

Basics of Complexity Analysis: The RAM Model and the Growth of Functions

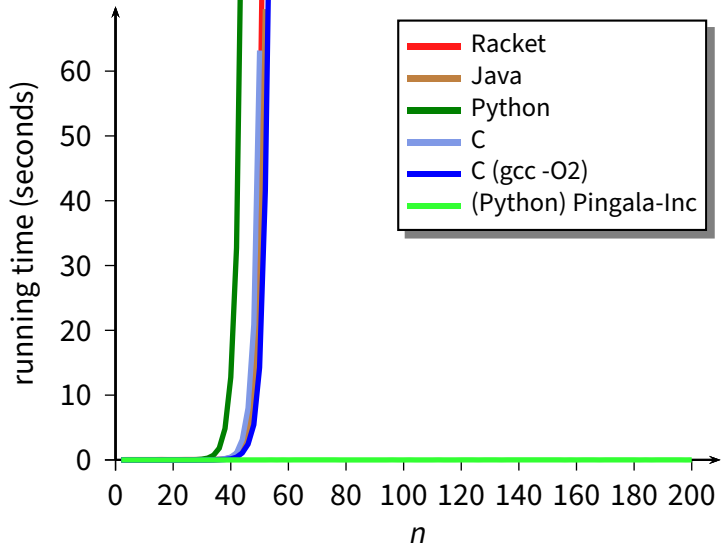
Antonio Carzaniga

Faculty of Informatics
Università della Svizzera italiana

February 19, 2025

- Informal analysis of two Pingala algorithms
- The *random-access machine* model
- Measure of complexity
- Characterizing functions with their asymptotic behavior
- Big- O , omega, and theta notations

Slow vs. Fast Pingala



Slow vs. Fast Pingala

- We informally characterized our two Pingala algorithms

Slow vs. Fast Pingala

- We informally characterized our two Pingala algorithms
 - ▶ **PINGALA**(n) is *exponential* in n
 - ▶ **PINGALA-INC**(n) is *linear* in n

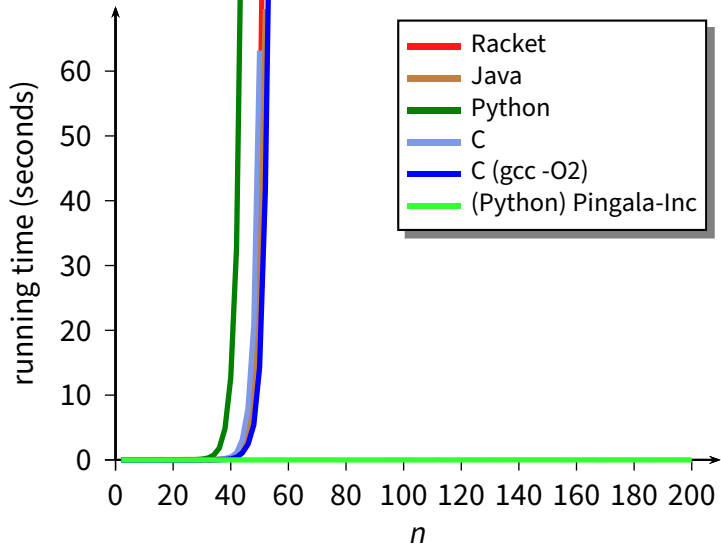
Slow vs. Fast Pingala

- We informally characterized our two Pingala algorithms
 - ▶ **PINGALA**(n) is *exponential* in n
 - ▶ **PINGALA-INC**(n) is *linear* in n
- How do we characterize the complexity of algorithms?
 - ▶ *in general*

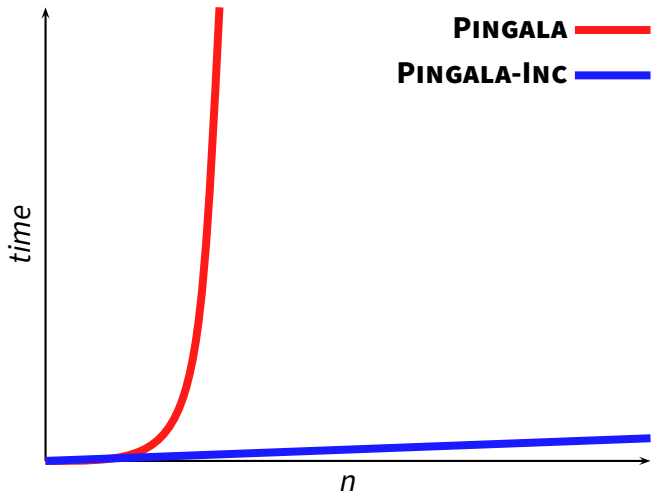
Slow vs. Fast Pingala

- We informally characterized our two Pingala algorithms
 - ▶ **PINGALA**(n) is *exponential* in n
 - ▶ **PINGALA-INC**(n) is *linear* in n
- How do we characterize the complexity of algorithms?
 - ▶ *in general*
 - ▶ in a way that is *specific to the algorithms*
 - ▶ but *independent of implementation details*

Slow vs. Fast Pingala



Slow vs. Fast Pingala



A Model of the Computer

- An informal model of the *Random-Access Machine (RAM)*

A Model of the Computer

- An informal model of the *Random-Access Machine (RAM)*
- *Basic types* in the RAM model

A Model of the Computer

- An informal model of the *Random-Access Machine (RAM)*
- *Basic types* in the RAM model
 - ▶ integer and floating-point numbers
 - ▶ limited size of each “word” of data (e.g., 64 bits)

A Model of the Computer

- An informal model of the *Random-Access Machine (RAM)*
- *Basic types* in the RAM model
 - ▶ integer and floating-point numbers
 - ▶ limited size of each “word” of data (e.g., 64 bits)
- *Basic steps* in the RAM model

A Model of the Computer

- An informal model of the *Random-Access Machine (RAM)*
- *Basic types* in the RAM model
 - ▶ integer and floating-point numbers
 - ▶ limited size of each “word” of data (e.g., 64 bits)
- *Basic steps* in the RAM model
 - ▶ *operations involving basic types*
 - ▶ load/store: assignment, use of a variable
 - ▶ arithmetic operations: addition, multiplication, division, etc.
 - ▶ branch operations: conditional branch, jump
 - ▶ subroutine call

A Model of the Computer

- An informal model of the ***Random-Access Machine (RAM)***
- ***Basic types*** in the RAM model
 - ▶ integer and floating-point numbers
 - ▶ limited size of each “word” of data (e.g., 64 bits)
- ***Basic steps*** in the RAM model
 - ▶ ***operations involving basic types***
 - ▶ load/store: assignment, use of a variable
 - ▶ arithmetic operations: addition, multiplication, division, etc.
 - ▶ branch operations: conditional branch, jump
 - ▶ subroutine call
- A ***basic step*** in the RAM model takes a ***constant time***

Analysis in the RAM Model

PINGALA-INC(n)

```
1  if  $n \leq 2$ 
2      return  $n$ 
3   $pprev = 1$ 
4   $prev = 2$ 
5  for  $i = 3$  to  $n$ 
6       $P = prev + pprev$ 
7       $pprev = prev$ 
8       $prev = P$ 
9  return  $P$ 
```

Analysis in the RAM Model

PINGALA-INC(n)

```
1  if  $n \leq 2$ 
2      return  $n$ 
3   $pprev = 1$ 
4   $prev = 2$ 
5  for  $i = 3$  to  $n$ 
6       $P = prev + pprev$ 
7       $pprev = prev$ 
8       $prev = P$ 
9  return  $P$ 
```

$cost$ $times$ ($n > 2$)

Analysis in the RAM Model

PINGALA-INC(n)

```
1  if  $n \leq 2$ 
2      return  $n$ 
3   $pprev = 1$ 
4   $prev = 2$ 
5  for  $i = 3$  to  $n$ 
6       $P = prev + pprev$ 
7       $pprev = prev$ 
8       $prev = P$ 
9  return  $P$ 
```

cost *times* ($n > 2$)

c_1	1
c_2	0
c_3	1
c_4	1
c_5	$n - 1$
c_6	$n - 2$
c_7	$n - 2$
c_8	$n - 2$
c_9	1

$$T(n) = c_1 + c_3 + c_4 + c_9 + (n - 1)c_5 + (n - 2)(c_6 + c_7 + c_8)$$

- Does a load/store operation cost more than, say, an arithmetic operation?

`x = 0` vs. `y + z`

- Does a load/store operation cost more than, say, an arithmetic operation?

$x = 0$ vs. $y + z$

- ***We do not care about the specific costs of each basic step***
 - ▶ these costs are likely to vary significantly with languages, implementations, and processors
 - ▶ we simplify our model by effectively considering only the maximal cost of any basic step
 - ▶ so, we assume $c_1 = c_2 = c_3 = \dots = c_j$

Analysis in the RAM Model

PINGALA-INC(n)

```
1  if  $n \leq 2$ 
2      return  $n$ 
3   $pprev = 1$ 
4   $prev = 2$ 
5  for  $i = 3$  to  $n$ 
6       $P = prev + pprev$ 
7       $pprev = prev$ 
8       $prev = P$ 
9  return  $P$ 
```

cost *times* ($n > 2$)

c_1	1
c_2	0
c_3	1
c_4	1
c_5	$n - 1$
c_6	$n - 2$
c_7	$n - 2$
c_8	$n - 2$
c_9	1

$$T(n) = c_1 + c_3 + c_4 + c_9 + (n - 1)c_5 + (n - 2)(c_6 + c_7 + c_8)$$

Analysis in the RAM Model

PINGALA-INC(n)

```
1  if  $n \leq 2$ 
2      return  $n$ 
3   $pprev = 1$ 
4   $prev = 2$ 
5  for  $i = 3$  to  $n$ 
6       $P = prev + pprev$ 
7       $pprev = prev$ 
8       $prev = P$ 
9  return  $P$ 
```

cost *times* ($n > 2$)

c_1	1
c_2	0
c_3	1
c_4	1
c_5	$n - 1$
c_6	$n - 2$
c_7	$n - 2$
c_8	$n - 2$
c_9	1

$T(n) = nC_1 + C_2 \Rightarrow T(n)$ is a linear function of n

A Model of *Any* Computer, Past, Present, and Future

A Model of *Any* Computer, Past, Present, and Future

- A *basic step* in the RAM model takes a *constant time*
 - ▶ “constant” means *independent of the input size*

A Model of *Any* Computer, Past, Present, and Future

- A *basic step* in the RAM model takes a *constant time*
 - ▶ “constant” means *independent of the input size*
- The *specific* constant is a *technological factor*

A Model of *Any* Computer, Past, Present, and Future

- A ***basic step*** in the RAM model takes a ***constant time***

 - ▶ “constant” means ***independent of the input size***

- The *specific* constant is a ***technological factor***

- ***Technology changes***

 - ...so ***we ignore any specific multiplicative or additive constants***

 - ...effectively we allow for ***any scaling factor***

Complexity as a Function of the *Size* of the Input

- We measure the complexity of an algorithm *as a function of the **size** of the input*
 - ▶ size measured in bits

Complexity as a Function of the *Size* of the Input

- We measure the complexity of an algorithm *as a function of the **size** of the input*
 - ▶ size measured in bits
 - ▶ did we do that for **PINGALA-INC**?

Complexity as a Function of the *Size* of the Input

- We measure the complexity of an algorithm *as a function of the **size** of the input*
 - ▶ size measured in bits
 - ▶ did we do that for **PINGALA-INC**?
- **Example:** given a sequence $A = \langle a_1, a_2, \dots, a_n \rangle$, and a value x , output TRUE if A contains x , or FALSE otherwise

Complexity as a Function of the *Size* of the Input

- We measure the complexity of an algorithm *as a function of the size of the input*
 - ▶ size measured in bits
 - ▶ did we do that for **PINGALA-INC**?
- **Example:** given a sequence $A = \langle a_1, a_2, \dots, a_n \rangle$, and a value x , output TRUE if A contains x , or FALSE otherwise

```
FIND( $A, x$ )  
1  for  $i = 1$  to  $\text{length}(A)$   
2      if  $A[i] == x$   
3          return TRUE  
4  return FALSE
```

Complexity as a Function of the *Size* of the Input

- We measure the complexity of an algorithm *as a function of the **size** of the input*
 - ▶ size measured in bits
 - ▶ did we do that for **PINGALA-INC**?
- **Example:** given a sequence $A = \langle a_1, a_2, \dots, a_n \rangle$, and a value x , output TRUE if A contains x , or FALSE otherwise

```
FIND( $A, x$ )  
1  for  $i = 1$  to  $\text{length}(A)$   
2      if  $A[i] == x$   
3          return TRUE  
4  return FALSE
```

$$T(n) = Cn$$

Worst-Case Complexity

- In general we measure the complexity of an algorithm *in the worst case*

Worst-Case Complexity

- In general we measure the complexity of an algorithm *in the worst case*
- **Example:** given a sequence $A = \langle a_1, a_2, \dots, a_n \rangle$, output TRUE if A contains two equal values $a_i = a_j$ (with $i \neq j$)

Worst-Case Complexity

- In general we measure the complexity of an algorithm *in the worst case*
- **Example:** given a sequence $A = \langle a_1, a_2, \dots, a_n \rangle$, output TRUE if A contains two equal values $a_i = a_j$ (with $i \neq j$)

FINDEQUALS(A)

```
1  for  $i = 1$  to  $\text{length}(A) - 1$ 
2      for  $j = i + 1$  to  $\text{length}(A)$ 
3          if  $A[i] == A[j]$ 
4              return TRUE
5  return FALSE
```

Worst-Case Complexity

- In general we measure the complexity of an algorithm *in the worst case*
- **Example:** given a sequence $A = \langle a_1, a_2, \dots, a_n \rangle$, output TRUE if A contains two equal values $a_i = a_j$ (with $i \neq j$)

FINDEQUALS(A)

```
1  for  $i = 1$  to  $\text{length}(A) - 1$ 
2      for  $j = i + 1$  to  $\text{length}(A)$ 
3          if  $A[i] == A[j]$ 
4              return TRUE
5  return FALSE
```

$$T(n) = C \frac{n(n-1)}{2}$$

Asymptotic Complexity

- We care about $T(n)$ as n *goes to infinity*
 - ▶ “for sufficiently large n ”

Asymptotic Complexity

- We care about $T(n)$ as n ***goes to infinity***
 - ▶ “for sufficiently large n ”
- We care only about the ***asymptotic order of growth*** of $T(n)$

Asymptotic Complexity

- We care about $T(n)$ as n *goes to infinity*
 - ▶ “for sufficiently large n ”
- We care only about the *asymptotic order of growth* of $T(n)$
 - ▶ so we ignore lower-order terms

Example:

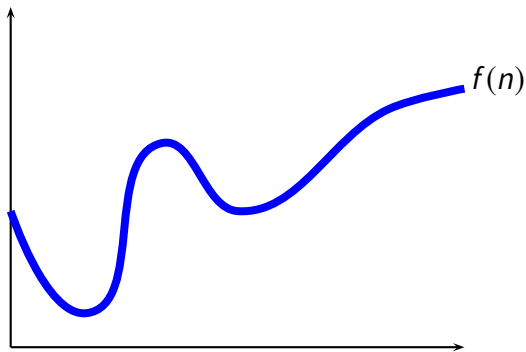
Algorithm 1 costs $T_1(n) = 100n + 3000$ basic steps

Algorithm 2 costs $T_2(n) = 0.02n^2 + 2$ basic steps

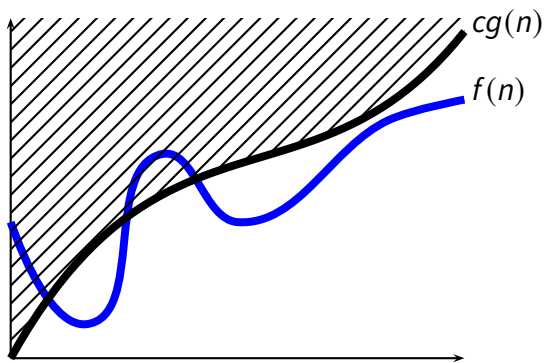
Which is best?

- Given a function $g(n)$, we define the *family of functions* $O(g(n))$

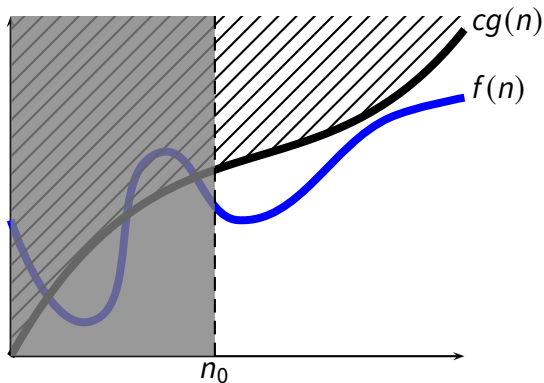
- Given a function $g(n)$, we define the *family of functions* $O(g(n))$



- Given a function $g(n)$, we define the *family of functions* $O(g(n))$

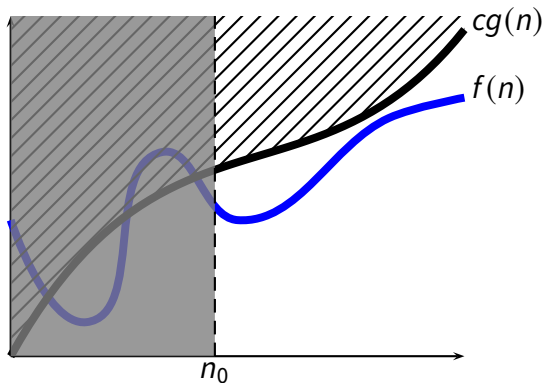


- Given a function $g(n)$, we define the *family of functions* $O(g(n))$



$$O(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \\ : 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

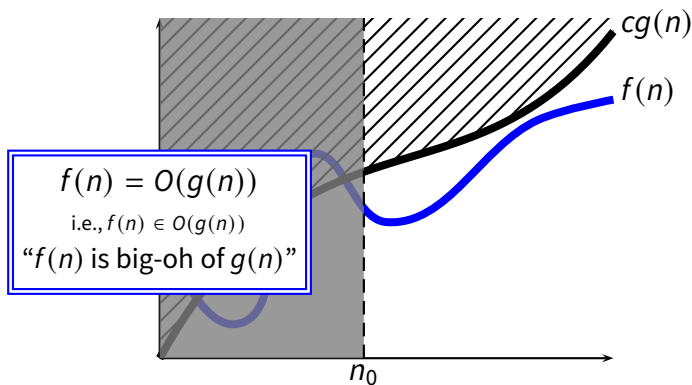
- Given a function $g(n)$, we define the *family of functions* $O(g(n))$



$$O(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \\ : 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

$f(n)$ is below $g(n)$ for all sufficiently large n , and for some scaling factor c

- Given a function $g(n)$, we define the *family of functions* $O(g(n))$



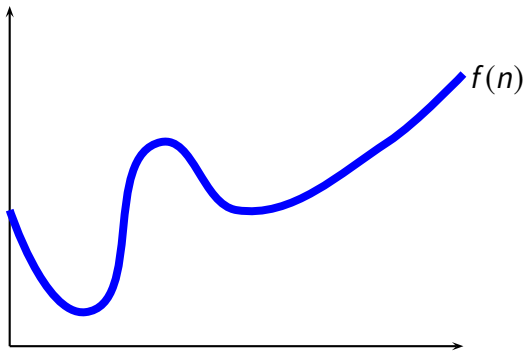
$$O(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0$$

$$: 0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$$

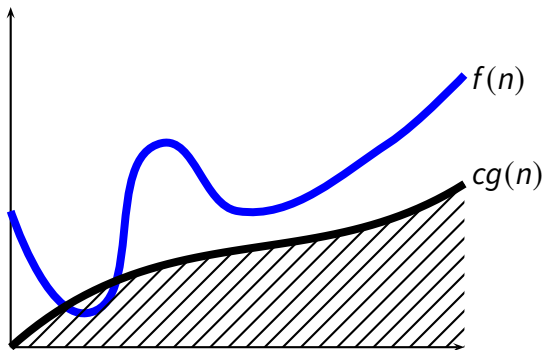
$f(n)$ is below $g(n)$ for all sufficiently large n , and for some scaling factor c

- Given a function $g(n)$, we define the *family of functions* $\Omega(g(n))$

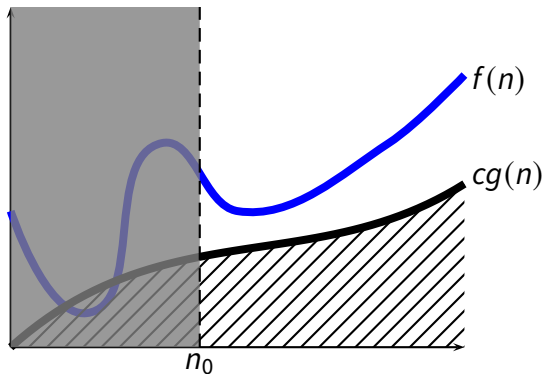
- Given a function $g(n)$, we define the *family of functions* $\Omega(g(n))$



- Given a function $g(n)$, we define the *family of functions* $\Omega(g(n))$

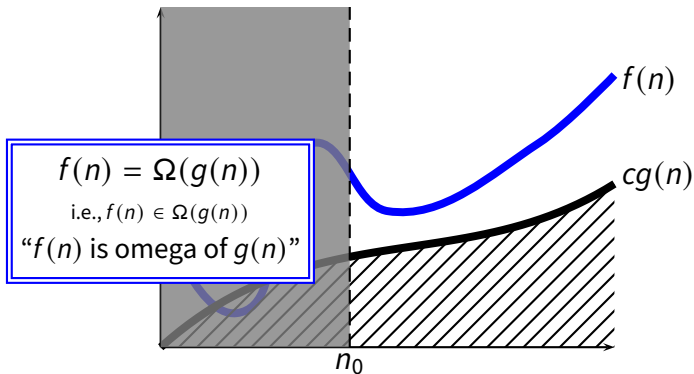


- Given a function $g(n)$, we define the *family of functions* $\Omega(g(n))$



$$\Omega(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \\ : 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$$

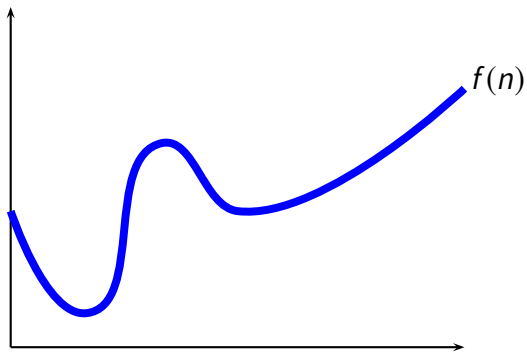
- Given a function $g(n)$, we define the *family of functions* $\Omega(g(n))$



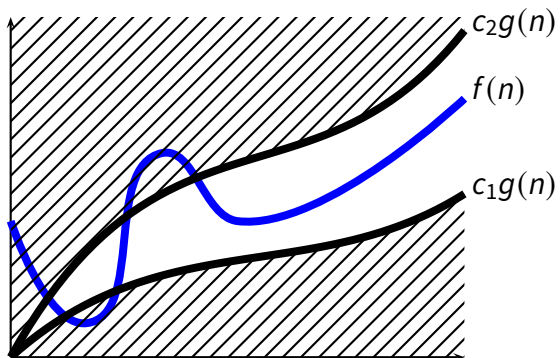
$$\Omega(g(n)) = \{f(n) : \exists c > 0, \exists n_0 > 0 \\ : 0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$$

- Given a function $g(n)$, we define the *family of functions* $\Theta(g(n))$

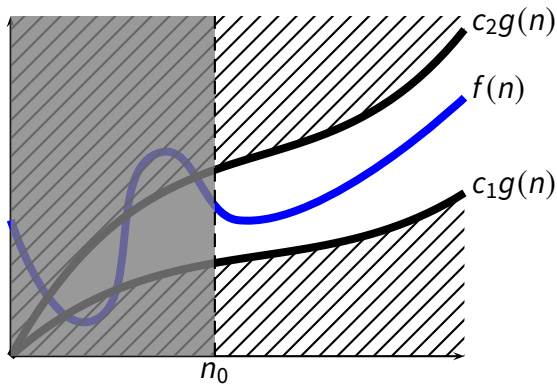
- Given a function $g(n)$, we define the *family of functions* $\Theta(g(n))$



- Given a function $g(n)$, we define the *family of functions* $\Theta(g(n))$

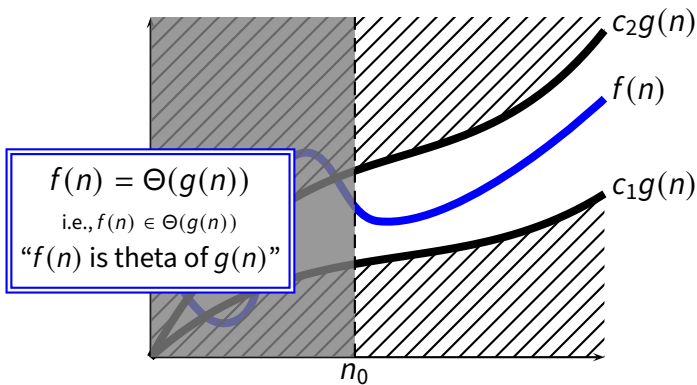


- Given a function $g(n)$, we define the *family of functions* $\Theta(g(n))$



$$\Theta(g(n)) = \{f(n) : \exists c_1 > 0, \exists c_2 > 0, \exists n_0 > 0 \\ : 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0\}$$

- Given a function $g(n)$, we define the *family of functions* $\Theta(g(n))$



$$\Theta(g(n)) = \{f(n) : \exists c_1 > 0, \exists c_2 > 0, \exists n_0 > 0$$
$$: 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0\}$$

- The O -notation defines an upper bound that might not be *asymptotically tight*

- The O -notation defines an upper bound that might not be *asymptotically tight*

E.g.,

$n \log n = O(n^2)$ is not asymptotically tight

so, $n \log n = O(n^2)$ but $n \log n \neq \Theta(n^2)$

$n^2 - n + 10 = O(n^2)$ is asymptotically tight

- The O -notation defines an upper bound that might not be ***asymptotically tight***

E.g.,

$n \log n = O(n^2)$ is not asymptotically tight

so, $n \log n = O(n^2)$ but $n \log n \neq \Theta(n^2)$

$n^2 - n + 10 = O(n^2)$ is asymptotically tight

- We use the o -notation to denote upper bounds that are *not* asymptotically tight. So, given a function $g(n)$, we define the family of functions $o(g(n))$

$$o(g(n)) = \{f(n) : \forall c > 0, \exists n_0 > 0 \\ : 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}$$

- The O -notation defines an upper bound that might not be ***asymptotically tight***

E.g.,

$n \log n = O(n^2)$ is not asymptotically tight

so, $n \log n = O(n^2)$ but $n \log n \neq \Theta(n^2)$

$n^2 - n + 10 = O(n^2)$ is asymptotically tight

- We use the o -notation to denote upper bounds that are *not* asymptotically tight. So, given a function $g(n)$, we define the family of functions $o(g(n))$

$$o(g(n)) = \{f(n) : \forall c > 0, \exists n_0 > 0 \\ : 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}$$

$f(n)$ is below $g(n)$ for all sufficiently large n , and for ALL scaling factors c

- The Ω -notation defines a lower bound that might not be *asymptotically tight*

- The Ω -notation defines a lower bound that might not be *asymptotically tight*

E.g.,

$2^n = \Omega(n \log n)$ is not asymptotically tight

$n + 4n \log n = \Omega(n \log n)$ is asymptotically tight

- The Ω -notation defines a lower bound that might not be *asymptotically tight*

E.g.,

$2^n = \Omega(n \log n)$ is not asymptotically tight

$n + 4n \log n = \Omega(n \log n)$ is asymptotically tight

- We use the ω -notation to denote lower bounds that are *not* asymptotically tight. So, given a function $g(n)$, we define the family of functions $\omega(g(n))$

$$\omega(g(n)) = \{f(n) : \forall c > 0, \exists n_0 > 0 \\ : 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\}$$

- The Ω -notation defines a lower bound that might not be *asymptotically tight*

E.g.,

$2^n = \Omega(n \log n)$ is not asymptotically tight

$n + 4n \log n = \Omega(n \log n)$ is asymptotically tight

- We use the ω -notation to denote lower bounds that are *not* asymptotically tight. So, given a function $g(n)$, we define the family of functions $\omega(g(n))$

$$\omega(g(n)) = \{f(n) : \forall c > 0, \exists n_0 > 0 \\ : 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\}$$

$f(n)$ **is above** $g(n)$ **for all sufficiently large** n , **and for ALL scaling factors** c

Characterizing *Unknown* Functions

- The idea of the O , Ω , and Θ notations is very often to characterize a function that is *not completely known*

Characterizing *Unknown* Functions

- The idea of the O , Ω , and Θ notations is very often to characterize a function that is *not completely known*

Example:

Let $\pi(n)$ be the number of *primes* less than or equal to n

What is the asymptotic behavior of $\pi(n)$?

Characterizing *Unknown* Functions

- The idea of the O , Ω , and Θ notations is very often to characterize a function that is *not completely known*

Example:

Let $\pi(n)$ be the number of *primes* less than or equal to n

What is the asymptotic behavior of $\pi(n)$?

▶ $\pi(n) = O(n)$

trivial ***upper bound***

Characterizing *Unknown* Functions

- The idea of the O , Ω , and Θ notations is very often to characterize a function that is *not completely known*

Example:

Let $\pi(n)$ be the number of *primes* less than or equal to n

What is the asymptotic behavior of $\pi(n)$?

▶ $\pi(n) = O(n)$

trivial ***upper bound***

▶ $\pi(n) = \Omega(1)$

trivial ***lower bound***

Characterizing *Unknown* Functions

- The idea of the O , Ω , and Θ notations is very often to characterize a function that is *not completely known*

Example:

Let $\pi(n)$ be the number of *primes* less than or equal to n

What is the asymptotic behavior of $\pi(n)$?

- ▶ $\pi(n) = O(n)$ trivial **upper bound**
- ▶ $\pi(n) = \Omega(1)$ trivial **lower bound**
- ▶ $\pi(n) = \Theta(n/\log n)$ non-trivial **tight bound**

Characterizing *Unknown* Functions

- The idea of the O , Ω , and Θ notations is very often to characterize a function that is *not completely known*

Example:

Let $\pi(n)$ be the number of *primes* less than or equal to n

What is the asymptotic behavior of $\pi(n)$?

- ▶ $\pi(n) = O(n)$ trivial **upper bound**
- ▶ $\pi(n) = \Omega(1)$ trivial **lower bound**
- ▶ $\pi(n) = \Theta(n/\log n)$ non-trivial **tight bound**

In fact, the fundamental *prime number theorem* says that

$$\lim_{n \rightarrow \infty} \frac{\pi(n) \ln n}{n} = 1$$

- $T(n) = n^2 + 10n + 100$

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$

■ $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$

■ $T(n) = n + 10 \log n$

■ $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$

■ $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$

■ $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$

■ $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$

■ $T(n) = n \log n + n\sqrt{n}$

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \Rightarrow T(n) = \Theta(n\sqrt{n})$

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \Rightarrow T(n) = \Theta(n\sqrt{n})$
- $T(n) = 2^{\frac{n}{6}} + n^7$

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \Rightarrow T(n) = \Theta(n\sqrt{n})$
- $T(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow T(n) = \Theta(2^{\frac{n}{6}})$

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \Rightarrow T(n) = \Theta(n\sqrt{n})$
- $T(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow T(n) = \Theta(2^{\frac{n}{6}})$
- $T(n) = \frac{10+n}{n^2}$

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \Rightarrow T(n) = \Theta(n\sqrt{n})$
- $T(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow T(n) = \Theta(2^{\frac{n}{6}})$
- $T(n) = \frac{10+n}{n^2} \Rightarrow T(n) = \Theta(\frac{1}{n})$

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \Rightarrow T(n) = \Theta(n\sqrt{n})$
- $T(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow T(n) = \Theta(2^{\frac{n}{6}})$
- $T(n) = \frac{10+n}{n^2} \Rightarrow T(n) = \Theta(\frac{1}{n})$
- $T(n) =$ complexity of **PINGALA-INC**

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \Rightarrow T(n) = \Theta(n\sqrt{n})$
- $T(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow T(n) = \Theta(2^{\frac{n}{6}})$
- $T(n) = \frac{10+n}{n^2} \Rightarrow T(n) = \Theta(\frac{1}{n})$
- $T(n) = \text{complexity of PINGALA-INC} \Rightarrow T(n) = \Theta(n)$

- $T(n) = n^2 + 10n + 100 \Rightarrow T(n) = \Theta(n^2)$
- $T(n) = n + 10 \log n \Rightarrow T(n) = \Theta(n)$
- $T(n) = n \log n + n\sqrt{n} \Rightarrow T(n) = \Theta(n\sqrt{n})$
- $T(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow T(n) = \Theta(2^{\frac{n}{6}})$
- $T(n) = \frac{10+n}{n^2} \Rightarrow T(n) = \Theta(\frac{1}{n})$
- $T(n) = \text{complexity of PINGALA-INC} \Rightarrow T(n) = \Theta(n)$
- We characterize the behavior of $T(n)$ *in the limit*
- The Θ -notation is an ***asymptotic notation***

- $f(n) = n^2 + 10n + 100$

- $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2)$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n}$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n} \Rightarrow f(n) = O(n^2)$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n} \Rightarrow f(n) = O(n^2)$

■ $f(n) = 2^{\frac{n}{6}} + n^7$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n} \Rightarrow f(n) = O(n^2)$

■ $f(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow f(n) = O((1.5)^n)$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n} \Rightarrow f(n) = O(n^2)$

■ $f(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow f(n) = O((1.5)^n)$

■ $f(n) = \frac{10+n}{n^2}$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n} \Rightarrow f(n) = O(n^2)$

■ $f(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow f(n) = O((1.5)^n)$

■ $f(n) = \frac{10+n}{n^2} \Rightarrow f(n) = O(1)$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n} \Rightarrow f(n) = O(n^2)$

■ $f(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow f(n) = O((1.5)^n)$

■ $f(n) = \frac{10+n}{n^2} \Rightarrow f(n) = O(1)$

■ $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n} \Rightarrow f(n) = O(n^2)$

■ $f(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow f(n) = O((1.5)^n)$

■ $f(n) = \frac{10+n}{n^2} \Rightarrow f(n) = O(1)$

■ $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$

■ $f(n) = \Theta(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$

■ $f(n) = n^2 + 10n + 100 \Rightarrow f(n) = O(n^2) \Rightarrow f(n) = O(n^3)$

■ $f(n) = n + 10 \log n \Rightarrow f(n) = O(2^n)$

■ $f(n) = n \log n + n\sqrt{n} \Rightarrow f(n) = O(n^2)$

■ $f(n) = 2^{\frac{n}{6}} + n^7 \Rightarrow f(n) = O((1.5)^n)$

■ $f(n) = \frac{10+n}{n^2} \Rightarrow f(n) = O(1)$

■ $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$

■ $f(n) = \Theta(g(n)) \wedge g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$

■ $f(n) = O(g(n)) \wedge g(n) = \Theta(h(n)) \Rightarrow f(n) = O(h(n))$

- $n^2 - 10n + 100 = O(n \log n)$?

- $n^2 - 10n + 100 = O(n \log n)$? NO

■ $n^2 - 10n + 100 = O(n \log n)$? NO

■ $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$?

■ $n^2 - 10n + 100 = O(n \log n)$? NO

■ $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$?

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2 \log_2 n})$?

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2 \log_2 n})$? YES

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2 \log_2 n})$? YES
- $f(n) = O(2^n) \Rightarrow f(n) = \Theta(n^2)$?

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2 \log_2 n})$? YES
- $f(n) = O(2^n) \Rightarrow f(n) = \Theta(n^2)$? NO

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2 \log_2 n})$? YES
- $f(n) = O(2^n) \Rightarrow f(n) = \Theta(n^2)$? NO
- $\sqrt{n} = O(\log^2 n)$?

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2 \log_2 n})$? YES
- $f(n) = O(2^n) \Rightarrow f(n) = \Theta(n^2)$? NO
- $\sqrt{n} = O(\log^2 n)$? NO

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2 \log_2 n})$? YES
- $f(n) = O(2^n) \Rightarrow f(n) = \Theta(n^2)$? NO
- $\sqrt{n} = O(\log^2 n)$? NO
- $n^2 + (1.5)^n = O(2^{\frac{n}{2}})$?

- $n^2 - 10n + 100 = O(n \log n)$? NO
- $f(n) = O(2^n) \Rightarrow f(n) = O(n^2)$? NO
- $f(n) = \Theta(2^n) \Rightarrow f(n) = O(n^2 2^n)$? YES
- $f(n) = \Theta(n^2 2^n) \Rightarrow f(n) = O(2^{n+2 \log_2 n})$? YES
- $f(n) = O(2^n) \Rightarrow f(n) = \Theta(n^2)$? NO
- $\sqrt{n} = O(\log^2 n)$? NO
- $n^2 + (1.5)^n = O(2^{\frac{n}{2}})$? NO

- So, what is the complexity of **FINDEQUALS**?

```
FINDEQUALS(A)
1  for  $i = 1$  to  $\text{length}(A) - 1$ 
2      for  $j = i + 1$  to  $\text{length}(A)$ 
3          if  $A[i] == A[j]$ 
4              return TRUE
5  return FALSE
```

- So, what is the complexity of **FINDEQUALS**?

```
FINDEQUALS(A)
1  for  $i = 1$  to  $\text{length}(A) - 1$ 
2      for  $j = i + 1$  to  $\text{length}(A)$ 
3          if  $A[i] == A[j]$ 
4              return TRUE
5  return FALSE
```

$$T(n) = \Theta(n^2)$$

- ▶ $n = \text{length}(A)$ is the **size of the input**
- ▶ we measure the **worst-case complexity**

Θ , O , and Ω as Relations

- *Theorem:* for any two functions $f(n)$ and $g(n)$,
 $f(n) = \Omega(g(n)) \wedge f(n) = O(g(n)) \Leftrightarrow f(n) = \Theta(g(n))$

Θ , O , and Ω as Relations

- *Theorem:* for any two functions $f(n)$ and $g(n)$,
 $f(n) = \Omega(g(n)) \wedge f(n) = O(g(n)) \Leftrightarrow f(n) = \Theta(g(n))$
- The Θ -notation, Ω -notation, and O -notation can be viewed as the “asymptotic” $=$, \geq , and \leq relations for functions, respectively

Θ , O , and Ω as Relations

- *Theorem:* for any two functions $f(n)$ and $g(n)$,
 $f(n) = \Omega(g(n)) \wedge f(n) = O(g(n)) \Leftrightarrow f(n) = \Theta(g(n))$
- The Θ -notation, Ω -notation, and O -notation can be viewed as the “asymptotic” $=$, \geq , and \leq relations for functions, respectively
- The above theorem can be interpreted as saying

$$f \geq g \wedge f \leq g \Leftrightarrow f = g$$

Θ , O , and Ω as Relations

- *Theorem:* for any two functions $f(n)$ and $g(n)$,
 $f(n) = \Omega(g(n)) \wedge f(n) = O(g(n)) \Leftrightarrow f(n) = \Theta(g(n))$
- The Θ -notation, Ω -notation, and O -notation can be viewed as the “asymptotic” $=$, \geq , and \leq relations for functions, respectively
- The above theorem can be interpreted as saying

$$f \geq g \wedge f \leq g \Leftrightarrow f = g$$

- When $f(n) = O(g(n))$ we say that $g(n)$ is an **upper bound** for $f(n)$, and that $g(n)$ **dominates** $f(n)$

Θ , O , and Ω as Relations

- *Theorem:* for any two functions $f(n)$ and $g(n)$,
 $f(n) = \Omega(g(n)) \wedge f(n) = O(g(n)) \Leftrightarrow f(n) = \Theta(g(n))$
- The Θ -notation, Ω -notation, and O -notation can be viewed as the “asymptotic”
 $=$, \geq , and \leq relations for functions, respectively
- The above theorem can be interpreted as saying

$$f \geq g \wedge f \leq g \Leftrightarrow f = g$$

- When $f(n) = O(g(n))$ we say that $g(n)$ is an **upper bound** for $f(n)$, and that $g(n)$ **dominates** $f(n)$
- When $f(n) = \Omega(g(n))$ we say that $g(n)$ is a **lower bound** for $f(n)$

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unsecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unspecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

- Examples

$$n^2 + 4n - 1 = n^2 + \Theta(n)?$$

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unspecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

- Examples

$$n^2 + 4n - 1 = n^2 + \Theta(n)? \quad \text{YES}$$

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unspecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

- Examples

$$n^2 + 4n - 1 = n^2 + \Theta(n)? \quad \text{YES}$$

$$n^2 + \Omega(n) - 1 = O(n^2)?$$

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unspecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

- Examples

$$n^2 + 4n - 1 = n^2 + \Theta(n)? \quad \text{YES}$$

$$n^2 + \Omega(n) - 1 = O(n^2)? \quad \text{NO}$$

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unspecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

- Examples

$$n^2 + 4n - 1 = n^2 + \Theta(n)? \quad \text{YES}$$

$$n^2 + \Omega(n) - 1 = O(n^2)? \quad \text{NO}$$

$$n^2 + O(n) - 1 = O(n^2)?$$

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unspecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

- Examples

$$n^2 + 4n - 1 = n^2 + \Theta(n)? \quad \text{YES}$$

$$n^2 + \Omega(n) - 1 = O(n^2)? \quad \text{NO}$$

$$n^2 + O(n) - 1 = O(n^2)? \quad \text{YES}$$

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unspecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

- Examples

$$n^2 + 4n - 1 = n^2 + \Theta(n)? \quad \text{YES}$$

$$n^2 + \Omega(n) - 1 = O(n^2)? \quad \text{NO}$$

$$n^2 + O(n) - 1 = O(n^2)? \quad \text{YES}$$

$$n \log n + \Theta(\sqrt{n}) = O(n\sqrt{n})?$$

Θ , O , and Ω as Anonymous Functions

- We can use the Θ -, O -, and Ω -notation to represent anonymous (unknown or unspecified) functions
E.g.,

$$f(n) = 10n^2 + O(n)$$

means that $f(n)$ is equal to $10n^2$ plus a function we don't know or we don't care to know that is asymptotically at most linear in n .

- Examples

$$n^2 + 4n - 1 = n^2 + \Theta(n)? \quad \text{YES}$$

$$n^2 + \Omega(n) - 1 = O(n^2)? \quad \text{NO}$$

$$n^2 + O(n) - 1 = O(n^2)? \quad \text{YES}$$

$$n \log n + \Theta(\sqrt{n}) = O(n\sqrt{n})? \quad \text{YES}$$

