CS 2009
Design and Analysis of Algorithms

Lecture 2 and 3:

Growth Of Function

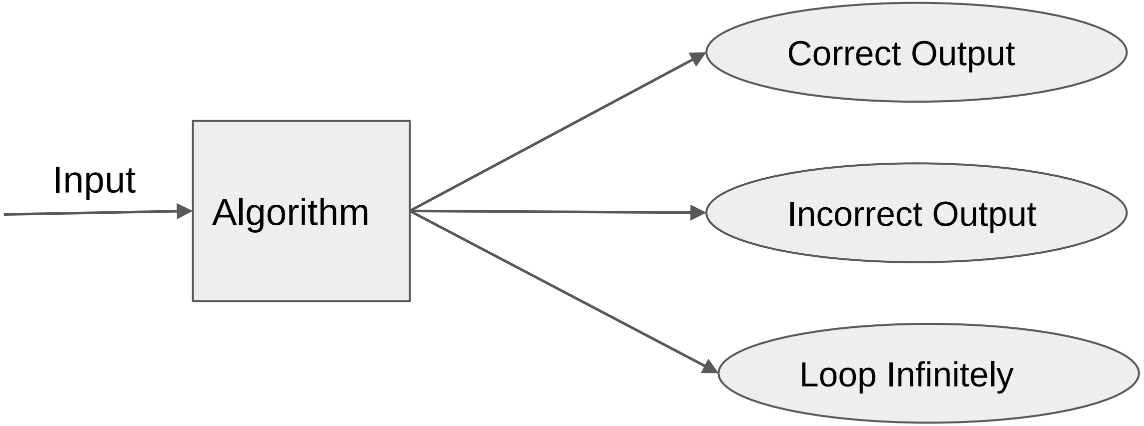
September 8-9, 2021

Prepared by: Waheed Ahmed

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

Correct Algorithm

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



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

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

MULTIPLICATION PROBLEM

$$\begin{array}{rr} & \overbrace{12345678998765432101}^{\boldsymbol{n}\ \textit{digits}} \\ \times & 98765432112345678901 \\ \hline \end{array}$$

How efficient is this algorithm?

(How many single-digit operations are required?)

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

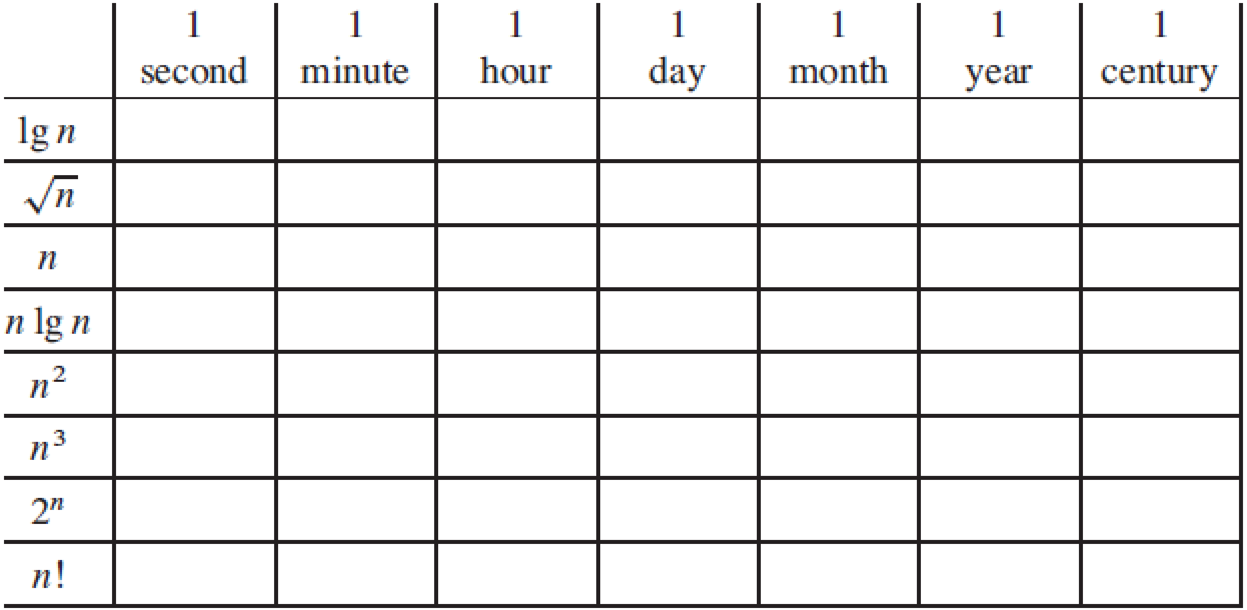
$$\frac{2 \,.\, (10^7)^2 \text{ Instructions}}{10^{10} \text{ Instructions/second}} = 20{,}000 \text{ seconds } (> 5.5 \text{ hours})$$

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

Growth Rate Ranking of Function?

Comparison of running times

For each function f (n) and time t in the following table, determine the largest size n of a problem that can be solved in time t, assuming that the algorithm to solve the problem takes f(n) microseconds.



	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Efficiency of Algorithm

INTRODUCING...

ASYMPTOTIC ANALYSIS

Some guiding principles:

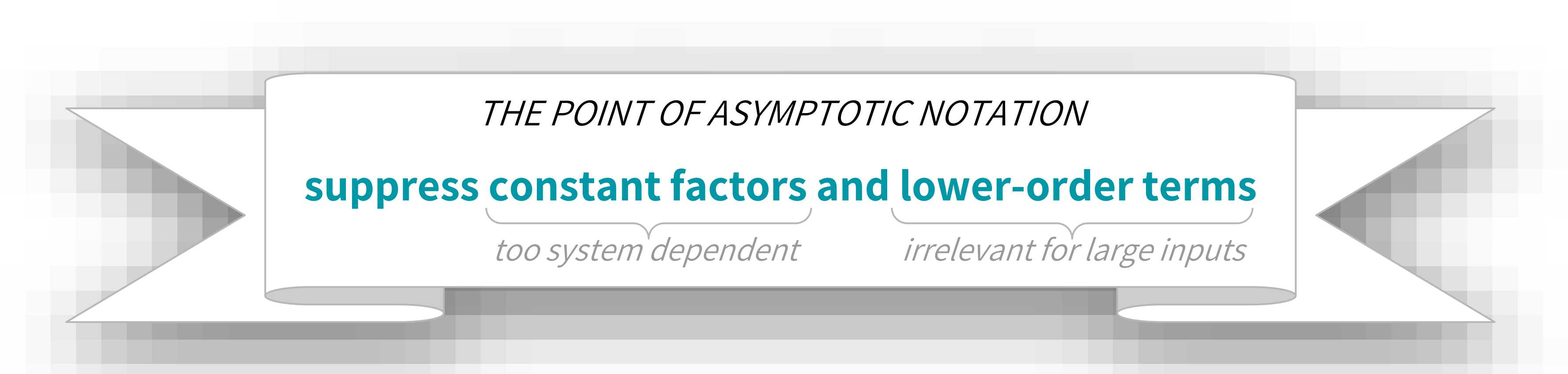
- we want some measure of runtime that's independent of hardware, programming language, memory layout, etc.
  - We want to reason about high-level algorithmic approaches rather than lower-level details
- we care about how the running time/number of operations *scales* with the size of the input (i.e. 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



THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent

irrelevant for large inputs

ASYMPTOTIC ANALYSIS (High Level Idea)

BIG-O NOTATION

THE POINT OF ASYMPTOTIC NOTATION

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

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

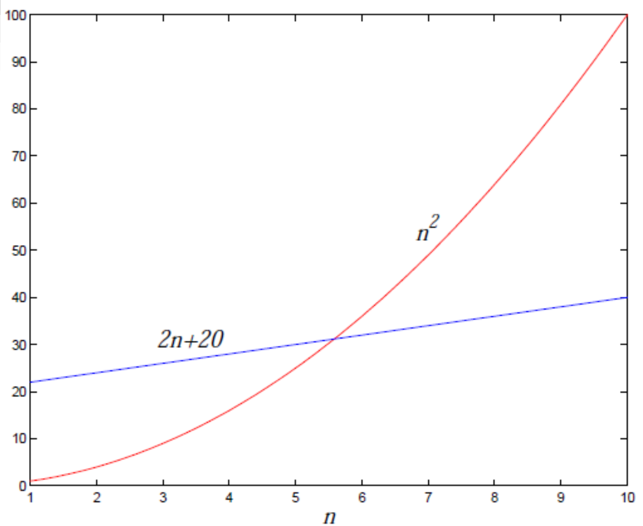
suppress constant factors and lower-order terms

too system dependent *irrelevant for large inputs*

$f_1\ (n) = n^2$

$f_2\ (n) = 2n + 20$

Which is better?



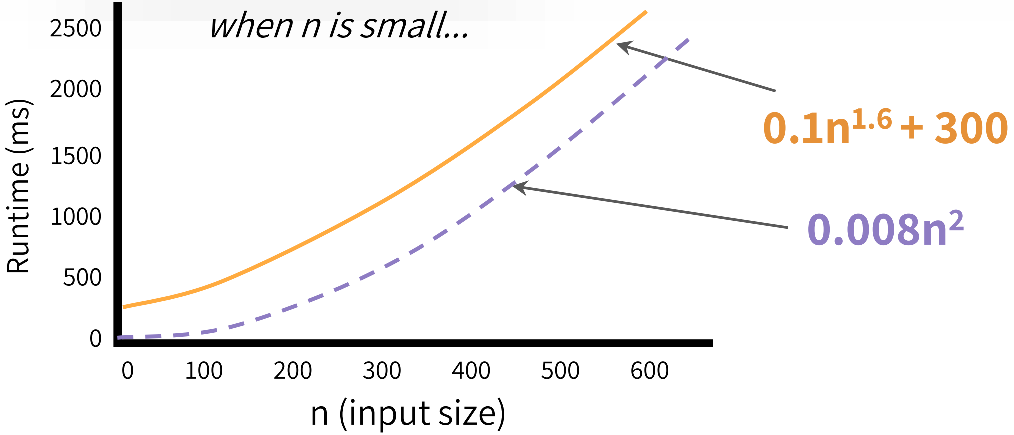
ASYMPTOTIC ANALYSIS (High Level Idea)

THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent (constant factors)

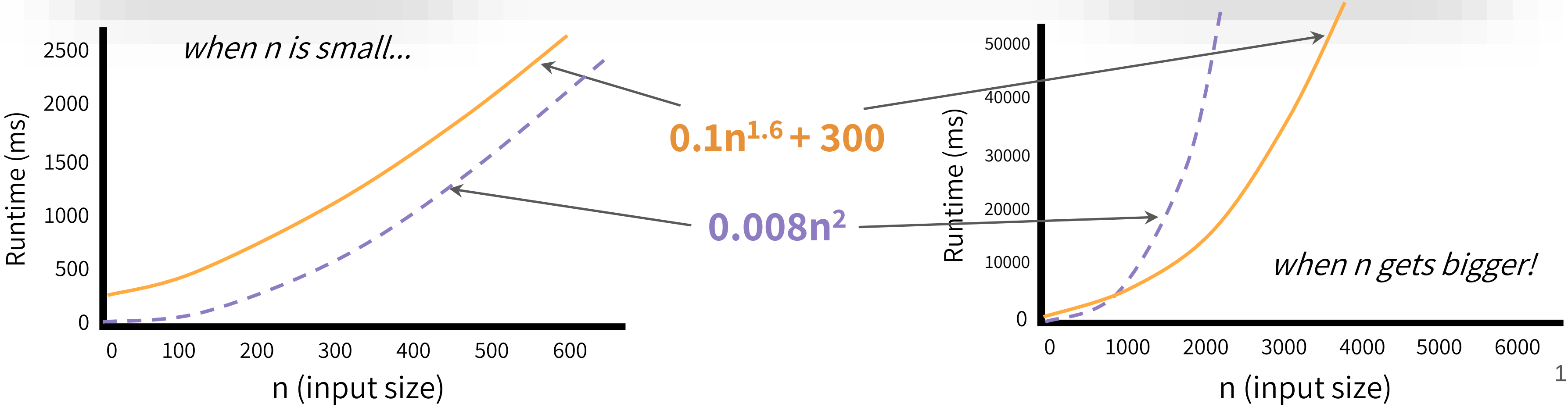
irrelevant for large inputs (lower-order terms)



ASYMPTOTIC ANALYSIS (High Level Idea)

THE POINT OF ASYMPTOTIC NOTATION

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

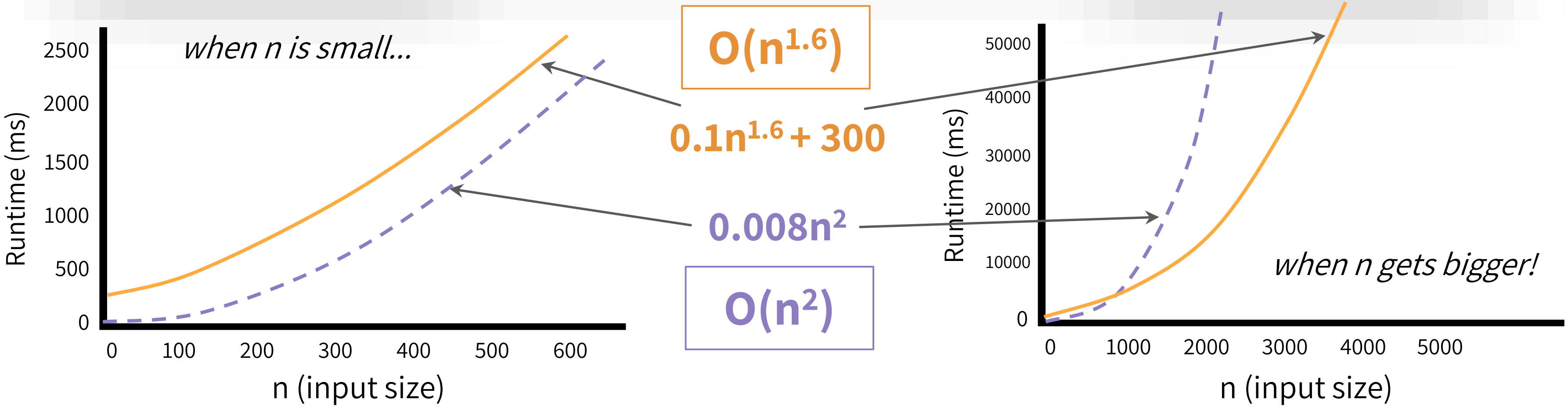


ASYMPTOTIC ANALYSIS (High Level Idea)

THE POINT OF ASYMPTOTIC NOTATION

suppress constant factors and lower-order terms

too system dependent *irrelevant for large inputs*



$O(n^{1.6})$

$0.1n^{1.6} + 300$

$0.008n^2$

$O(n^2)$

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 $O(1)$?

Which Running Time Is Better?

Skip in Class

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

Growth of Function

Skip in Class

n	$\log_2 n$	$n \log_2 n$	n^2	2^n
1	0	0	1	1
2	1	2	4	8
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65536
32	5	160	1024	4294967296

Skip in Class

Growth Rate Ranking of Function?

$$f(n) = n^n$$

$$f(n) = 2^n$$

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

$$f(n) = n^2$$

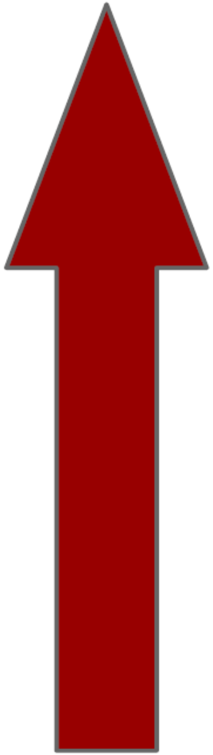
$$f(n) = n \log n$$

$$f(n) = n$$

$$f(n) = \sqrt{n}$$

$$f(n) = \log n$$

$$f(n) = 1$$



grow fast

grow slowly