CS 2009
Design and Analysis of Algorithms

Prepared by: Waheed Ahmed

Edited by: Farrukh Salim Shaikh
Email : farrukh.salim@nu.edu.pk

Lecture 1:

Introduction & Course Overview

Grading Policy (CS 2009)

Assessment Type	**Weight**
Assignments	10
Midterms (Week 6 & Week 11)	30 (15 each)
Project	10
Final	50

Text & Reference Books

- Required Textbook
  - Thomas H. Cormen “Introduction to Algorithms” 2nd Edition

- Reference Books
- Anany Levitin “Introduction to the Design and Analysis of Algorithms” 3rd edition
- Jon Kleinberg and Éva Tardos “Algorithm Design”
- Sanjoy Dasgupta et al. “Algorithms”
- Steven S. Skiena “The Algorithm Design Manual”

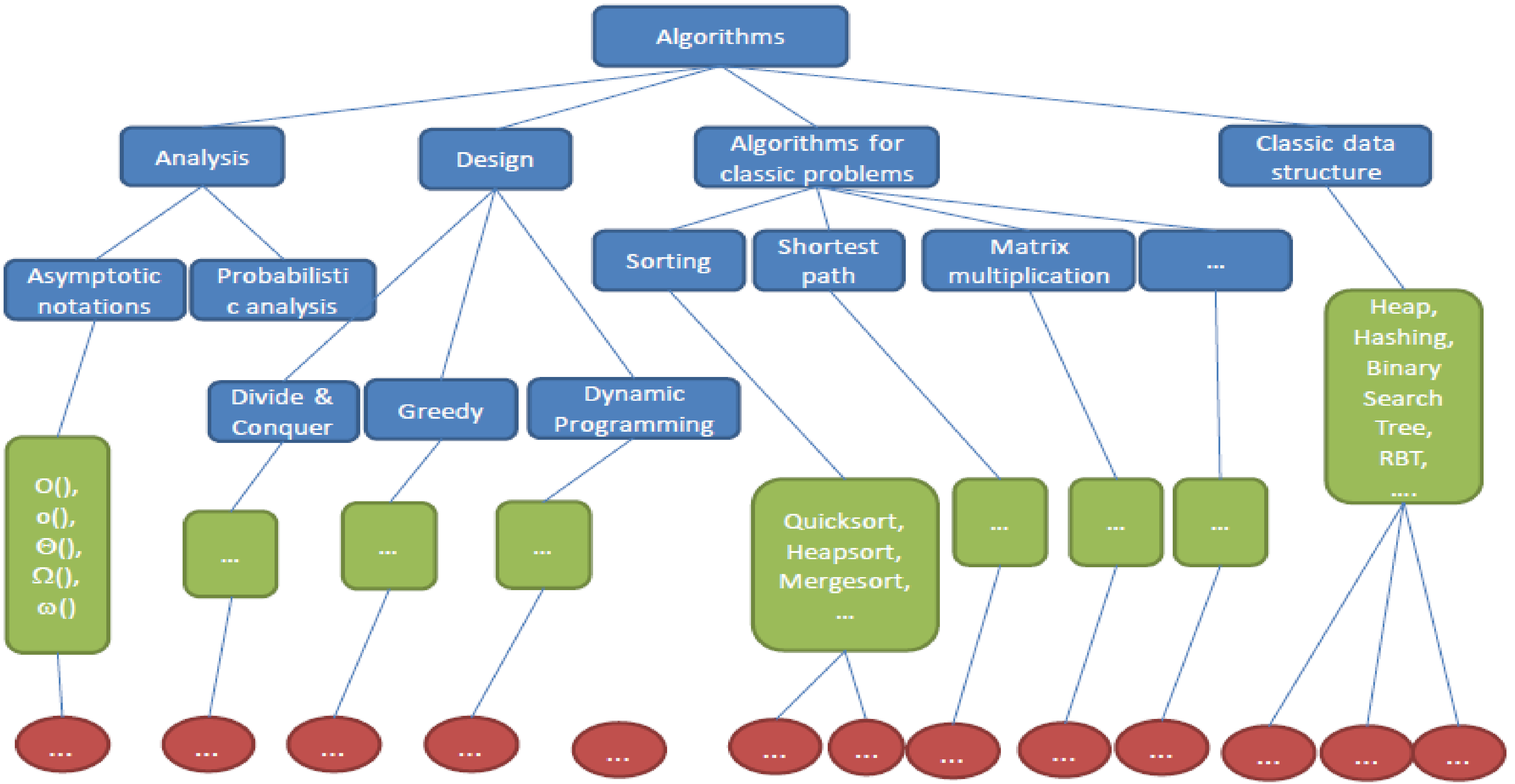
Contents & Tentative Schedule

Week	Topics
Week 1 & 2	Basics of Algorithms, Mathematical Foundation, Growth of Function, Asymptotic Notations. Data Structures Review (Stack, Queue, Linked List, Hash Table, Binary Tree).
Week 3 & 4	Divide and Conquer, Substitution Method, Recurrence-Tree Method, Master’s Method.
Week 5	Sorting (Merge, Insertion, Quick, Heap, Counting, Radix, Bucket)
Week 6	Mid term 1 Exam
Week 7	Dynamic Programming

Contents & Tentative Schedule

Week	Topics
Week 8	Dynamic Programming & Greedy Algorithms
Week 9, 10 & 12	Graph Theory (Graph Categorization, Graph Terminology, Representation of Graphs, BFS & DFS, Strongly Connected Components, Greedy Algorithms: Kruskal’s Algorithm, Prim’s Algorithms, Bellman-Ford Algorithms, Dijkstra’s Algorithm)
Week 11	Midterm 2
Week 13 & 14	Geometric Algorithms (Introduction,Graham Scan, Close Points). String Matching
Week 15 & 16	NP Complete Problems and Solutions using Approximation Algorithm, Amortized algorithms
Week 17	Review & Project Presentations

Knowledge tree



What is an algorithm?

What is Algorithm

- An algorithm is any well-defined computational procedure that takes some value as input and produces some value as output. (Thomas H. CORMEN)

- An ***algorithm*** is a sequence of **computational** steps for solving a problem.

  E.g.

  - Multiply Two Numbers.
  - Algorithms to Sort Array.

What is an algorithm?

- Algorithm: cook a cup of instant noodles
  1. Pull back lid to the dotted line.
  2. Fill the cup to the inside line with boiling water from a kettle or from the microwave
  3. Close lid and let stand for 3 minutes.
  4. Stir well and add a pinch of salt and pepper to taste.

MULTIPLICATION PROBLEM

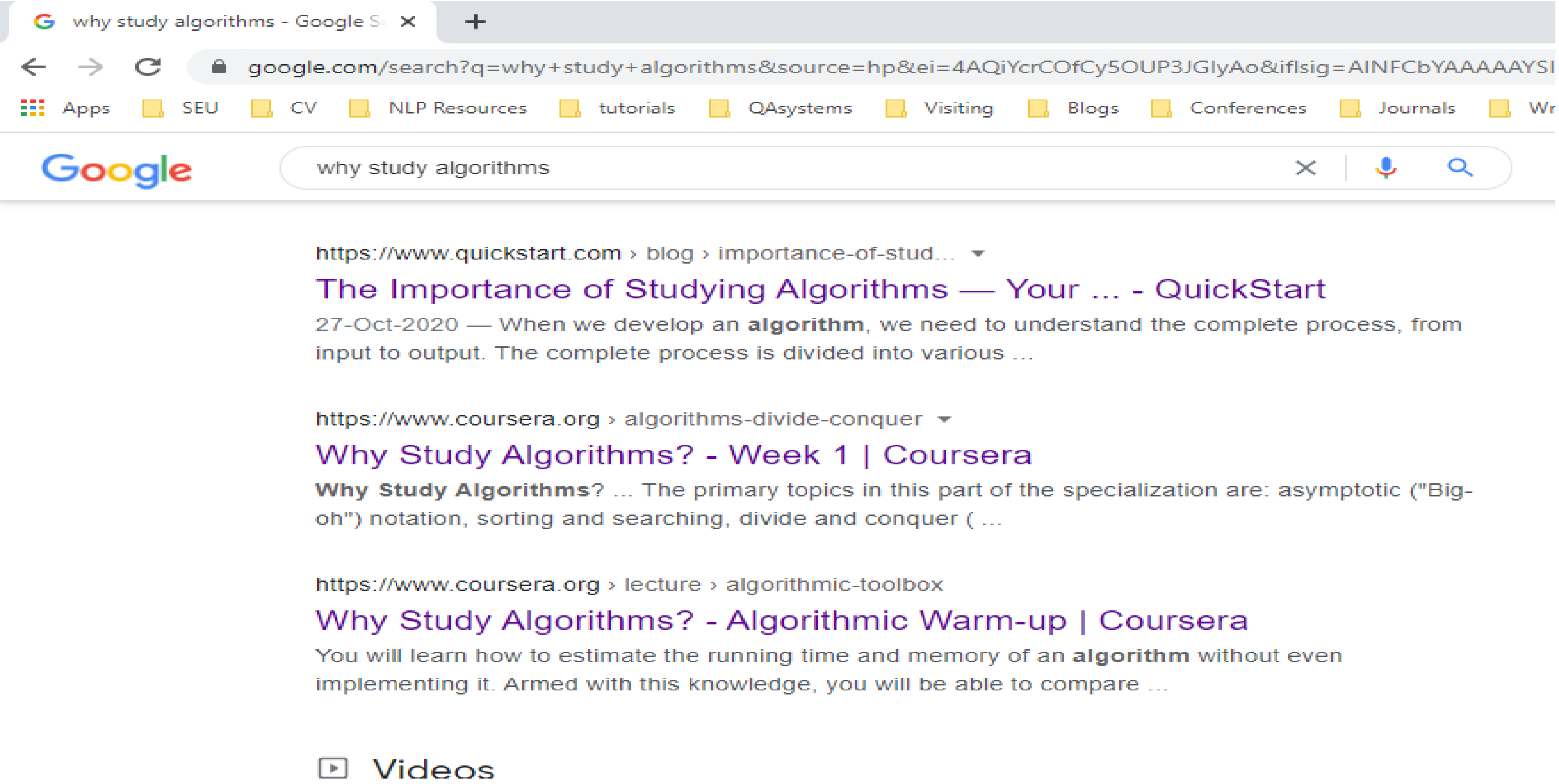
Input: 2 numbers, x and y (n digits each)

Output: the product x · y

$$\begin{array}{r} 2143 \\ \times\ 9112 \\ \hline \mathbf{19427016} \end{array}$$

Why Study Algorithms ?

Web Search


why study algorithms - Google S
google.com/search?q=why+study+algorithms&source=hp&ei=4AQiYcrCOfCy5OUP3JGlyAo&iflsig=AINFCbYAAAAAYSI
Apps
SEU
CV
NLP Resources
tutorials
QAsystems
Visiting
Blogs
Conferences
Journals
Google
why study algorithms
https://www.quickstart.com › blog › importance-of-stud...
The Importance of Studying Algorithms — Your ... - QuickStart
27-Oct-2020 — When we develop an algorithm, we need to understand the complete process, from input to output. The complete process is divided into various ...
https://www.coursera.org › algorithms-divide-conquer
Why Study Algorithms? - Week 1 | Coursera
Why Study Algorithms? ... The primary topics in this part of the specialization are: asymptotic ("Big-oh") notation, sorting and searching, divide and conquer (...
https://www.coursera.org › lecture › algorithmic-toolbox
Why Study Algorithms? - Algorithmic Warm-up | Coursera
You will learn how to estimate the running time and memory of an algorithm without even implementing it. Armed with this knowledge, you will be able to compare ...
Videos

Personalized Recommendation

- More than **70% of what people watch** on YouTube is determined by its **recommendation algorithm**.

- News Feed, Friend Suggestions

Lot of Applications

Internet. Web search, Packet routing, distributed file sharing,...

Biology. Human genome project, protein folding, …

Data Mining. Text Classification, Text Clustering, Page Rank

Security. E-commerce, Cell phones, Voting machine

Web programing. Sorting Algorithms, Searching algorithms

Graphics. Video Games, Virtual Reality,

Social networks. Recommendations, news feed

Machine Learning AI. Linear Regression Algorithm, Deep Neural Networks such RNN, CNN

Robotics. Planning Algorithms.

Why Study Algorithms?

- To become proficient programmer.
- To solve problems that could not be solved.
- For fun and profit.

What we are interested in Algorithms

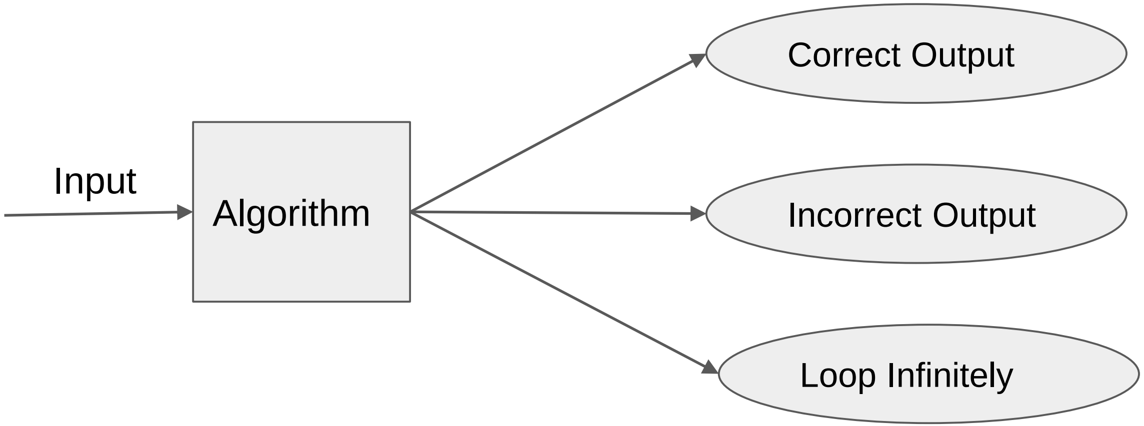
- Correctness
  - Does it work correctly?
- Performance/Efficiency
  - How much time will it take? (Time Complexity)
  - How much space will it take? (Space Complexity)
- Can We do it better?

What’s more important than performance?

- Correctness
- Robustness
- User-friendliness
- Simplicity
- Extensibility
- Reliability

Correct Algorithm

An algorithm is said to be ***correct*** if, for every input instance, it halts with the correct output.



Which Running Time Is Better?

Computer A **(Faster)**: Run algorithm of $\mathbf{2n^2}$ complexity. Run 10 billions instruction per second.

Computer B **(Slower)**: Run Algorithm **50 n log n** complexity. Run 10 millions instruction per second.

Input length **n = 10** millions

$$\frac{2 \cdot (10^7)^2 \text{ Instructions}}{10^{10} \text{ Instructions/second}} = 20{,}000 \text{ seconds } (> 5.5 \text{ hours})$$

$$\frac{50 \cdot 10^7 \log 10^7 \text{ Instructions}}{10^7 \text{ Instructions/second}} = 1163 \text{ seconds } (< 20 \text{ minutes})$$

Which Running Time Is Better?

Is $1000000n$ operations better than $4n^2$?

Is $0.000001n^3$ operations better than $4n^2$?

Is $3n^2$ operations better than $4n^2$?

- **The answers for the first two depend on what value n is...**
  - $1000000n < 4n^2$ only when n exceeds a certain value (in this case, 250000)
- **These constant multipliers are too environment-dependent...**
  - An operation could be faster/slower depending on the machine, so $3n^2$ ops on a slow machine might not be "better" than $4n^2$ ops on a faster machine

MULTIPLICATION PROBLEM

How efficient is this algorithm?

(How many single-digit operations are required?)

Algorithm description (informal*):

compute partial products (using multiplication & "carries" for digit overflows), and add all (properly shifted) partial products together

```
     2143
   x 9112
---------
     4286
    21430
   214300
 19187000
---------
 19427016
```

MULTIPLICATION PROBLEM

How efficient is this algorithm?

(How many single-digit operations are required?)

n partial products: ~$2n^2$ ops (at most n multiplications & n additions per partial product)

adding n partial products: ~$2n^2$ ops

(a bunch of additions & "carries")

~ $4n^2$ operations in the worst case

```
    2143
  x 9112
--------
    4286
   21430
  214300
19187000
--------
19427016
```

MULTIPLICATION PROBLEM

How efficient is this algorithm?

(How many single-digit operations are required?)

n partial products: ~$2n^2$ ops (at most n multiplications & n additions per partial product)

adding n partial products: ~$2n^2$ ops

(a bunch of additions & "carries")

~ $4n^2$ operations in the worst case

$$
\begin{array}{r}
2143 \\
\times\ 9112 \\
\hline
4286 \\
21430 \\
214300 \\
19187000 \\
\hline
\mathbf{19427016}
\end{array}
$$

MULTIPLICATION PROBLEM

$$\overbrace{1234567899876543210 1}^{\textbf{\textit{n}}\ \textit{digits}}$$

$$\times\ 98765432112345678901$$

How efficient is this algorithm?

(How many single-digit operations are required?)

n partial products: ~$2n^2$ ops (at most n multiplications & n additions per partial product)

adding n partial products: ~$2n^2$ ops
(a bunch of additions & “carries”)

~ $4n^2$ operations in the worst case

Complexity analysis- One Loop

Problem: Does array A contain the integer t? Given A (array of length n) and t (an integer). \\\\

```
for (i = 0; i<n ; i++):
        if A[i] == t:
                return true
return false
```

Question: What is the running time?

Complexity analysis- One Loop

The running time is:

- 1 assignment (i = 0)
- n+1 comparisons (i < n)
- n increments (i++)
- n array offset calculations (a[i])
- n comparisons (a[i] == K)
- a + b(n + 1) + cn + dn + en, where a, b, c, d, and e are constants depend upon machine
- Easier just to say O(n) (constant-time) operations

```
for (i = 0; i<n ; i++):
        if A[i] == t:
                return true
return false
```

Complexity analysis- One Loop

Problem: Does array A contain the integer t in first 5 elements? Given A (array of length n) and t (an integer). \\\\

```
for (i = 0; i<5 ; i++):
        if A[i] == t:
                return true
return false
```

Question: What is the running time? O (k) where k = 5

Complexity analysis- Two Loops

Problem: Given A;B (arrays of length n) and t (an integer). [Does A or B contain t?] \\\\

```
for (i = 0; i<n ; i++):
        if A[i] == t:
          return true
for (i = 0; i<n ; i++):
        if B[i] == t:
           return true
return false
```

Question: What is the running time? O(n)

Complexity analysis- two Nested Loops

Problem: Do arrays A;B have a number in common? Given arrays A; B of length n \\\\

```
for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
                        if (A[i] == B[j]):
                            return true
        }
}
return false
```

Question: What is the running time? $O(n^2)$

Growth of Function

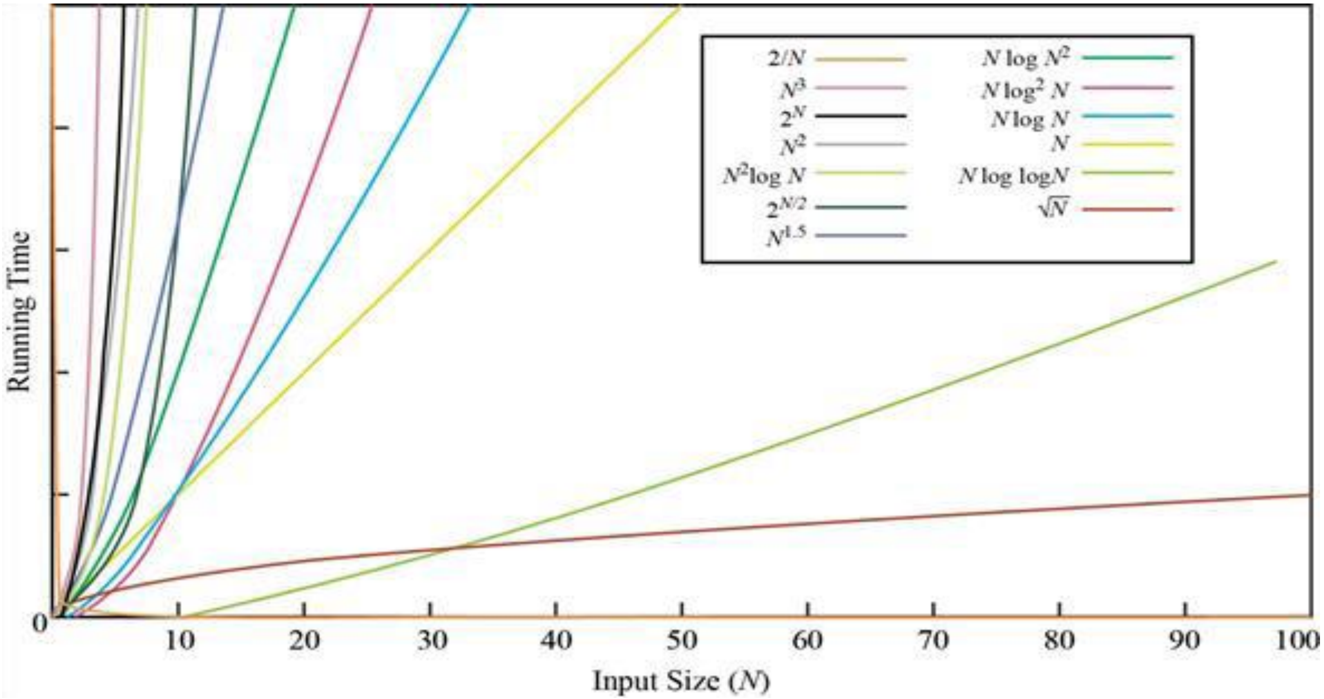
Growth functions are used to estimate the number of steps an algorithm uses as its input grows.

- Common Big-O functions in algorithm analysis
  - $g(n) = 1$ (growth is constant)
  - $g(n) = \log_2 n$ (growth is logarithmic)
  - $g(n) = n$ (growth is linear)
  - $g(n) = n \log_2 n$ (growth is faster than linear)
  - $g(n) = n^2$ (growth is quadratic)
  - $g(n) = 2^n$ (growth is exponential)

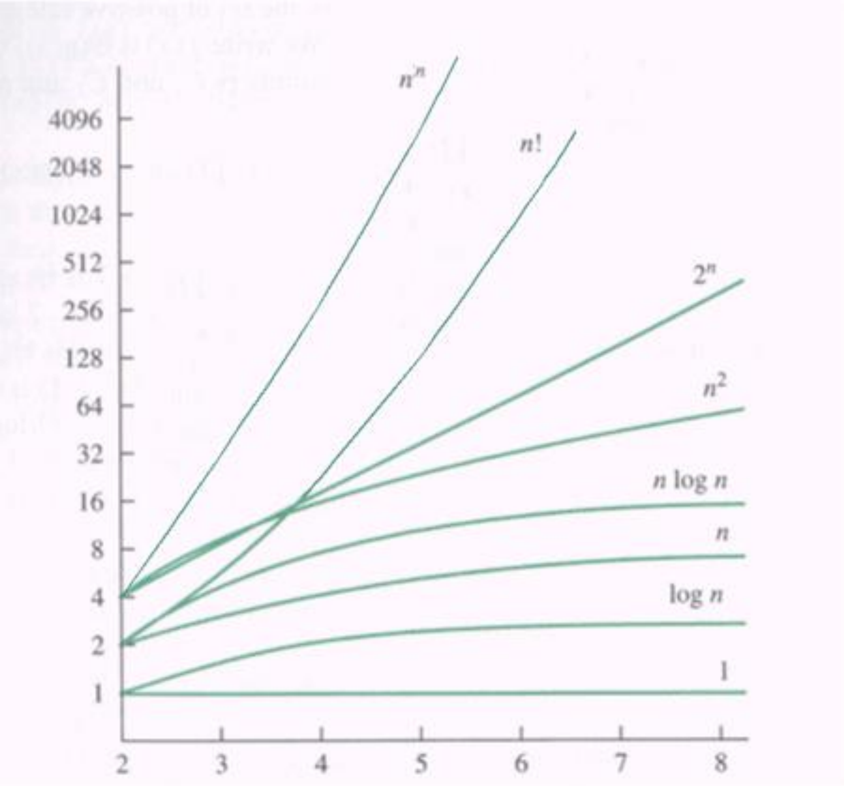
Growth of Function

n	lgn	nlgn	n^2	n^3	2^n
0			0	0	1
1	0	0	1	1	2
2	1	2	4	8	4
4	2	8	16	64	16
8	3	24	64	512	256
16	4	64	256	4096	65536
32	5	160	1024	32768	4294967296
64	6	384	4096	262144	1.84467E+19
128	7	896	16384	2097152	3.40282E+38
256	8	2048	65536	16777216	1.15792E+77
512	9	4608	262144	134217728	1.3408E+154
1024	10	10240	1048576	1073741824	
2048	11	22528	4194304	8589934592	

Growth of Function



Growth of Function



Efficiency of Algorithm

INTRODUCING…

ASYMPTOTIC ANALYSIS

Efficiency of Algorithm

INTRODUCING...

ASYMPTOTIC ANALYSIS

Some guiding principles:

- we want some measure of runtime that's independent of hardware, programming language, memory layout, etc.
  - We want to reason about high-level algorithmic approaches rather than lower-level details
- we care about how the running time/number of operations *scales* with the size of the input (i.e. the runtime's *rate of growth*),
- Not concerned with small values of n, Concerned with VERY LARGE values of n.
- Asymptotic –refers to study of function f as n approaches infinity

ASYMPTOTIC ANALYSIS (High Level Idea)

We'll express the asymptotic runtime of an algorithm using

BIG-O NOTATION

- We would say Multiplication **"runs in time $O(n^2)$"**
  - Informally, this means that the runtime "scales like" n^2

ASYMPTOTIC ANALYSIS (High Level Idea)

We'll express the asymptotic runtime of an algorithm using

BIG-O NOTATION

- We would say Multiplication **"runs in time $O(n^2)$"**
  - Informally, this means that the runtime "scales like" n^2

THE POINT OF ASYMPTOTIC NOTATION

suppress $\underbrace{\text{constant factors}}_{\textit{too system dependent}}$ and $\underbrace{\text{lower-order terms}}_{\textit{irrelevant for large inputs}}$

ASYMPTOTIC ANALYSIS (High Level Idea)

BIG-O NOTATION

THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent *irrelevant for large inputs*

Example $f(n) = 2n^2 + 4n + 1$

$f(n) = O(n^2)$: 2 is constant, n^2 is the dominant term, and the term $4n + 1$ becomes insignificant as n grows larger.

ASYMPTOTIC ANALYSIS (High Level Idea)

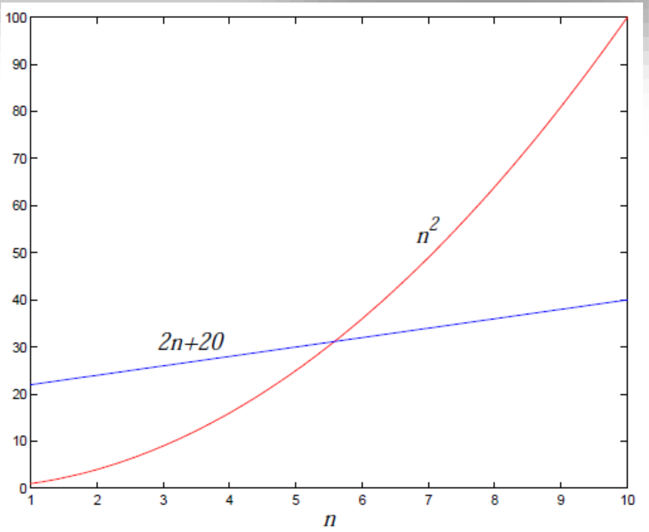
THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent | *irrelevant for large inputs*

$$f_1(n) = n^2$$
$$f_2(n) = 2n + 20$$

Which is better?

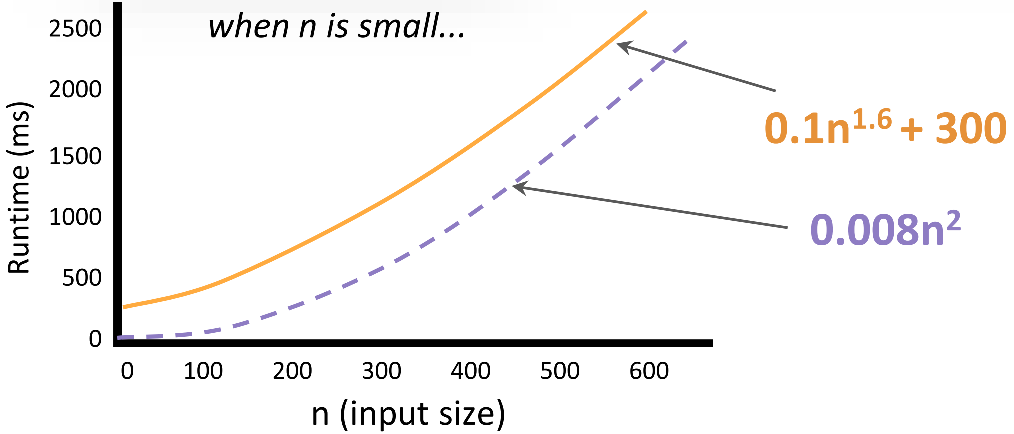


ASYMPTOTIC ANALYSIS (High Level Idea)

THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent — *irrelevant for large inputs*

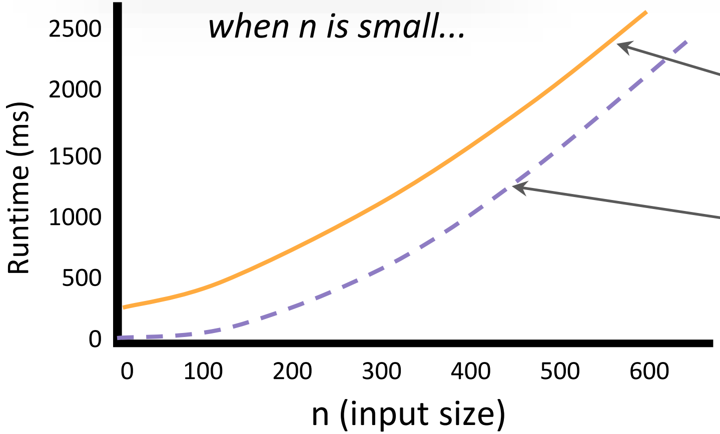


ASYMPTOTIC ANALYSIS (High Level Idea)

THE POINT OF ASYMPTOTIC NOTATION

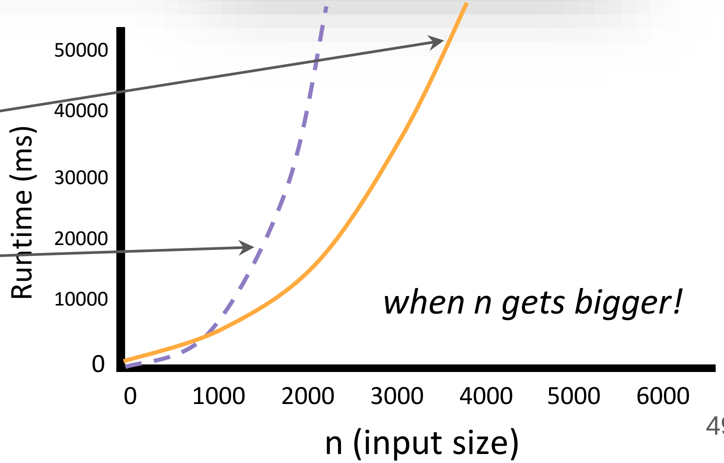
suppress constant factors and lower-order terms

too system dependent — *irrelevant for large inputs*



$0.1n^{1.6} + 300$

$0.008n^2$

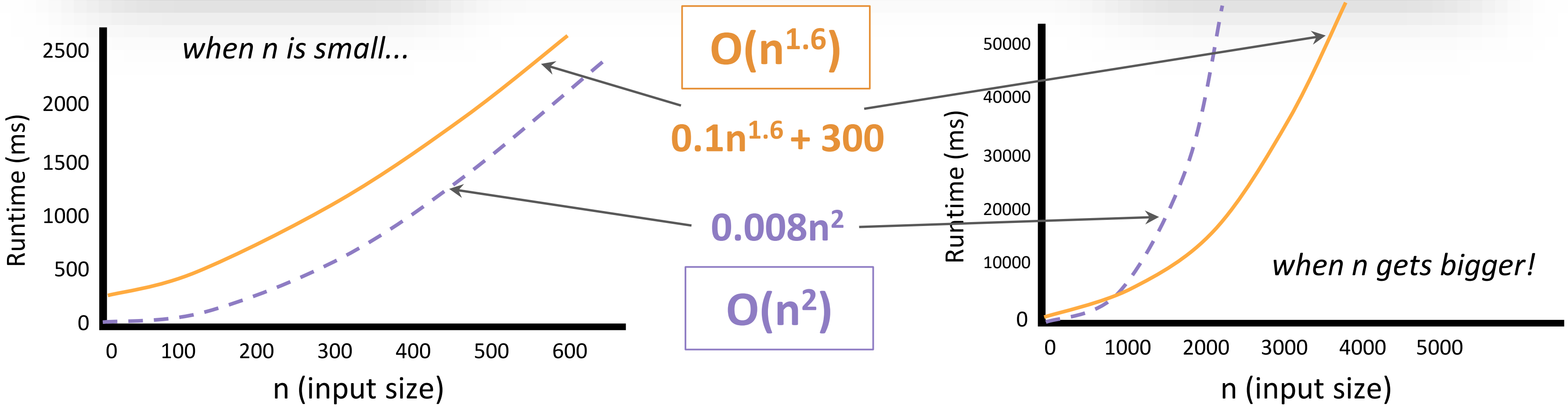


ASYMPTOTIC ANALYSIS (High Level Idea)

THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent — *irrelevant for large inputs*



ASYMPTOTIC ANALYSIS (High Level Idea)

- To compare algorithm runtimes in this class, we compare their Big-O runtimes
  - Ex: a runtime of $O(n^2)$ is considered “better” than a runtime of $O(n^3)$
  - Ex: a runtime of $O(n^{1.6})$ is considered “better” than a runtime of $O(n^2)$
  - Ex: a runtime of $O(1/n)$ is considered “better” than $O(1)$?

RUNTIME ANALYSIS

There are a few different ways to analyze the runtime of an algorithm:

We'll mainly focus on worst case analysis since it tells us how fast the algorithm is on *any* kind of input

Worst-case analysis:
What is the runtime of the algorithm on the *worst* possible input?

Best-case analysis:
What is the runtime of the algorithm on the *best* possible input?

Average-case analysis:
What is the runtime of the algorithm on the *average* input?

Big-O Notation

Let f(n) & g(n) be functions defined on the positive integers.

What do we mean when we say "f(n) is O(g(n))"?

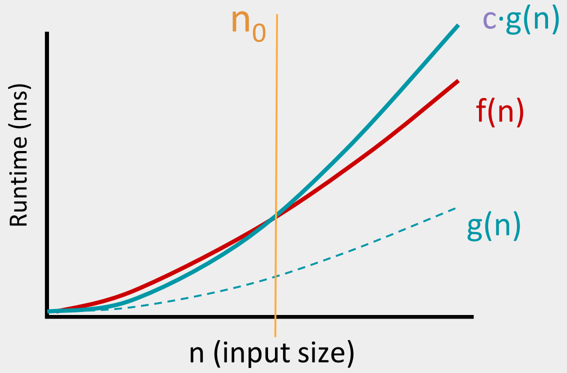
f(n) grows no faster than g(n)
or
g(n) is upper bound on f(n)

In Math

$f(n) = O(g(n))$
if and only if
there exists positive **constants**
c and $\mathbf{n_0}$ such that *for all* $n \geq n_0$

$$f(n) \leq c \cdot g(n)$$

In Picture



Big-O Notation

Let f(n) & g(n) be functions defined on the positive integers.

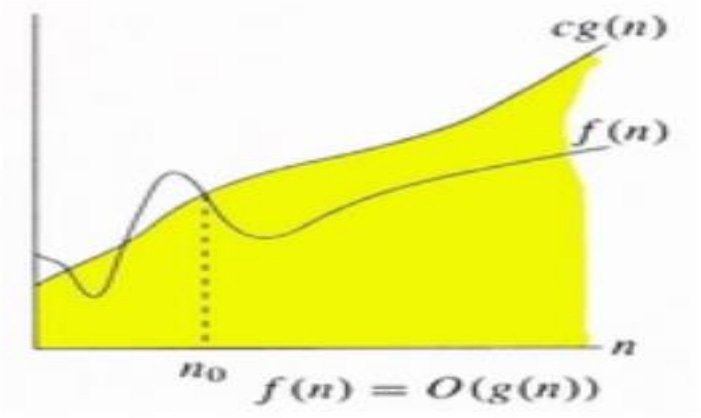
What do we mean when we say "f(n) is O(g(n))"?

In Math

$f(n) = O(g(n))$
if and only if
there exists positive **constants**
c and **n_0** such that *for all $n \geq n_0$*

$f(n) \leq c \cdot g(n)$

In Picture



Big-O Notation

Let $f(n)$ & $g(n)$ be functions defined on the positive integers.

What do we mean when we say "f(n) is O(g(n))"?

In Math

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

if and only if

there exists positive **constants**

c and **n_0** such that *for all $n \geq n_0$*

$f(n) \leq c \cdot g(n)$

In Math

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

$\Leftrightarrow$

$\exists\ c, n_0 > 0$ s.t. $\forall\ n \geq n_0$,

$f(n) \leq c \cdot g(n)$

Proving Big-O Bounds

If you're ever asked to formally prove that f(n) is O(g(n)), use the *MATH* definition:

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

"if and only if" $\Leftrightarrow$

"for all"

$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$

$$f(n) \leq c \cdot g(n)$$

"there exists"

"such that"

must be constants!
i.e. c & n_0 cannot
depend on n!

- To **prove** f(n) = O(g(n)), you need to announce your c & n_0 up front!
  - Play around with the expressions to **find appropriate choices of c & n_0** (positive constants)
  - Then you can write the proof! Here how to structure the start of the proof:

Proving Big-O Bounds: Example # 1 (Method # 1)

$$f(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$
$$f(n) \leq c \cdot g(n)$$

Prove that $3n^2 + 5n = O(n^2)$.

find a c & n_0 such that for all $n \geq n_0$:

$$\mathbf{3n^2 + 5n \leq c \bullet n^2}$$

rearrange this inequality just to see things a bit more clearly:

$$\mathbf{5n \leq (c - 3) \bullet n^2}$$

Now let's cancel out the n:

$$\mathbf{5 \leq (c - 3)\, n}$$

Let's choose:

c = 4

n_0= 5

(other choices work too!

e.g. c= 5, n_0 = 4

c= 10, n_0 = 10)

Proving Big-O Bounds: Example # 2 (Method # 2)

Prove that $f(n) = 3n^2 + 5n + 7 = O(n^2)$.

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

$\Leftrightarrow$

$\exists\ c, n_0 > 0$ s.t. $\forall\ n \geq n_0$,

$f(n) \leq c \cdot g(n)$

find a c & n_0 such that for all $n \geq n_0$:

$$3n^2 + 5n + 7 \leq c \bullet n^2$$

$3n^2 \leq 3n^2$ for $n \geq 0$

$5n \leq 5n^2$ for $n \geq 0$

$7 \leq 7n^2$ for $n \geq 1$

$3n^2 + 5n + 7 \leq 3n^2 + 5n^2 + 7n^2$ for $n \geq 1$

$3n^2 + 5n + 7 \leq 15n^2$ for $n \geq 1$

Proved that $f(n) = 3n^2 + 5n + 7 = O(n^2)$ [for $c = 15$, $n_0 = 1$]

Proving Big-O Bounds: Example # 2 (Method # 1)

Prove that $f(n) = 3n^2 + 5n + 7 = O(n^2)$.

find a c & n_0 such that for all $n \geq n_0$:

$$3n^2 + 5n + 7 \leq c \bullet n^2$$

Divide both sides by n^2, we get:

$3n^2 / n^2 + 5n/ n^2 + 7/n^2 \leq c \bullet n^2/n^2$

$3 + 5/n + 7/n^2 \leq c$

If we choose n_0 equal to 1 then we have value of c

$3 + 5 + 7 \leq c$

$c \geq 15$

$3n^2 + 5n + 7 \leq 15n^2$ for $n \geq 1$

Proved that $f(n) = 3n^2 + 5n + 7 = O(n^2)$ [for $c = 15$, $n_0 = 1$]

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

$\Leftrightarrow$

$\exists\, c, n_0 > 0$ s.t. $\forall\, n \geq n_0$,

$f(n) \leq c \cdot g(n)$

Proving Big-O Bounds: Example # 1 (Method # 2)

Prove that $f(n) = 3n^2 + 5n = O(n^2)$.

find a c & n_0 such that for all $n \geq n_0$:

$$3n^2 + 5n \leq c \bullet n^2$$

$3n^2 \leq 3n^2$ for $n \geq 0$

$5n \leq 5n^2$ for $n \geq 0$

$3n^2 + 5n \leq 3n^2 + 5n^2$ for $n \geq 0$

$3n^2 + 5n \leq 8n^2$ for $n \geq 0$

So $f(n) = 3n^2 + 5n = O(n^2)$ [for $c = 8$, $n_0 = 0$]

The c & n_0 are selected as positive constants, so:

Proved that $f(n) = 3n^2 + 5n = O(n^2)$ [for $c = 8$, $n_0 = 1$]

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

$\Leftrightarrow$

$\exists\, c, n_0 > 0$ s.t. $\forall\, n \geq n_0$,

$f(n) \leq c \cdot g(n)$

Proving Big-O Bounds: Example # 3 (Method # 2)

Show that $f(n) = 5n \log_2 n + 8n + 200 = O(n \log_2 n)$.

find a c & n_0 such that for all $n \geq n_0$:

$$5n \log_2 n + 8n + 200 \leq c \bullet n \log_2 n$$

$$5n \log_2 n + 8n + 200 \leq 5n \log_2 n + 8n \log_2 n + 200n \log_2 n \quad \text{for } n \geq 2$$

$$5n \log_2 n + 8n + 200 \leq 213n \log_2 n \quad \text{for } n \geq 2$$

Thus

$$f(n) = 5n \log_2 n + 8n + 200 = O(n \log_2 n) \ [\text{for } c = 213,\ n_0 = 2]$$

Proving Big-O Bounds: Example # 3(ii) (Method # 2)

Show that $f(n) = 5n \log_2 n + 8n - 200 = O(n \log_2 n)$.

$$f(n) = O(g(n))$$
$$\Leftrightarrow$$
$$\exists\, c, n_0 > 0 \text{ s.t. } \forall\, n \geq n_0,$$
$$f(n) \leq c \cdot g(n)$$

find a c & n_0 such that for all $n \geq n_0$:

$$5n \log_2 n + 8n - 200 \leq c \bullet n \log_2 n$$

While finding c and n0, in

$$5n \log_2 n + 8n - 200 \leq 5n \log_2 n + 8n \log_2 n$$

$$5n \log_2 n + 8n - 200 \leq 13n \log_2 n \quad \text{for} \quad n \geq 2$$

Thus

$$f(n) = 5n \log_2 n + 8n - 200 = O(n \log_2 n) \; [\text{for } c = 13,\ n_0 = 2]$$

Proving Big-O Bounds: Example # 2(ii) (Method # 2)

Prove that $f(n) = 3n^2 + 5n - 7 = O(n^2)$.

$$f(n) = O(g(n)) \Leftrightarrow \exists\, c, n_0 > 0 \text{ s.t. } \forall\, n \geq n_0,\ f(n) \leq c \cdot g(n)$$

find a c & n_0 such that for all $n \geq n_0$:

$$3n^2 + 5n - 7 \leq c \bullet n^2$$

$3n^2 \leq 3n^2$ for $n \geq 0$

$5n \leq 5n^2$ for $n \geq 0$

$\cancel{-7 \leq -7n^2 \quad \text{for } n \geq 1}$

$3n^2 + 5n - 7 \leq 3n^2 + 5n^2$ for $n \geq 0$

$3n^2 + 5n - 7 \leq 8n^2$ for $n \geq 1$

Proved that $f(n) = 3n^2 + 5n - 7 = O(n^2)$ [for $c = 8$, $n_0 = 1$]

Disproving Big-O Bounds

If you're ever asked to formally disprove that T(n) is O(f(n)), use **proof by contradiction!**

This means you need to show that NO POSSIBLE CHOICE of c & n_0 exists such that the Big-O definition holds

Disproving Big-O Bounds

Skip in Class

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

$\Leftrightarrow$

$\exists\, c, n_0 > 0$ s.t. $\forall\, n \geq n_0$,

$f(n) \leq c \cdot g(n)$

Prove that $3n^2 + 5n$ is *not* $O(n)$.

For sake of contradiction, assume that $3n^2 + 5n$ is $O(n)$. This means that there exists positive constants c & n_0 such that $3n^2 + 5n \leq c \cdot n$ for all $n \geq n_0$. Then, we would have the following:

$$3n^2 + 5n \leq c \cdot n$$

$$3n + 5 \leq c$$

$$n \leq (c - 5)/3$$

However, since $(c - 5)/3$ is a constant, we've arrived at a contradiction since n cannot be bounded above by a constant for all $n \geq n_0$. For instance, consider $n = n_0 + c$: we see that $n \geq n_0$, but $n > (c - 5)/3$. Thus, our original assumption was incorrect, which means that $3n^2 + 5n$ is not $O(n)$. ■

Big-Ω Notation

Let f(n) & g(n) be functions defined on the positive integers.

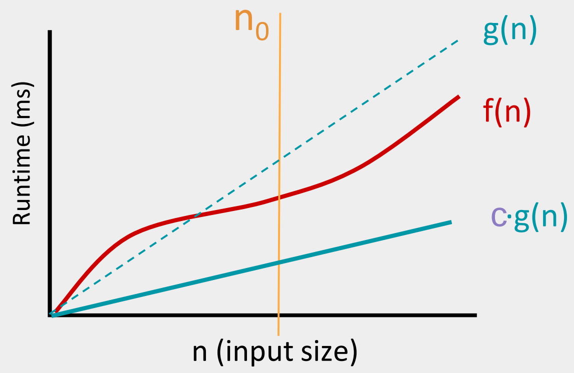
What do we mean when we say "f(n) is Ω(g(n))"?

In *Math*

$$f(n) = \Omega(g(n))$$
$$\Leftrightarrow$$
$$\exists\, c, n_0 > 0 \text{ s.t. } \forall\, n \geq n_0,$$
$$f(n) \geq c \cdot g(n)$$

inequality switched directions!

In Pictures



Big-Ω Notation

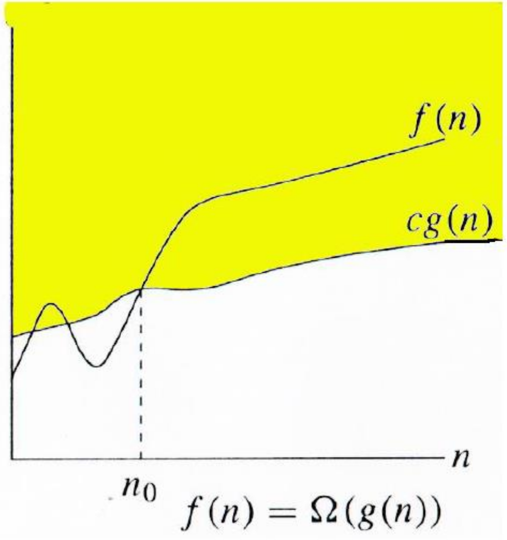
Let f(n) & g(n) be functions defined on the positive integers.

What do we mean when we say "f(n) is Ω(g(n))"?

In *Math*

$$f(n) = \Omega(g(n))$$
$$\Leftrightarrow$$
$$\exists\ c, n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$
$$f(n) \geq c \cdot g(n)$$

inequality switched directions!

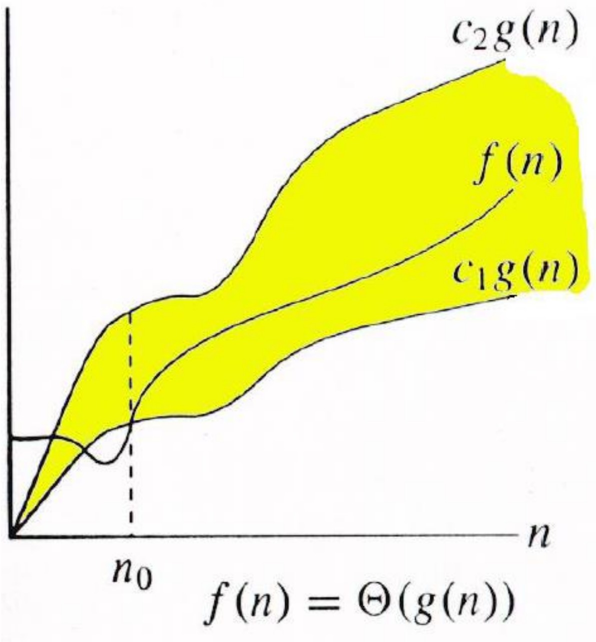


Big-Θ Notation

We say **"f(n) is Θ(g(n))"**
if and only if both

f(n) = O(g(n))
and
f(n) = Ω(g(n))

$$f(n) = \Theta(g(n))$$
$$\Leftrightarrow$$
$$\exists\, c_1, c_2, n_0 > 0 \text{ s.t. } \forall\, n \geq n_0,$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$



PROVING BIG-ϴ NOTATION

Prove that $n^2 + 4n^2 = \Theta(n^2)$.

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

$\Leftrightarrow$

$\exists\, c_1, c_2, n_0 > 0$ s.t. $\forall\, n \geq n_0$,

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

$n^2 + 4n^2 = \Theta(n^2)$ $c_1 = ?, c_2 = ?\ n_0 = ?$

$$c_1 \times n^2 \leq n^2 + 4n^2 \leq c_2\, n^2$$

$$c_1 \times n^2 \leq 5n^2 \leq c_2\, n^2$$

$$1 \times n^2 \leq 5n^2 \leq 5\, n^2$$

$$1 \times n^2 \leq n^2 + 4n^2 \leq 5\, n^2$$

$c_1 = 1, c_2 = 5\ n_0 = 1$

Asymptotic Notations (continued)

O(1) - Constant Time

- Algorithm requires same fixed number of steps regardless of the size of the task.
- For example: Push/Pop in Stack or Insert or Remove for a Queue.
- Constant Time Algorithms are best algorithms unless that time is very long.
- 25 = O(1), i.e. [any constant] = O(1)

O(n) – Linear Time

- Algorithm requires number of steps proportional to the size of the task.
- For example: Traversal of linked-list or array, finding max./min. element in a list etc.

O (lg n)

- Algorithm having running time growing more slowly than the size of the input.
- Double the input, and the running time only gets a little longer, not doubled.
- For example: Binary Search.

Asymptotic Notations (continued)

$O(n^2)$ - Quadratic Time

- The number of operations is proportional to the size of task squared.
- Example 1: Selection sort of n elements.
- Example 2: Comparing two-dimensional array of size n by n

Big-O notation

- Big-O only gives sensible comparison of algorithms in different complexity classes when n is large.
- Big-O notation cannot compare algorithms in the same complexity class.
- For example: $O(n^2)$ is a set, or family, of fucntion with the same of smaller order of growth like $n^2 + n$, $100n + 5$, $4n^2 – n \lg n + 12$, $n^2/5 – 100n$, $n \log n$, $50n$, and so forth. Moreover, note! $n^3 \notin O(n^2)$

Arithmetic of of Big-O, Ω and Ɵ Notations

Transitivity

- $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ $\Rightarrow$ $f(n) \in O(h(n))$
- $f(n) \in \Omega(g(n))$ and $g(n) \in \Omega(h(n))$ $\Rightarrow$ $f(n) \in \Omega(h(n))$
- $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n))$ $\Rightarrow$ $f(n) \in \Theta(h(n))$

Scaling

- If $f(n)) \in O(g(n))$ then for any $k > 0$, $f(n)) \in O(k.g(n))$

Reflexivity

- $f(n) \in \Theta(g(n))$ then $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$

Arithmetic of of Big-O, Ω and ϴ Notations

Sums

- If $f_1(n)) \in O\ (g_1(n))$ and $f_2(n)) \in O\ (g_2(n))$ then $(f_1 + f_2)(n) \in O\ (\max(g_1(n), g_2(n))$

Symmetry

$f(n) \in \Theta\ (g(n))$ if and only if $g(n) \in \Theta\ (f(n))$

Transpose Symmetry

- $f(n) \in O\ (g(n))$ if and only if $g(n) \in \Omega\ (f(n))$
- $f(n) \in o\ (g(n))$ if and only if $g(n) \in \omega\ (f(n))$

Arithmetic of of Big-O, Ω and Ɵ Notations

- $f_1(n) * f_2(n) = O(g_1(n) * g_2(n))$

- $O(n^{c1}) \subset O(n^{c2})$ for any $c1 < c2$

- For any costants $a, b, c > 0$

  $O(a) \subset O(\log n) \subset O(n^b) \subset O(c^n)$

- Multipying with n, will result in:

  $O(an) \subset O(n.\log n) \subset O(n^{b+1}) \subset O(nc^n)$

Little-o Notation

Let $f(n)$ & $g(n)$ be functions defined on the positive integers.

What do we mean when we say "$f(n)$ is $o(g(n))$"?

In Math

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

$$\Leftrightarrow$$

$$\forall\ c > 0,\ \exists\ n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$

$$f(n) < c \cdot g(n)$$

$f(n)$ becomes insignificant relative to $g(n)$ as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = 0$$

$g(n)$ is an **upper bound** *for $f(n)$ that is* ***not asymptotically tight****.*

o notation

f(n) = o(g(n)) for **'any'** **constant** **$c > 0$** there is a constant **n_0** > 0 such that

$$0 \le f(n) < c \cdot g(n)$$

$3n^2 + 5n = O(n^2)$ *asymptotically tight.*

But

$3n^2 + 5n = O(n^3)$ is not *asymptotically tight.*

$3n + 5 = O(n^2)$ is not *asymptotically tight.*

$3n + 5 = o(n^2)$

$3n^2 + 5 \neq o(n^2)$

Little- ω Notation

Let f(n) & g(n) be functions defined on the positive integers.

What do we mean when we say "f(n) is ω(g(n))"?

In Math

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

$$\Leftrightarrow$$

$$\forall\ c > 0,\ \exists\ n_0 > 0 \text{ s.t. } \forall\ n \geq n_0,$$

$$f(n) > c \cdot g(n)$$

f(n) becomes very large relative to g(n) as n approaches infinity:

$$\lim_{n \to \infty} [f(n) / g(n)] = \infty$$

g(n) is an **lower bound** *for f(n) that is* ***not asymptotically tight***.

ω notation

f(n) = ω(g(n)) for ‘**any**’ **constant**
$c > 0$ there is a constant **n_0** > 0 such that

$$0 \le f(n) > c \cdot g(n)$$

$$3n^2 + 5 = \omega(n)$$

$$3n + 5 \neq \omega(n)$$

g(n) is **lower bound** *for f(n) that is not asymptotically tight.*

Asymptotic Notation Summary

Bound	Definition (How To Prove)	It Represents
$f(n) = O(g(n))$	$\exists\ c > 0,\ \exists\ n_0 > 0$ s.t. $\forall\ n \geq n_0,\ f(n) \leq c \cdot g(n)$	upper bound
$f(n) = o(g(n))$	$\forall\ c > 0, \exists\ n_0 > 0$ s.t. $\forall\ n \geq n_0,\ f(n) < c \cdot g(n)$	upper bound Not asymptotically tight
$f(n) = \Omega(g(n))$	$\exists\ c > 0,\ \exists\ n_0 > 0$ s.t. $\forall\ n \geq n_0,\ f(n) \geq c \cdot g(n)$	lower bound
$f(n) = \omega(g(n))$	$\forall\ c > 0, \exists\ n_0 > 0$ s.t. $\forall\ n \geq n_0,\ f(n) > c \cdot g(n)$	lower bound Not asymptotically tight
$f(n) = \Theta(g(n))$	$f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$	tight bound

Comparison Of functions

$f(n) = O(g(n)) \approx a \leq b$

$f(n) = \Omega(g(n)) \approx a \geq b$

$f(n) = \Theta(g(n)) \approx a = b$

$f(n) = o(g(n)) \approx a < b$

$f(n) = \omega(g(n)) \approx a > b$

Proving Big- Θ Bounds: Example

Show that $\mathrm{f(n)}\,\frac{1}{2}\,n^2 - 3n = \theta(n^2)$

f(n) = Θ(g(n))

⇔

$\exists\ c_1, c_2, n_0 > 0$ s.t. $\forall\ n \geq n_0$,

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

find c_1, c_2 & n_0 such that for all $n \geq n_0$:

$$c_1 . n^2 \leq \frac{1}{2}\, n^2 - 3n \leq c_2 \,.\, n^2$$

$$0 \leq c_1 . n^2 \leq \frac{1}{2}\, n^2 - 3n \leq c_2 \,.\, n^2$$

Divide by n^2: $0 \leq c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

$c_1 \leq \frac{1}{2} - \frac{3}{n}$ **holds for $n \geq 7$** and $c_1 \leq \frac{1}{14}$

$\frac{1}{2} - \frac{3}{n} \leq c_2$ **holds for $n \geq 7$** and $c_2 \geq \frac{1}{14}$

$$0 \leq \frac{1}{14} n^2 \leq \frac{1}{2}\, n^2 - 3n \leq \frac{1}{14} . n^2$$

Thus it is shown that $\frac{1}{2}\, n^2 - 3n = \theta(n^2)$ **[for $c_1 = \frac{1}{14}$, $c_2 = \frac{1}{14}$, $n_0 = 7$]**

Proving Big- ϴ Bounds: Example (Method # 1)

Show that $f(n)\frac{1}{2}n^2 - 3n = \theta(n^2)$

find c_1, c_2 & n_0 such that for all $n \geq n_0$:

$c_1 . n^2 \leq \frac{1}{2} n^2 - 3n \leq c_2 . n^2$

$0 \leq c_1 . n^2 \leq \frac{1}{2} n^2 - 3n \leq c_2 . n^2$

Divide by n^2: $0 \leq c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$

$c_1 \leq \frac{1}{2} - \frac{3}{n}$ holds for **$n \geq 7$ and $c1 \leq \frac{1}{14}$**

$\frac{1}{2} \leq c_2$ holds for **$n \geq 1$ and $c_2 \geq \frac{1}{2}$**

$0 \leq \frac{1}{14} . n^2 \leq \frac{1}{2} n^2 - 3n \leq \frac{1}{2} . n^2$

Thus it is shown that $\frac{1}{2} n^2 - 3n = \theta(n^2)$ [for $c_1 \leq \frac{1}{14}$, $c_2 \geq \frac{1}{2}$, $n_0 = 7$]

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

$\Leftrightarrow$

$\exists\ c_1, c_2, n_0 > 0$ s.t. $\forall\ n \geq n_0$,

$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$