CMSC201
Computer Science I for Majors

Lecture 22 – Algorithmic Analysis & Hexadecimal Numbers

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 hexadecimal numbers
  - To be able to convert between bin, dec, and hex
- 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

Hexadecimal Numbers

Decimal Representation

- Decimal uses 10 digits
  - Decimal, *deci* = 10
  - The digits used are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9

ten millions	millions	hundred thousands	ten thousands	thousands	hundreds	tens	ones
9	8	7	5	4	2	1	0
10^7	10^6	10^5	10^4	10^3	10^2	10^1	10^0

Binary Representation

- Binary uses 2 digits
  - Binary, *bi* = 2
  - The digits used are 0 and 1

one hundred and twenty-eights	sixty-fours	thirty-twos	sixteens	eights	fours	twos	ones
1	0	0	1	1	1	1	0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Hexadecimal Representation

- Hexadecimal (or just "hex") uses 16 digits
  - Hexadecimal; *hex* = 6 plus *deci* = 10 → 16
  - The digits used are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
    - And letters A (10), B (11), C (12), D (13), E (14), and F (15)

two hundred sixty-eight million four hundred thirty-five thousand four hundred and fifty-sixes	sixteen million seven hundred seventy-seven thousand two hundred and sixteens	one million forty-eight thousand five hundred and seventy-sixes	sixty-five thousand five hundred and thirty-sixes	four thousand and ninety-sixes	two hundred and fifty-sixes	sixteens	ones
F	D	A	8	6	3	1	0
16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0

Hexadecimal Representation

- Hexadecimal (or just "hex") uses 16 digits
  - Hexadecimal, *hex* = 6 plus *deci* = 10 → 16
  - The digits used are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9
    - And letters A (10), B (11), C (12), D (13), E (14), and F (15)

268435456s	16777216s	1048576s	65536s	4096s	256s	sixteens	ones
F	D	A	8	6	3	1	0
16^7	16^6	16^5	16^4	16^3	16^2	16^1	16^0

Hex to Binary Conversion

- A hexadecimal digit can be easily represented as four digits of binary (with leading zeros)

Hex	Binary	Hex	Binary	Hex	Binary	Hex	Binary
0	0000	4	0100	8	1000	C	1100
1	0001	5	0101	9	1001	D	1101
2	0010	6	0110	A	1010	E	1110
3	0011	7	0111	B	1011	F	1111

- This makes conversion very simple
  - **7A0F** becomes **0111 1010 0000 1111**
  - **1100 0010 0110 1001** becomes **C269**

Hex to Decimal Conversion

- Possible to convert between decimal and hex
  - But it requires calculating out multiples of 16
- Simpler to make a "side trip" to binary as an in-between step when converting
  - 240 becomes **`1111 0000`** becomes **`F0`**
    - **`F0`** is equal to $(15 * 16^1) + (0 * 16^0) = 240 + 0 = 240$
  - **`7D`** becomes **`0111 1101`** becomes 125
    - **`7D`** is equal to $(7 * 16^1) + (13 * 16^0) = 112 + 13 = 125$

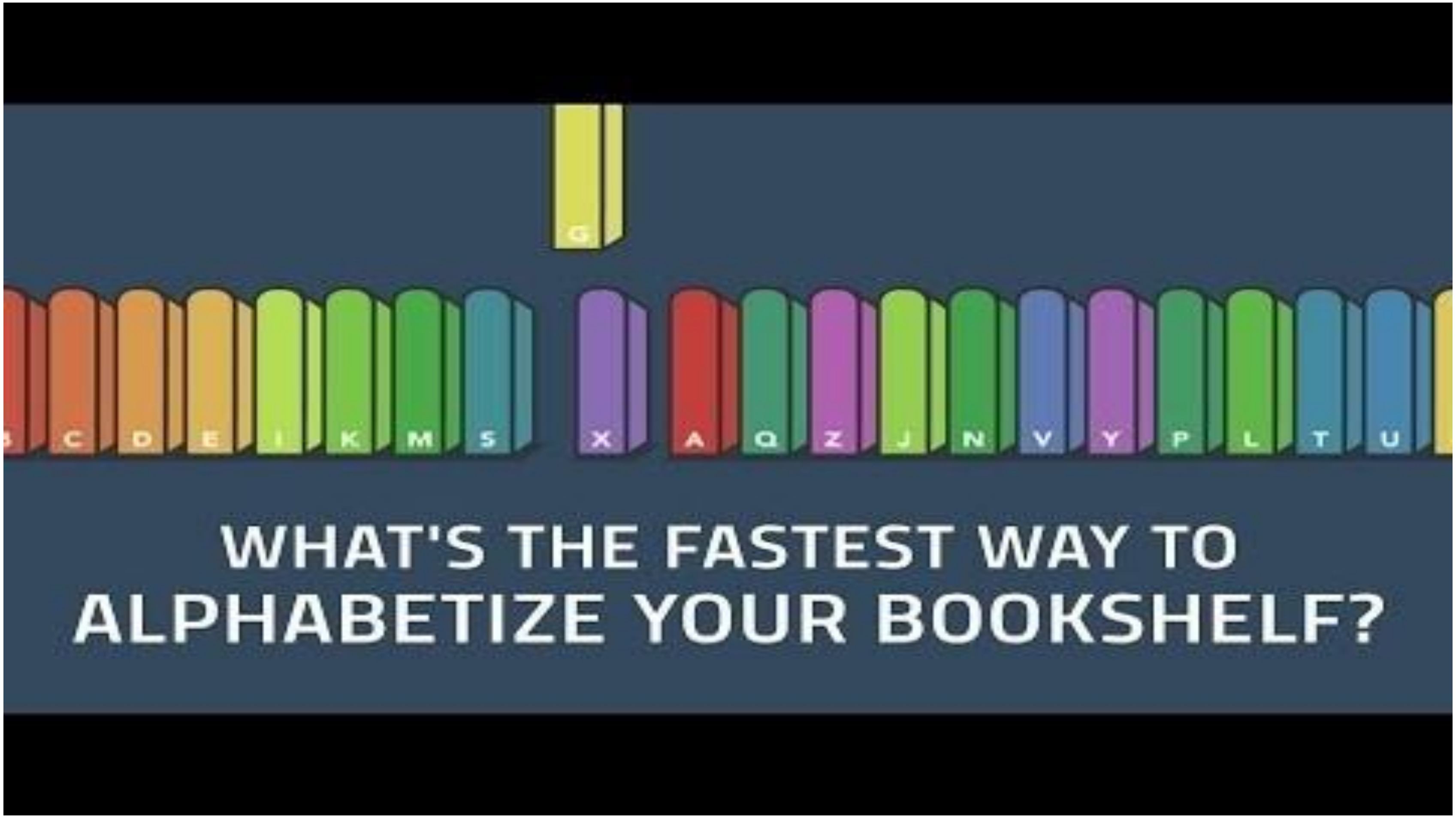
Number System Notation

- Because number systems share a subset of the same digits, it may be confusing which is which
  - For example, what is the value of 10?
    - In decimal it's 10, in binary it's 2, and in hex it's 16

- To prevent this, numbers may often be prefixed with **`0b`**, **`0d`**, or **`0x`** (binary, decimal, hex):
  - **`0b1100`** is binary, and has a value of 12
  - **`0x15`** is hexadecimal, and has a value of 21

Run Time

Alphabetizing a Bookshelf



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

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 positions
  - But the larger the position we ask for, the longer and longer it takes

Fibonacci Recursion

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

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

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

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


Fibonacci Recursion

`python fibEx.py (with position = 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`, how many elements do you 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 "pivot" to the left and everything greater than the "pivot" 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

A Simple Example

- Say we write an algorithm that takes in an list of numbers and returns the maximum
  - What is the absolute fastest it can run?
    - Linear time – **Ω(N)**
  - What is the absolute slowest it can run?
    - Linear time – **O(N)**
  - Are these two values the same?
    - YES – so we can also say it's **Θ(N)**

Simplification

- We are only interested in the growth rate as an "order of magnitude"
  - As the problem grows really, really, really large

- We are <u>not</u> concerned with the fine details
  - Constant multipliers are dropped
    - So `O(3 * N`^{2}`)` becomes simply `O(N`^{2}`)`
  - Lower order terms are dropped
    - So `O(N`^{3}` + 4N`^{2}`)` becomes simply `O(N`^{3}`)`

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 best case for run time?
  - What is the worst 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 "pivot" 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, pivot 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 same worst case run times...
  - Quicksort often runs very, very quickly
  - Bubble Sort often runs much faster than Selection

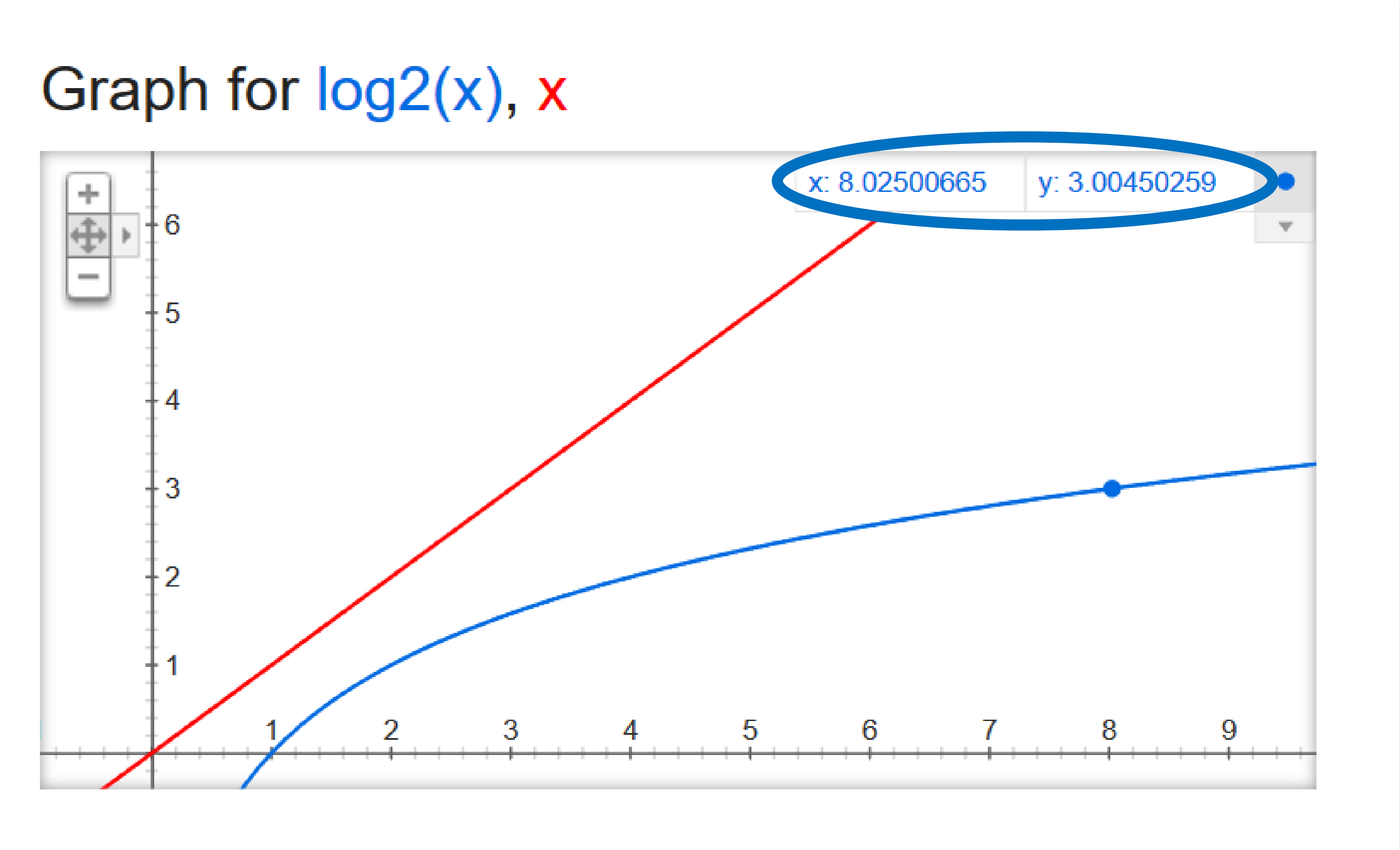
- How does this apply to linear search and binary search? What are the best and worst run times for these?

Search Run Times

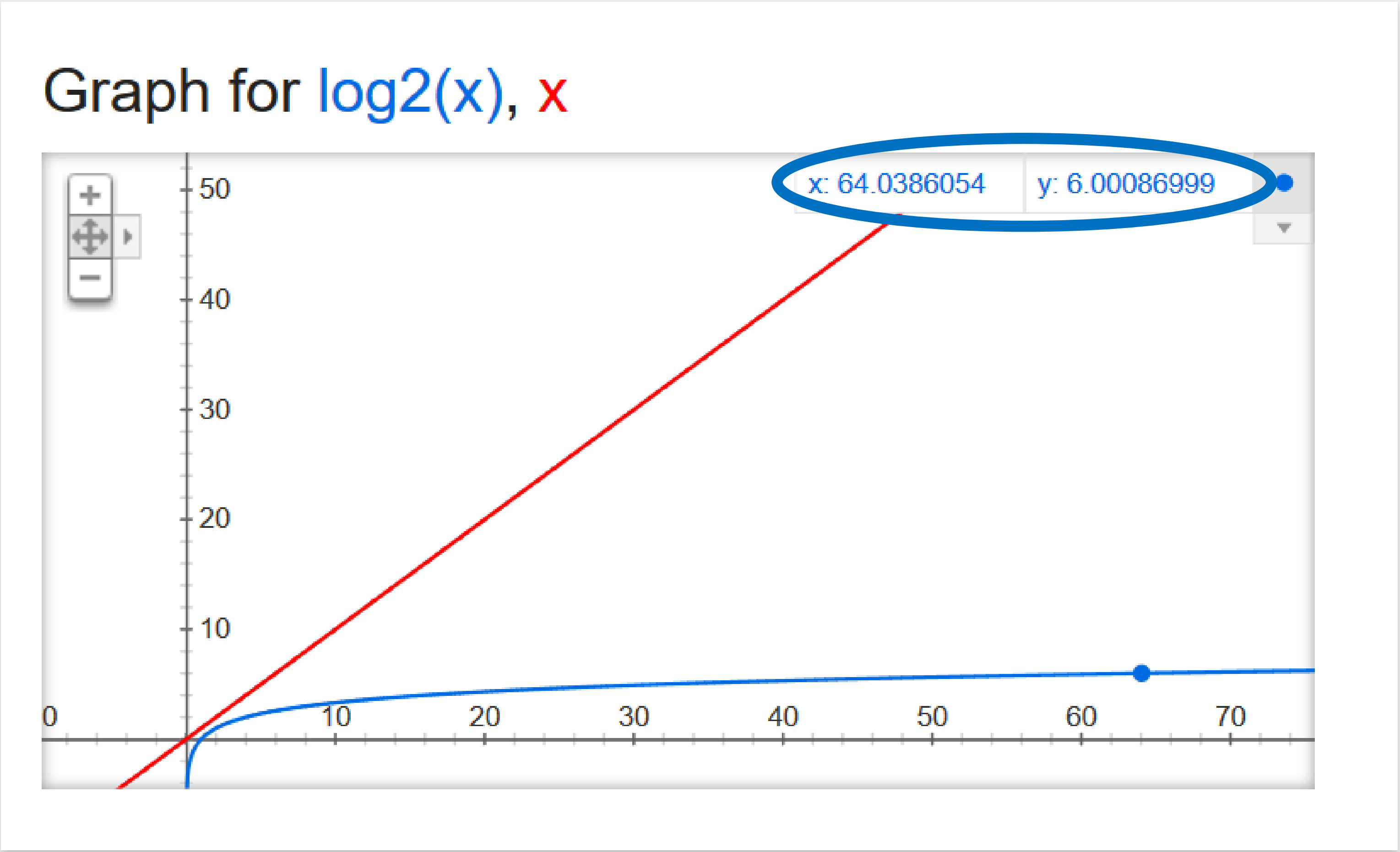
- Linear search:
  - Best case: **Ω(1)**
  - Worst case: **O(N)**

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

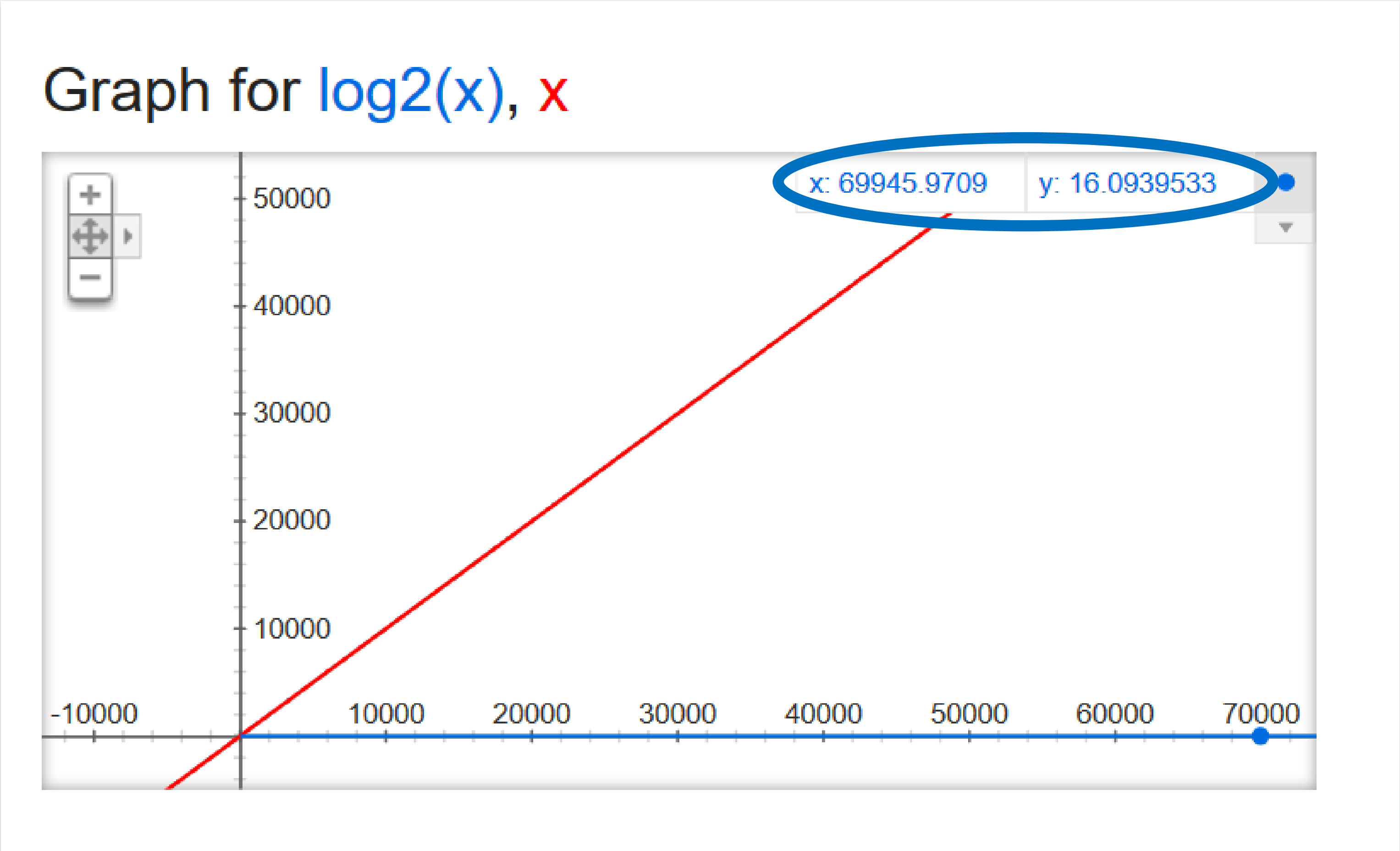
Why Care?



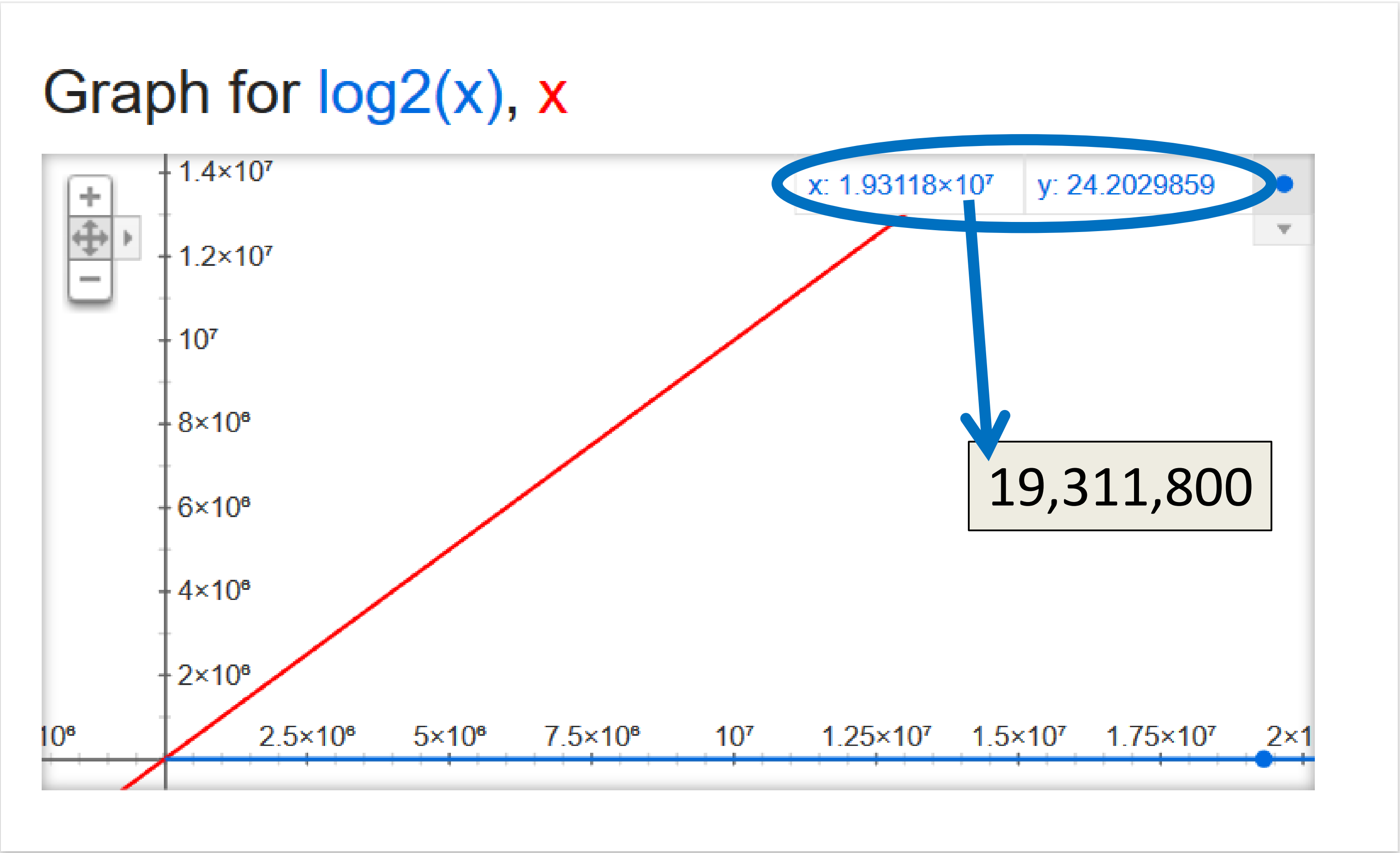
Why Care?


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

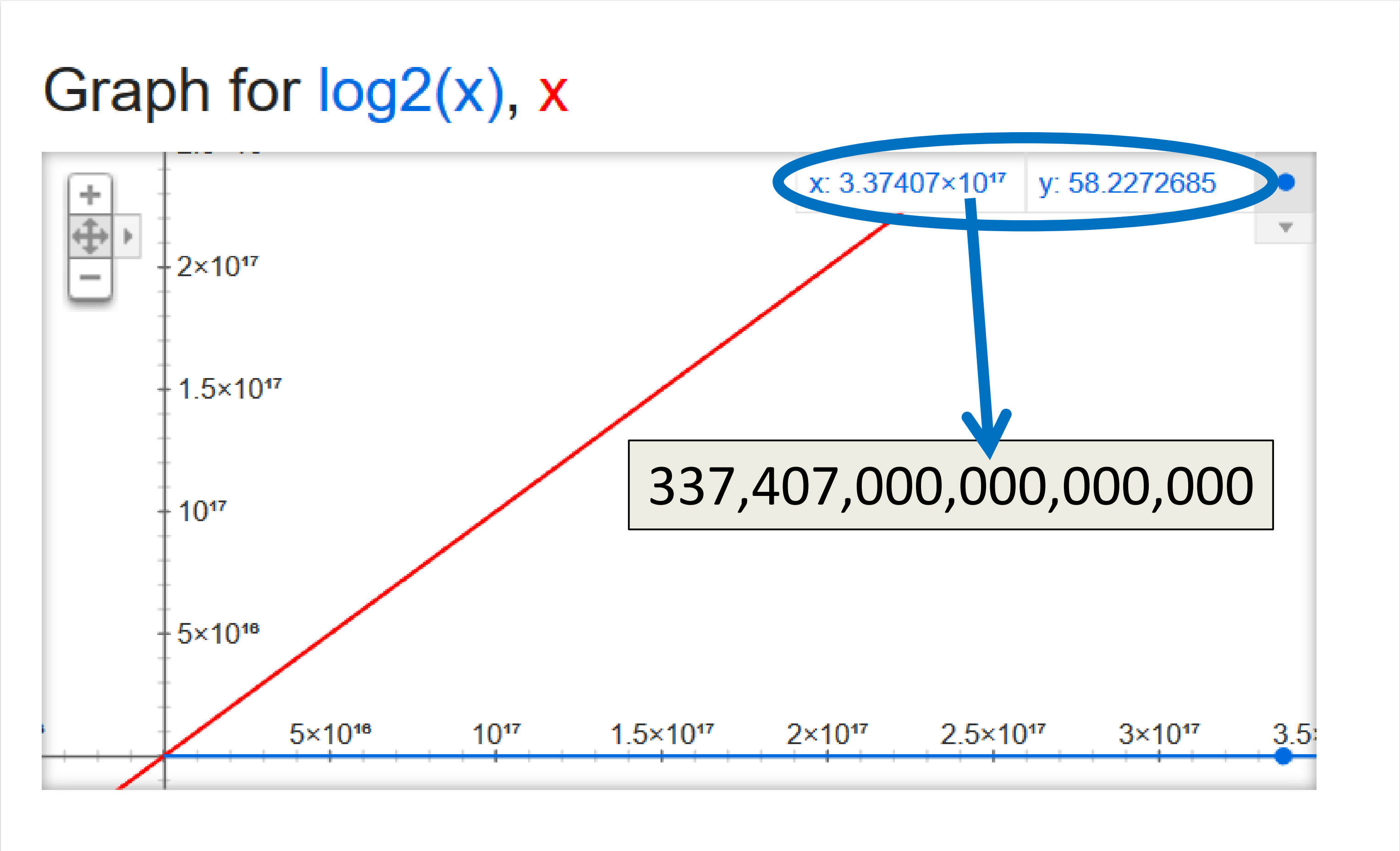
Why Care?


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?


Graph for log2(x), x
x: 1.93118×10⁷
y: 24.2029859
19,311,800
1.4×10⁷
1.2×10⁷
10⁷
8×10⁶
6×10⁶
4×10⁶
2×10⁶
2.5×10⁶
5×10⁶
7.5×10⁶
1.25×10⁷
1.5×10⁷
1.75×10⁷

Why Care?


Graph for log2(x), x
x: 3.37407×10¹⁷
y: 58.2272685
337,407,000,000,000,000
2×10¹⁷
1.5×10¹⁷
10¹⁷
5×10¹⁶
5×10¹⁶
10¹⁷
1.5×10¹⁷
2×10¹⁷
2.5×10¹⁷
3×10¹⁷

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 quadrillion 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

Daily CS History

- Hedy Lamarr
  - Film star in 1930s - 1950s
  - Patented a frequency-hopping system that would make radio-guided torpedoes hard to detect or jam during World War II
  - Technologies like Bluetooth and Wi-Fi use similar methods

Final Exam Locations

- Find your room ahead of time!

- **Engineering 027** - Sections 2, 3, 4, 5, 6, 25
  Section 22

- **Meyerhoff 030** - Sections 8, 9, 10, 11, 12
  Sections 14, 15, 16, 17

Announcements

- Project 3 is due on Friday, May 10th
- Survey #3 is out now
- Course evaluations also out, please complete
- Final exam is Friday, May 17th from 6 to 8 PM

Image Sources

- Alphabetizing a Bookshelf video screenshot:
  - https://www.youtube.com/watch?v=WaNLJf8xzC4
- Graphs of x and $\log_2(x)$ courtesy of Google equation grapher
- Hedy Lamarr:
  - https://commons.wikimedia.org/wiki/File:Hedy_lamarr_-_1940.jpg

