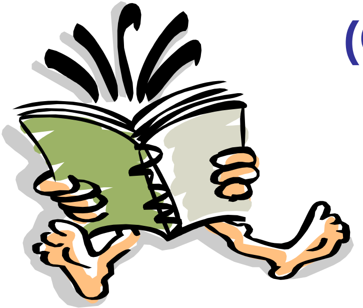
Analysis of Algorithms

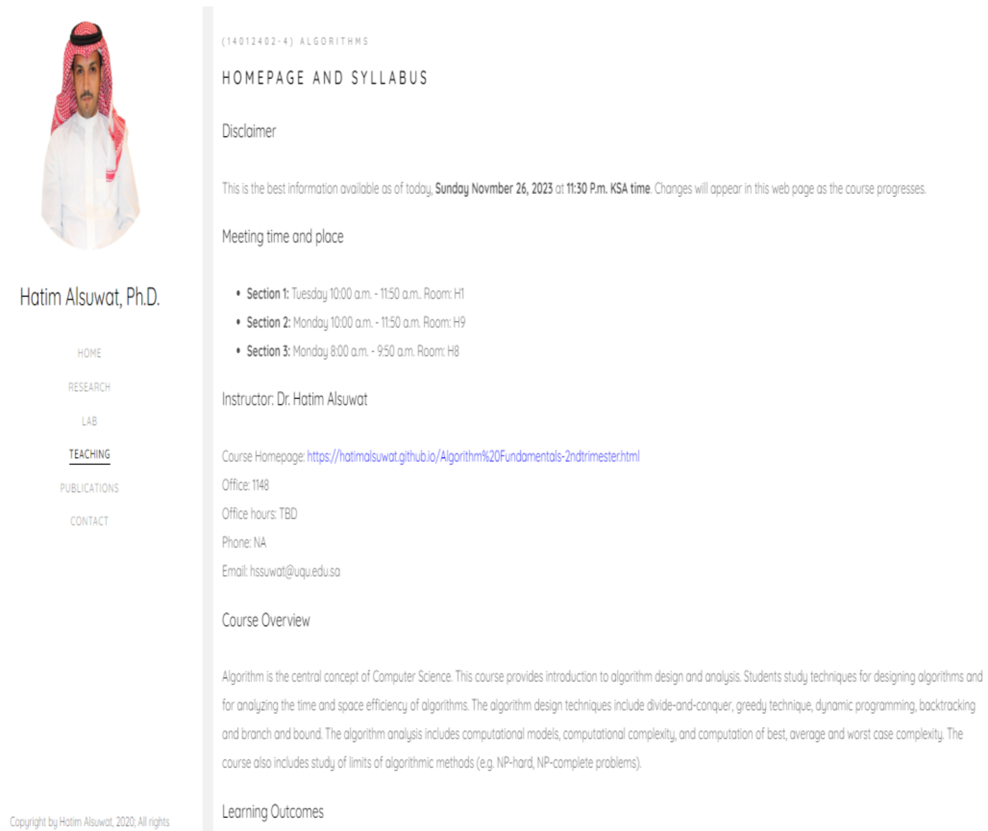
Asymptotic Analysis



(Chapter 3, Appendix A)

Course Information

https://hatimalsuwat.github.io/Algorithm%20Fundamentals-2ndtrimester.html



Communication:

- Announcements on webpage/ emails/ blackboard
- Questions? Email me.
- Staff email: hssuwat@uqu.edu.sa

Course technology:

- Website
- UQU Blackboard
- Regular homework
- Help us make it awesome!

Course Information

- Course Website https://hatimalsuwat.github.io/Algorithm%20Fundamentals-2ndtrimester.html
- Discussion:
  - Please ask any question during the lecture (don’t be shy)
  - There is no such thing as a stupid question.
  - Answer others’ questions - if you know the answer ;-)
  - Learn from others’ questions and answers

Course Information

- **Assignments:**
  - ***Quizzes:*** there will be several quizzes randomly given
  - ***Homework assignments*****:** there will be several homework assignments during the semester.
  - ***Exams*****:** One Midterm Exam and One Final Exam. Closed book tests will cover the course material.
  - Assignments are always due on the announced day and time. Exams must be taken as scheduled except in cases of extenuating circumstances such as a documented emergency.
- Participation can help on margins

Course Information

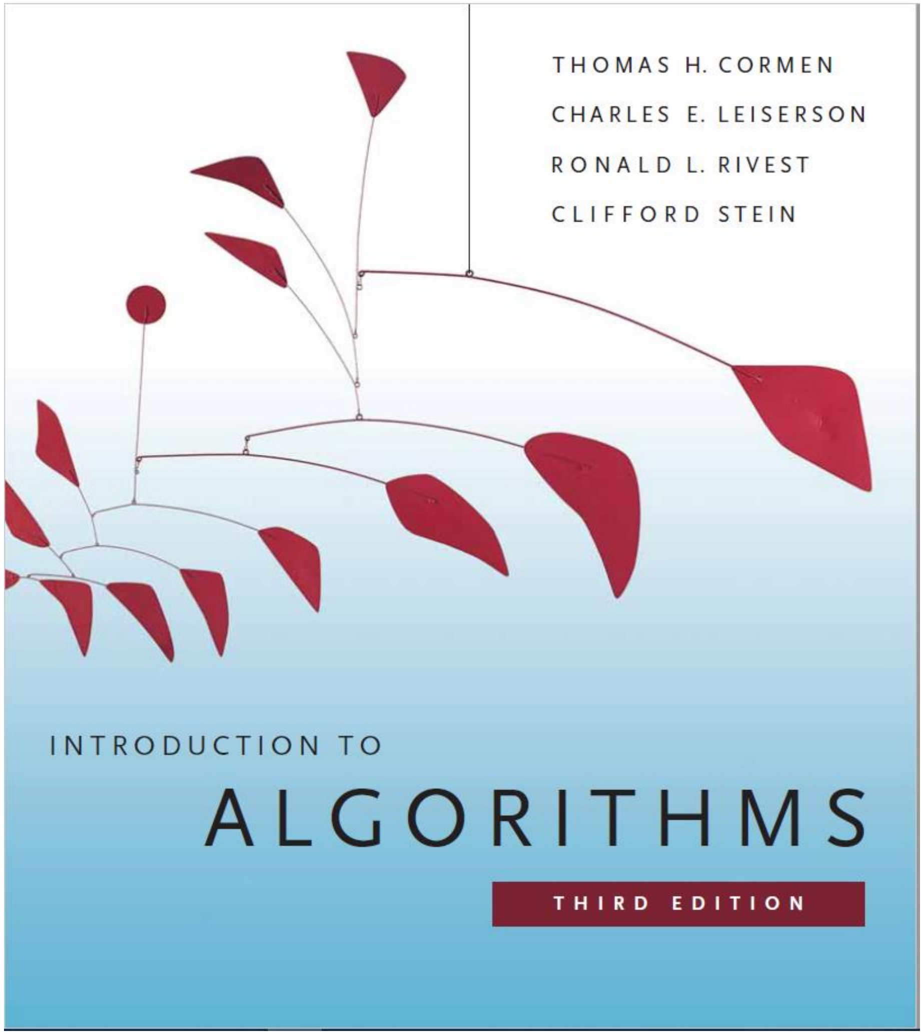
- **Grading:**
  - **Midterm Exam: 20%**
  - **Practical: 25%**
  - **Homework Assignments: 10%**
  - **Participation and Quizzes: 5%**
  - **Final Exam: 40%**

- **Total score that can be achieved: 100**

Course Information

- **Meeting time and place:**
  - **Office:** Department of Computer Science (office #1148)
  - **Office hours:** Please email me if you have any question. If necessary, I will arrange a phone call or in-person meeting
  - **Email:** Hssuwat@uqu.edu.sa

Textbook


THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN
INTRODUCTION TO
ALGORITHMS
THIRD EDITION

Course Information: Feedback

- Please give feedback positive or negative as early as you can via email.

Analysis of Algorithms

- An *algorithm* is a finite set of precise instructions for performing a computation or for solving a problem.
- What is the goal of analysis of algorithms?
  - To compare algorithms mainly in terms of running time but also in terms of other factors (e.g., memory requirements, programmer's effort etc.)
- What do we mean by running time analysis?
  - **Determine how running time increases as the size of the problem increases**.

Input Size

- Input size (number of elements in the input)
  - size of an array
  - polynomial degree
  - # of elements in a matrix
  - # of bits in the binary representation of the input
  - vertices and edges in a graph

Types of Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are
- Best case
  - Provides a lower bound on running time
  - Input is the one for which the algorithm runs the fastest

$$\boxed{Lower\ Bound \leq Running\ Time \leq Upper\ Bound}$$

- Average case
  - Provides a prediction about the running time
  - Assumes that the input is random

How do we compare algorithms?

- We need to define a number of <u>objective measures</u>.

(1) Compare execution times?

Not good: times are specific to a particular computer !!

(2) Count the number of statements executed?

Not good: number of statements vary with the programming language as well as the style of the individual programmer.

Ideal Solution

- Express running time as a function of the input size *n* (i.e., *f(n)*).
- Compare different functions corresponding to running times.
- Such an analysis is independent of machine time, programming style, etc.

Example

- Associate a "cost" with each statement.
- Find the "total cost" by finding the total number of times each statement is executed.

Algorithm 1

	Cost
arr[0] = 0;	c_1
arr[1] = 0;	c_1
arr[2] = 0;	c_1
...	...
arr[N-1] = 0;	c_1

$c_1+c_1+...+c_1 = c_1 \times N$

Algorithm 2

	Cost
for(i=0; i<N; i++)	c_2
arr[i] = 0;	c_1

$(N+1) \times c_2 + N \times c_1 =$

$(c_2 + c_1) \times N + c_2$

Another Example

- ***Algorithm 3*** *Cost*

```
sum = 0;                          c1
for(i=0; i<N; i++)                c2
   for(j=0; j<N; j++)             c2
         sum += arr[i][j];        c3
                              ------------
```

$$c_1 + c_2 \times (N+1) + c_2 \times N \times (N+1) + c_3 \times N^2$$

Asymptotic Analysis

- To compare two algorithms with running times *f(n)* and *g(n),* we need a **rough measure** that characterizes **how fast each function grows.**
- *Hint:* use *rate of growth*
- Compare functions in the limit, that is, **asymptotically!**

  (i.e., for large values of *n*)

Rate of Growth

- Consider the example of buying *elephants* and *goldfish:*

  Cost: cost_of_elephants + cost_of_goldfish

  Cost ~ cost_of_elephants (approximation)

- The low order terms in a function are relatively insignificant for **large** *n*

$$n^4 + 100n^2 + 10n + 50 \quad \sim \quad n^4$$

i.e., we say that $n^4 + 100n^2 + 10n + 50$ and n^4 have the same **rate of growth**

Asymptotic Notation

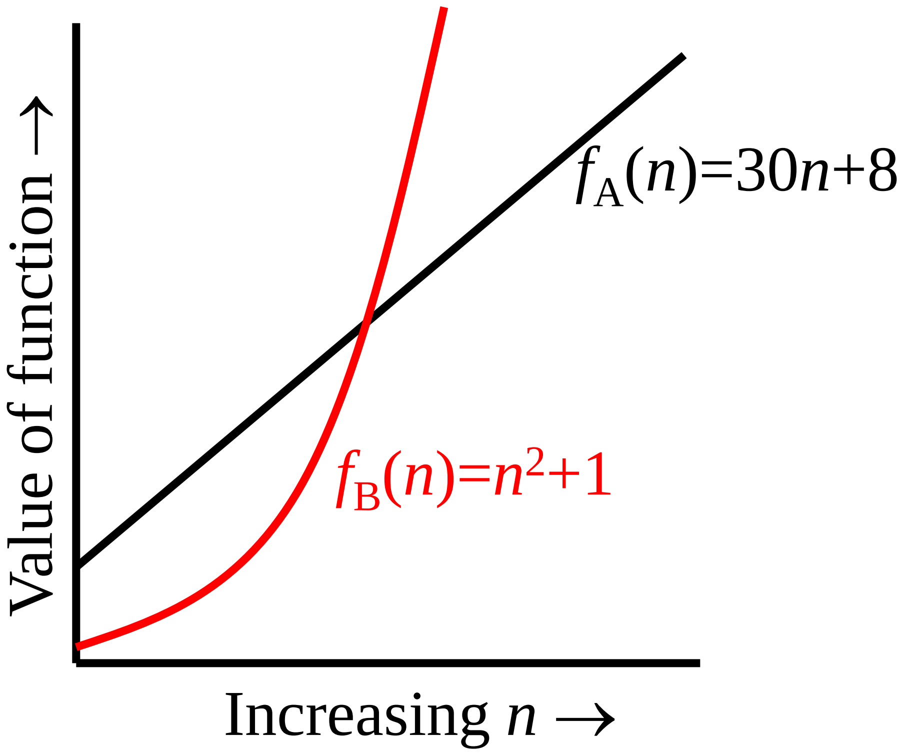
- O notation: asymptotic "less than":
  - $f(n)=O(g(n))$ implies: $f(n)$ "$\leq$" $g(n)$
- Ω notation: asymptotic "greater than":
  - $f(n)= \Omega (g(n))$ implies: $f(n)$ "$\geq$" $g(n)$
- Θ notation: asymptotic "equality":
  - $f(n)= \Theta (g(n))$ implies: $f(n)$ "$=$" $g(n)$

Big-O Notation

- We say $f_A(n)=30n+8$ is *order n*, or O (n)
  It is, at most, roughly *proportional* to n.
- $f_B(n)=n^2+1$ is *order* n^2, or $O(n^2)$. It is, at most, roughly proportional to n^2.
- In general, any $O(n^2)$ function is faster-growing than any $O(n)$ function.

Visualizing Orders of Growth

- On a graph, as you go to the right, a faster growing function eventually becomes larger...



More Examples …

- $n^4 + 100n^2 + 10n + 50$ is $O(n^4)$
- $10n^3 + 2n^2$ is $O(n^3)$
- $n^3 - n^2$ is $O(n^3)$
- constants
  - 10 is $O(1)$
  - 1273 is $O(1)$

Back to Our Example

Algorithm 1

	Cost
arr[0] = 0;	c_1
arr[1] = 0;	c_1
arr[2] = 0;	c_1
...	
arr[N-1] = 0;	c_1

$c_1+c_1+...+c_1 = c_1 \times N$

Algorithm 2

	Cost
for(i=0; i<N; i++)	c_2
arr[i] = 0;	c_1

$(N+1) \times c_2 + N \times c_1 =$

$(c_2 + c_1) \times N + c_2$

- Both algorithms are of the same order: *O(N)*

Example (cont'd)

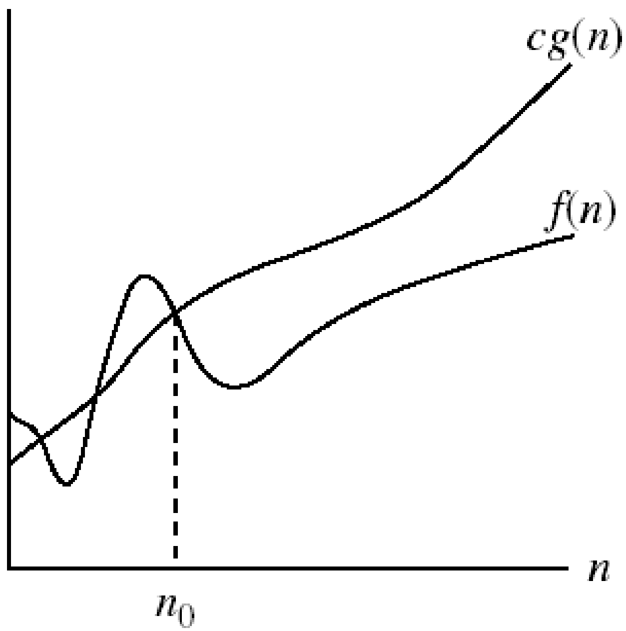
Algorithm 3	***Cost***
sum = 0;	c_1
for(i=0; i<N; i++)	c_2
for(j=0; j<N; j++)	c_2
sum += arr[i][j];	c_3

$c_1 + c_2 \; x \; (N+1) + c_2 \; x \; N \; x \; (N+1) + c_3 \; x \; N^2 = O(N^2)$

Asymptotic notations

- *O-notation*

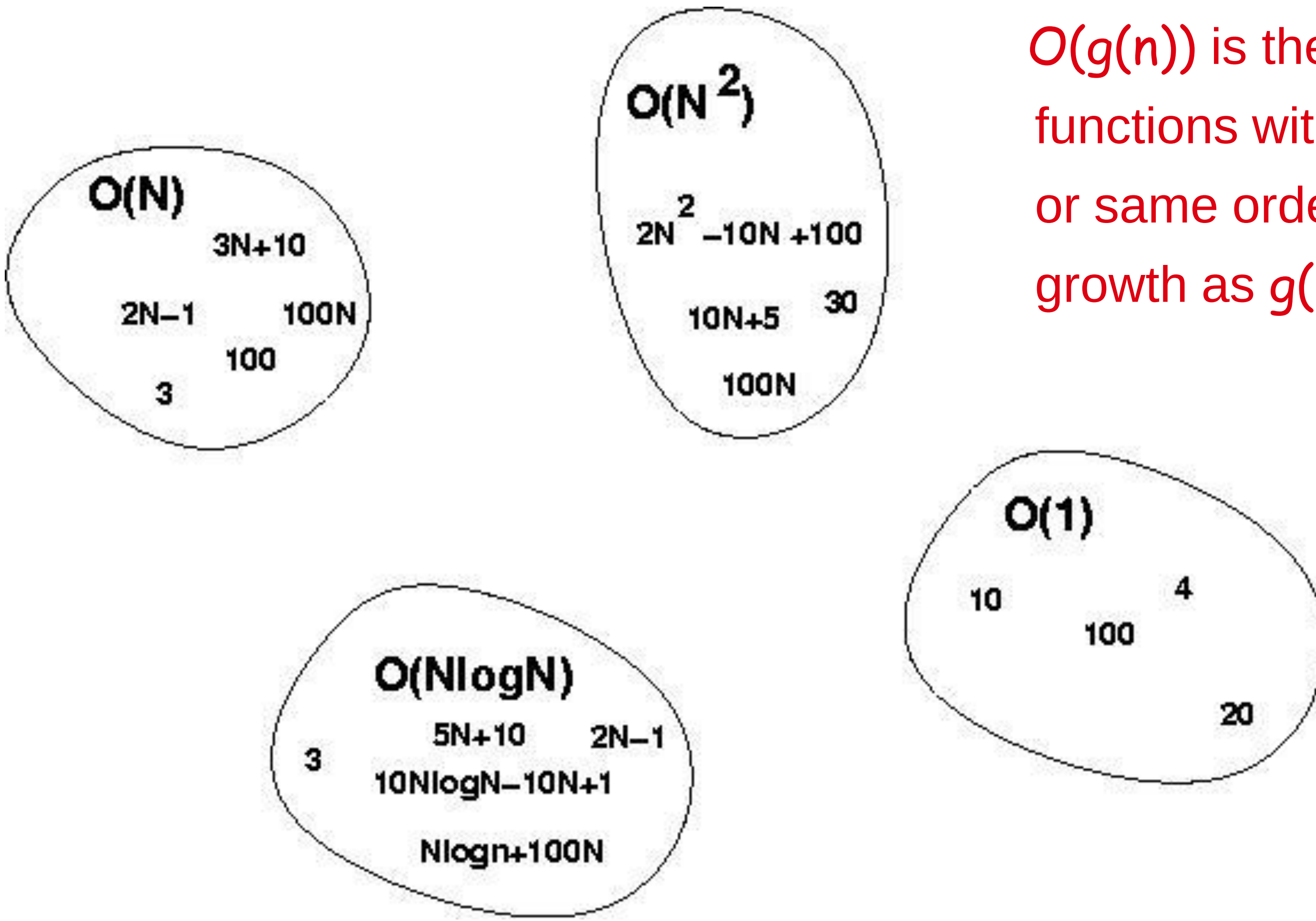
$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\} .$$



$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

Big-O Visualization

O(*g*(*n*)) is the set of functions with smaller or same order of growth as *g*(*n*)



Examples

- $2n^2 = O(n^3)$: $2n^2 \le cn^3 \Rightarrow 2 \le cn \Rightarrow c = 1$ and $n_0 = 2$

- $n^2 = O(n^2)$: $n^2 \le cn^2 \Rightarrow c \ge 1 \Rightarrow c = 1$ and $n_0 = 1$

- $1000n^2+1000n = O(n^2)$:

$1000n^2+1000n \le 1000n^2+ n^2 =1001n^2 \Rightarrow c=1001$ and $n_0 = 1000$

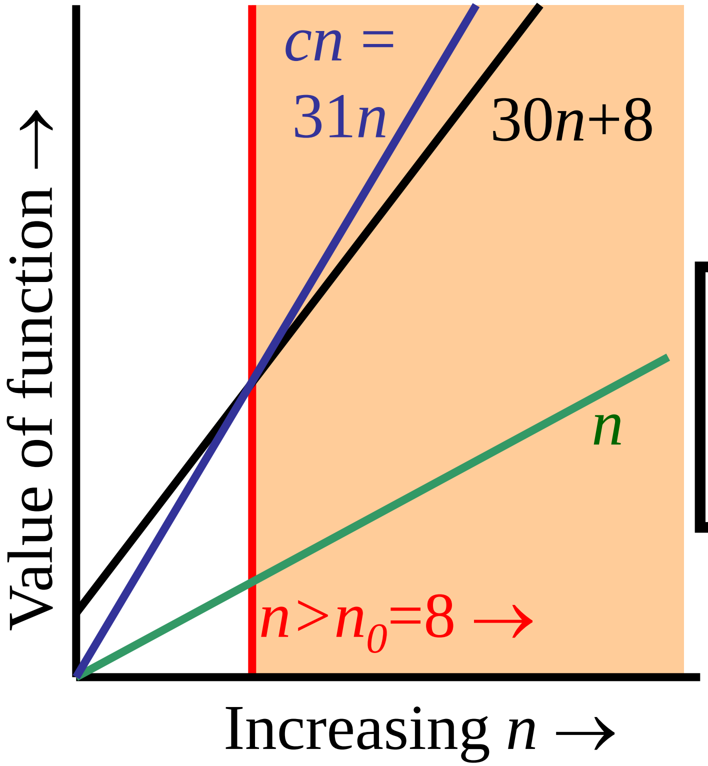
- $n = O(n^2)$: $n \le cn^2 \Rightarrow cn \ge 1 \Rightarrow c = 1$ and $n_0 = 1$

More Examples

- Show that $30n+8$ is $O(n)$.
  - Show $\exists c,n_0$: $30n+8 \le cn$, $\forall n>n_0$.
    - Let $c=31$, $n_0=8$. Assume $n>n_0=8$. Then $cn = 31n = 30n + n > 30n+8$, so $30n+8 < cn$.

Big-O example, graphically

- Note $30n+8$ isn't less than n *anywhere* ($n>0$).
- It isn't even less than $31n$ *everywhere*.
- But it *is* less than $31n$ <u>everywhere to the right of $n=8$</u>.



$$30n+8 \in O(n)$$

No Uniqueness

- There is no unique set of values for n_0 and c in proving the asymptotic bounds
- Prove that $100n + 5 = O(n^2)$
  - $100n + 5 \le 100n + n = 101n \le 101n^2$

    for all $n \ge 5$

    $n_0 = 5$ and $c = 101$ is a solution
  - $100n + 5 \le 100n + 5n = 105n \le 105n^2$

    for all $n \ge 1$

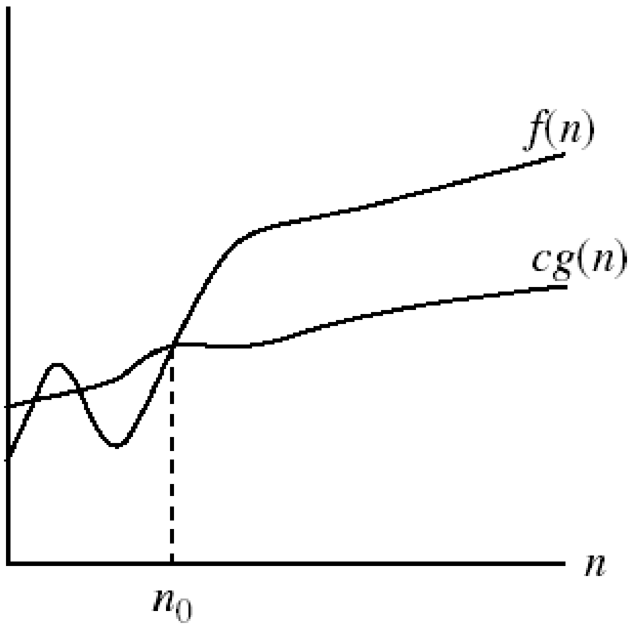
    $n_0 = 1$ and $c = 105$ is also a solution

Must find **SOME** constants c and n_0 that satisfy the asymptotic notation relation

Asymptotic notations (cont.)

- *Ω - notation*

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}\ .$$



$\Omega(g(n))$ is the set of functions with larger or same order of growth as $g(n)$

$g(n)$ is an ***asymptotic lower bound*** for $f(n)$.

Examples

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

  $\exists\ c, n_0$ such that: $0 \le cn \le 5n^2 \Rightarrow cn \le 5n^2 \Rightarrow c = 1$ and $n_0 = 1$

- $100n + 5 \ne \Omega(n^2)$

  $\exists$ c, n_0 such that: $0 \le cn^2 \le 100n + 5$

  $100n + 5 \le 100n + 5n$ ($\forall\ n \ge 1$) $= 105n$

  $cn^2 \le 105n \Rightarrow n(cn - 105) \le 0$

  Since n is positive $\Rightarrow cn - 105 \le 0 \Rightarrow n \le 105/c$

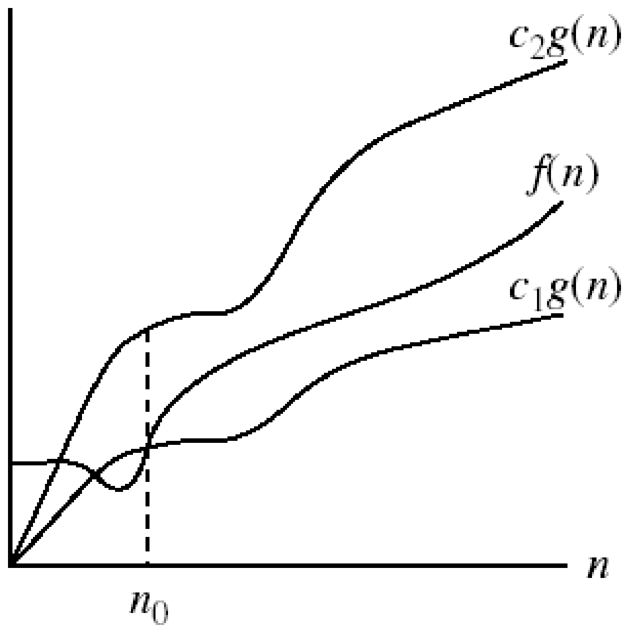
  $\Rightarrow$ contradiction: n cannot be smaller than a constant

- $n = \Omega(2n)$, $n^3 = \Omega(n^2)$, $n = \Omega(\log n)$

Asymptotic notations (cont.)

- *Θ-notation*

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\} .$$



$\Theta(g(n))$ is the set of functions with the same order of growth as $g(n)$

$g(n)$ is an ***asymptotically tight bound*** for $f(n)$.

Examples

- $n^2/2 - n/2 = \Theta(n^2)$
  - $\frac{1}{2} n^2 - \frac{1}{2} n \le \frac{1}{2} n^2 \; \forall n \ge 0 \quad \Rightarrow \quad c_2 = \frac{1}{2}$
  - $\frac{1}{2} n^2 - \frac{1}{2} n \ge \frac{1}{2} n^2 - \frac{1}{2} n * \frac{1}{2} n \; (\forall n \ge 2) = \frac{1}{4} n^2$

    $\Rightarrow \quad c_1 = \frac{1}{4}$

- $n \ne \Theta(n^2)$: $c_1 n^2 \le n \le c_2 n^2$

  $\Rightarrow$ only holds for: $n \le 1/c_1$

Examples

- $6n^3 \neq \Theta(n^2)$: $c_1 n^2 \leq 6n^3 \leq c_2 n^2$

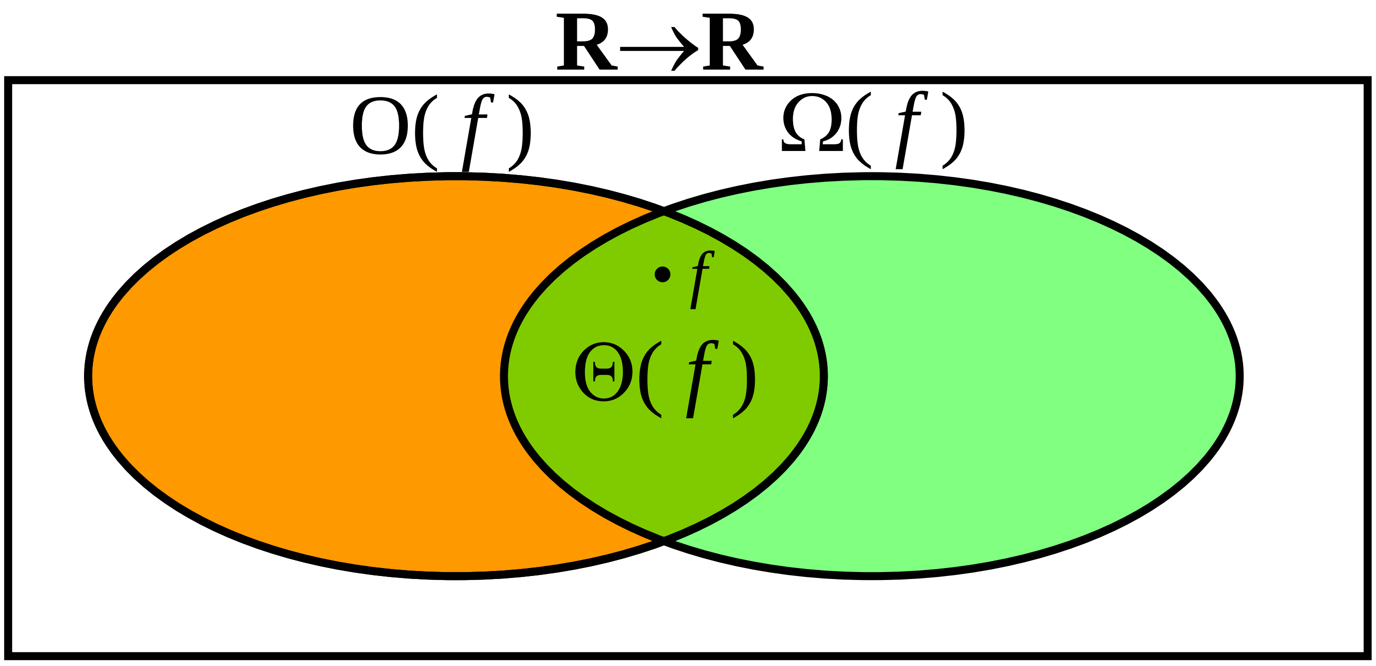
  $\Rightarrow$ only holds for: $n \leq c_2 /6$

- $n \neq \Theta(\log n)$: $c_1 \log n \leq n \leq c_2 \log n$

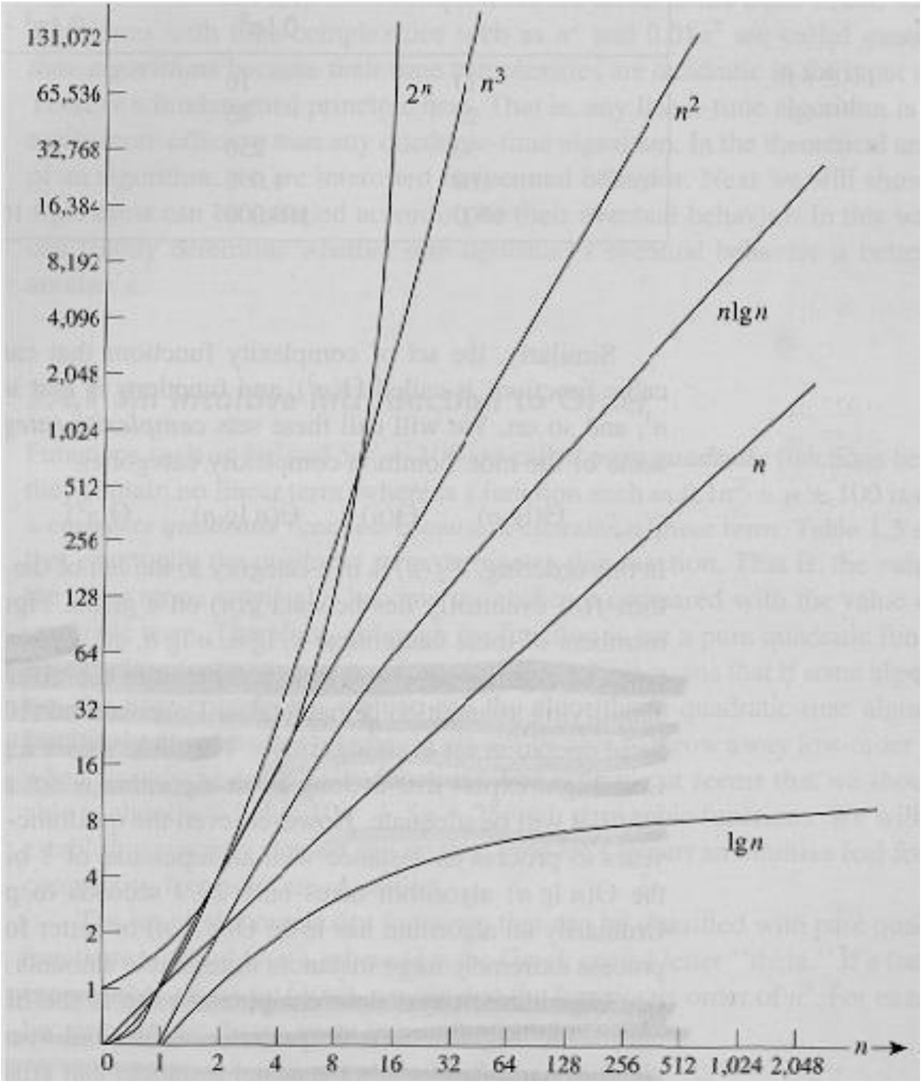
  $\Rightarrow c_2 \geq n/\log n, \forall\, n \geq n_0$ – impossible

Relations Between Different Sets

- Subset relations between order-of-growth sets.



Common orders of magnitude



Common orders of magnitude

Table 1.4 Execution times for algorithms with the given time complexities

n	$f(n) = \lg n$	$f(n) = n$	$f(n) = n \lg n$	$f(n) = n^2$	$f(n) = n^3$	$f(n) = 2^n$
10	0.003 μs*	0.01 μs	0.033 μs	0.1 μs	1 μs	1 μs
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	8 μs	1 ms†
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	27 μs	1 s
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	64 μs	18.3 min
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	125 μs	13 days
10^2	0.007 μs	0.10 μs	0.664 μs	10 μs	1 ms	4×10^{13} years
10^3	0.010 μs	1.00 μs	9.966 μs	1 ms	1 s	
10^4	0.013 μs	10 μs	130 μs	100 ms	16.7 min	
10^5	0.017 μs	0.10 ms	1.67 ms	10 s	11.6 days	
10^6	0.020 μs	1 ms	19.93 ms	16.7 min	31.7 years	
10^7	0.023 μs	0.01 s	0.23 s	1.16 days	31,709 years	
10^8	0.027 μs	0.10 s	2.66 s	115.7 days	3.17×10^7 years	
10^9	0.030 μs	1 s	29.90 s	31.7 years		

*1 μs = 10^{-6} second.
†1 ms = 10^{-3} second.

Logarithms and properties

- In algorithm analysis we often use the notation "log n" without specifying the base

Binary logarithm $\lg n = \log_2 n$

Natural logarithm $\ln n = \log_e n$

$$\lg^k n = (\lg n)^k$$

$$\lg \lg n = \lg(\lg n)$$

$$\log x^y = y \log x$$

$$\log xy = \log x + \log y$$

$$\log \frac{x}{y} = \log x - \log y$$

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

$$\log_b x = \frac{\log_a x}{\log_a b}$$

More Examples

- For each of the following pairs of functions, either f(n) is O(g(n)), f(n) is Ω(g(n)), or f(n) = Θ(g(n)). Determine which relationship is correct.
  - $f(n) = \log n^2$; $g(n) = \log n + 5$ $\quad f(n) = \Theta(g(n))$
  - $f(n) = n$; $g(n) = \log n^2$ $\quad f(n) = \Omega(g(n))$
  - $f(n) = \log \log n$; $g(n) = \log n$ $\quad f(n) = O(g(n))$
  - $f(n) = n$; $g(n) = \log^2 n$ $\quad f(n) = \Omega(g(n))$
  - $f(n) = n \log n + n$; $g(n) = \log n$ $\quad f(n) = \Omega(g(n))$
  - $f(n) = 10$; $g(n) = \log 10$ $\quad f(n) = \Theta(g(n))$
  - $f(n) = 2^n$; $g(n) = 10n^2$ $\quad f(n) = \Omega(g(n))$
  - $f(n) = 2^n$; $g(n) = 3^n$ $\quad f(n) = O(g(n))$

Properties

- *Theorem:*

  $$f(n) = \Theta(g(n)) \Leftrightarrow f = O(g(n)) \text{ and } f = \Omega(g(n))$$

- Transitivity:
  - $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$
  - Same for O and Ω
- Reflexivity:
  - $f(n) = \Theta(f(n))$
  - Same for O and Ω
- Symmetry:
  - $f(n) = \Theta(g(n))$ if and only if $g(n) = \Theta(f(n))$
- Transpose symmetry:
  - $f(n) = O(g(n))$ if and only if $g(n) = \Omega(f(n))$

Asymptotic Notations in Equations

- On the right-hand side
  - $\Theta(n^2)$ stands for some anonymous function in $\Theta(n^2)$

  $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ means:

  There exists a function $f(n) \in \Theta(n)$ such that

  $$2n^2 + 3n + 1 = 2n^2 + f(n)$$

- On the left-hand side

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

  No matter how the anonymous function is chosen on the left-hand side, there is a way to choose the anonymous function on the right-hand side to make the equation valid.

Common Summations

- Arithmetic series: $$\sum_{k=1}^{n} k = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$
- Geometric series: $$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \ldots + x^n = \frac{x^{n+1} - 1}{x - 1} \left(x \neq 1\right)$$
  - Special case: $|x| < 1$: $$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$
- Harmonic series: $$\sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \ldots + \frac{1}{n} \approx \ln n$$
- Other important formulas: $$\sum_{k=1}^{n} \lg k \approx n \lg n$$

$$\sum_{k=1}^{n} k^p = 1^p + 2^p + \ldots + n^p \approx \frac{1}{p+1} n^{p+1}$$

Mathematical Induction

- A powerful, rigorous technique for proving that a statement S(*n*) is true for *every* natural number *n*, no matter how large.

- Proof:

  – **Basis step**: prove that the statement is true for $n = 1$

  – **Inductive step:** assume that $S(n)$ is true and prove that $S(n+1)$ is true for all $n \geq 1$

- Find case n “within” case $n+1$

Example

- Prove that: $2n + 1 \leq 2^n$ for all $n \geq 3$
- **Basis step:**
  - $n = 3$: $2 * 3 + 1 \leq 2^3 \Leftrightarrow 7 \leq 8$ TRUE
- **Inductive step:**
  - Assume inequality is true for n, and prove it for (n+1):

$2n + 1 \leq 2^n$ must prove: $2(n + 1) + 1 \leq 2^{n+1}$

$2(n + 1) + 1 = (2n + 1) + 2 \leq 2^n + 2 \leq$

$\leq 2^n + 2^n = 2^{n+1}$, since $2 \leq 2^n$ for $n \geq 1$

Summations – Review

Review on Summations

- Why do we need summation formulas?

For computing the running times of iterative constructs (loops). (CLRS – Appendix A)

Review on Summations

- **Constant Series:** For integers *a* and *b*, $a \le b$,

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

- **Linear Series (Arithmetic Series):** For $n \ge 0$,

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

- **Quadratic Series:** For $n \ge 0$,

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

Review on Summations

- **Cubic Series:** For $n \geq 0$,

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

- **Geometric Series:** For real $x \neq 1$,

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

For $|x| < 1$, $\displaystyle\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$

Review on Summations

- **Linear-Geometric Series:** For $n \geq 0$, real $c \neq 1$,

$$\sum_{i=1}^{n} ic^{i} = c + 2c^{2} + \cdots + nc^{n} = \frac{-(n+1)c^{n+1} + nc^{n+2} + c}{(c-1)^{2}}$$

- **Harmonic Series:** nth harmonic number, $n \in \mathrm{I}^{+}$,

$$H_{n} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

$$= \sum_{k=1}^{n} \frac{1}{k} = \ln(n) + O(1)$$

Review on Summations

- **Telescoping Series:**

$$\sum_{k=1}^{n} a_k - a_{k-1} = a_n - a_0$$

- **Differentiating Series:** For $|x| < 1$,

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

Review on Summations

- **Approximation by integrals:**
  - For monotonically increasing *f*(*n*)

$$\int_{m-1}^{n} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m}^{n+1} f(x)dx$$

  - For monotonically decreasing *f*(*n*)

$$\int_{m}^{n+1} f(x)dx \le \sum_{k=m}^{n} f(k) \le \int_{m-1}^{n} f(x)dx$$

- **How?**

Review on Summations

- ***n*th harmonic number**

$$\sum_{k=1}^{n} \frac{1}{k} \geq \int_{1}^{n+1} \frac{dx}{x} = \ln(n+1)$$

$$\sum_{k=2}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{dx}{x} = \ln n$$

$$\Rightarrow \sum_{k=1}^{n} \frac{1}{k} \leq \ln n + 1$$