

CMSC201

Computer Science I for Majors

Lecture 25 – Algorithmic Analysis

Last Class We Covered

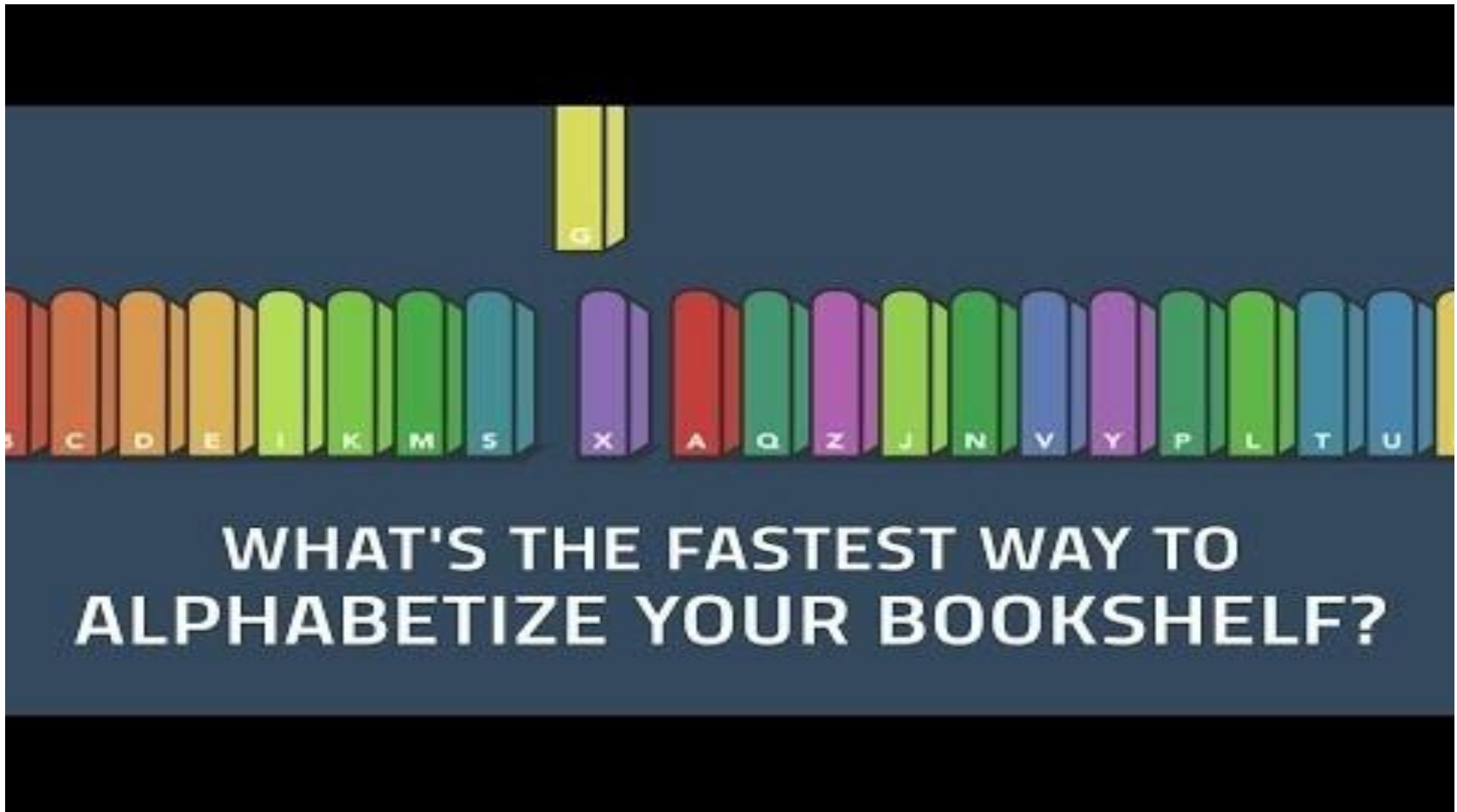
- Sorting algorithms
 - Bubble Sort
 - Selection Sort
 - Quicksort
- Searching algorithms
 - Linear search
 - Binary search

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 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 does not 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 does not 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, $\log_2(\mathbf{x})$ is basically asking “2 to what power equals \mathbf{x} ?” (normally shown as $\lg(\mathbf{x})$)
 - This is the same as saying, “how many times must we divide \mathbf{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?
 - 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?
- **N²**

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 much 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 – $\Omega(N)$
 - 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 $\Theta(N)$

Simplification

- We are only interested in the growth rate as an “order of magnitude”
 - As the problem grows really, really, really large
- We are not 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 \mathbf{N} , linear search does \mathbf{N} operations. So we say it is $\mathbf{O(N)}$ (pronounced “big Oh of n”)
- For a list of size \mathbf{N} , binary search does $\mathbf{\lg(N)}$ operations, so we say it is $\mathbf{O(\lg(N))}$
- The function inside the $\mathbf{O()}$ 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 – $\Omega(N \lg N)$
- 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?

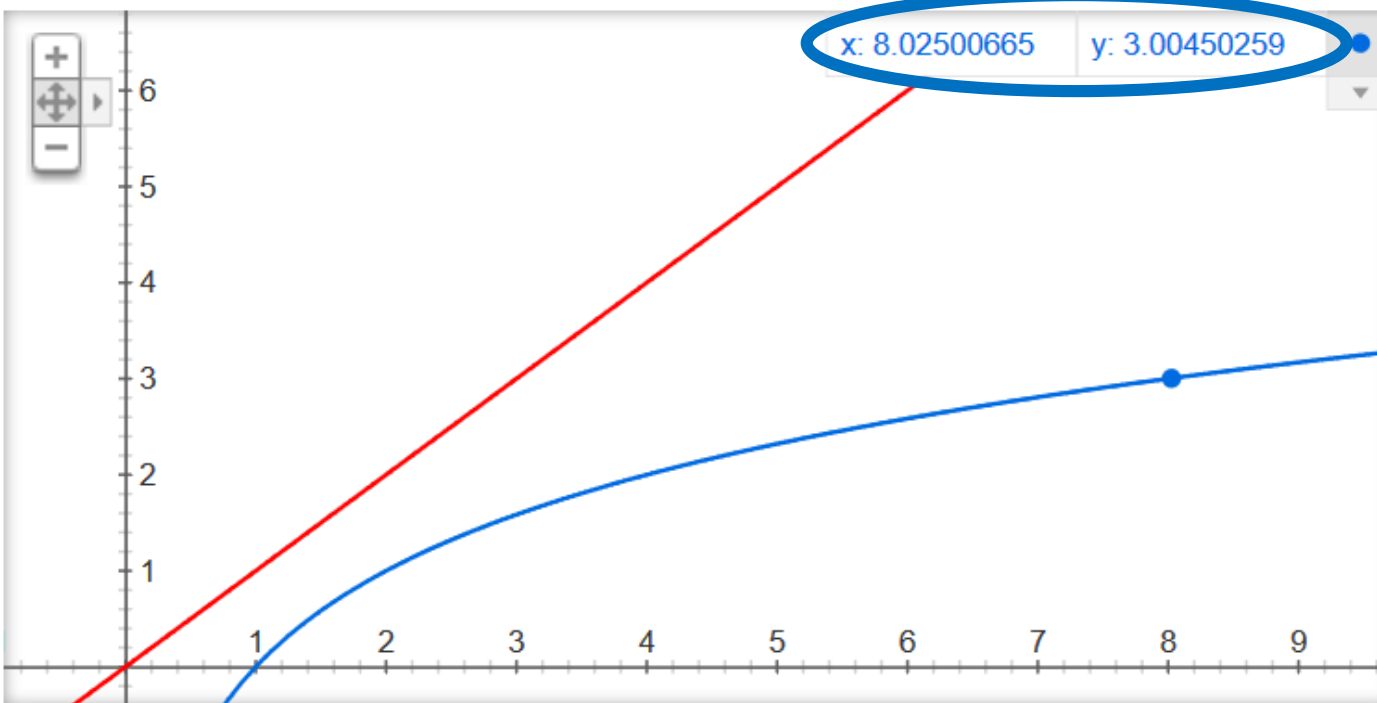
Search Run Times

- Linear search:
 - Best case: $\Omega(1)$
 - Worst case: $O(N)$

- Binary search:
 - Best case: $\Omega(1)$
 - Worst case: $O(\lg(N))$

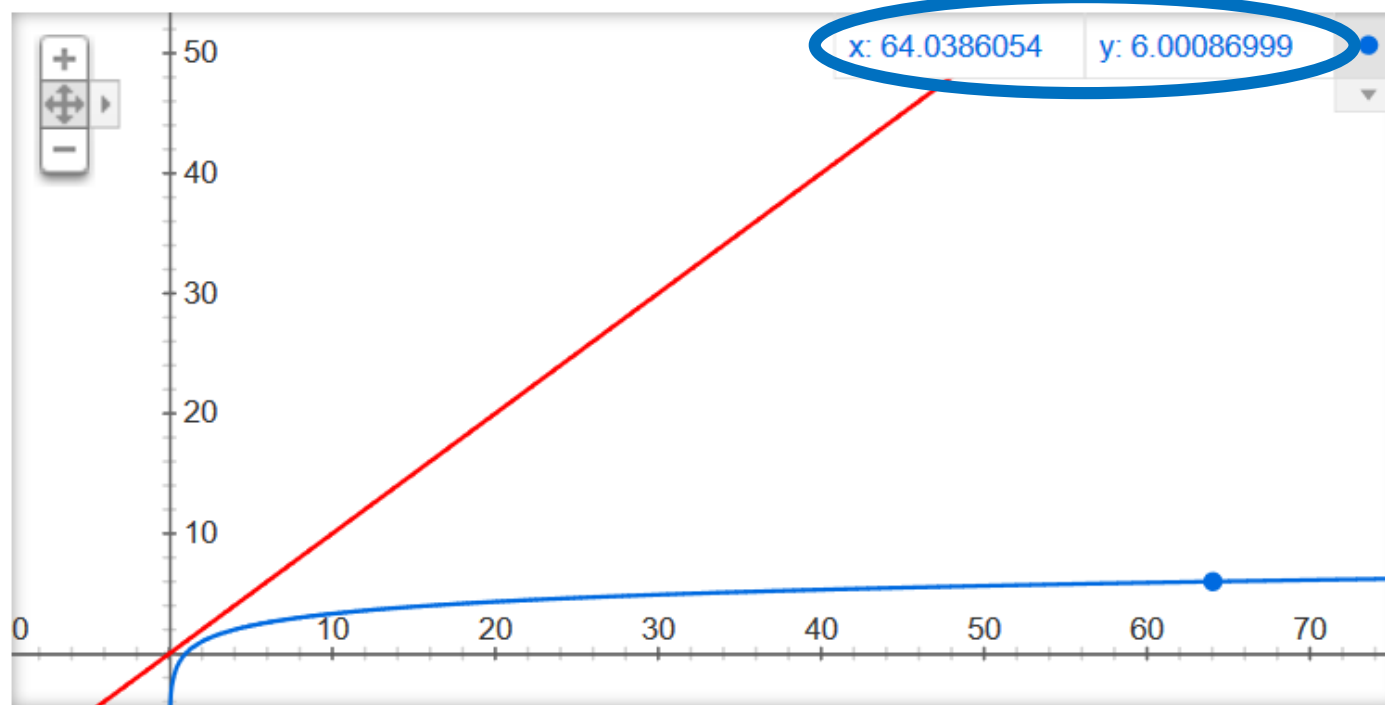
Why Care?

Graph for $\log_2(x)$, x



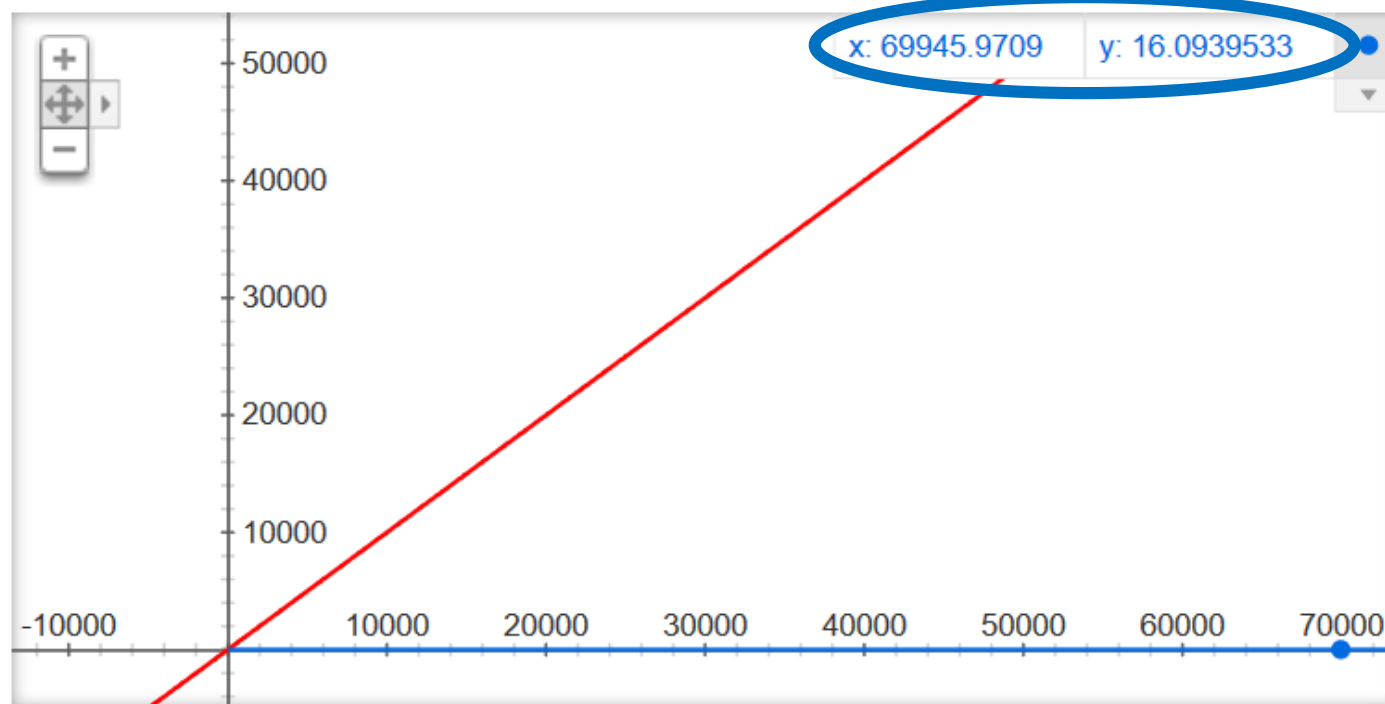
Why Care?

Graph for $\log_2(x)$, x



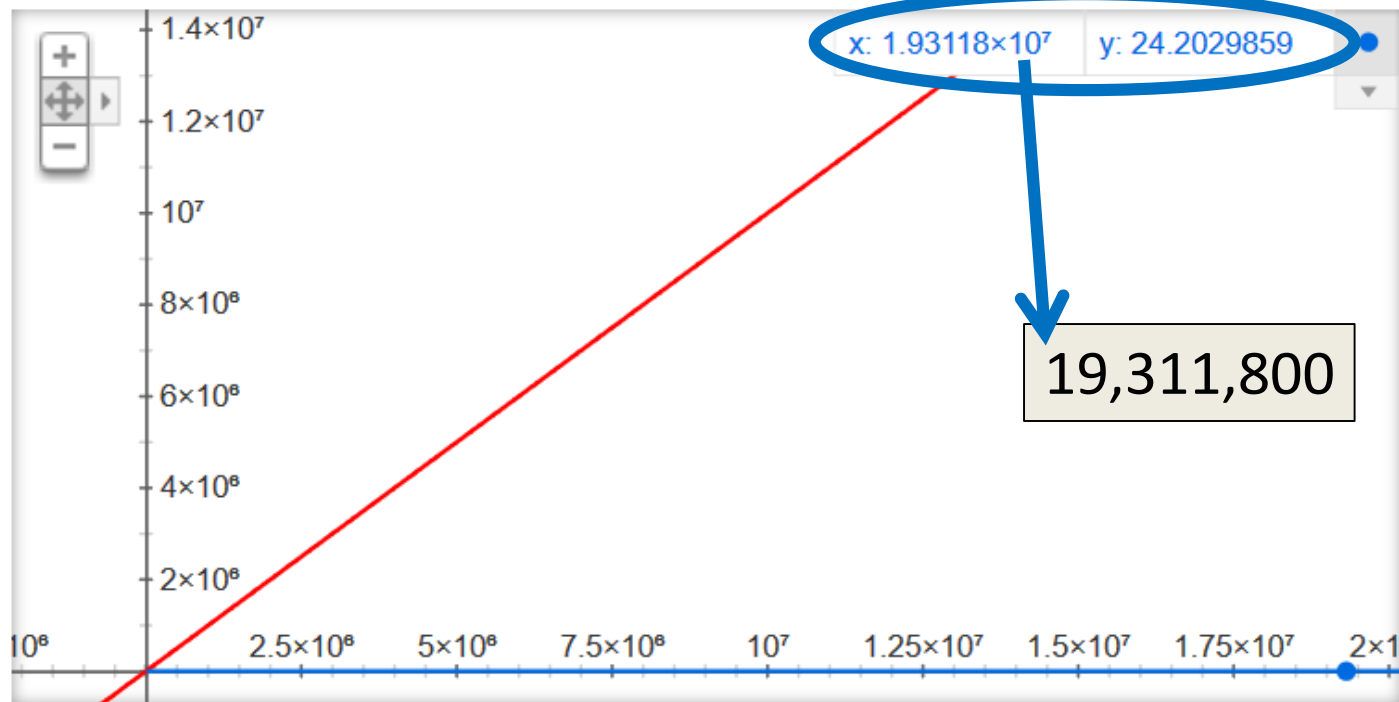
Why Care?

Graph for $\log_2(x)$, x



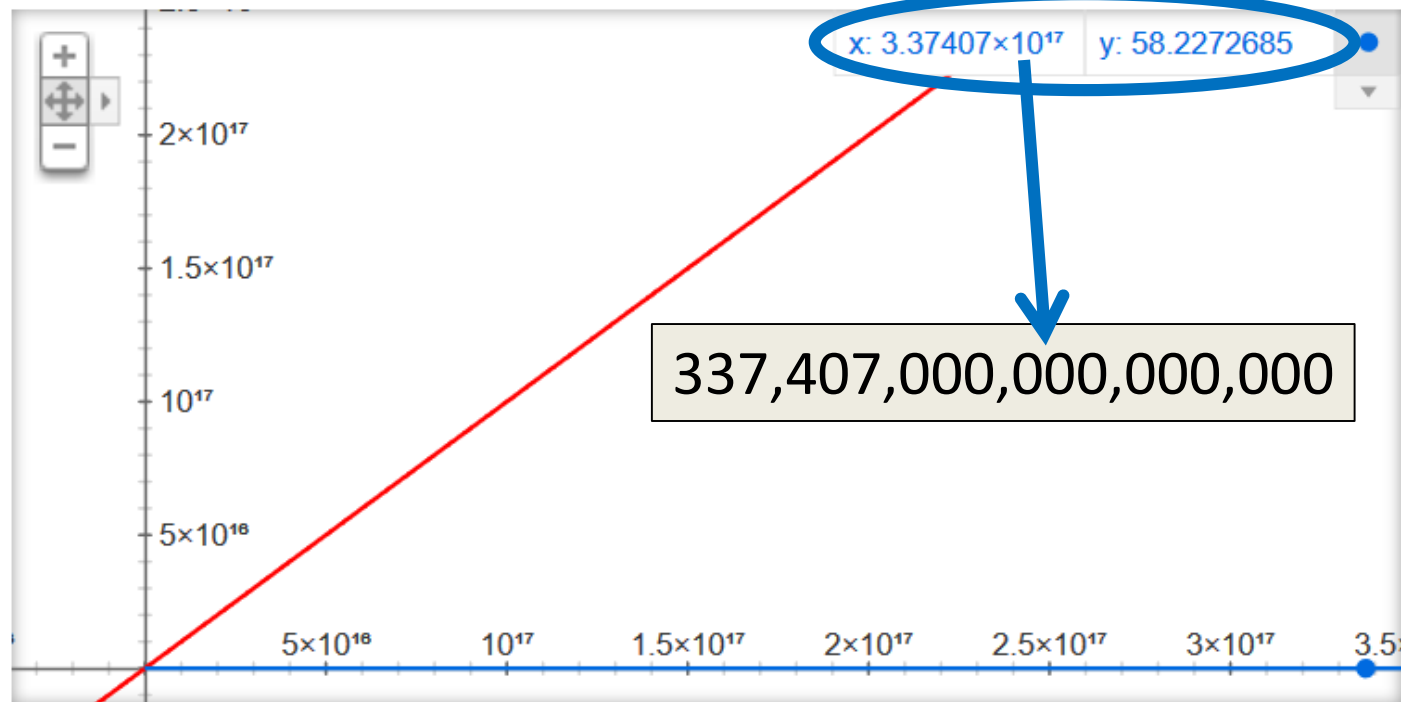
Why Care?

Graph for $\log_2(x)$, x



Why Care?

Graph for $\log_2(x)$, x



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



Announcements

- Project 3 is due on Friday, May 11th
- Assignments out now
 - Survey #3
 - Final exam metacognition quiz out on BB same day
- Course evaluations also out, please complete
- Final exam is Friday, May 18th from 6 to 8 PM

Final Exam Locations

- Find your room ahead of time!
- **Engineering 027** - Sections 8, 9, 10, 11, 12
Section 6
- **Meyerhoff 030** - Sections 2, 3, 4, 5
Sections 14, 15, 16, 17, 30

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