CMSC201 Computer Science I for Majors

Lecture 24 – Algorithmic Analysis

Last Class We Covered

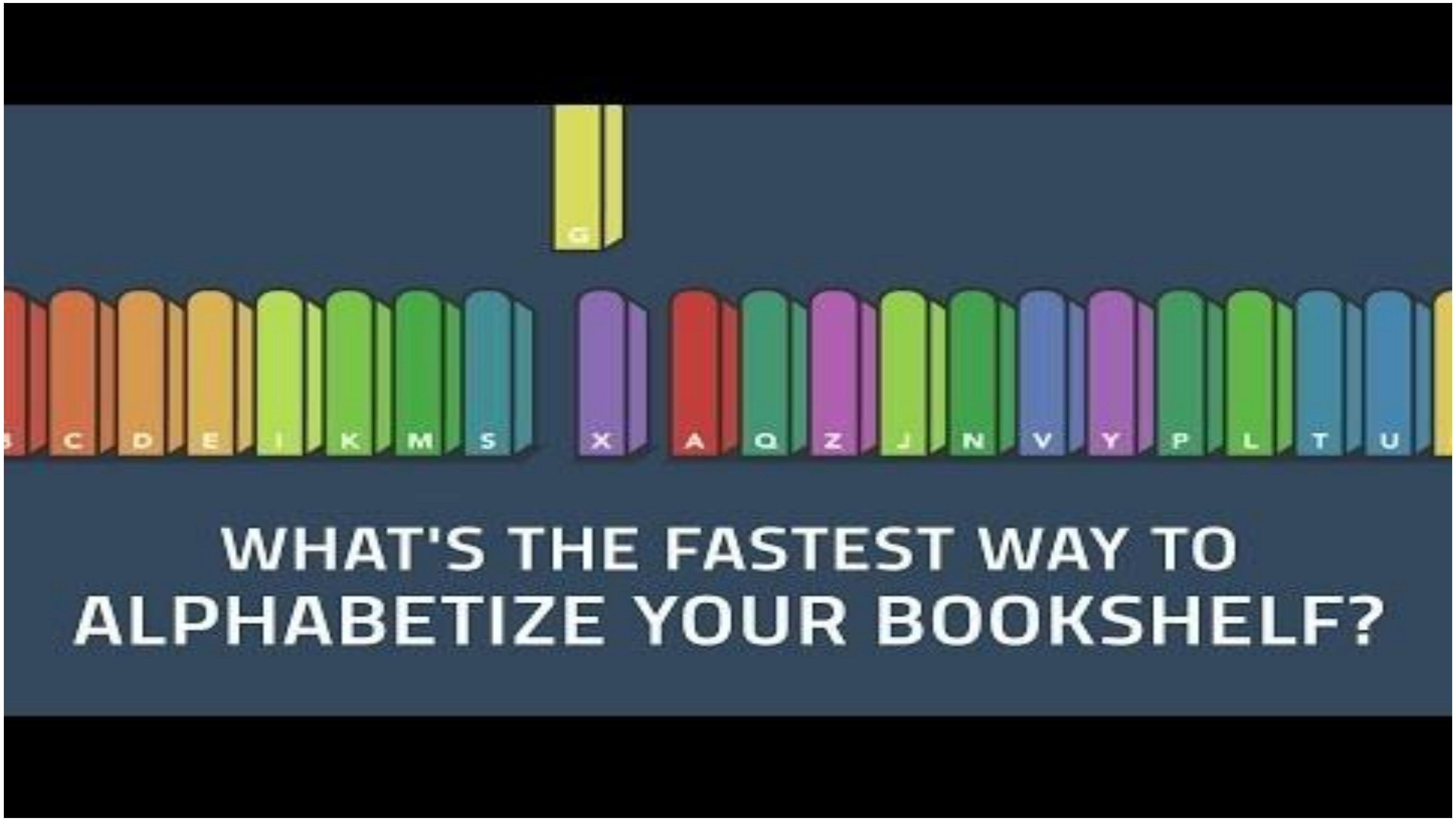
- Sorting algorithms
  - Bubble Sort
  - Selection Sort
  - Quicksort
- Searching algorithms
  - Linear search
  - Binary search

Any Questions from Last Time?

Today's Objectives

- To learn about asymptotic analysis
  - What it is
  - Why it's important
  - How to calculate it
- To discuss "run time" of algorithms
  - Why one algorithm is "better" than another

Alphabetizing a Bookshelf



Video from https://www.youtube.com/watch?v=WaNLJf8xzC4

Run Time

Run Time

- An algorithm's ***run time*** is the amount of "time" it takes for that algorithm to run
  - "Time" normally means number of operations or something similar, and not seconds or minutes
- Run time is shown as an expression, which updates based on how large the problem is
- Run time shows how an algorithm *scales*, or changes with the size of the problem

Example: Fibonacci Recursion

- Ideally, we want an algorithm that runs in a reasonable amount of time, no matter how large the problem
- Remember the recursive Fibonacci program?
  - It runs within one second for smaller numbers
  - But the larger the number we ask for, the longer and longer it takes

Fibonacci Recursion

```
python fibEx.py (with num < 30):
    < 1 second

python fibEx.py (with num = 30):
    2 seconds

python fibEx.py (with num = 35):
    8 seconds

python fibEx.py (with num = 40):
    76 seconds
```

Fibonacci Recursion

```
python fibEx.py (with num = 50):
    Guess!

    9,493 seconds

    2 hours, 38 minutes, 13 seconds!!!
```

Run Time for Linear Search

- Say we have a list that <u>does not</u> contain what we're looking for.
- How many things in the list does linear search have to look at for it to figure out the item's not there for a list of 8 things?
- 16 things?
- 32 things?

Run Time for Binary Search

- Say we have a list that <u>does not</u> contain what we're looking for.
- What about for binary search?
  - How many things does it have to look at to figure out the item's not there for a list of 8 things?
  - 16 things?
  - 32 things?
- Notice anything different?

Different Run Times

- These algorithms scale differently!
  - Linear search does an amount of work equal to the number of items in the list
  - Binary search does an amount of work equal to the **`log`**$_2$ of the numbers in the list!
- By the way, **`log`**$_2$**`(x)`** is basically asking "2 to what power equals x?" (normally shown as **`lg(x)`**)
  - This is the same as saying, "how many times must we divide x in half before we hit 1?"

Bubble Sort Run Time

- For a list of size **`N`**, how much work do we do for a single pass?
  - **`N`**
- How many passes will we have to do?
  - **`N`**
- What is the run time of Bubble Sort?
  - $\mathbf{N}^2$

Selection Sort Run Time

- What is the run time of finding the lowest number in a list?

- For a list of size **`N`**, what is the worst case number of elements you'd have to look through to find the min?
- **`N`**

Selection Sort Run Time

- For a list of size **N**, how many times would we have to find the min to sort the list?
- **N**

- What is the run time of this sorting algorithm?
- $\mathbf{N}^2$

Quicksort Run Time

- For a list of size **N**, how many steps does it take to move everything less than the last number to the left and everything greater than the last number to the right?

- **N**

Quicksort Run Time

- How many times will the algorithm divide the list in half?
- **`lg(N)`**

- What is the run time of Quicksort?
- **`N * lg(N)`**

Different Run Times

- As our list gets bigger and bigger, which of the **search** algorithms is faster?
  - Linear or binary search?

- How <u>much</u> faster is binary search?
  - A lot!
  - But exactly how much is "a lot"?

Asymptotic Analysis

What is “Big O” Notation?

- Big O notation is a concept in Computer Science
  - Used to describe the complexity (or performance) of an algorithm
- Big O describes the **worst-case** scenario
  - Big Omega (Ω) describes the best-case
  - Big Theta (Θ) is used when the best and worst case scenarios are the same

Asymptotic Analysis

- For a list of size **`N`**, linear search does **`N`** operations. So we say it is **`O(N)`** (pronounced "big Oh of n")

- For a list of size **`N`**, binary search does **`lg(N)`** operations, so we say it is **`O(lg(N))`**

- The function inside the **`O()`** parentheses indicates how fast the algorithm scales

Worst Case vs Best Case

- Why differentiate between the two?

- Think back to selection sort
  - What is the best case for run time?
  - What is the worst case for run time?
- They're the same!
  - Always have to find each minimum by looking through the entire list every time – **$\Theta(N^2)$**

Bubble Sort Run Times

- What about bubble sort?
  - What is the <u>best</u> case for run time?
  - What is the <u>worst</u> case for run time?

- Very different!
  - Best case, everything is already sorted – $\Omega(N)$
  - Worst case, it's completely backwards – $O(N^2)$

Quicksort Run Times

- What about quicksort?
  - Depends on what the "hinge" or "partition" is
- This determines how many times we split
  - But each split, we'll need to compare each item to the hinge in their respective part: **`O( N )`**

- Best case, hinge is exact center – **`Ω( N*lgN )`**
- Worst case, it's an "edge" item – **`O(`** N^2 **`)`**

Worst-case vs Best-case

- This is why, even though all three sorting algorithms have the same run times...
  - Quicksort often runs very, very quickly
  - Bubble Sort often runs much faster than Selection

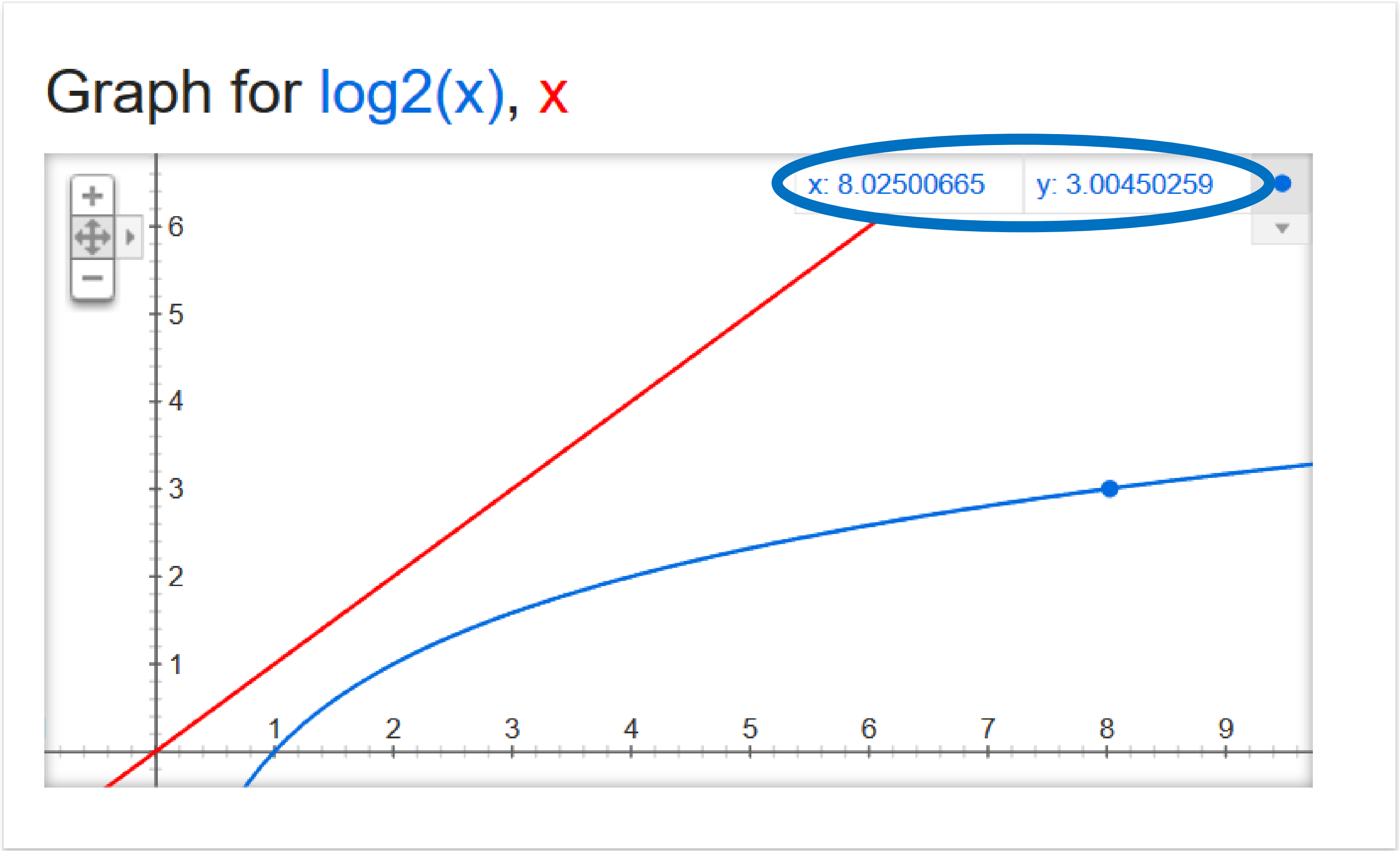
- How does this apply to linear search and binary search? What are the best and worst run times for these?

Search Run Times

- Linear search:
  - Best case: `Ω( 1 )`
  - Worst case: `O( N )`

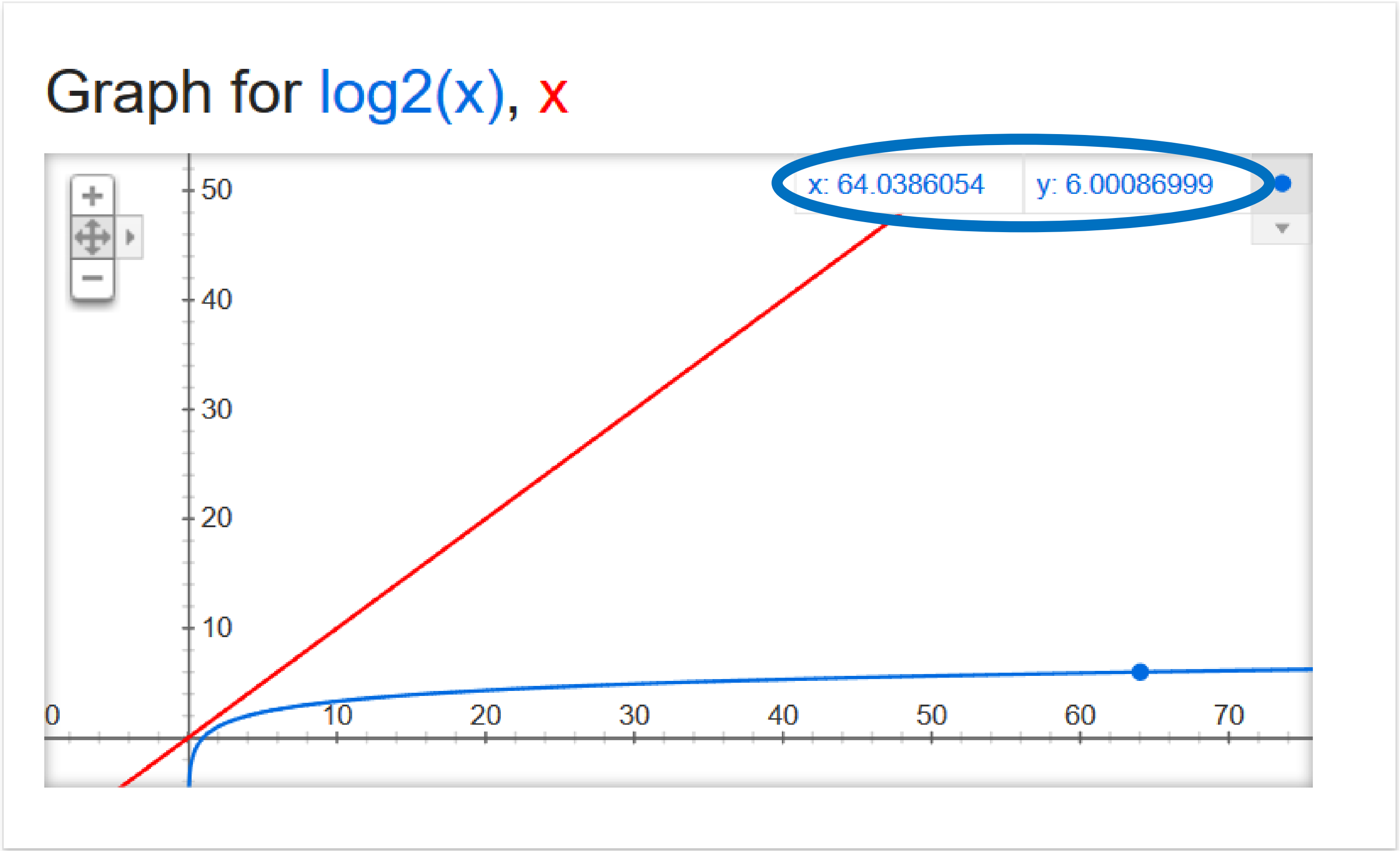
- Binary search:
  - Best case: `Ω( 1 )`
  - Worst case: `O( lg(N) )`

Why Care?

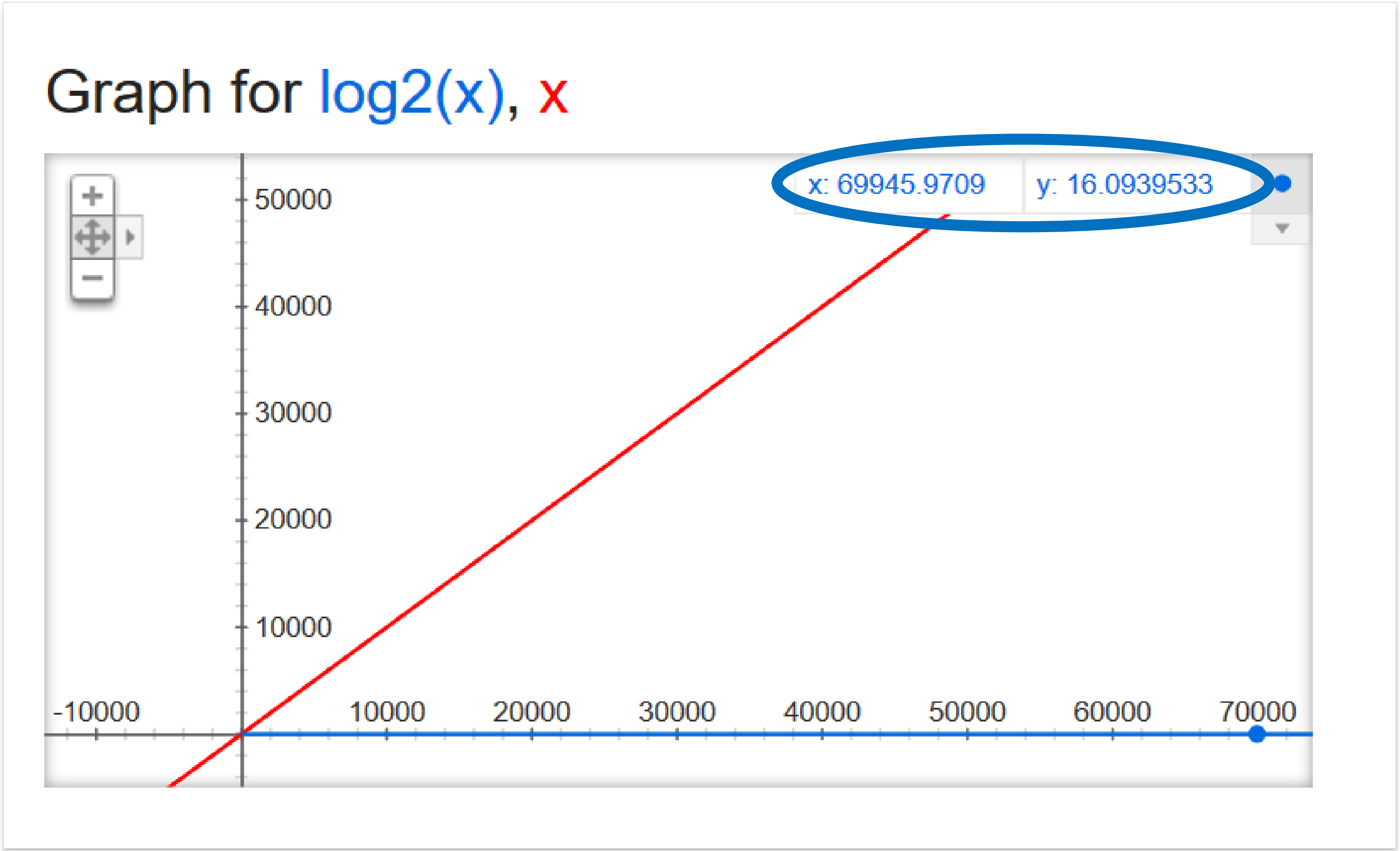

Graph for log2(x), x

x: 8.02500665 y: 3.00450259

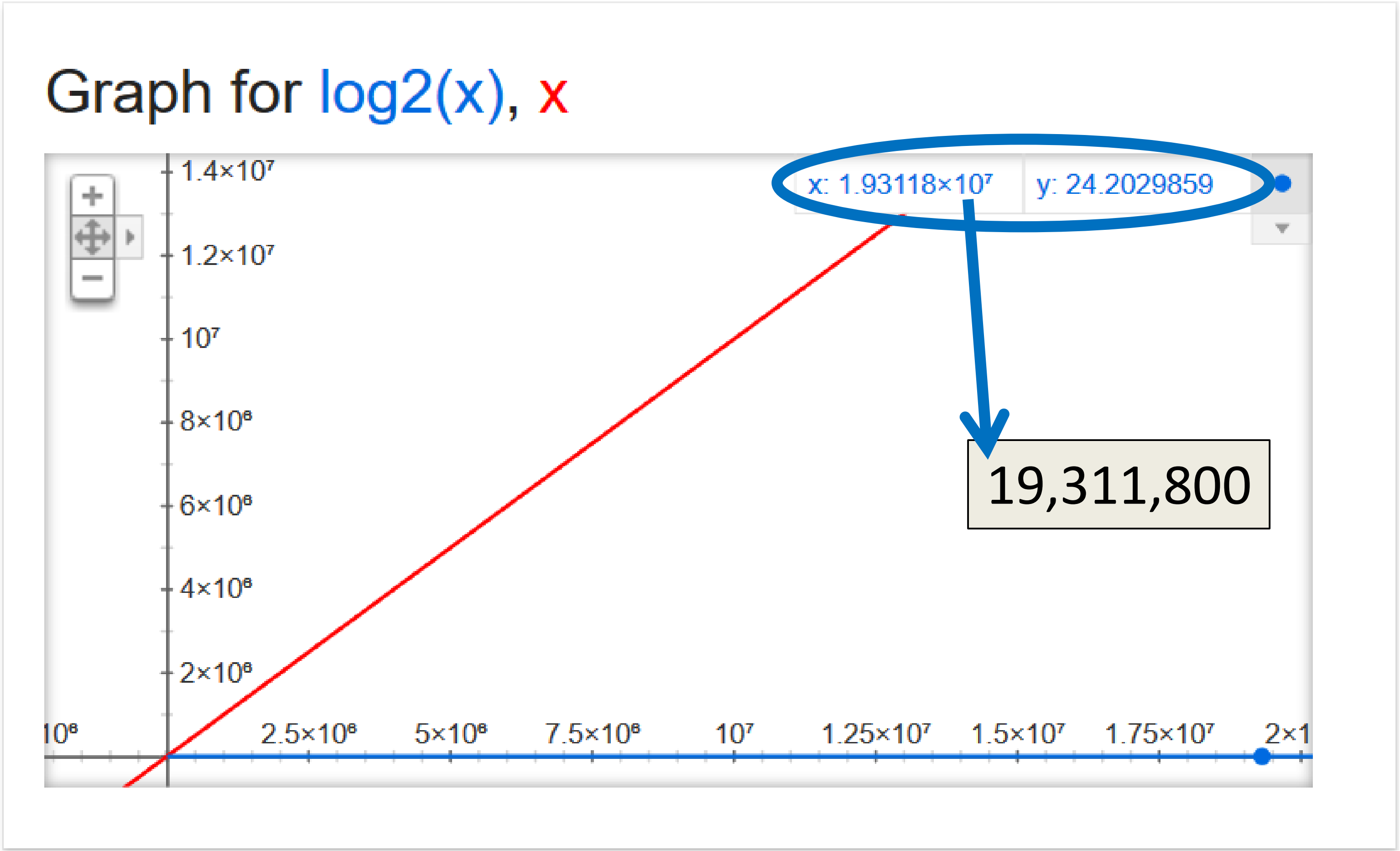
Why Care?


Graph for log2(x), x
x: 64.0386054
y: 6.00086999
50
40
30
20
10
0
10
20
30
40
50
60
70

Why Care?

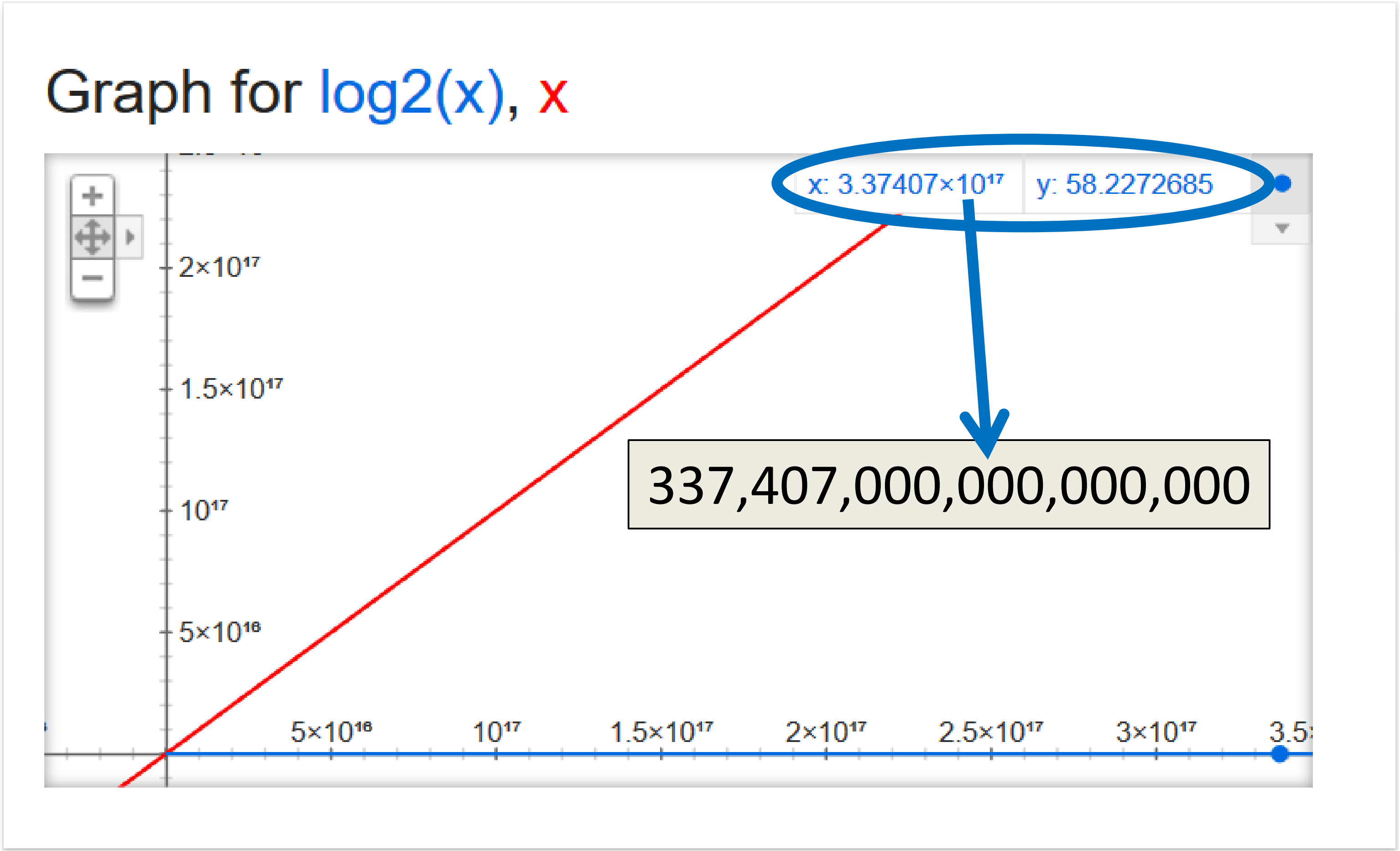

Graph for log2(x), x
x: 69945.9709
y: 16.0939533
50000
40000
30000
20000
10000
-10000
10000
20000
30000
40000
50000
60000
70000

Why Care?



19,311,800

Why Care?



337,407,000,000,000,000

Why Care?

- For large problems, there's a *huge* difference!
- If we can do 1,000,000 operations per second, and the list is 337.4 <u>quadrillion</u> items
  - Binary search takes 0.000058 seconds
  - Linear search takes 337,407,000,000 seconds
    5,623,450,000 minutes
    93,724,166 hours
    3,905,173 days
    10,699 years

Announcements

- Final is Friday, May 19th from 6 to 8 PM
- Project 3 out now
  - Design due on Friday, May 5th @ 8:59:59 PM
  - Project due on Friday, May 12th @ 8:59:59 PM
- Survey #3 also out – follow link in announcement
- Now we'll talk about SEEQs

SEEQs

The Student Evaluation of Educational Quality (SEEQ) is a standardized course evaluation instrument used to provide measures of an instructor’s teaching effectiveness. The results of this questionnaire will be used by promotion and tenure committees as part of the instructor’s evaluation.

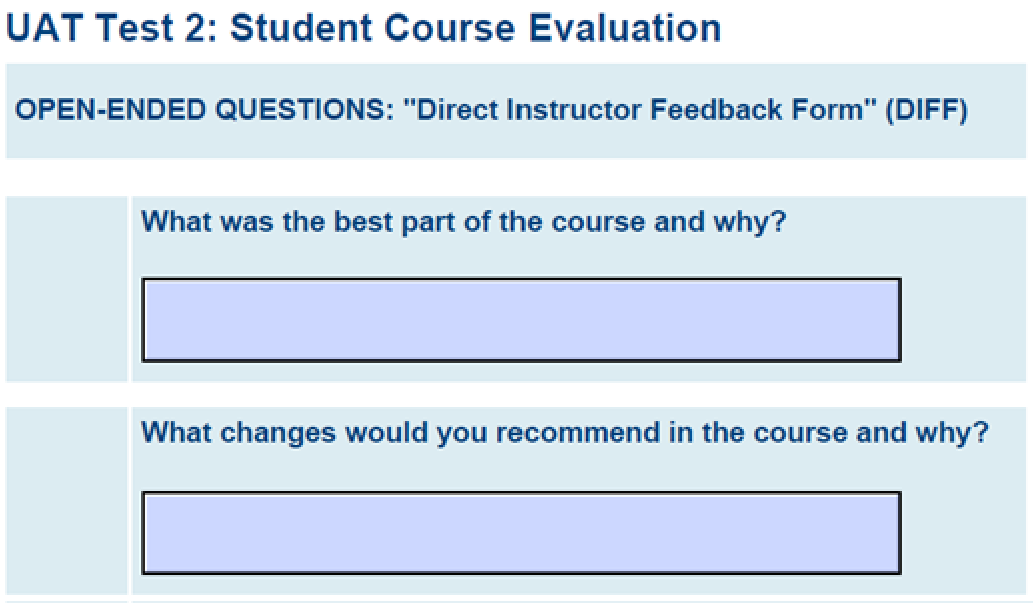
The Direct Instructor Feedback Forms (DIFFs) were designed to provide feedback to instructors and they are not intended for use by promotion and tenure committees.

The responses to the SEEQ and the DIFFs will be kept confidential and will not be distributed until final grades are in.

Completing SEEQs

- Please take the time now, if you haven't already, to complete the SEEQ online
- You can access it via the link in your email, or via Blackboard

This is the part →
I will get to see



UAT Test 2: Student Course Evaluation

OPEN-ENDED QUESTIONS: "Direct Instructor Feedback Form" (DIFF)

What was the best part of the course and why?

What changes would you recommend in the course and why?