Analysis of Algorithms

Mason Vail
Boise State University Computer Science

What Are We Doing?

There may be multiple correct solutions, but they are not all equally efficient.

Efficiency can refer to amount of work for the processor, amount of memory used, I/O costs, etc.

How do we compare alternate solutions?

Washing Dishes

We have a stack of *n* dishes to wash and dry by hand. There's more than one way to do this.

Careful Smart Way:

- Prepare sink (5 minutes)
- Wash and rinse each dish (½ minute each)
- Dry each dish (½ minute each)

Example adapted from *Java Foundations: Introduction to Program Design & Data Structures, 4th Ed.*, by Lewis, DePasquale, and Chase, 2017.

Washing Dishes

A **growth function** describes the amount of work required to carry out an algorithm. For our Careful Smart Way of washing *n* dishes, the growth function would be

$$f(n) = 5 + \frac{1}{2} n + \frac{1}{2} n$$

or

$$f(n) = 5 + n$$

Washing Dishes

Careless Sloppy Way:

- Prepare sink (5 minutes)
- Wash and rinse each dish (0.5 minutes each)
- Dry (and redry) each washed dish ($\sum_{i=1}^{n} 1/2\ i$)

$$f(n) = 5 + 1/2\ n + \sum_{i=1}^{n} 1/2\ i$$

$$f(n) = 5 + 1/2\ n + 1/4\ n^2 + 1/4\ n$$

$$f(n) = 5 + 3/4\ n + 1/4\ n^2$$

Washing Dishes

n	5 + n	5 + ¾ n + ½ n^2
1	6	6
10	15	37.5
20	25	120
30	35	252.5
40	45	435
50	55	667.5
...	...	...
110	115	3112.5

Washing Dishes

n	5 + n	% from 5	% from n
1	6	83.33%	16.67%
20	25	20.00%	80.00%
40	45	11.11%	88.89%
60	65	7.69%	92.31%
80	85	5.88%	94.74%
100	105	4.76%	95.24%

n	5 + ¾ n + ½ n^2	% from 5	% from ¾ n	% from ¼ n^2
1	6	83.33%	12.50%	4.17%
20	120	4.17%	12.50%	83.33%
40	435	1.15%	6.90%	91.95%
60	950	0.53%	4.74%	94.74%
80	1665	0.30%	3.60%	96.10%
100	2580	0.19%	2.91%	96.90%

Washing Dishes

The dominant factor as n becomes large is the **order** of the algorithm. We represent the order with **Big-O notation**.

$f(n) = 5 + \mathbf{n}$

$O(n)$

$f(n) = 5 + \frac{3}{4}\, n + \frac{1}{4}\, \mathbf{n^2}$

$O(n^2)$

Washing Dishes - Your Turn

Germ-o-phobe Way:

Every dish must be washed completely independently of all others, so no washed-off grime contaminates the next dish.

- Clean and prepare sink before each dish (10 minutes each)
- Wash and rinse each dish (0.5 minutes each)
- Dry each dish (0.5 minutes each)

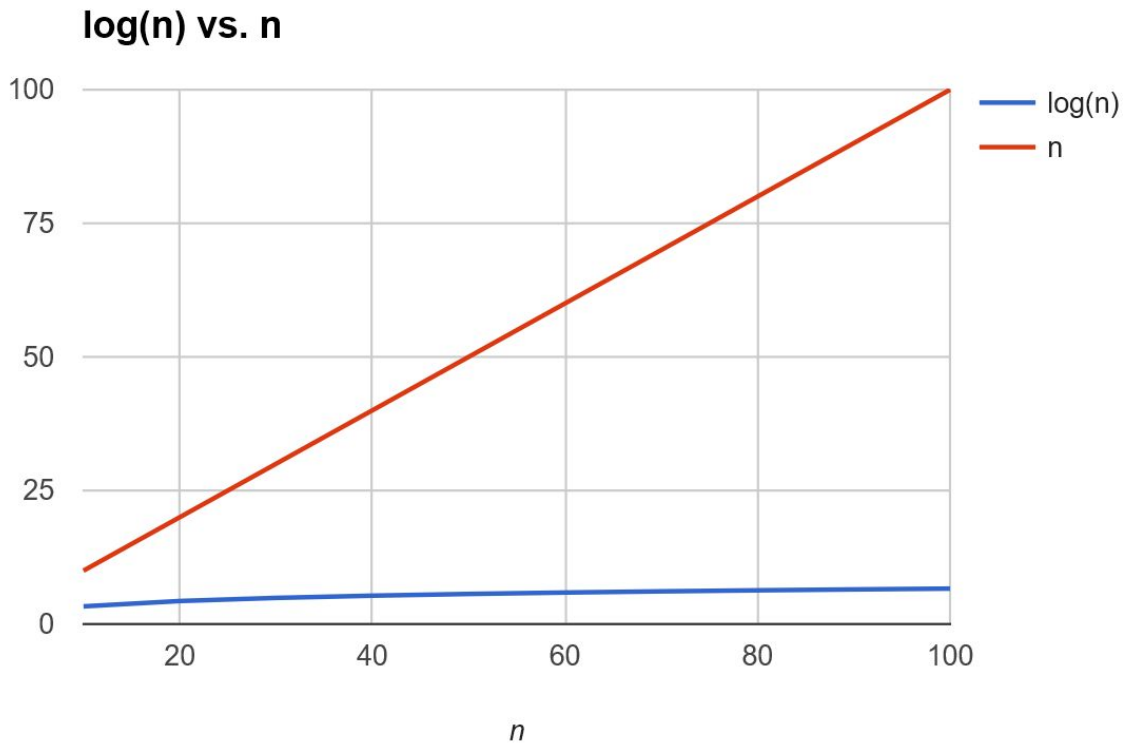
What is the growth function and order? How does it compare to the other two dishwashing algorithms?

Common Orders for Single Variable Algorithms

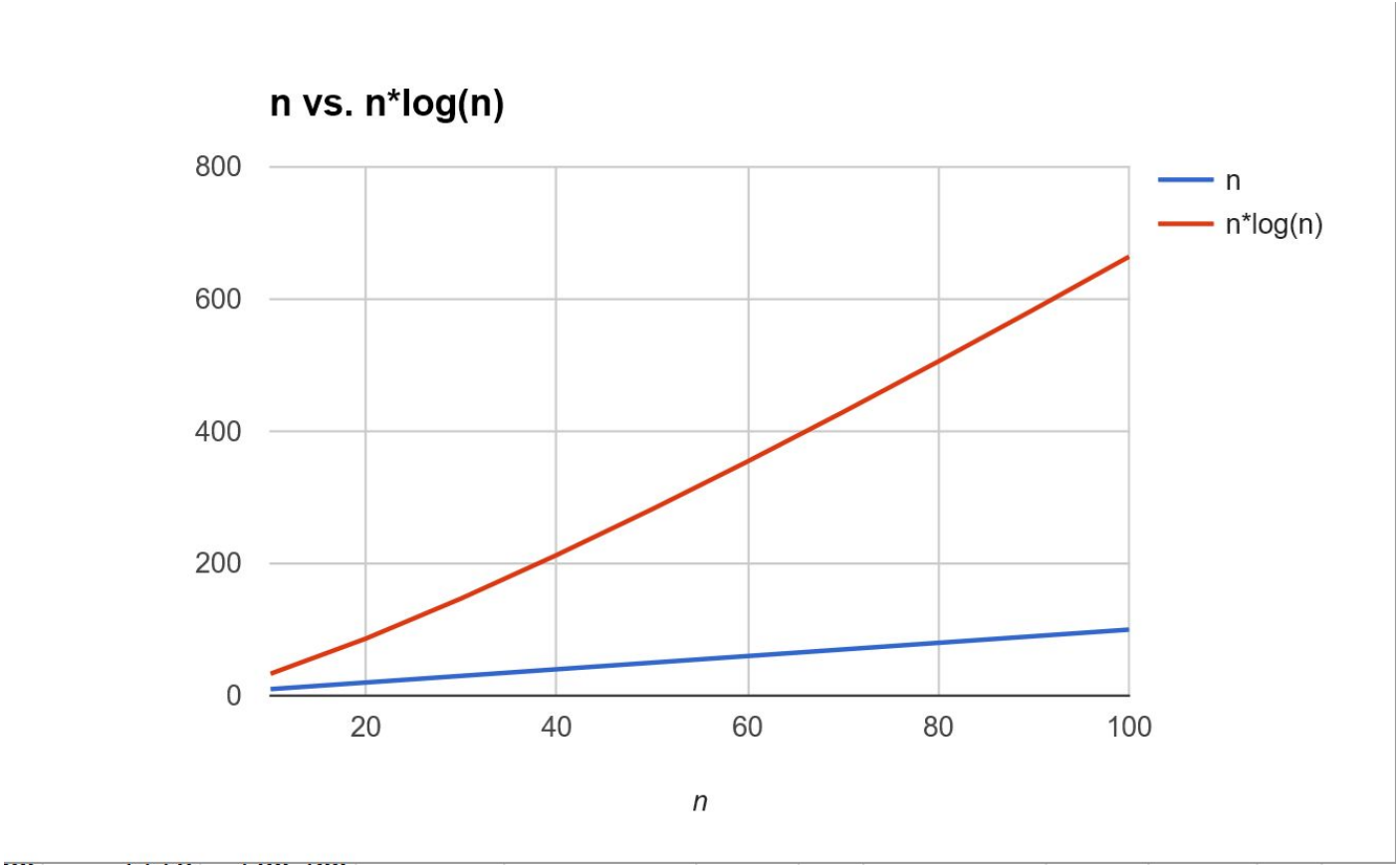
n	O(1) Constant	O(log n) Logarithmic	O(n) Linear	O(n log n) Log-linear	$O(n^2)$ Quadratic	$O(n^3)$ Cubic	$O(2^n)$ Exponential
1	3	0	1	1	1	1	2
2	3	1	2	2	4	8	4
4	3	2	4	8	16	64	16
8	3	3	8	24	64	512	256
16	3	4	16	64	256	4,096	65,536
32	3	5	32	160	1,024	32,768	4,294,967,296

Adapted from *Object-Oriented Data Structures Using Java, 3rd Ed.*, by Dale, Joyce, and Weems, 2012.

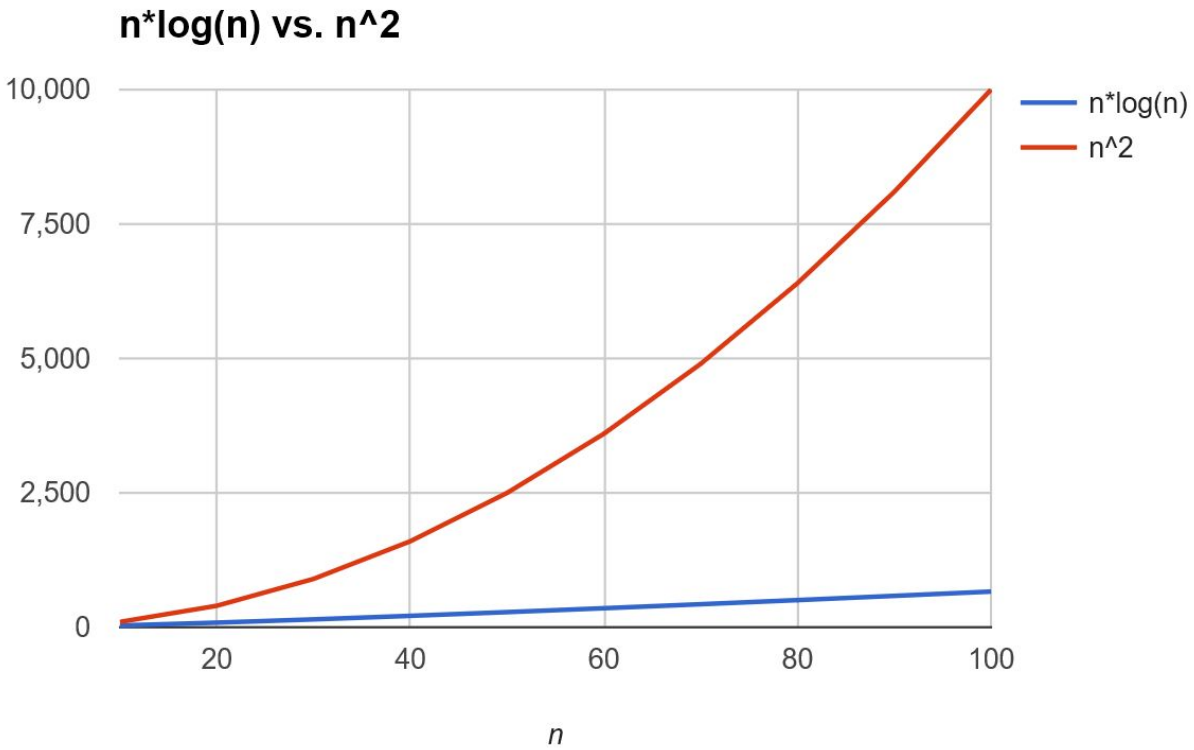
Comparing Orders: Logarithmic vs. Linear



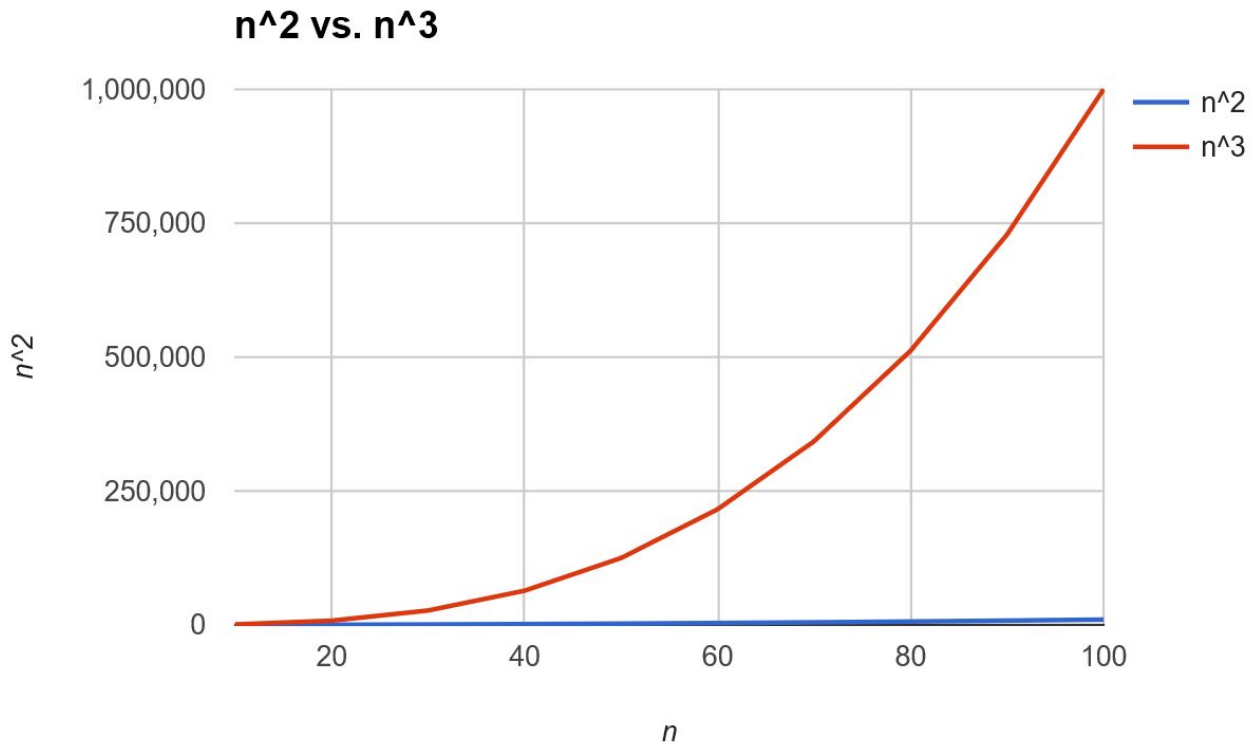
Comparing Orders: Linear vs. Log-Linear



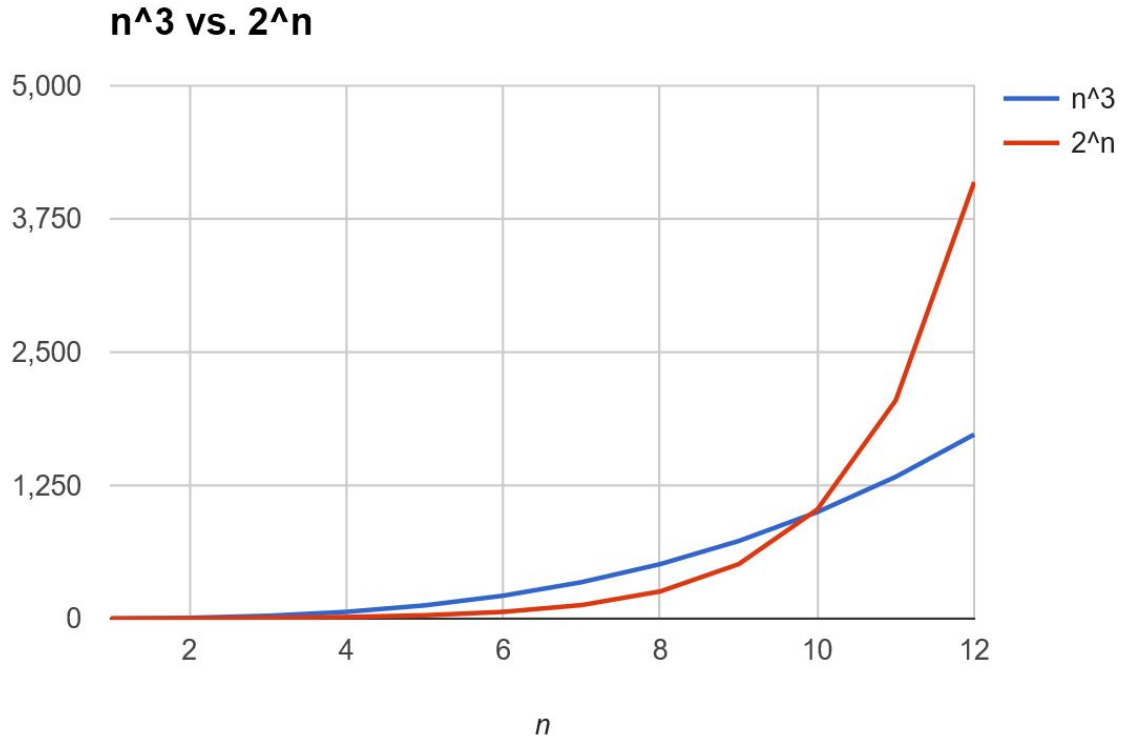
Comparing Orders: Log-Linear vs. Quadratic



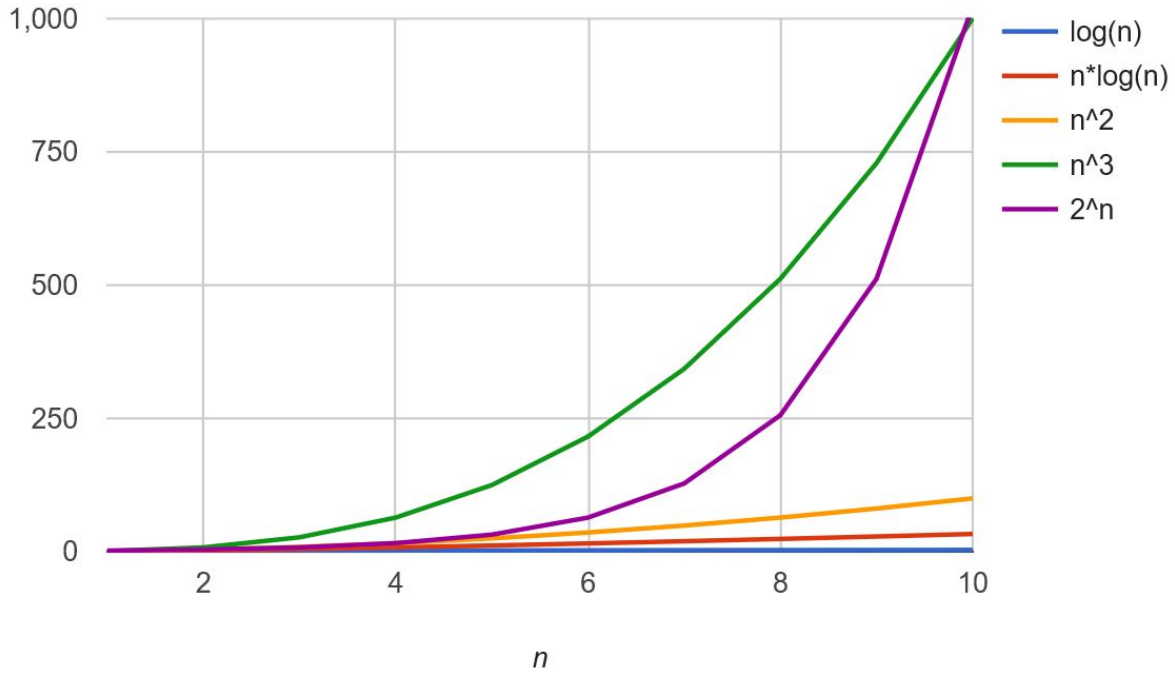
Comparing Orders: Quadratic vs. Cubic



Comparing Orders: Cubic vs. Exponential



Comparing Orders: All Together



Example O(n) Algorithm

```
//the length of the String is the size of the problem, n
public int countChars (String str, char c) {
    int count = 0;
    for (int index = 0; index < str.length(); index++) {
        if (str.charAt(index) == c) {
            count++;
        }
    }
    return count;
}
```

Example O(n^2) Algorithm

```
//String length is n
public String sortString (String str)
{
  char[] chars = str.tochars();

  for (int idx = 0;
      idx < chars.length; idx++) {
    int lowIndex = findLow(chars,idx);
    swap(chars, idx, lowIndex);
  }

  return new String(chars);
}
```

```
//String length is n
private int findLow(char[] chars,
    int startIdx) {
  int lowIdx = startIdx;
  for (int idx = startIdx + 1;
      idx < chars.length; idx++) {
    if (chars[idx] < chars[lowIdx]) {
      lowIdx = idx;
    }
  }
  return lowIdx;
}

private void swap(char[] chars,
    int idx1, int idx2) {
  char tmp = chars[idx1];
  chars[idx1] = chars[idx2];
  chars[idx2] = tmp;
}
```

Basic Analysis Walkthrough

Constant Factor. How many statements would be executed in a call to algorithm() when the array size is zero (n == 0)? Is it the same for n == 1?

Best and Worst Case Growth Functions. Under what conditions would the minimum number of statements be executed for all n > 0 (or n > 1)? Under what conditions would the maximum number of statements be executed? In other words, is there some arrangement or other characteristic of the problem that affects how many statements get executed? What is the growth function under these conditions?

Expected Average Case Growth Function. Assuming a “usual” (random, perhaps) input to the algorithm, what is the expected average number of statements (the expected growth function) for a call to algorithm()?

Order. Based on the growth function analysis above, what is the runtime order (big-O) of algorithm()?

Analysis of Algorithms

Mason Vail
Boise State University Computer Science