Algorithmic Complexity I

CSE 120 Spring 2017

Instructor: Justin Hsia

Teaching Assistants: Anupam Gupta, Braydon Hall, Eugene Oh, Savanna Yee

Administrivia

- Assignments:
  - Binary Practice (4/21)
  - Creativity Assignment (4/24)

- Midterm in class on Wednesday, 4/26
  - 1 sheet of notes (2-sided, letter, handwritten)
  - Fill-in-the-blank(s), short answer questions, maybe simple drawing
    - Questions will cover lectures, assignments, and readings
  - Midterm Review sheet will be released tonight (4/19), will be covered in lab next week (4/25)

Outline

- **Algorithm Analysis: The Basics**
- Comparing Algorithms
- Orders of Growth

Algorithm Correctness

- An algorithm is considered correct if for every input, it reports the correct output and doesn't run forever or cause an error

- Incorrect algorithms may run forever, crash, or not return the correct answer
  - But they could still be useful!
  - *e.g.* an approximation algorithm

- Showing correctness
  - Mathematical proofs for algorithms
  - Empirical verification of implementations

Algorithm Analysis

- One commonly used criterion for analyzing algorithms is computation time
  - How long does the algorithm take to run and finish its task?
  - Can be used to compare different algorithms for the same computational problem

- How to measure this time?
  - Counting in my head
  - Stopwatch
  - Within your program

Aside: Computation Time

- Computers take time to complete their tasks
  - Under the hood, it's sort of like a bunch of buckets of water filling up – you have to wait for water to reach the top of a bucket for a single computation to complete
  - Buckets take about a billionth of a second to fill (~ 1 nanosecond)
    - There are billions of them on a single chip!

- A CPU can generally only execute one instruction at a time

Timing in Processing

- The function `millis()` returns the number of milliseconds since starting your program (as an `int`)
  - To start timing, call and store the value in a variable
  - Call again after your computation and subtract the values

```
void draw() {
  int time = millis();
  someComputation();
  println("Took " + (millis()-time) + " milliseconds to compute.");
  noLoop();
}
```

Outline

- Algorithm Analysis: The Basics
- **Comparing Algorithms**
- Orders of Growth

Algorithm: Searching A Sorted Array

- Input: Numbers in a sorted array, desired number
- Output: If desired number is in the array (`true`/`false`)

- Algorithm 1:
  - Check each index starting from 0 for desired number
    - If equal, then report `true`
    - If not equal, then move to next index
    - If at end of array, then report `false`
  - Called Linear Search (also works for unsorted array)

Algorithm: Searching A Sorted Array

- Input: Numbers in a sorted array, desired number
- Output: If desired number is in the array (`true`/`false`)

- Algorithm 2:
  - Check "middle" index for desired number
    - If equal, then report `true`
    - If less than desired number, check *halfway* forwards next
    - If greater than desired number, check halfway backwards next
  - If no halfway point left, then report `false`
  - Called Binary Search
    - http://www.cs.armstrong.edu/liang/animation/web/BinarySearch.html

Peer Instruction Question

- On average, which algorithm would take less time to complete a search?
  - Vote at http://PollEv.com/justinh

A. **Algorithm 1 (Linear Search)**

B. **Algorithm 2 (Binary Search)**

C. **They'd take about the same amount of time**

Measuring Linear Search

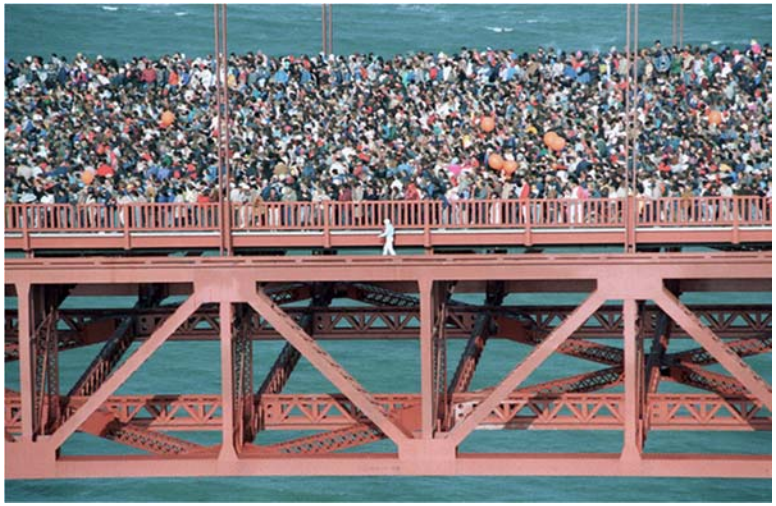
- Let's time Linear Search:

```
void draw() {
  int n = 3;
  println("Is " + n + " in intArr?");
  int time = millis();
  println(linearSearch(n));
  println("Took " + (millis()-time) + " milliseconds to compute.");
  noLoop();
}
```

- One issue: our algorithm seems to be too fast to measure!
  - How can we fix this?

Best Case vs. Worst Case vs. Average Case

- We were measuring close to the best case!
  - Didn’t matter how long our array was
- Could measure average case instead
  - Run many times on random numbers and average results
- Instead, we’ll do worst case analysis. Why?
  - Nice to know the most time we’d ever spend
  - Worst case happens often
  - Average case is often similar to worst case



Example of Worst Case in Action

- Many web servers out there run something called "The Apache HTTP Server" (or just Apache for short)
  - When a user enters a particular URL, Apache delivers the correct files from the server to the person on the internet
  - An old version of Apache had a bug where if you entered a URL with tons of consecutive slashes, it could take *hours* to complete the request
    - *e.g.* http://someurl.com////////////////////

- Bottom line: an algorithm is often judged by its worst case behavior

What is the Worst Case?

- Discuss with your neighbor (no voting):
  - Assume `intArr.length` is `1000000` and `intArr[i] = i;`
  - What is a worst case argument for `num` for Linear Search?
  - What is a worst case argument for `num` for Binary Search?

A. 1

B. 500000

C. 1000000

D. 1000001

E. Something else

```
boolean linearSearch(int num) {
  for(int i = 0; i < intArr.length; i = i + 1) {
    if(intArr[i] == num) {
      return true;
    }
  }
  return false;
}
```

Timing Experiments

- Let's try running Linear Search on a worst case argument value
  - Results:
- Now let's run Binary Search on a worst case argument value
  - Results:

Runtime Intuition

- Does it seem reasonable that the runtimes were inconsistent?

- Some reasons:
  - Your computer isn't just running Processing – there's a lot of other stuff running (*e.g.* operating system, web browser)
  - The computer hardware does lots of fancy stuff to avoid slowdown due to physical limitations
    - These may not work as well each execution based on other stuff going on in your computer at the time

Empirical Analysis Conclusion

- We’ve shown that Binary Search is seemingly much faster than Linear Search
  - Similar to having two sprinters race each other
- Limitations:
  - Different computers may have different runtimes
  - Same computer may have different runtime on same input
  - Need to implement the algorithm in order to run it
- Goal: come up with a “universal algorithmic classifier”
  - Analogous to coming up with a metric to compare all athletes (or fighters)

Outline

- Algorithm Analysis: The Basics
- Comparing Algorithms
- **Orders of Growth**

Characterizing Algorithms

- The method computer scientists use is roughly:

1) Measure the algorithm's runtime on many different input sizes N (*e.g.* arrays of length 100, 200, 400, 800, …)
   - To avoid runtime issues, can also count number of "steps" involved

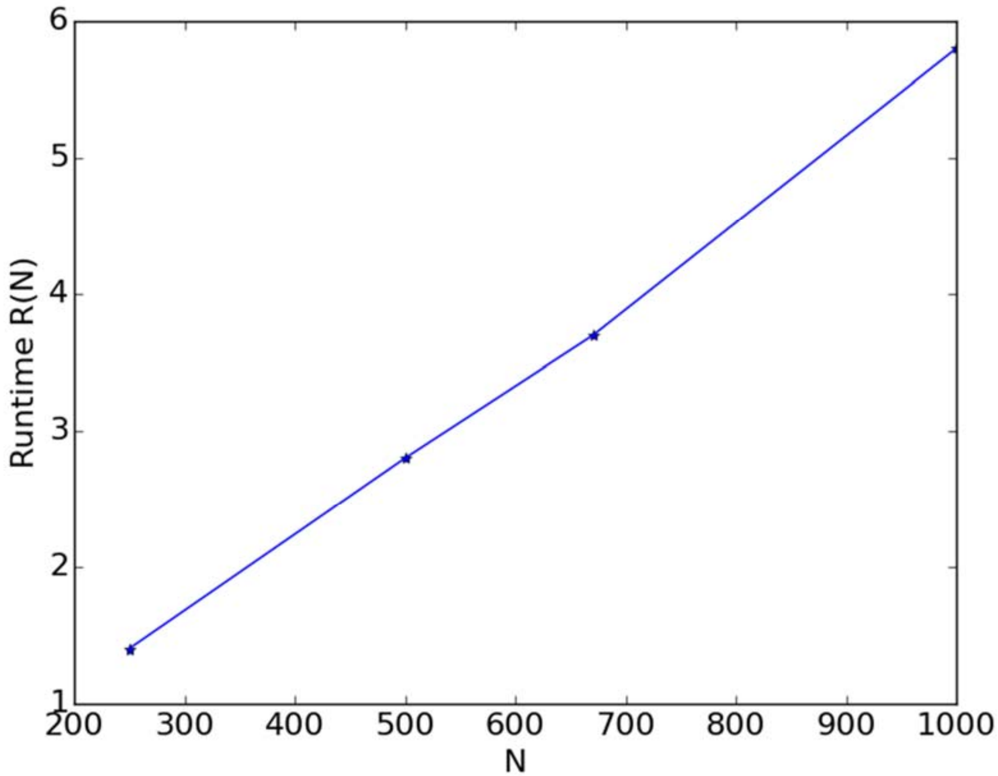
2) Make a plot of the runtime as a function of N, which we'll call $\mathrm{R(N)}$

3) Determine the general *shape* of $\mathrm{R(N)}$
   - Does $\mathrm{R(N)}$ look like N (linear), N^2 (quadratic), N^3 (cubic), $\log \mathrm{N}$ (logarithmic), etc.

Linear Search

- As the name implies, Linear Search is linear
  - If you double N, then R(N) should roughly double

N (input size)	R(N) (time)
250 items	1.4 sec
500 items	2.8 sec
671 items	3.8 sec
1000 items	5.7 sec

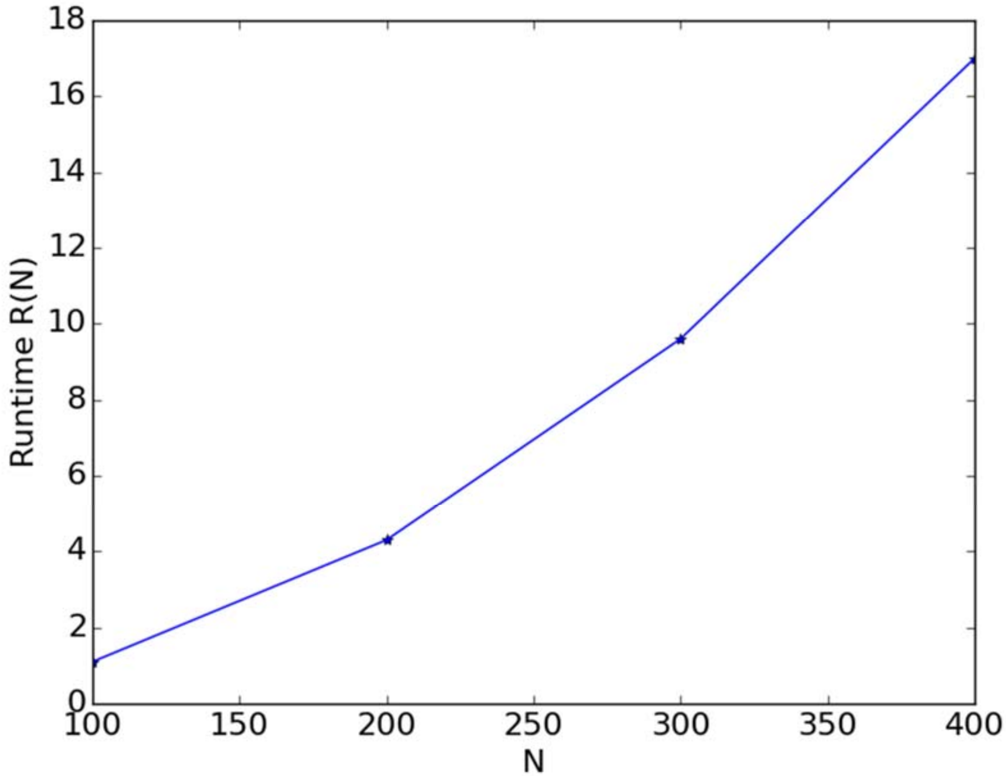


Peer Instruction Question

- Algorithm for: do any pairs in array sum to zero?
- Which function does R(N) look like?
  - Vote at http://PollEv.com/justinh

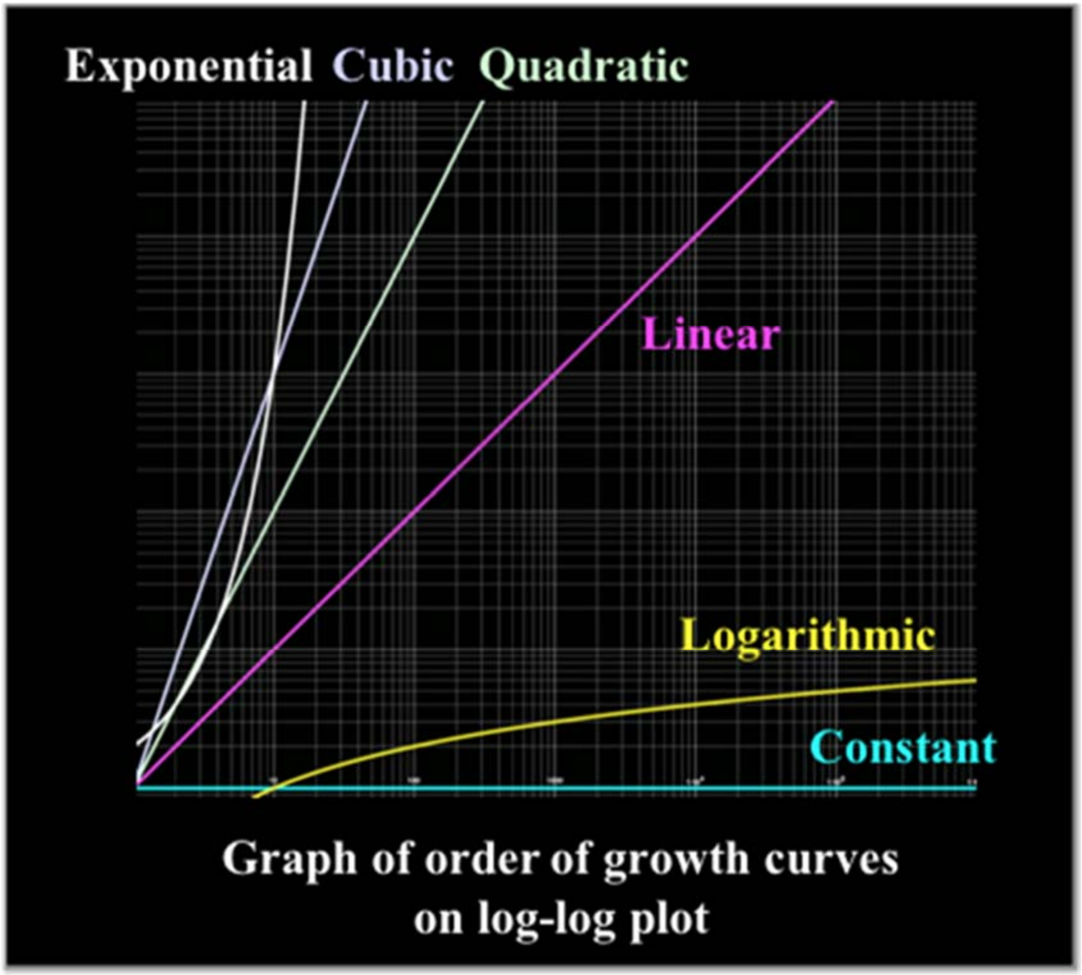
A. sqrt(N)
B. log(N)
C. N
D. N^2
E. 2^N

N (input size)	R(N) (time)
100 items	1.1 seconds
200 items	4.3 seconds
300 items	9.6 seconds
400 items	17.0 seconds



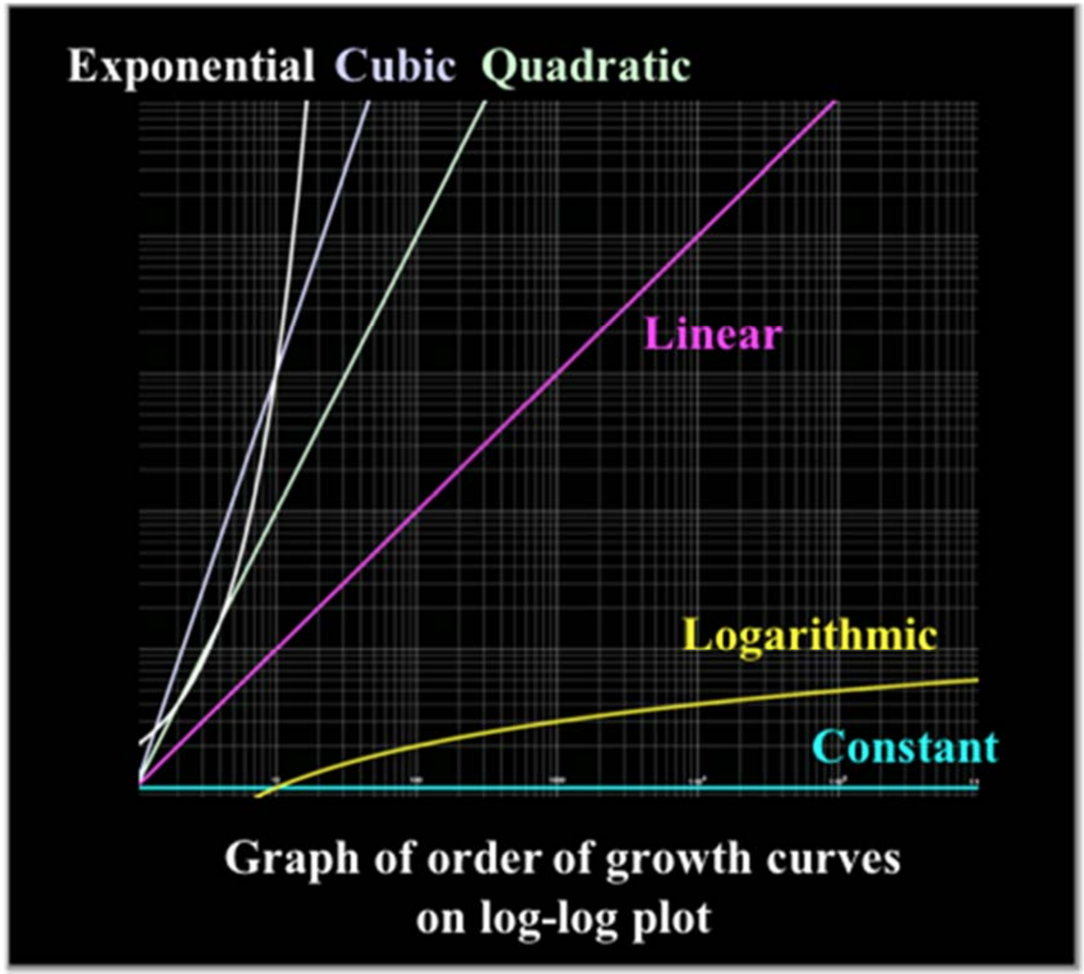
Orders of Growth

- The order of growth of $R(N)$ is its general shape:
  - Constant 1
  - Logarithmic $\log N$
  - Linear N
  - Quadratic N^2
  - Cubic N^3
  - Exponential 2^N
  - Factorial $N!$



Orders of Growth

- The order of growth of $\mathrm{R(N)}$ is its general shape:
  - Use *dominant* term
  - *e.g.* $10\mathrm{N}^2 + 4\log \mathrm{N}$ is quadratic



Binary Search

- What order of growth is Binary Search?
  - Analyze using number of "steps" in worst case

N (input size)	Indices to Check
1 items	
2 items	
4 items	
8 items	
16 items	

Which is Faster?

- Suppose we have two algorithms: one is linear in N and the other is quadratic in N
  - No voting

- Which is faster?
  A. **Linear Algorithm**
  B. **Quadratic Algorithm**
  C. **It depends**

The Reason Order of Growth Matters

- Roughly speaking, we care about really big N in real world applications
  - *e.g.* For Facebook, N (users) is ~ 1 billion
    - Want to generate list of suggested friends? Better be a fast algorithm as a function of N
- Order of growth is just a rough rule of thumb
  - There are limited cases where an algorithm with a worse order of growth can actually be faster
  - In almost all cases, order of growth works very well as a representation of an algorithm's speed

Orders of Growth Comparison

- The numbers below are rough estimates for a "typical" algorithm on a "typical" computer – provides a qualitative difference between the orders of growth

	Linear	Linearithmic	Quadratic	Cubic	Exponential	Exponential	Factorial
	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

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.