never grow asymptotically slower than n^3

Ω-NOTATION
ASYMPTOTIC LOWER BOUND



Trend of running time

Ω-NOTATION : Example

Prove that

- $5n^2$ is $\Omega(n)$

Proof:

- According to the definition of Ω *()*, we should find a constant $\boldsymbol{c}$ s.t.

$$\boldsymbol{c \times g(n) \leq f(n) \; \forall \; n \geq \; n_0}$$

$$c \times n \leq 5 \times n^2 \; \forall \; n > \; n_0 \text{ , divide by n}$$

$$c \leq 5 \times n \quad \forall \; n \geq \; n_0$$

- Substitute n=1 (for example), then any **c ≤ 5** will satisfy the inequality ∀ n ≥ 1

➔ it's true

Θ-NOTATION: DEFINITION

- Θ notation provides an asymptotic tight bound of an algorithm.
- For function F(n), we define Θ (g(n)), big-Theta of n, as:

f(n) is Θ (g(n)) If

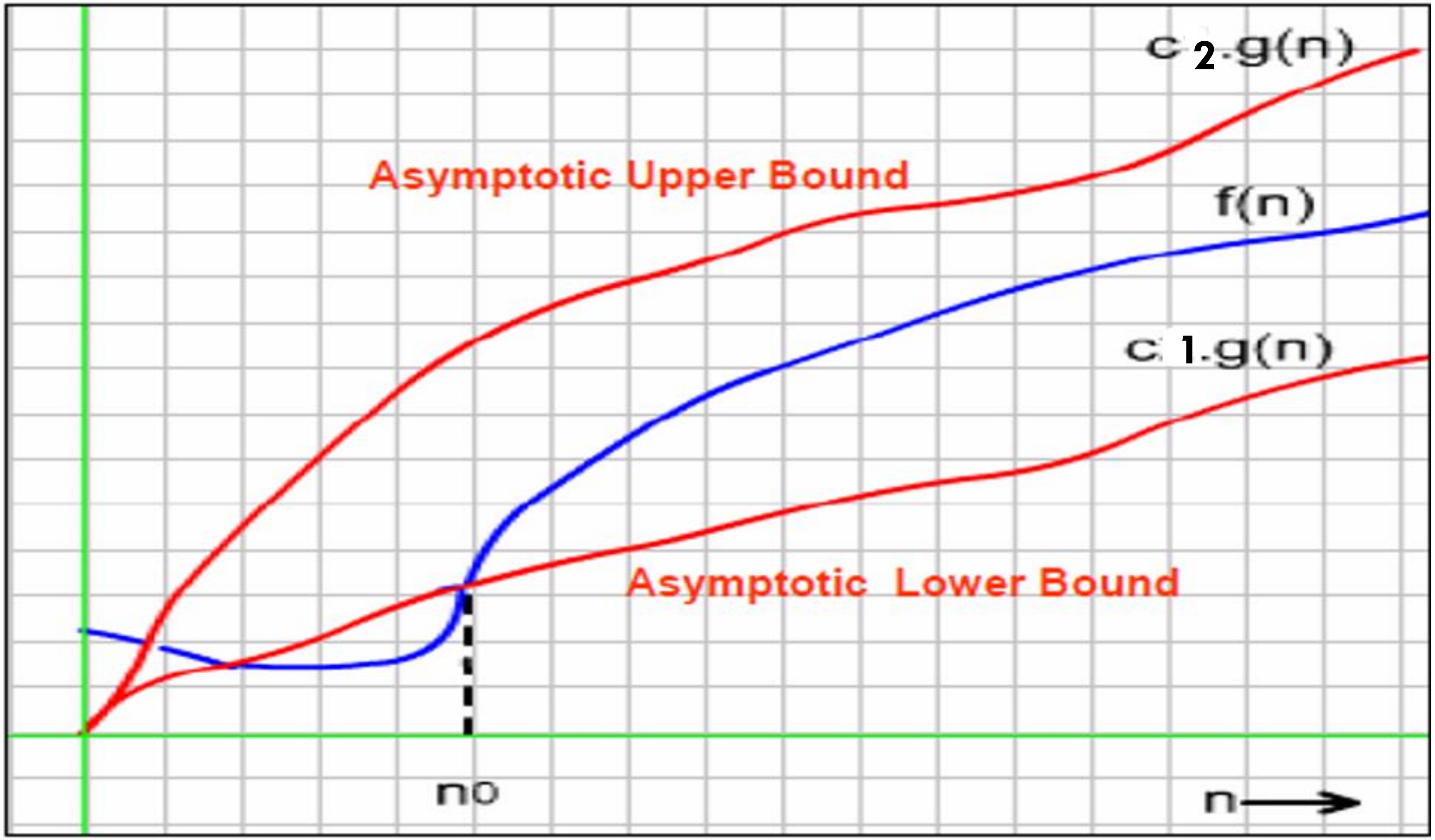
∃ positive constants c1,c2 ,and n_0, such that

$$c1 \times g(n) \le f(n) \le c2 \times g(n)$$

$$\forall n \ge n_0$$

- *Θ():* Exact order (most difficult to compute in some algorithms)
- For example:-
  - $T(n) = \Theta(n^3)$
    - T(n) grows asymptotically as fast as n^3.

Θ-NOTATION: DEFINITION



Asymptotic tight bound of f(n)

Θ -NOTATION : Example

Prove that

- **$5n^2$ is $\Theta(n^2)$)**)

Proof:

- According to the definition of $\Theta()$, we should find 2 constants c_1 & c_2 s.t.

$$c1 \times \boldsymbol{g}(\boldsymbol{n}) \leq \boldsymbol{f}(\boldsymbol{n}) \leq c2 \times \boldsymbol{g}(\boldsymbol{n}) \quad \forall \boldsymbol{n} \geq \boldsymbol{n_0}$$

$$c_1 \times \mathbf{n^2} \leq 5 \times n^2 \leq c_2 \times \mathbf{n^2} \quad \forall n \geq n_0 \text{ , divide by } n^2$$

$$c_1 \leq 5 \leq c_2 \quad \forall n \geq n_0$$

- So, there's exists $c_1 = 4$ and $c_2 = 6$ that satisfy the inequality for all $n \geq 1$ ($n_0 = 1$)

➔ it's true

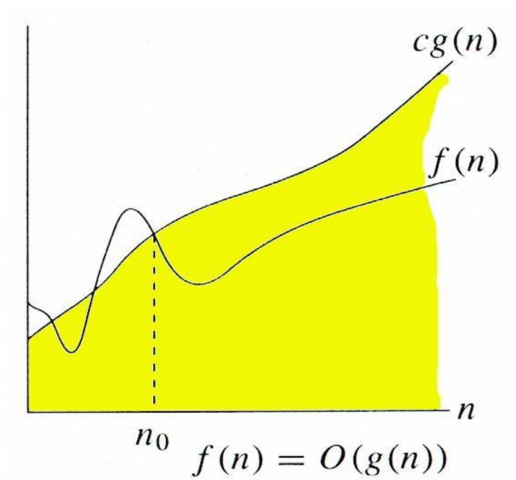
Relations Between Bounds (asymptotic notations)

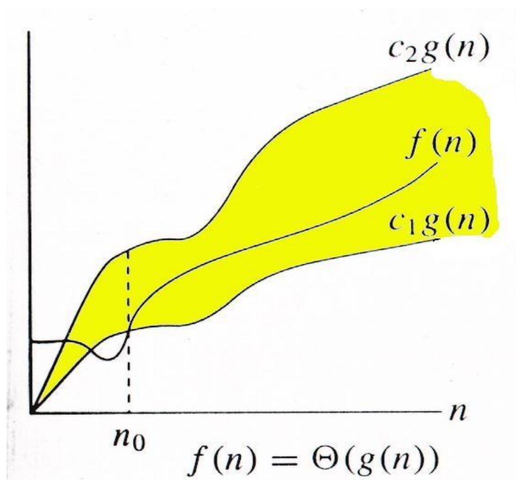
O Θ Ω

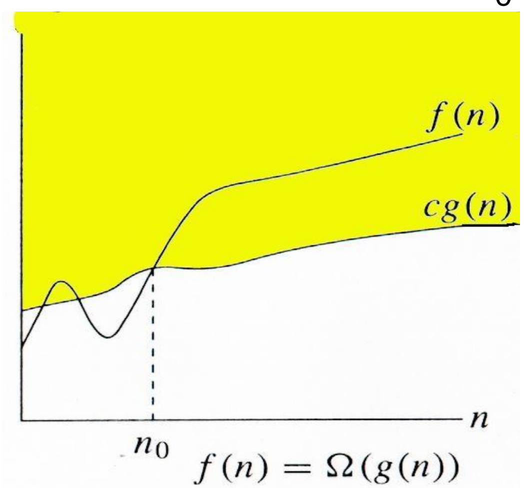
$f(n) \le c \times g(n)$
for all $n \ge n_0$

$c1 \times g(n) \le f(n) \le c2 \times g(n)$
for all $n \ge n_0$

$f(n) \ge c \times g(n)$
for all $n \ge n_0$







Relations Between Bounds (asymptotic notations)

Theorem :

For any two functions $g(n)$ and $f(n)$,

$f(n) = \Theta(g(n))$ iff

$f(n) = O(g(n))$ **and** $f(n) = \Omega(g(n))$.

I.e. f(N) is $\Theta(g(N))$ **iff** it's $O(g(N))$ & $\Omega(g(N))$ (by their definitions)

Relations Between Bounds : Example

Prove that

- **$5n^2$ is $\Theta(n^2)$ ➔ $O(N^2)$ & $\Omega(N^2)$**

Proof:

- According to the definition of $\Theta()$, we should find 2 constants c_1 & c_2 s.t.

$$c1 \times \boldsymbol{g}(\boldsymbol{n}) \le \boldsymbol{f}(\boldsymbol{n}) \le c2 \times \boldsymbol{g}(\boldsymbol{n}) \quad \forall\, \boldsymbol{n} > \boldsymbol{n_0}$$

$$c_1 \times \mathbf{n^2} \le 5 \times n^2 \le c_2 \times \mathbf{n^2} \quad \forall\, n > n_0 \text{ , divide by } n^2$$

$$c_1 \le 5 \le c_2 \quad \forall\, n > n_0$$

- So, there's exists $c_1 = 4$ and $c_2 = 6$ that satisfy the inequality for all $n \ge 1$ ($n_0 = 1$)
- But the left side of the inequality is the prove of the definition of *$\Omega(N^2)$* and right side is the prove of *$O(N^2)$*

➔ it's true

Examples

- $\Theta(n^3)$: n^3

  $5n^3 + 4n$

  $105n^3 + 4n2 + 6n$

- $\Theta(n^2)$: n^2

  $5n^2 + 4n + 6$

  $n^2 + 5$

Examples

- **True or false?**
  - $N^2 = O(N^2)$ **true**
  - $2N = O(N^2)$ **true**
  - $N = O(N^2)$ **true**

  - $N^2 = O(N)$ **false**
  - $2N = O(N)$ **true**
  - $N = O(N)$ **true**

Examples

- $N^2 = \Theta(N^2)$ **true**
- $2N = \Theta(N^2)$ **false**
- $N = \Theta(N^2)$ **false**

- $N^2 = \Theta(N)$ **false**
- $2N = \Theta(N)$ **true**
- $N = \Theta(N)$ **true**

- $4+3n = O(n)$ **true**
- $n+2\log n = O(\log n)$ **false**
- $\log n+2 = O(1)$ **False**

What is the difference between worst case and Big-O?

- Worst, best and average cases describe distinct runtime functions (Different types of analysis) : one for the sequence of highest runtime of any given n, one for that of lowest, and so on..
- **asymptotic** notations are just representation methods of any time complexity of an algorithm.

What is the difference between worst case and Big-O?

- **Worst case:-** happen
- **Upper Bound:-** may not happen

Example1:

- **Compute complexity for this code:-**

  For k=1 to N

  Statements That take exectly *k* statements

  End for

- **<u>Solution:-</u>**

- **By applying rules of order: Complexity Time = O(**N^2**)**

- **By exact iteration:**

  - **Complexity Time** $=1+2+\cdots\cdots+N = \sum_{k=1}^{N} k = \frac{N(N+1)}{2} = \Theta(N^2)$

Example2:

- Indicate, for each pair of expressions (A, B) in the table below, whether A is O, Ω, or Θ of B. Log are base 2. Your answer should be in the form of the table with "yes" or "no" written in each box:

A	B	O	Ω	Θ
2^{N+1}	2^N			
$N (Log(N))^2$	$N Log(N)$			
$(N-1)!$	$N!$			

Example2:

- Indicate, for each pair of expressions (A, B) in the table below, whether A is O, Ω, or Θ of B. Log are base 2. Your answer should be in the form of the table with "yes" or "no" written in each box:

A	B	O	Ω	Θ
2^{N+1}	2^N	yes	yes	yes
$N (Log(N))^2$	$N Log(N)$	no	yes	no
$(N-1)!$	$N!$	yes	no	no

Example3:

- Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following:
  - $f(n) = O(g(n))$ implies $g(n) = O(f(n))$.
  - $h(n) + k(n) = \Theta(\min(h(n), k(n)))$.

Example3:

- Let $f(n)$ and $g(n)$ be asymptotically positive functions. Prove or disprove each of the following:
  - $f(n) = O(g(n))$ implies $g(n) = O(f(n))$. **#Disproved**
  - $h(n) + k(n) = \Theta(\min(h(n), k(n)))$. **#Disproved**

Example 4:

- Write the **big-Θ** expression (**tight** bound) to describe the number of operations required for the following algorithms:

1)

```
z = 0 ;
  for (w = 0 ; w < K ; w++)
        for (x = 0 ; x < N ; x++)
              z = z + x
        End for
        for (y = 0 ; y < M ; y++)
              z = z + y
        End for
  End for
```

Solution:

By applying rules of order:

Complexity Time $= \Theta(K \times (N + M)) = \Theta(K \times N) = \Theta(N^2)$ if K is $O(N)$

Note: $\Theta(N+M)=\Theta(N)$ say $N \Rightarrow \max$

Example 4:

- Write the **big-Θ** expression (**tight** bound) to describe the number of operations required for the following algorithms:

2)

```
F1 (N)
    For i = 1 to N Do
        j = N
        While I > j Do
            j = j – 1
        End while
    End for
EndF1
```

Solution:-

- **By applying rules of order:**

  Complexity Time = $O(N^2)$

- **By exact iteration:**

  Complexity Time

  $=0+1+2+\cdots\cdots+N-1 = \sum_{i=0}^{N-1} i$

  $= \frac{(N-1)N}{2} = \Theta(N^2)$

Example 4:

- Write the **big-Θ** expression (**tight** bound) to describe the number of operations required for the following algorithms:

3)

```
F2 (N)
   I = 1
    While I < N Do
            For J = 1 to I
                 K = 1
            End for
      I = 2 × I
    End while
F2
```

- **By applying rules of order:**

  Complexity Time = O (NlogN)

Solution:

- By summing the # iterations in the inner loop: **1 + 2 + 4 + 8 + 16 + … + 2^L** ,where L is # iterations
- **Termination:**

  when $2^L = N \rightarrow \log 2^L = \log N$

  $\rightarrow$ **$L = \log_2 (N)$**
- **1 + 2 + 4 + 8 + 16 + … + 2^L = $\sum_0^l 2^i$=**

  $\sum_0^{\log N} 2^i = (2^{\log N+1} - 1) = (2^{\log N}\, 2 - 1)$

  $= (N^{\log 2}\, 2 - 1)$ = N 2-1= Θ (N)

Example 4:

- Write the **big-Θ** expression (**tight** bound) to describe the number of operations required for the following algorithms:

4)

```
I=N
while (I > 1)
        I = I - 10;
 End while
```

- **<u>Solution:</u>**
  - **Iterator:** N, N – 10, N – 2×10, N – 3×10, …, N – L×10, where **L** is # iterations
  - **Termination:** when N – L×10 = 1 ➔ L = $\frac{N-1}{10}$

Complexity Time = Θ(N)

Example 4:

- Write the **big-Θ** expression (**tight** bound) to describe the number of operations required for the following algorithms:

5)

For i = 1 to N Do

A[i] = i

B[i] = 1

Endfor

InsertionSort(A, N)

Solution: $\Theta(N+N)=\Theta(N) \Longrightarrow$ since the array is already sorted (insertion sort best case)

Eight growth functions

- Eight functions O(n) that occur frequently in the analysis of algorithms (in order of increasing rate of growth relative to n):
  - Constant ≈ ***1***
  - Logarithmic ≈ log ***n***
  - Linear ≈ ***n***
  - Log Linear ≈ ***n*** log ***n***
  - Quadratic ≈ ***n***2
  - Cubic ≈ ***n***3
  - Exponential ≈ $\mathbf{2}^{n}$
  - Factorial≈ ***n***!

Growth rates compared

	n=1	n=2	n=4	n=8	n=16	n=32
1	1	1	1	1	1	1
logn	0	1	2	3	4	5
n	1	2	4	8	16	32
nlogn	0	2	8	24	64	160
n^2	1	4	16	64	256	1024
n^3	1	8	64	512	4096	32768
2^n	2	4	16	256	65536	4294967296
n!	1	2	24	40320	20.9T	Don’t ask!

Thanks