

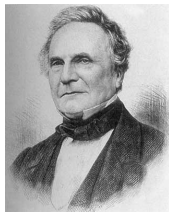
CS 320 Fall 2021 Runtime Analysis (Big O)

Sanjay Rajopadhye
Colorado State University

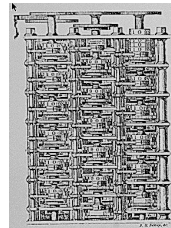
1

Runtime Analysis

As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time? - *Charles Babbage*



Charles Babbage (1864)



Analytic Engine (schematic)

2

Outline: four topics

- Algorithm time complexity
- Plotting data and the function clubs
- Digression: line of sight algorithm
- A survey of common running times

Colorado State University ³

3

Algorithm Time Complexity

Colorado State University ⁴

4

Algorithm Time Complexity

- How do we measure the **complexity** (time, space requirements) of an algorithm?
 - As a function of its input **size** (an integer, n) denoting:
 - Number of inputs (e.g., sorting)
 - Number of bits to represent the input (e.g., primality)
 - Sometimes multiple parameters, e.g., knapsack
 - Number of objects, n
 - Knapsack capacity, C

We want to determine the running time as a function of problem sizes, and analyze them asymptotically

Colorado State University ⁵

5

How to measure time?

- Seconds/nano-seconds?
 - **No, too specific & machine dependent**
- Number of instructions executed?
 - **No, still too specific & machine dependent**
- # of code fragments that take constant time?
 - **Yes**
- What kind of fragments/instructions?
 - **Arithmetic operations, memory accesses, finite combinations of these**

Colorado State University ⁶

6

How to measure space?

- Bits?
 - Too detailed, but sometimes necessary (e.g., knapsack capacity)
- Integers?
 - Nicer, but dangerous – we can code a whole program in a single arbitrary sized integer, so we have to be careful about the size. Better to use machine words
i.e, fixed size (e.g., 64, collections of bits)

Colorado State University ⁷

7

Worst/average case time

- A bound on the maximum possible running time of the algorithm of inputs of size n
 - Usually captures the notion, but may be an overestimate
- Average case
 - More accurate but difficult – need to describe what is the range of inputs, and what is the distribution, statistical analysis. Let I be the set of inputs, and P_i and C_i be the probability and computation time of input i

$$\sum_{i \in I} P_i C_i$$

- Often a constant factor of worst case time

Same considerations for space and other measures.

Colorado State University ⁸

8

Why it matters

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Colorado State University 9

9

Tractable = polynomial time

- For many problems, there is a natural, but likely naïve, brute force search algorithm that checks every possible solution
 - Enumerating such solutions is usually an exponential function of n (recall counting from CS220).
 - Hence naïve
- **Definition:** an algorithm is said to be polynomial time if there exist positive constants c , and d , such that on any input of size n , the running time is bounded by $c n^d$

What about an algorithm whose running time is $c n \lg n$?

Colorado State University 10

10

Justification/issues

(Why) is the distinction important?

- On the one hand, a polynomial function like $6.03 \times 10^{23} n^{20}$ is polynomial, it is too large in practice (e.g., for $n=10$)
- On the other hand, some algorithm whose worst-case execution time is exponential **behave much better** in practice because the worst-case instances are (seem to be) rare
 - Simplex method for solving **linear programming**

So why?

- In practice, the polynomials have a low degree and coefficients
- The difference between the polynomial-exponential barrier reveals interesting and crucial structure of the problem

Colorado State University ¹¹

11

Asymptotic growth rates

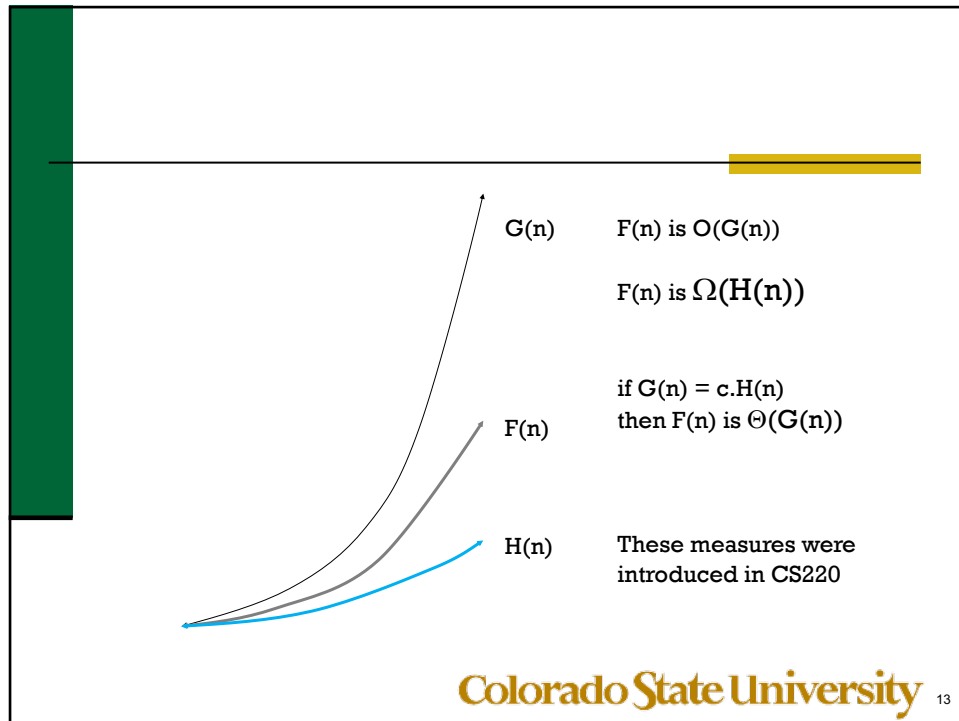
- We are building mathematical functions that model the execution time (or other properties) of programs and algorithms.
- Need a mechanism to **compare** them.
 - How do we compare numbers? Using the relations: $<$, $>$, \leq , and \geq
 - Partial/total orders
- The Big-Oh, Big-Omega and Big-Theta notation (introduced in CS 220) is such an order relation.

Here, $f \leq g$ means that f grows **slower** than g (and also that g grows **faster** than f). We may also use $g > f$. So the following claims mean the same thing

- $f(n) \leq g(n)$
 - $g \geq f$ or $g > f$
 - $f = O(g)$
 - $g(n) = \Omega(f(n))$
- Often, one of the functions is our (complicated) model $T(n)$ and the other is a simpler function (e.g., a monomial)

Colorado State University ¹²

12



13

Basic definitions

A function $T(n)$ is $O(f(n))$ if there exist constants $c > 0$ and $n_0 > 0$ such that

$$\text{for all } n \geq n_0 : T(n) \leq c f(n)$$

■ Example: $T(n) = 32n^2 + 16n + 32$.

■ $T(n)$ is $O(n^2)$

■ **ALSO TRUE:**

■ $T(n)$ is $O(n^3)$

■ $T(n)$ is $O(2^n)$

■ Many possible upper bounds for one function!
 We always look for the best (lowest) upper bound, but it is not always easy to establish

Colorado State University 14

14

Properties of \ll , \gg and O

- **Transitivity**
 - $f \ll g$ and $g \ll h$ implies $f \ll h$
- **Additivity (Additive slowdown)**
 - $f \ll h$ and $g \ll h$ implies $f + g \ll h$
- **Multiplication by a constant**
 - $f \ll g$ implies $c \times f \ll g$ (and of course $f \ll c \times g$ holds by definition)

Colorado State University ¹⁵

15

Lower bounds (convention)

Although Big-Oh and Big-Omega are equivalent, a special need arises when our model $T(n)$ is quantified over **all algorithms to solve the given problem**

- **Example:** consider the claim that any comparison based algorithm must make at least $c \times n \log n$ comparisons, for some constant, c . We say that comparison based sorting is **lower bounded** by $n \lg n$, i.e., that $T(n)$ is $\Omega(n \lg n)$ and we often reserve the Ω notation for this.
- **Problems have lower bounds**
 - A common lower bound is the size of the input itself (any algorithm to solve the problem must read all the inputs)
 - Sometimes we can prove better/tighter lower bounds (e.g., sorting above and searching is structured data ([CS 420](#)))

Colorado State University ¹⁶

16

Tight Bounds

- If $T(n)$ is $\Omega(f(n))$ and $T(n)$ is also $O(f(n))$ we have a tight bound, and we write that $T(n)$ is $\Theta(f(n))$.
- It means that we have **closed the problem**, since the **algorithm** that we have attains the lower bound on the **problem**

Colorado State University ¹⁷

17

Closed and Open Problems

Sorting is a **closed problem**

- It has a lower bound of $n \log n$. We say that **sorting is $\Omega(n \log n)$**
- There are many sorting algorithms whose execution time is $O(n \log n)$ (**see how we use big-Oh to talk about an algorithm**)

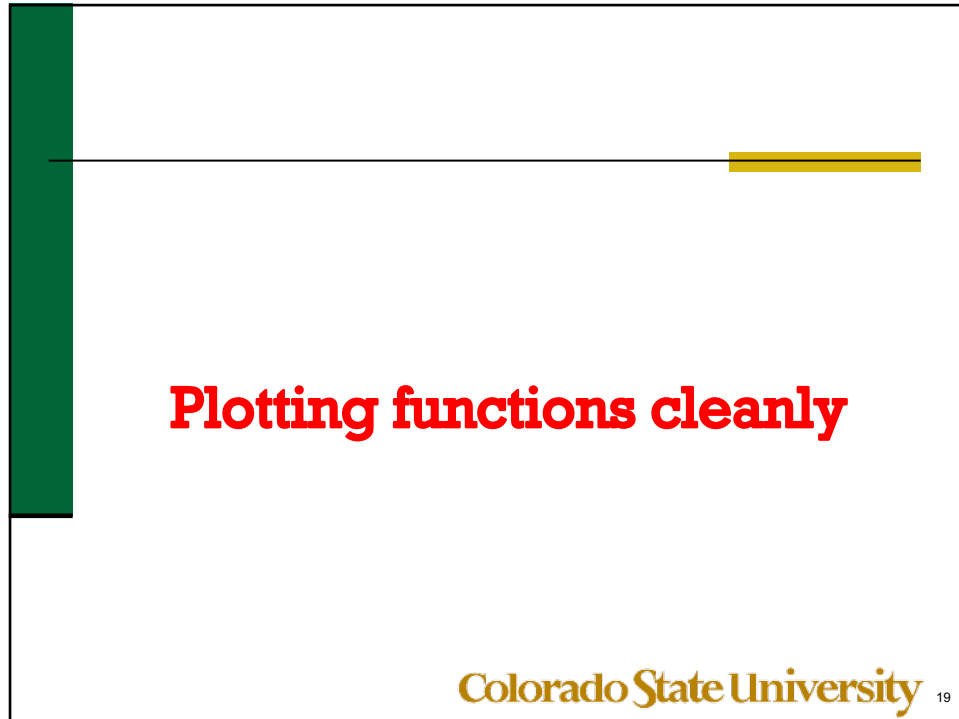
Matrix multiplication is an **open problem**

- It is $\Omega(n^2)$. **Why?**
 - The standard algorithm is $O(n^3)$
 - Another well known algorithm is $O(n^{2.376})$ and further improvements reduce the polynomial degree even further

See how the polynomial degree does not have to be integer

Colorado State University ¹⁸

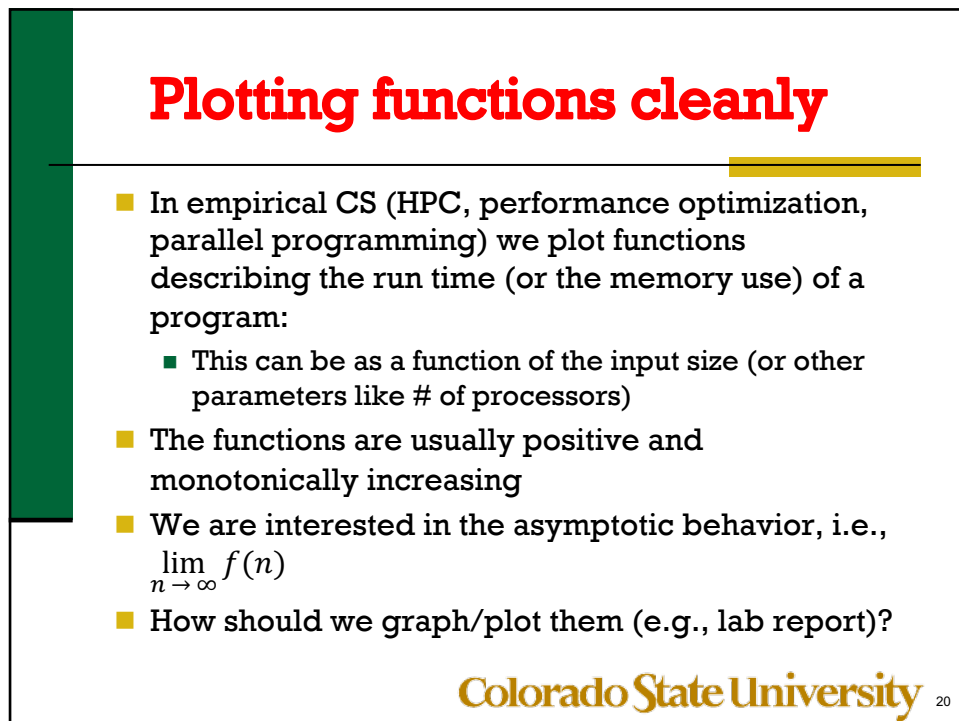
18



Plotting functions cleanly

Colorado State University ¹⁹

19



Plotting functions cleanly

- In empirical CS (HPC, performance optimization, parallel programming) we plot functions describing the run time (or the memory use) of a program:
 - This can be as a function of the input size (or other parameters like # of processors)
- The functions are usually positive and monotonically increasing
- We are interested in the asymptotic behavior, i.e.,
 $\lim_{n \rightarrow \infty} f(n)$
- How should we graph/plot them (e.g., lab report)?

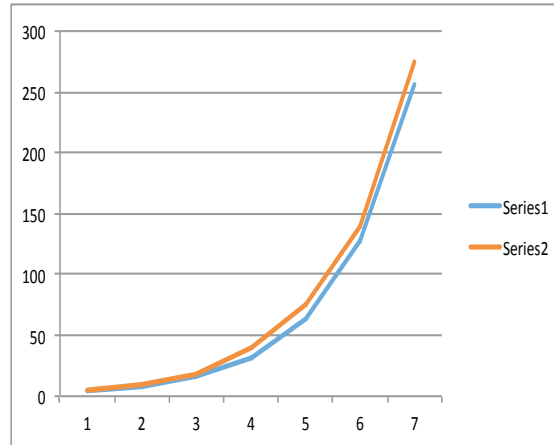
Colorado State University ²⁰

20

Ungraded Quiz (survey)

- The plot shows the **ideal (expected)** vs **empirical (observed)** values. Which one is **ideal**, and which is “just a bit off?”

- Series 1 (blue)
- Series 2 (orange)

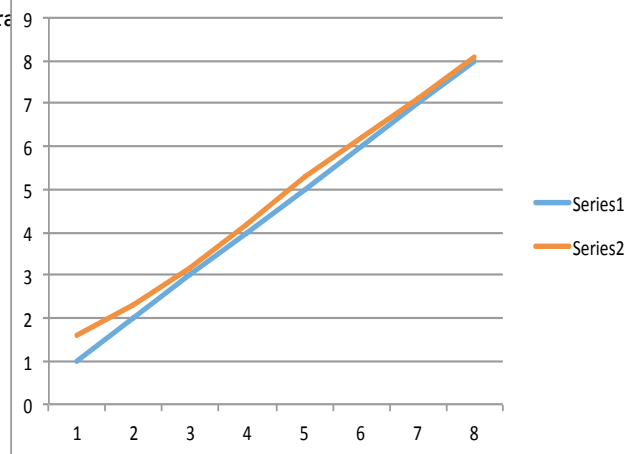


21

Ungraded Quiz (survey)

- Same question, data is plotted differently.

- Series 1 (blue)
- Series 2 (orange)



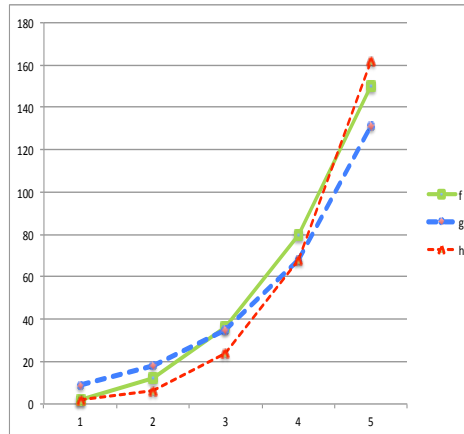
22

Three functions: f, g and h

n	f(n)	g(n)	h(n)
1	2	9	2
2	12	18	6
3	36	35	24
4	80	68	68
5	150	131	162

n	f(n)	g(n)	h(n)
1	2	9	2
2	12	18	6
3	36	35	24
4	80	68	68
5	150	131	162

- What class of functions are f, g, and h?
 - Polynomial? What degree?
 - Exponential? What base?
- Impossible/hard to tell



Colorado State University 23

23

Why is it hard?

- The human visual system is very good at identifying linear (straight line) plots.
- Everything else is approximate.
- Asymptotically increasing functions just “swoosh up,” i.e., $\lim_{n \rightarrow \infty} f(n) = \infty$
- Not enough range of data in second set of examples here just 1 ... 5)

Colorado State University 24

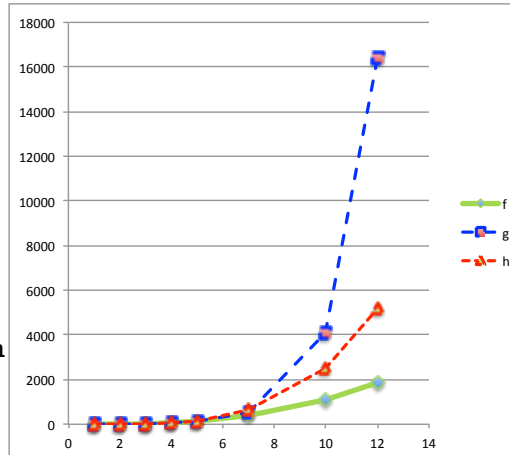
24

Larger domains for f, g and h

n	f(n)	g(n)	h(n)
1	2	9	2
2	12	18	6
3	36	35	24
4	80	68	68
5	150	131	162
7	400	520	624
10	1100	4106	2510
12	1872	16396	5196

Much better idea now about which function may be polynomial vs exponential? But still

- all is not clear (order, base ...)
- h(n) may spike up later...



Colorado State University 25

25

Straight lines

We get the most information from **straight lines!**

- We can easily recognize a straight line

$$y = ax + b$$
 - The **slope (a)** and **y intercept (b)** tells us all.
 - **How to “massage the data” into straight lines.**
 - Change the scale to logarithmic: it turns a **multiplicative** factor into a **shift** (y axis crossing b), and an **exponential** into a **multiplicative factor** (slope a)

Colorado State University 26

26

Four exponential functions

$$y = 2^n \quad \log_{10}(y) = n \log_{10} 2 \quad \text{linear in } n$$

$$y = 3^n \quad \log_{10}(y) = n \log_{10} 3$$

the slope is the (log of the) base of the exponent

$$y = 6 \times 3^n \quad \log_{10}(y) = n \log_{10} 3 + \log_{10} 6$$

6 shifts up (in log scale)

$$y = 3^n / 5 \quad \log_{10}(y) = n \log_{10} 3 - \log_{10} 5$$

5 shifts down

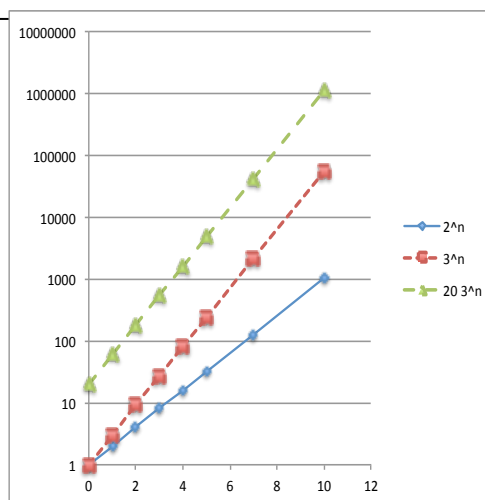
Colorado State University 27

27

Use a semi-log plot

n	2 ⁿ	3 ⁿ	20*3 ⁿ
0	1	1	20
1	2	3	60
2	4	9	180
3	8	27	540
4	16	81	1620
5	32	243	4860
7	128	2087	41740
10	1024	56349	1126980

semi-log plot:
 y-axis on log scale
 x-axis linear
 angle: base
 shift: multiplicative factor



Colorado State University 28

28

What about polynomials?

What is the logarithm of a polynomial (actually a monomial)?

- $y = 5n^3$

- $\log_{10}(y) = \log_{10} 5 + \log_{10} n^3 = \log_{10} 5 + 3 \log_{10} n$

Definitely not a straight line.

- But what about this?
- So we use a log-log scale/plot

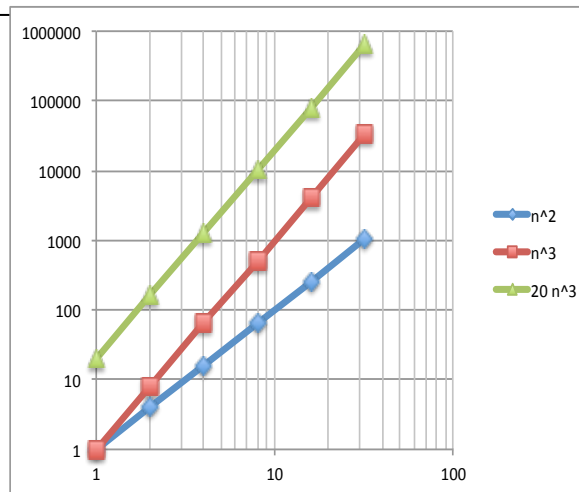
Colorado State University 29

29

Polynomial on log-log plot

n	n ²	n ³	20*n ³
1	1	1	20
2	4	8	160
4	16	64	1280
8	64	512	10240
16	256	4096	81820
32	1024	32768	655360

slope: degree
shift: multiplicative factor



Colorado State University 30

30

Handling multiple terms

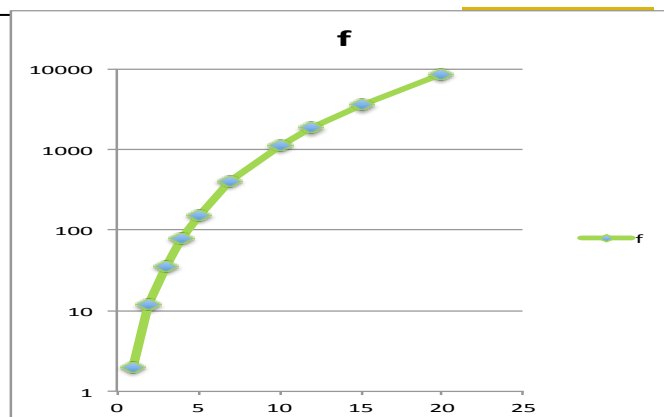
- Functions like $f(n) = 3^n + 4^n$ and polynomials that have more than one term. We don't have a simple algebraic rule to compute logarithms of the sum of multiple terms
 - Now, $f(n) = 3^n + 4^n = 4^n \left(1 + \left(\frac{3}{4}\right)^n\right)$
 - and since $\left(\frac{3}{4}\right) < 1$, so $\lim_{n \rightarrow \infty} \left(1 + \left(\frac{3}{4}\right)^n\right) = 1$
 - so, as $n \rightarrow \infty$, we have $\log f(n) \rightarrow \log 4^n \times 1 = \log 4 \times n$ i.e., only the dominant term matters
- For a polynomial like $f(n) = 4 \times n^3 + 3 \times n^2$ we do the same thing $f(n) = n^3 \left(4 + \frac{3}{4n}\right)$ and as $n \rightarrow \infty$, the term in parentheses approaches 4, so $\log f(n) \rightarrow \log 4 \times n^3 = \log 4 + 3 \log n$
- **Message: when plotting your data, look for the trend among the points with larger input values**

Colorado State University 31

31

Infer the function

n	f(n)
1	2
2	12
3	36
4	80
5	150
7	400
10	1100
12	1872



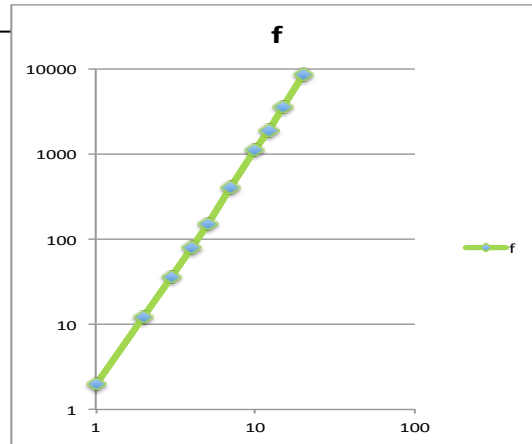
The semi-log plot does not give a straight line, so f is not exponential

Colorado State University 32

32

What about a log-log plot

n	f(n)
1	2
2	12
3	36
4	80
5	150
7	400
10	1100
12	1872



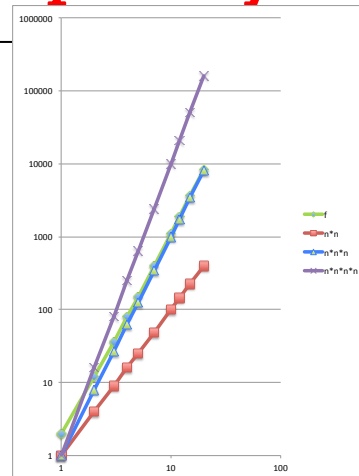
YES! The log-log plot is asymptotically a straight line, so f is polynomial, but what is its leading term?

Colorado State University 33

33

Continue empirically

n	f(n)	n ²	n ³	n ⁴
1	2	1	1	1
2	12	4	8	16
3	36	9	27	81
4	80	16	64	256
5	150	25	125	625
7	400	49	343	2401
10	1100	100	1000	10000
12	1872	144	1728	20736



Compare with n, n^2, n^3, n^4

It is degree 3, no multiplicative factor

Colorado State University 34

34

Polynomial club of functions

- All **polynomial** functions are members
- **Membership test:** to enter the club you scan your id
 - checker is just a **log-log plotter** you're in if it's a straight line with slope between 0° to 90°
- **Slowest** growing polynomial (**fastest** algorithms) are polynomials $f(n) = n^\epsilon$ where, ϵ is an arbitrarily small constant.
- **Fastest** growing polynomial (**slowest** algorithms) $f(n) = n^\Gamma$ where, Γ is an arbitrarily large constant

Colorado State University ³⁵

35

The exponential club

- All **exponential** functions are members
- **Membership test:** to enter the club you scan your id
 - checker is just a **semi-log plotter** you're in if it's a straight line with slope between 0° to 90°
- **Slowest** growing exponential (**fastest** algorithms) are exponential $f(n) = \epsilon^n$ where, ϵ is an arbitrarily small constant.
- **Fastest** growing exponential (**slowest** algorithms) $f(n) = \Gamma^n$ where, Γ is an arbitrarily large constant.

Colorado State University ³⁶

36

Are there other clubs?

- The basic mathematical definition of \ll , \gg , O and Ω still hold: for large enough n one function exceeds the other,
- The plotting trick is simply to compress the x or y axis plotting, and it doesn't change asymptotic behavior
- What if we compress the x axis and not the y axis: a so-called **log-semi** plot (but this naming convention is soon going to prove inadequate)
 - These are the **poly-log functions**: polynomials of $\log n$
 - The worst poly-log algorithm is faster the fastest polynomial algorithm $\log^\Gamma n < n^\epsilon$
- **Super-exponential functions**: straight line when we plot $\log \log f(n)$ vs n

Colorado State University 37

37

Competition between clubs

- Each club conducts their internal tournaments, and ranks their members.
- Algorithm designers try to invent new algorithms for open problems
 - When they give a new algorithm when the previous best was in the same club, they **reduce the slope by a constant**, and it's a big accomplishment, e.g., going from $O(n^3)$ to $O(n^{2.7})$
 - even if that improvement comes at a "cost" of a factor that is equal the slowest member of a faster club
 - A new algorithm that's in a **faster club** a major breakthrough.
- Breakthroughs between the exponential and polynomial clubs are **increasingly unlikely**

Colorado State University 38

38

Strictly vs asymptotically linear

First, we define what it means for a function to be strictly linear and asymptotically linear.

- A function $y = g(x)$ is said to be **strictly linear** if there are constants m and b such that $y = mx + b$
- A function $y = g(x)$ is said to be **asymptotically linear** if $y = mx + b$ when $\lim x \rightarrow \infty$ (we often drop the asymptotically adjective)

Colorado State University 39

39

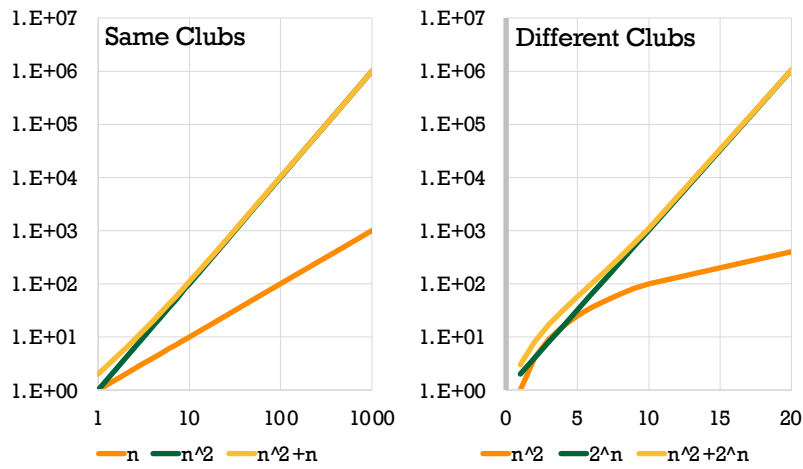
Additive Slowdown

- What happens when you add two functions
 - $g(x) = f1(x) + f2(x)$
- If they are members of the same polynomial club
 - say, $f1(x) = x$, $f2(x) = x^2$
 - $g(x)$ is a member of the polynomial club and asymptotically linear
- If they are members of different clubs, exponential and polynomial
 - say, $f1(x) = 2^x$, $f2(x) = x^2$
 - $g(x)$ is a member of the exponential club and asymptotically linear

Colorado State University 40

40

Additive Slowdown



Colorado State University 41

41

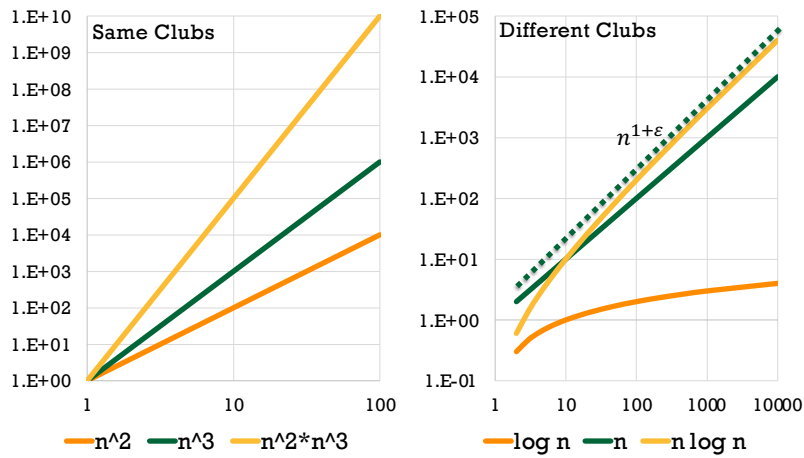
Multiplicative Slowdown

- What happens when we multiply two functions?
 - $g(x) = f1(x) * f2(x)$
- If they are members of the same club
 - say, $f1(x) = x^2$, $f2(x) = x^3$
 - $g(x)$ is a member of the same club (asymptotically linear) with a slope of the sum of the slopes
- If they are members of different clubs
 - say, $f1(x) = x^2$, $f2(x) = \log(x)$
 - $g(x)$ is not a member of the faster growing polynomial club, but it is indistinguishable from it. While not a straight line, even asymptotically it will continue to increase, but you won't be able to notice the difference.

Colorado State University 42

42

Multiplicative Slowdown



Colorado State University ⁴³

43

Scaling Variables

Let $y = f(x)$ be an arbitrary (asymptotically monotonically increasing) function that represents the execution time of a program on an input of size x .

We can introduce four **scaling variables** to massage the input or output data.

- $y' = \log y$
- $y'' = \log \log y$
- $x' = \log x$
- $x'' = \log \log x$

Colorado State University ⁴⁴

44

Massaging Functions

Substituting the **scaling variables** yields the following nine **massaging functions** for the normal, log, and log log cases of each variable. This allows us to massage the input and output data for the different frames of reference in the graphs.

- $y = h_0(x)$ is linear for linear functions
- $y' = h_1(x)$ is linear for exponential functions
- $y'' = h_2(x)$ is linear for doubly exponential
- $y = h_3(x')$ is linear for logarithmic functions
- $y = h_4(x'')$
- $y' = h_5(x')$ is linear for polynomial functions
- $y' = h_6(x'')$ is linear for poly logarithmic functions
- $y'' = h_7(x')$
- $y'' = h_8(x'')$

Colorado State University ⁴⁵

45

Swooping

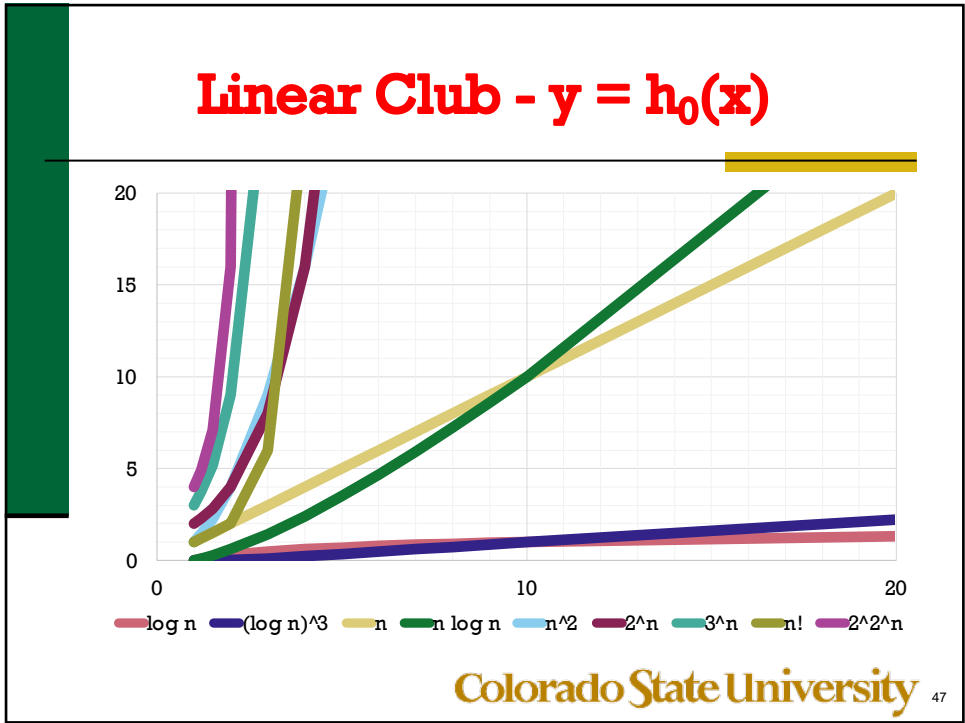
Consider three clubs, C_a , C_b and C_c , with **membership defining functions**, h_a , h_b , and h_c , where C_a grows faster than C_b and C_c grows slower than C_b . Membership defining functions are the massaging functions corresponding to these clubs.

Consider a function $f(x) \in C_b$.

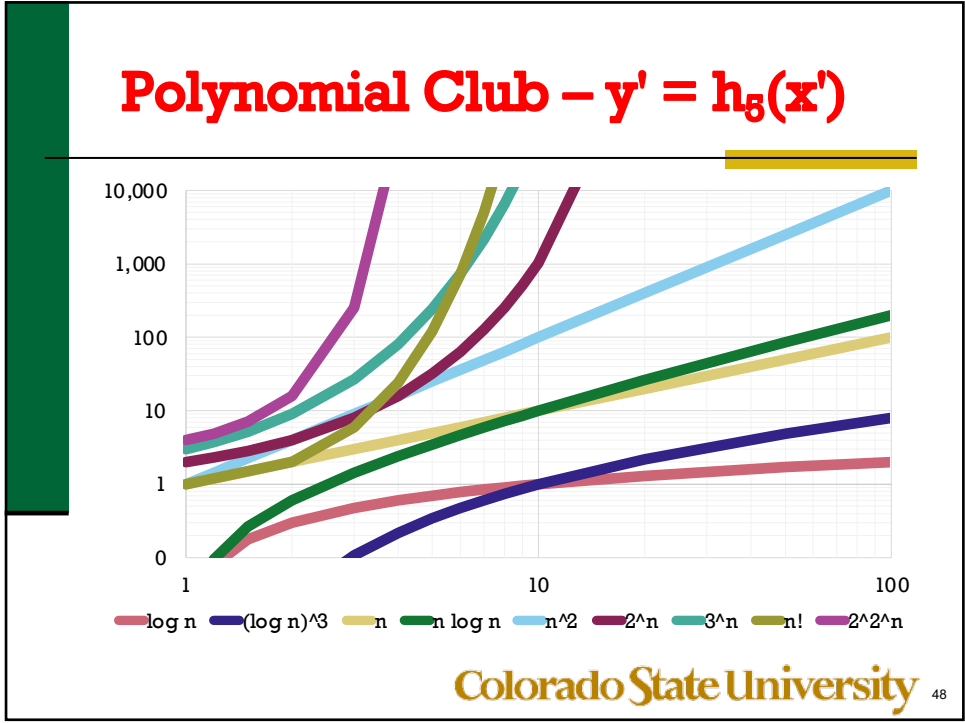
- $f(x)$ **swoops up** if it grows strictly faster than a membership defining function, i.e., even faster than a straight line with slope approaching 90° .
- $f(x)$ **swoops right** if it grows strictly slower than a membership defining function, i.e., even slower than a straight line with slope approaching 0° .

Colorado State University ⁴⁶

46

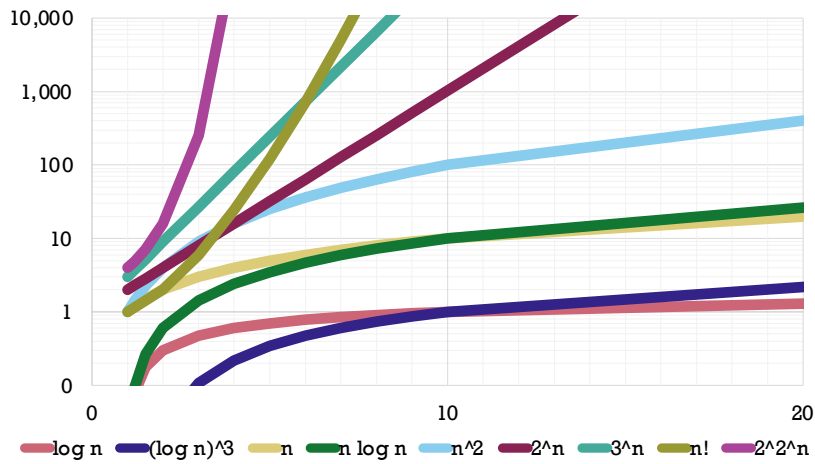


47



48

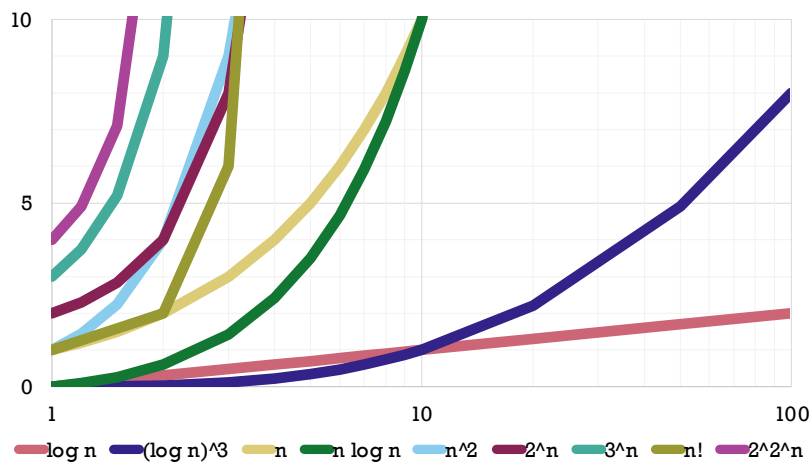
Exponential Club - $y' = h_1(x)$



Colorado State University 49

49

Logarithmic Club - $y = h_3(x')$



Colorado State University 50

50