

# EECS 3101

Prof. Andy Mirzaian



Computer Science  
and Engineering

120 Campus Walk

Assignment Project Exam Help

# Machine Model

<https://powcoder.com>

Add WeChat powcoder

# &

# Time Complexity

# STUDY MATERIAL:

- [CLRS] chapters 1, 2, 3
  - Lecture Note 2
- Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Example

**Time complexity** shows dependence of algorithm's running time on input size.

Let's assume: Computer speed =  $10^6$  IPS,  
Input: a data base of size  $n = 10^6$

Time Complexity	Execution time
$n$	1 sec.
$n \log n$	20 sec.
$n^2$	12 days
$2^n$	40 quadrillion ( $10^{15}$ ) years

# Machine Model

- **Algorithm Analysis:**

- should reveal intrinsic properties of the algorithm itself.
- should not depend on any computing platform, programming language, compiler, computer speed, etc.

- **Random Access machine (RAM):**

an idealized computer model

- **Elementary steps:**

- arithmetic:  $+$   $-$   $\times$   $\div$
- logic:  $\text{and}$   $\text{or}$   $\text{not}$
- comparison:  $=$   $<$   $>$   $\neq$   $\leq$   $\geq$
- assigning a value to a scalar variable:  $\leftarrow$
- input/output a scalar variable
- following a pointer or array indexing
- on rare occasions:  $\sqrt{\quad}$ ,  $\sin$ ,  $\cos$ , ...

# Time Complexity

- **Time complexity** shows dependence of algorithm's running time on input size.
  - Worst-case
  - Average or expected-case
  - Amortized (studied in EECS 4101)
- **What is it good for?**
  - Tells us how efficient our design is before its costly implementation.
  - Reveals inefficiency bottlenecks in the algorithm.
  - Can use it to compare efficiency of different algorithms that solve the same problem.
  - Is a tool to figure out the true complexity of the problem itself!  
How fast is the “fastest” algorithm for the problem?
  - Helps us classify problems by their time complexity.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Input size & Cost measure

## ▪ Input size:

- **Bit size:** # bits in the input.

This is often cumbersome but sometimes necessary when running time depends on numeric data values, e.g., factoring an  $n$ -bit integer.

- **Combinatorial size:** e.g., # items in an array, a list, a tree, # vertices and # edges in a graph, ...  
<https://powcoder.com>

Add WeChat powcoder

## ▪ Cost measure:

- **Bit cost:** charge one unit of time to each bit operation
- **Arithmetic cost:** charge one unit of time to each elementary RAM step

## ▪ Complexity:

- **Bit complexity**
- **Arithmetic complexity**

# Input size & Cost measure

## ■ Input size:

- **Bit size:** # bits in the input.

This is often cumbersome but sometimes necessary when running time depends on numeric data values, e.g., factoring an  $n$ -bit integer.

- **Combinatorial size:** e.g., # items in an array, a list, a tree, # vertices and # edges in a graph, ...  
<https://powcoder.com>

## ■ Cost measure:

- **Bit cost:** charge one

- **Arithmetic cost:** charge one M step

## ■ Complexity:

- **Bit complexity**
- **Arithmetic complexity**

We choose these unless stated otherwise

# Time Complexity Analysis

**Worst-case** time complexity is a function of input size

$$T: \mathcal{N} \rightarrow \mathbb{R}^+$$

$T(n)$  = **maximum** # steps by the algorithm on any input of size  $n$ .

```
Algorithm InversionCount(A[1..n])  
  count  $\leftarrow$  0  
  for  $i \leftarrow 1 \dots n$  do  
    for  $j \leftarrow i+1 \dots n$  do  
      if  $A[i] > A[j]$  then count++  
  return count  
end
```

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

$$T(n) = 3n^2 + 6n + 4$$

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

Incidentally, this  
can be improved to  
 $\Theta(n \log n)$   
time by  
divide-&-conquer

Which line do you prefer?  
The 2<sup>nd</sup> is simpler and  
platform independent.



# Caution!

```
Algorithm SUM(A[1..n])  
  S ← 0  
  for i ← 1 .. n do  
    S ← S + A[i]  
  return S  
end
```

Worst Case:  $\Theta(n)$  time

```
Algorithm PRIME(P)      § integer P > 1  
  for i ← 2 .. P - 1 do  
    if P mod i = 0 then return NO  
  return YES  
end
```

Worst Case:  $\Theta(P)$  time. Linear?

<https://powcoder.com> #input bits  $n \approx \log P$   
LINEAR  $\Rightarrow P \approx 2^n$

Add WeChat [powcoder](https://powcoder.com)  $T(n) = \Theta(P) = \Theta(2^n)$ .  
EXPONENTIAL!

**PRIMALITY TESTING:** used in cryptography ...

It was a long standing open problem whether there is any deterministic algorithm that solves this problem in time polynomial in the input bit size. This was eventually answered affirmatively by:

M. Agrawal, N. Kayal, N. Saxena, "PRIMES is in P,"  
Annals of Mathematics 160, pp: 781-793, 2004.

# Asymptotic Notations

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

$$T(n) = \Theta(f(n))$$

$$T(n) = 23 n^3 + 5 n^2 \log n + 7 n \log^2 n + 4 \log n + 6.$$

drop constant  
multiplicative factor

drop lower order terms

Assignment Project Exam Help

$$T(n) = \Theta(n^3)$$

<https://powcoder.com>

Add WeChat powcoder

**Why do we want to do this?**

1. Asymptotically (at very large values of  $n$ ) the leading term largely determines function behaviour.
2. With a new computer technology (say, 10 times faster) the leading coefficient will change (be divided by 10). So, that coefficient is technology dependent any way!
3. This simplification is still capable of distinguishing between important but distinct complexity classes, e.g., linear vs. quadratic, or polynomial vs exponential.

# Asymptotic Notations: $\Theta$ $O$ $\Omega$ $o$ $\omega$

Rough, intuitive meaning worth remembering:

Theta	$f(n) = \Theta(g(n))$	$f(n) \approx c g(n)$
Big Oh	$f(n) = O(g(n))$	$f(n) \leq c g(n)$
Big Omega	$f(n) = \Omega(g(n))$	$f(n) \geq c g(n)$
Little Oh	$f(n) = o(g(n))$	$f(n) \ll c g(n)$
Little Omega	$f(n) = \omega(g(n))$	$f(n) \gg c g(n)$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Asymptotics by ratio limit

$L = \lim_{n \rightarrow \infty} f(n)/g(n)$ . If  $L$  exists, then:

Theta	$f(n) = \Theta(g(n))$	$0 < L < \infty$
Big Oh	$f(n) = O(g(n))$	$L < \infty$
Big Omega	$f(n) = \Omega(g(n))$	$0 < L$
Little Oh	$f(n) = o(g(n))$	$L = 0$
Little Omega	$f(n) = \omega(g(n))$	$L = \infty$

## $\forall$ & $\exists$ quantifiers

**Logic:** “and” commutes with “and”, “or” commutes with “or”,  
but “and” & “or” do not commute.

Similarly, same type quantifiers commute, but different types do not:

$$\exists x \exists y: P(x,y) = \exists y \exists x: P(x,y) = \exists x,y: P(x,y)$$

$$\forall x \forall y: P(x,y) = \forall y \forall x: P(x,y) = \forall x,y: P(x,y)$$

$$\forall x \exists y: P(x,y) \neq \exists y \forall x: P(x,y)$$

<https://powcoder.com>

Counter-example for the 3<sup>rd</sup>:

LHS: for every person  $x$ , there is a date  $y$ , such that  $x$  is born on date  $y$ .

RHS: there is a date  $y$ , such that for every person  $x$ ,  $x$  is born on date  $y$ .

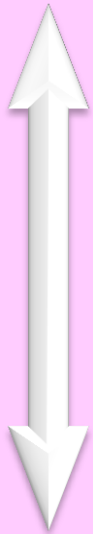
Each person has a birth date, but not every person is born on the same date!

Give natural examples for the following to show their differences:

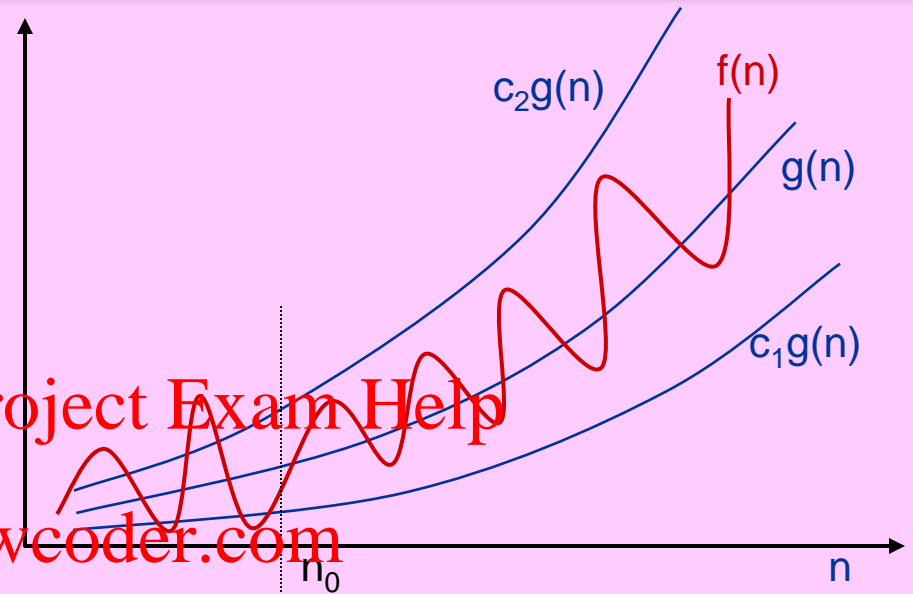
1.  $\forall x \exists y \forall z: P(x,y,z)$
2.  $\forall x \forall z \exists y: P(x,y,z)$
3.  $\exists y \forall x \forall z: P(x,y,z)$

# Theta : Asymptotic Tight Bound

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



$$\underbrace{\exists c_1, c_2, n_0 > 0}_{\in \mathbb{R}^+} : \forall n \geq n_0, \quad c_1 g(n) \leq f(n) \leq c_2 g(n).$$



# Theta : Example

$$f(n) = \Theta(g(n)) \iff \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$f(n) = 23n^3 - 10n^2 \log n + 7n + 6$$

$$f(n) = [23 - (10 \log n)/n + 7/n^2 + 6/n^3]n^3 \quad \text{factor out leading term}$$

$$\forall n \geq 10 : f(n) \leq (23 + 0 + 7/100 + 6/1000)n^3$$

$$= (23 + 0 + 0.07 + 0.006)n^3$$

$$= 23.076n^3 < 24n^3$$

$$\forall n \geq 10 : f(n) \geq (23 - \log 10 + 0 + 0)n^3 > (23 - \log 16)n^3 = 19n^3$$

$$\forall n \geq 10 : 19n^3 \leq f(n) \leq 24n^3$$

$$(\text{Take } n_0 = 10, c_1 = 19, c_2 = 24, g(n) = n^3)$$

$$f(n) = \Theta(n^3)$$

Now you see why we can drop lower order terms & the constant multiplicative factor.



# Big Oh: Asymptotic Upper Bound

$$f(n) = O(g(n))$$

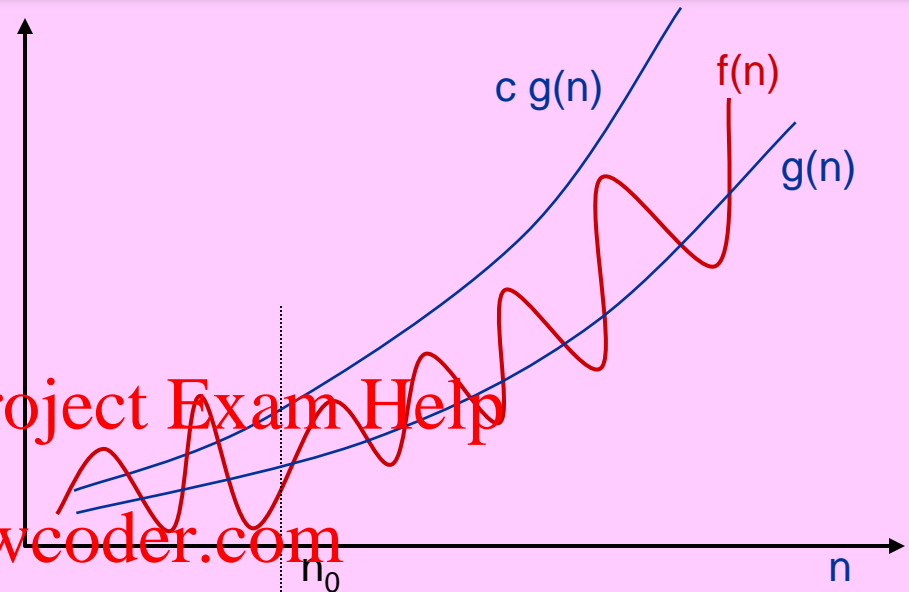


$$\underbrace{\exists c, n_0 > 0}_{\in \mathbb{R}^+} : \forall n \geq n_0, \quad f(n) \leq c g(n).$$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



# Big Omega : Asymptotic Lower Bound

$$f(n) = \Omega(g(n))$$

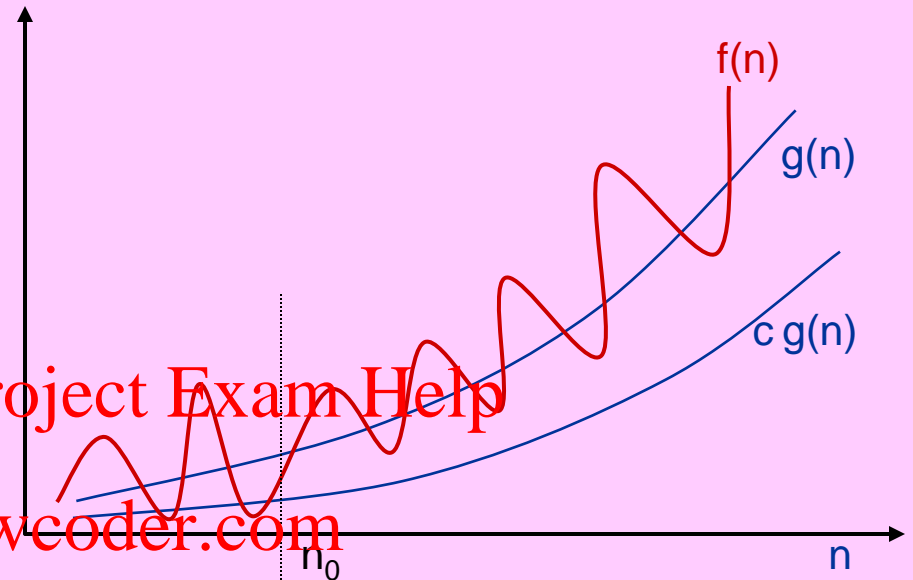


$$\underbrace{\exists c, n_0}_{\in \mathbb{R}^+} > 0 : \forall n \geq n_0, \quad cg(n) \leq f(n).$$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



# Little oh : Non-tight Asymptotic Upper Bound

$$f(n) = o(g(n))$$



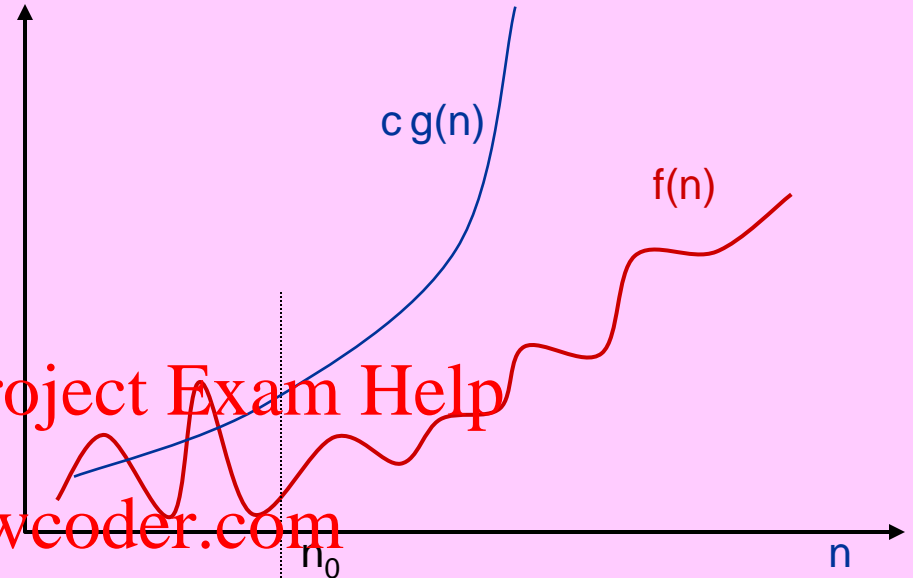
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

$$\forall c > 0, \exists n_0 > 0 : \forall n \geq n_0, f(n) < c g(n).$$

No matter how small  $\in \mathbb{R}^+$



# Little omega : Non-tight Asymptotic Lower Bound

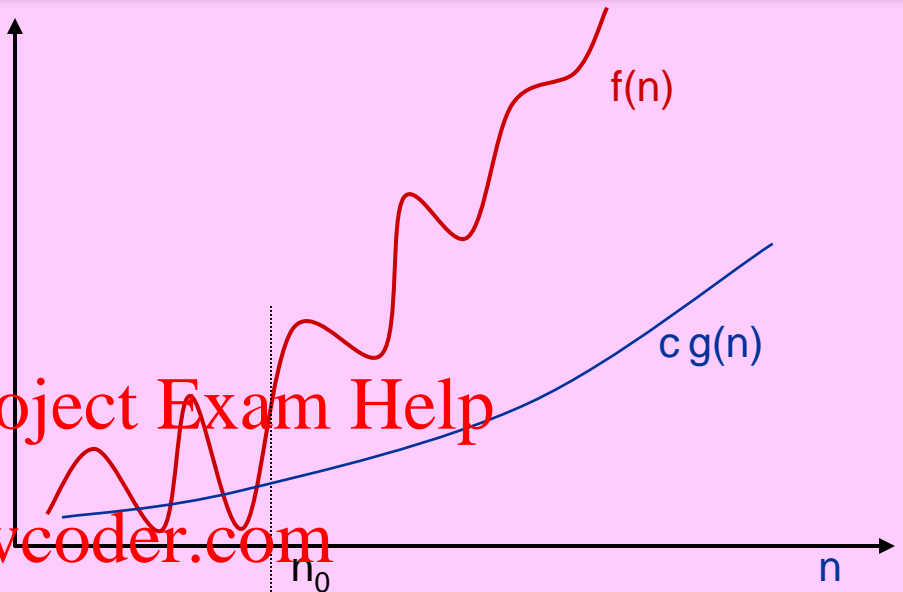
$$f(n) = \omega(g(n))$$



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder



$$\forall c > 0, \exists n_0 > 0 : \forall n \geq n_0, f(n) > c g(n).$$

No matter how large  $\in \mathbb{R}^+$

# Definitions of Asymptotic Notations

$$f(n) = \Theta(g(n)) \quad \exists c_1, c_2 > 0, \exists n_0 > 0: \forall n \geq n_0, c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$$f(n) = O(g(n)) \quad \exists c > 0, \exists n_0 > 0: \forall n \geq n_0, f(n) \leq c g(n)$$

$$f(n) = \Omega(g(n)) \quad \exists c > 0, \exists n_0 > 0: \forall n \geq n_0, c g(n) \leq f(n)$$

<https://powcoder.com>

Add WeChat powcoder

$$f(n) = o(g(n)) \quad \forall c > 0, \exists n_0 > 0: \forall n \geq n_0, f(n) < c g(n)$$

$$f(n) = \omega(g(n)) \quad \forall c > 0, \exists n_0 > 0: \forall n \geq n_0, c g(n) < f(n)$$

# Example Proof

$$f(n) = o(g(n))$$

$$\forall c > 0, \exists n_0 > 0: \forall n \geq n_0, f(n) < c g(n)$$

**CLAIM:**  $n^2 \neq o(n)$ .

**Proof:** Need to show:  $\neg (\forall c > 0, \exists n_0 > 0: \forall n \geq n_0 (n^2 < c n))$ .

**Move  $\neg$  inside:**

$$\Leftrightarrow (\exists c > 0, \forall n_0 > 0, \exists n \geq n_0 (n^2 \geq c n)).$$

**Work inside-out:**

The diagram illustrates the logical structure of the proof using nested braces and annotations:

- The innermost expression is  $n \geq c$ .
- A brace groups  $n \geq c$  with the condition  $n \geq n_0$  to form  $\exists n (n \geq n_0 \text{ and } n \geq c)$ .
- A larger brace groups the entire expression  $\exists n (n \geq n_0 \text{ and } n \geq c)$  with the quantifier  $\forall n_0 > 0$  to form  $\forall n_0 > 0, \exists n \geq \max\{n_0, c\}$ .

Annotations in red text are placed near the braces:

- <https://powcoder.com> is written above the innermost brace.
- Add WeChat is written to the left of the middle brace.
- powcoder is written to the right of the middle brace.

**Now browse  
outside-in.**

**Everything OK!**

choose  $c=1$ , then  $\forall n_0 > 0$ , choose  $n = \max\{n_0, c\}$

# Any short cuts?

$$f(n) = o(g(n))$$

$$\forall c > 0, \exists n_0 > 0: \forall n \geq n_0, f(n) < c g(n)$$

Assignment Project Exam Help

<https://powcoder.com>

If you are a starter on these notations, you are advised to exercise your predicate logic muscles for a while to get the hang of it!

Add WeChat powcoder

However, in the next few slides, we will study some **FACTS & RULES** that help us manipulate and reason about asymptotic notations in most common situations with much ease.

# A Classification of Functions

**Recall:**  $X < Y \Leftrightarrow \log X < \log Y$ . Also,  $\log(X^Y) = Y \log X$ .

- **$o(1)$**  e.g.,  $1/\omega(1)$ ,  $(\log n)^{-3}$ ,  $n^{-2}$ ,  $2^{-3n}$ ,  $0$ .
- **$O(1)$**  asymptotically upper-bounded by a constant
- **$\Theta(1)$**  e.g.,  $f(n) = 5 + 1/n$ ,  $(2n + 1)/(6n + \log n)$
- **Poly-Logarithmic:** e.g.,  $\log^3 n$   
 $f(n) = \log^{\Theta(1)} n$  <https://powcoder.com>
- **Polynomial:** Add WeChat powcoder  
 $f(n) = n^{\Theta(1)}$  e.g.,  $f(n) = 3n + 1$ ,  $n^{1.5} \log^{-3} n$ ,  $n^{100}$ .  
 $\log f(n) = \Theta(1) \log n = \Theta(\log n)$
- **Super-Polynomial but Sub-Exponential:**  
 $f(n) = n^{\omega(1)} \cap 2^{o(n)}$  e.g.,  $f(n) = n^{\log n}$ ,  $2^{\sqrt{n}}$ ,  $2^{\frac{n}{\log n}}$   
 $\log f(n) = \omega(\log n) \cap o(n)$
- **Exponential or more:**  
 $f(n) = 2^{\Omega(n)}$  e.g.,  $f(n) = 2^{3n \log n}$ ,  $2^{n^2}$ ,  $2^{2^n}$ .  
 $\log f(n) = \Omega(n)$



# Order of Growth Rules

1. Below  $X \ll Y$  means  $X = o(Y)$  :

$o(1) \ll \Theta(1) \ll \text{Poly-Log} \ll \text{Polynomial} \ll \text{Exponential or more}$

$o(1) \ll \Theta(1) \ll \log^{\Theta(1)} n \ll n^{\Theta(1)} \ll 2^{\Omega(n)}$

<https://powcoder.com>

**Examples:** Add WeChat powcoder

$$\log^{100} n = o(n^{0.01})$$

$$n^{100} = o((1.1)^{3n+1})$$

# Asymptotic Relation Rules

2. **Skew Symmetric:**  $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$   
 $f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$
3.  $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \ \& \ f(n) = \Omega(g(n))$   
 $\Leftrightarrow f(n) = O(g(n)) \ \& \ g(n) = O(f(n))$
4. **Symmetric** [only Assignment Project Exam Help]
5. **Reflexive:**  $f(n) = \Theta(f(n)), f(n) = O(f(n)), f(n) = \Omega(f(n)),$   
 $f(n) \neq o(f(n)), f(n) \neq \omega(f(n)).$
6. **Transitive:**  $f(n) = O(g(n)) \ \& \ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$   
works for  $\Theta, \Omega, o, \omega$  too.
7.  $f(n) = o(g(n)) \Rightarrow f(n) = O(g(n)) \ \& \ f(n) \neq \Omega(g(n)) \Rightarrow f(n) \neq \Theta(g(n))$
8.  $f(n) = \omega(g(n)) \Rightarrow f(n) = \Omega(g(n)) \ \& \ f(n) \neq O(g(n)) \Rightarrow f(n) \neq \Theta(g(n))$
9.  $f(n) = g(n) + o(g(n)) \Rightarrow f(n) = \Theta(g(n))$  [can hide lower order terms]

# Arithmetic Rules

10. Given:  $f(n) = O(F(n))$ ,  $g(n) = O(G(n))$ ,  $h(n) = \Theta(H(n))$ , constant  $d > 0$ .

**Sum:**  $f(n) + g(n) = O(F(n)) + O(G(n)) = O(\max\{F(n), G(n)\})$ .

**Product:**  $f(n) \cdot g(n) = O(F(n)) \cdot O(G(n)) = O(F(n) \cdot G(n))$ .  
 $f(n) \cdot h(n) = O(F(n)) \cdot \Theta(H(n)) = O(F(n) \cdot H(n))$ .

**Division:**  $\frac{f(n)}{h(n)} = \frac{O(F(n))}{\Theta(H(n))} = O\left(\frac{F(n)}{H(n)}\right)$

**Power:**  $f(n)^d = O(F(n)^d)$ .

*In addition to  $O$ , these rules also apply to  $\Theta, \Omega, o, \omega$ .*

## Examples:

- $16n^2 + 9n \log n = O(n^2) + O(n \log n) = O(\max\{n^2, n \log n\}) = O(n^2)$ .
- $n^2 = \Theta(n^2)$ ,  $\log n = o(n^{0.01}) \Rightarrow n^2 \log n = o(n^{2.01})$ .
- $\frac{3n+7 \log n}{4n \log n+9} = \frac{\Theta(n)}{\Theta(n \log n)} = \Theta\left(\frac{n}{n \log n}\right) = \Theta\left(\frac{1}{\log n}\right)$ .
- $3n + 1 = \Theta(n) \Rightarrow (3n + 1)^{3.7} = \Theta(n^{3.7})$ .

# Proof of Rule of Sum

$$f(n) = O(F(n)) \ \& \ g(n) = O(G(n)) \implies f(n)+g(n) = O(\max\{F(n), G(n)\})$$

**Proof:** From the premise we have:

$$\exists n_1, c_1 > 0: \forall n \geq n_1, \quad f(n) \leq c_1 F(n)$$

$$\exists n_2, c_2 > 0: \forall n \geq n_2, \quad g(n) \leq c_2 G(n)$$

Now define  $n_0 = \max\{n_1, n_2\} > 0$  &  $c = c_1 + c_2 > 0$ .

We get:

$$\begin{aligned} \forall n \geq n_0, \quad f(n)+g(n) &\leq c_1 F(n) + c_2 G(n) \\ &\leq c_1 \max\{F(n), G(n)\} + c_2 \max\{F(n), G(n)\} \\ &= (c_1 + c_2) \max\{F(n), G(n)\} \\ &= c \max\{F(n), G(n)\} \end{aligned}$$

We have shown:

$$\exists n_0, c > 0: \forall n \geq n_0, \quad f(n) + g(n) \leq c \max\{F(n), G(n)\}.$$

Therefore,  $f(n) + g(n) = O(\max\{F(n), G(n)\})$ .

# Where did we go wrong!

“CLAIM”  $\Theta(n) = \Theta(1)$  .

“Proof”: By repeated application of the max rule of sum:  $\Theta(1) + \Theta(1) = \Theta(1)$  .

$$\begin{aligned}
 \Theta(n) &= \overbrace{\Theta(1) + \Theta(1) + \Theta(1) + \dots + \Theta(1) + \Theta(1) + \underbrace{\Theta(1) + \Theta(1)}}^{n \text{ } \Theta(1)'s} \\
 &= \underbrace{\Theta(1) + \Theta(1) + \Theta(1) + \dots + \Theta(1) + \Theta(1)}_{\Theta(1)} + \underbrace{\Theta(1)}_{\Theta(1)} \\
 &= \Theta(1) + \Theta(1) + \Theta(1) + \dots + \underbrace{\Theta(1) + \Theta(1)}_{\Theta(1)} \\
 &= \dots \underbrace{\Theta(1) + \Theta(1)}_{\Theta(1)} \\
 &= \underbrace{\Theta(1) + \Theta(1)}_{\Theta(1)} \\
 &= \Theta(1)
 \end{aligned}$$

Do you buy this?      Where did we go wrong?

**META RULE:** Within an expression, you can apply the asymptotic simplification rules only a **constant** number of times!

**WHY?**

# Algorithm Complexity

VS

Assignment Project Exam Help

# Problem Complexity

<https://powcoder.com>

Add WeChat powcoder

# Algorithm Time Complexity

$T(n)$  = worst-case time complexity of **algorithm ALG**.

- $T(n) = O(f(n))$ :

You must prove that on **every** input of size  $n$ , for all sufficiently large  $n$ , ALG takes at most  $O(f(n))$  time. That is, even the worst input of size  $n$  cannot force ALG to require more than  $O(f(n))$  time.

Assignment Project Exam Help

<https://powcoder.com>

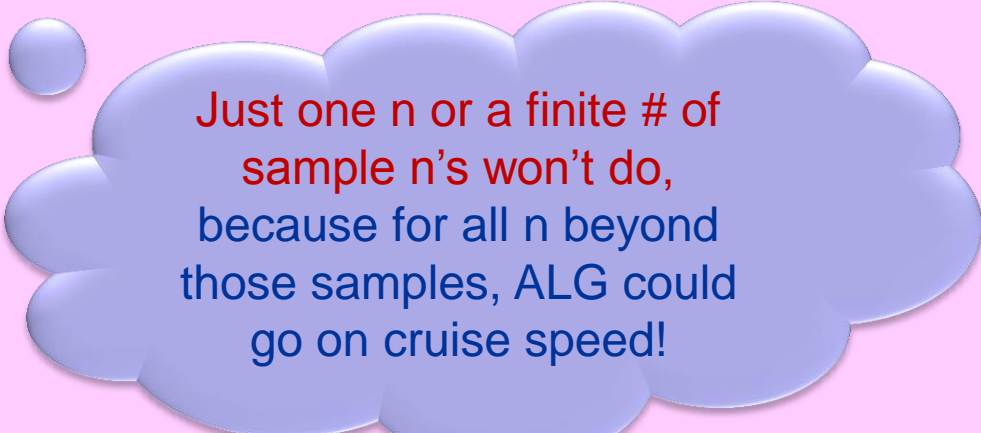
- $T(n) = \Omega(f(n))$ :

Demonstrate the existence of an input instance  $I$  of size  $n$  for infinitely many  $n$ , for which ALG requires at least  $\Omega(f(n))$  time.

Add WeChat powcoder

- $T(n) = \Theta(f(n))$ :

Do both!



Just one  $n$  or a finite # of sample  $n$ 's won't do, because for all  $n$  beyond those samples, ALG could go on cruise speed!

# Problem Time Complexity

**$T(n)$**  = worst-case time complexity of **problem P**  
is the time complexity of the “best” algorithm that solves P.

- **$T(n) = O(f(n))$ :**

Need to demonstrate (the existence of) an algorithm ALG that correctly solves problem P, and that worst-case time complexity of ALG is at most  $O(f(n))$ .

Assignment Project Exam Help

This is usually the much

harder task

- **$T(n) = \Omega(f(n))$ :**

Prove that for **every** algorithm ALG that correctly solves every instance of problem P, worst-case time complexity of ALG must be at least  $\Omega(f(n))$ .

<https://powcoder.com>

Add WeChat powcoder

- **$T(n) = \Theta(f(n))$ :** Do both!



# Example Problem: Sorting

Some sorting algorithms and their worst-case time complexities:

Quick-Sort:	$\Theta(n^2)$
Insertion-Sort:	$\Theta(n^2)$
Selection-Sort:	$\Theta(n^2)$
Merge-Sort:	$\Theta(n \log n)$
Heap-Sort:	$\Theta(n \log n)$

there are infinitely many sorting algorithms!

<https://powcoder.com>

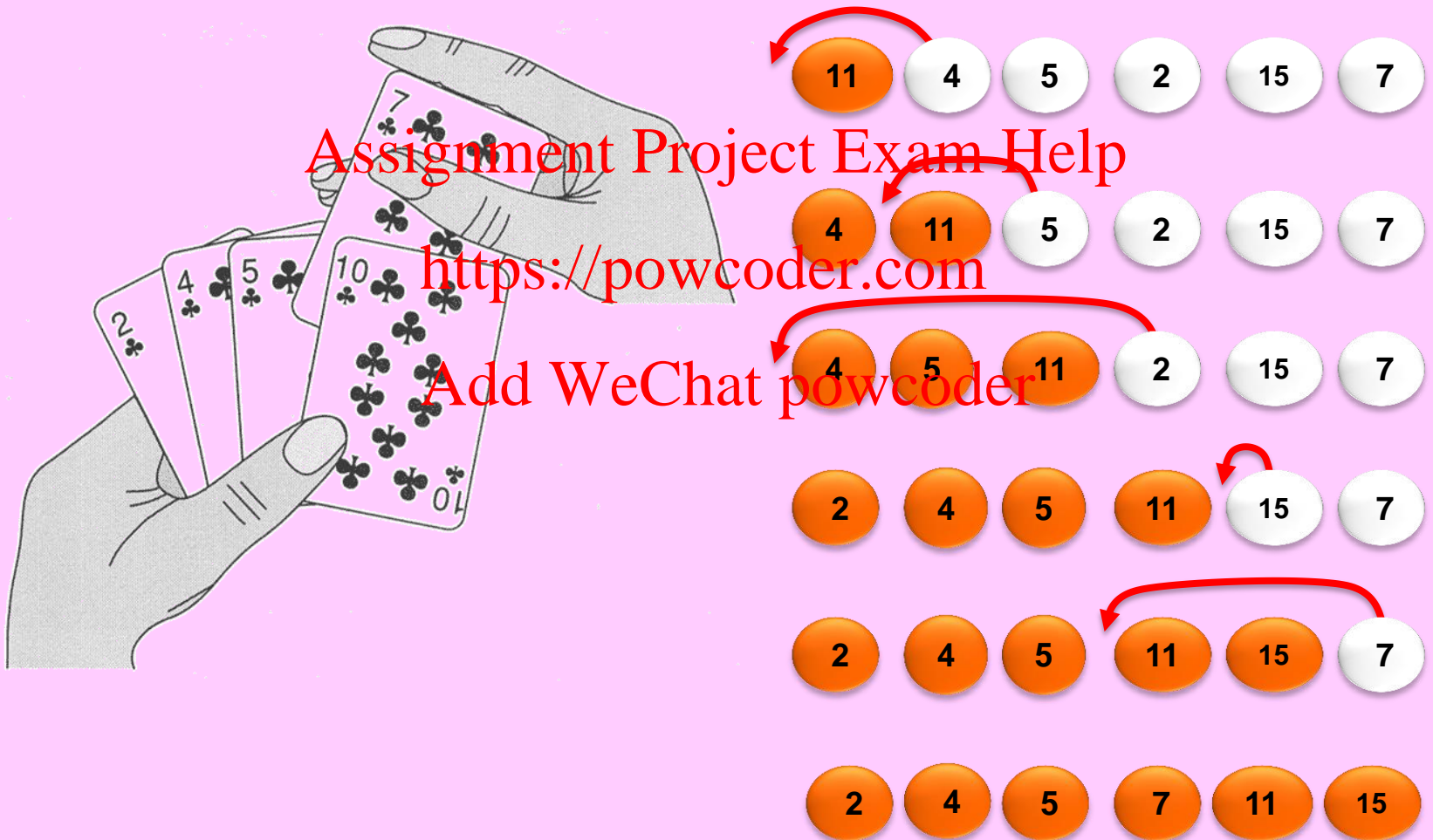
**Shown in Slide 5:** essentially every algorithm that solves the  
**SORTING Problem**  
requires at least  
 $\Omega(n \log n)$   
time in the worst-case.

So, Merge-Sort and Heap-Sort are worst-case optimal, and

**SORTING** complexity is  $\Theta(n \log n)$ .

# Insertion Sort

an incremental algorithm



Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

# Insertion Sort: Time Complexity

**Algorithm InsertionSort(A[1..n])**

Pre-Condition: A[1..n] is an array of n numbers

Post-Condition: A[1..n] permuted in sorted order

1. **for** i ← 2 .. n **do**

LI: A[1..i-1] is sorted, A[i..n] is untouched.

§ insert A[i] into sorted prefix A[1..i-1] by right-cyclic-shift:

2. key ← A[i]

3. j ← i-1

4. **while** j > 0 and A[j] > key **do**

5. A[j+1] ← A[j]

6. j ← j-1

7. **end-while**

8. A[j+1] ← key

9. **end-for**

**end**

$T(n)$

$$= \Theta\left(\sum_{i=2}^n (1 + t_i + 1)\right)$$

$$= \Theta\left(n + \sum_{i=2}^n t_i\right)$$

$$= \Theta\left(n + \sum_{i=2}^n i\right)$$

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

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

Worst-case:  $t_i = i$  iterations (reverse sorted).

$$\sum_{i=2}^n i = \frac{n(n+1)}{2} - 1 = \Theta(n^2).$$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Assignment Project Exam Help  
**Exercises**

<https://powcoder.com>

Add WeChat powcoder

1. We listed a number of facts and showed the proofs of some of them. Prove the rest.
2. Give the most simplified answer to each of the following with a brief explanation.

a)  $f(n) = 4n^3 + 6n^2 \log n + 1200 = \Theta(?)$

b)  $f(n) = \frac{5n^6 + O(n^3 \log n)}{3n^2 \log n + O(n)} = \Theta(?)$

c)  $f(n) = 109n^2 + 10^{10} n \log n + 3 \cdot 2^n = \Theta(?)$

d)  $f(n) = n^{\frac{\log \log n}{\log n}} = \Theta(?)$

e) Does  $f(n) = o(n) + \Theta(n^2 \log n)$  imply  $f(n) = \Omega(n \log \log n)$ ?

f) Indicate whether or not each function below is  $\omega(n \log n)$ :

$3n + 2, \quad 5n^2 / \log^3 n, \quad \log((n^2)!)$ .

g) Let  $f(n) = 3n^6 / \log^{15} n + 7n\sqrt{n}$ . Is  $f(n) = n^{\Theta(1)}$ ?

3. Are there any differences between  $\Theta(1)^n$  and  $\Theta(2^n)$  and  $2^{\Theta(n)}$ ? Explain.
4. Are any of the following implications always true? Prove or give a counter-example.
  - a)  $f(n) = \Theta(g(n)) \Rightarrow f(n) = cg(n) + o(g(n))$ , for some real constant  $c > 0$ .
  - b)  $f(n) = \Theta(g(n)) \Rightarrow f(n) = cg(n) + O(g(n))$ , for some real constant  $c > 0$ .
5. Show that  $2^{O(\log \log n)} = \log^{O(1)} n$ .
6. **Prove or disprove:**  $n^5 = \Theta(n^{5+o(1)})$ .

7. [CLRS: Problem 3-4, p. 62] Asymptotic notation properties

Let  $f(n)$  and  $g(n)$  be asymptotically positive functions.

Prove or disprove each of the following conjectures.

(a)  $f(n) = O(g(n))$  implies  $g(n) = O(f(n))$ .

(b)  $f(n) + g(n) = \Theta(\min(f(n), g(n)))$ .

(c)  $f(n) = O(g(n))$  implies  $\log(f(n)) = O(\log(g(n)))$ ,  
where  $\log(g(n)) \geq 1$  and  $f(n) \geq 1$  for all sufficiently large  $n$ .

(d)  $f(n) = O(g(n))$  implies  $2^{f(n)} = O(2^{g(n)})$ .

(e)  $f(n) = O((f(n))^2)$ .

(f)  $f(n) = O(g(n))$  implies  $g(n) = \Omega(f(n))$ .

(g)  $f(n) = \Theta(f(n/2))$ .

(h)  $f(n) + o(f(n)) = \Theta(f(n))$ .

8. We have:  $2^{2^{\log \log n}} = 2^{\log n} = n$ , and

$$2^{2^{\log \log n - 1}} = \left(2^{2^{\log \log n}}\right)^{\frac{1}{2}} = \sqrt{n}.$$

So, is  $2^{2^{\lfloor \log \log n \rfloor}}$  equal to  $\Theta(n)$  or  $\Theta(\sqrt{n})$ ? Which one?

9. **Prove or disprove:** The following implications hold.

(a)  $f(n) = O(g(n)) \Rightarrow f(n)^{h(n)} = O(g(n)^{h(n)})$ .

(b)  $f(n) = \Theta(g(n))$  and  $h(n) = \Theta(1) \Rightarrow f(n)^{h(n)} = \Theta(g(n)^{h(n)})$ .

[Note: Compare with “Power Rule”]

10. Is it true that for every pair of functions  $f(n)$  and  $g(n)$  we must have at least one of:  $f(n) = O(g(n))$  or  $g(n) = O(f(n))$ ?

[Hint: Consider  $f(n) = \begin{cases} n & \text{for odd } n \\ n^2 & \text{for even } n \end{cases}$ ,  $g(n) = \begin{cases} n^3 & \text{for odd } n \\ n & \text{for even } n \end{cases}$ .]

11. Demonstrate two functions  $f: \mathcal{N} \rightarrow \mathcal{N}$  and  $g: \mathcal{N} \rightarrow \mathcal{N}$  with the following properties:  
Both  $f$  and  $g$  are strictly monotonically increasing, and  $f \neq O(g)$ , and  $g \neq O(f)$ .

12. Show that  $\left(1 + \frac{1}{\Theta(n)}\right)^{\Theta(n)} = \Theta(1)$ .

[Hint:  $\lim_{x \rightarrow \infty} \left(1 + \frac{1}{x}\right)^x = e \approx 2.7182$  is Neper's constant.]

13. Rank the following functions by asymptotic order of growth. Put them in equivalence classes based on  $\Theta$ -equality. That is,  $f$  and  $g$  are in the same class iff  $f = \Theta(g)$ , and  $f$  is in an earlier class than  $g$  if  $f = o(g)$ . [Hint: see also the previous exercise.]

$$5, \quad \sqrt{\log n}, \quad \log n, \quad n^{\log n}, \quad \log^n n, \quad \log(n!), \quad (\lceil \log n \rceil)!$$

$$n^{3/\log n}, \quad n^{\log \log n}, \quad \sqrt{4^n}, \quad \sqrt{4}^{\sqrt{n}}, \quad 2^{2^{n+3}}, \quad 2^{2^{n-3}},$$

$$\left(\frac{n^4 + 4n + 1}{n^4 + 1}\right)^{6n^3 + 5}, \quad \left(\frac{n^2 + 4n + 1}{n^2 + 1}\right)^{n \log n}, \quad \left(\frac{2n + 1}{2n}\right)^{2n^2} n^{-4}.$$



14. We say a function  $f: \mathcal{N} \rightarrow \mathbb{R}^+$  is **asymptotically additive** if  $f(n) + f(m) = \Theta(f(n+m))$ .

Which of the following functions are asymptotically additive? Justify your answer.

- i) Logarithm:  $\log n$ , (Assume  $n > 1$ .)
- ii) Monomial:  $n^d$ , for any real constant  $d \geq 0$  (this includes, e.g.,  $\sqrt{n}$  with  $d=0.5$ .)
- iii) Harmonic:  $1/n$ . (Assume  $n > 0$ .)
- iv) Exponential:  $a^n$ , for any real constant  $a > 1$ .

**Prove or disprove:** If  $f$  and  $g$  are asymptotically additive and are positive. then so is:

- i)  $a \cdot f$ , for any real constant  $a > 0$ ,
- ii)  $f + g$ ,
- iii)  $f - g$ , assuming  $(f - g)(n)$  is always positive. Extra credit is given if  $f - g$  is monotonically increasing.
- iv)  $f \cdot g$ .
- v)  $f^g$ .

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

15. The **Set Disjointness Problem** is defined as follows: We are given two sets  $A$  and  $B$ , each containing  $n$  arbitrary numbers. We want to determine whether their intersection is empty, i.e., whether they have any common element.

- a) Design the most efficient algorithm you can to solve this problem and analyze worst-case time complexity of your algorithm.
- b) What is the time complexity of the Set Disjointness Problem itself?

[You are not yet equipped to answer part (b).

The needed methods will be covered in Lecture Slide 5.]

Assignment Project Exam Help

END

<https://powcoder.com>

Add WeChat powcoder