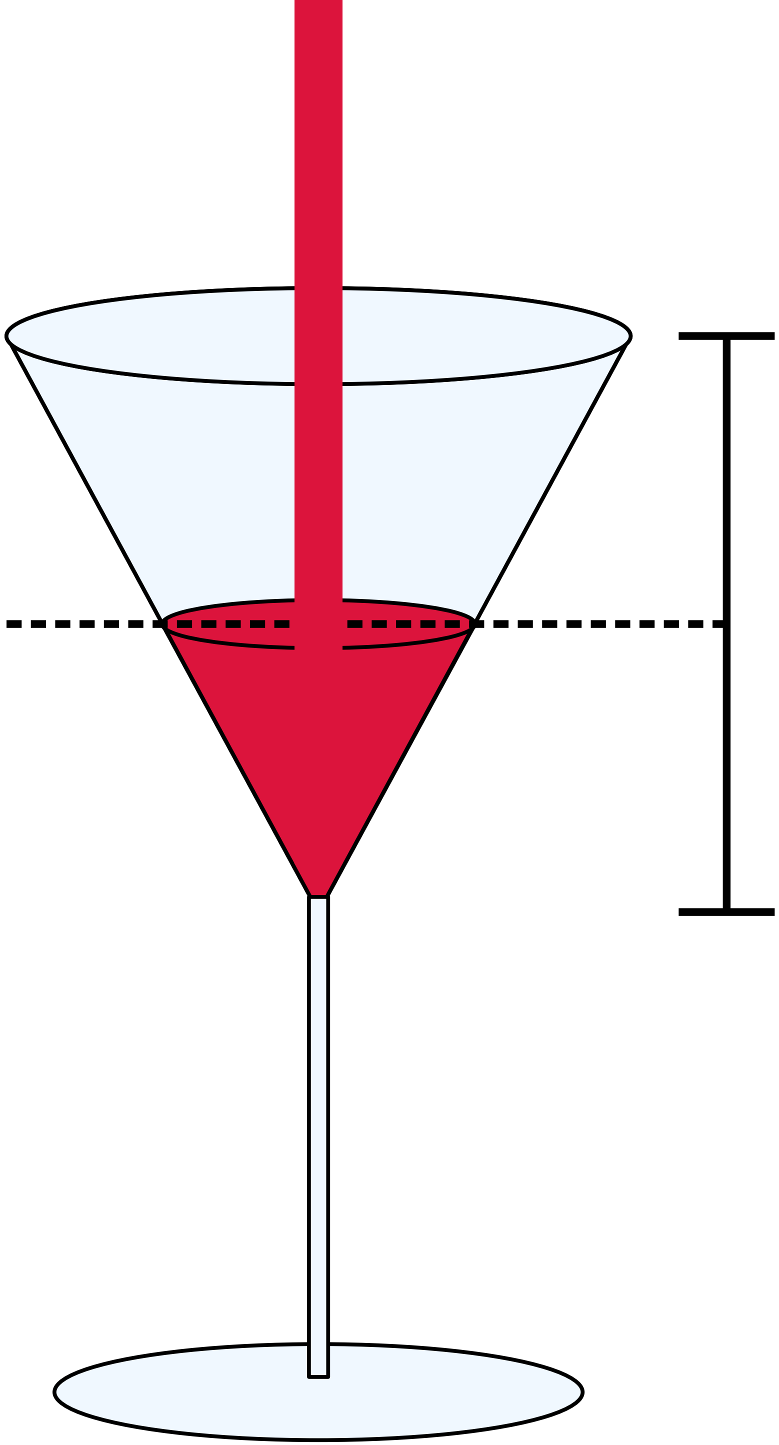
Big-O Notation

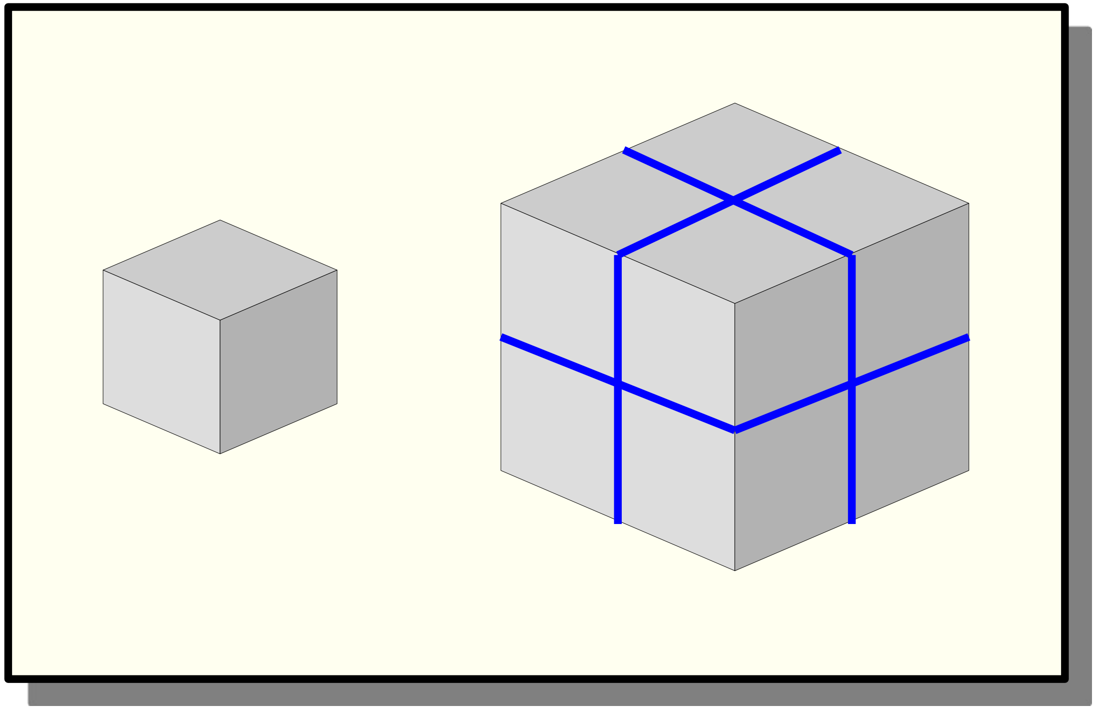
A Motivating Question

```
void printTrigrams_v1(const string& str) {
    for (int i = 0; i + 3 <= str.length(); i++) {
        string trigram = str.substr(i, 3);
        cout << trigram << endl;
    }
}
```

```
void printTrigrams_v2(const string& str) {
    string s = str;
    while (s.length() >= 3) {
        cout << s[0] << s[1] << s[2] << endl;
        s = s.substr(1);
    }
}
```

Estimating Quantities





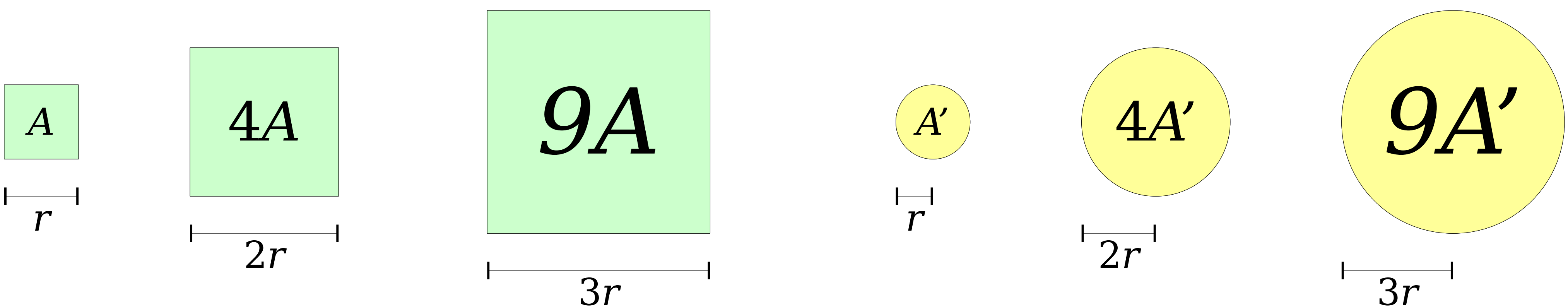
The glass is filled to half its height.
What fraction of the glass is full?

Answer at
https://cs106b.stanford.edu/pollev

Knowing the rate at which some quantity scales allows you to predict its value in the future, even if you don’t have an exact formula.

Big-O Notation

- ***Big-O notation*** is a way of quantifying the rate at which some quantity grows.
- For example:
  - A square of side length r has area $O(r^2)$.
  - A circle of radius r has area $O(r^2)$.

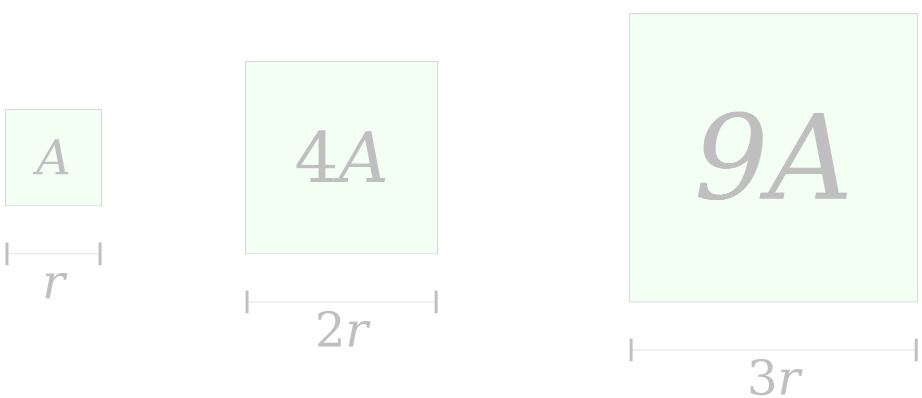


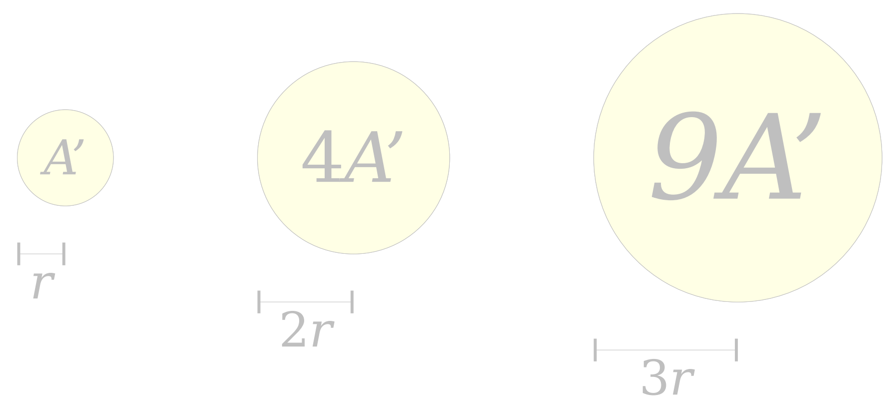
Doubling r increases area 4×.
Tripling r increases area 9×.

Doubling r increases area 4×.
Tripling r increases area 9×.

Big-O Notation

- ***Big-O notation*** is a way of quantifying the rate at which some quantity grows.
- For example:
  - A square of side length r has area $O(r^2)$.
  - A circle of radius r has area $O(r^2)$.





Doubling r increases area 4×.
Tripling r increases area 9×.

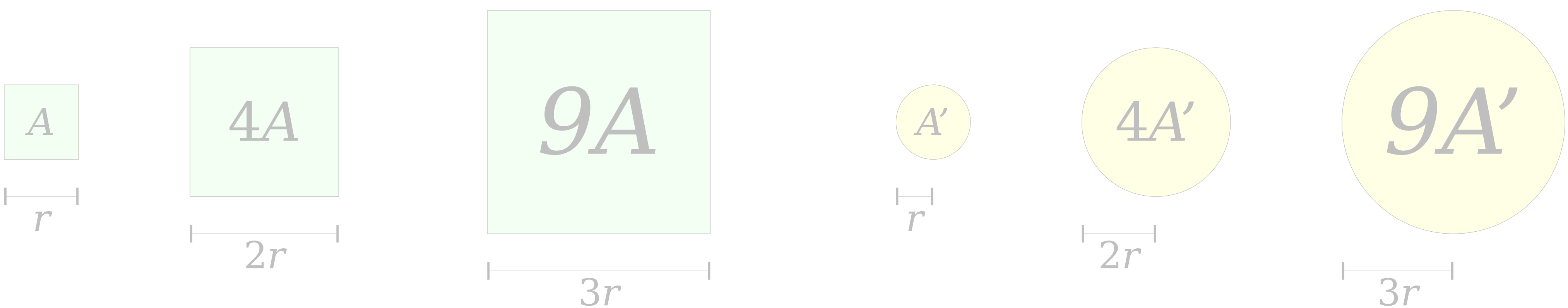
Doubling r increases area 4×.
Tripling r increases area 9×.

Big-O Notation

- ***Big-O notation*** is a wa
  rate at which some quan

This just says that these quantities grow at the same relative rates. It does not say that they're equal!

- For example:
  - A square of side length r has area $O(r^2)$.
  - A circle of radius r has area $O(r^2)$.



Doubling r increases area 4×.
Tripling r increases area 9×.

Doubling r increases area 4×.
Tripling r increases area 9×.

Big-O Notation

- ***Big-O notation*** is a way of quantifying the rate at which some quantity grows.
- For example:
  - A square of side length r has area $O(r^2)$.
  - A circle of radius r has area $O(r^2)$.
  - A cube of side length r has volume $O(r^3)$.
  - A sphere of radius r has volume $O(r^3)$.
  - A sphere of radius r has surface area $O(r^2)$.
  - A cube of side length r has surface area $O(r^2)$.

Example: Metcalfe's Law

- ***Metcalfe's Law*** says that

  The value of a communications network with n users is $O(n^2)$.

Example: Metcalfe's Law

- ***Metcalfe's Law*** says that

  The value of a communications network with n users is $O(n^2)$.

- Imagine a social network has 10,000,000 users and is worth $10,000,000. Estimate how many users it needs to have to be worth $1,000,000,000.

Example: Metcalfe's Law

- ***Metcalfe's Law*** says that

  The value of a communications network with n users is $O(n^2)$.

- Imagine a social network has 10,000,000 users and is worth $10,000,000. Estimate how many users it needs to have to be worth $1,000,000,000.

Answer at ***https://cs106b.stanford.edu/pollev***

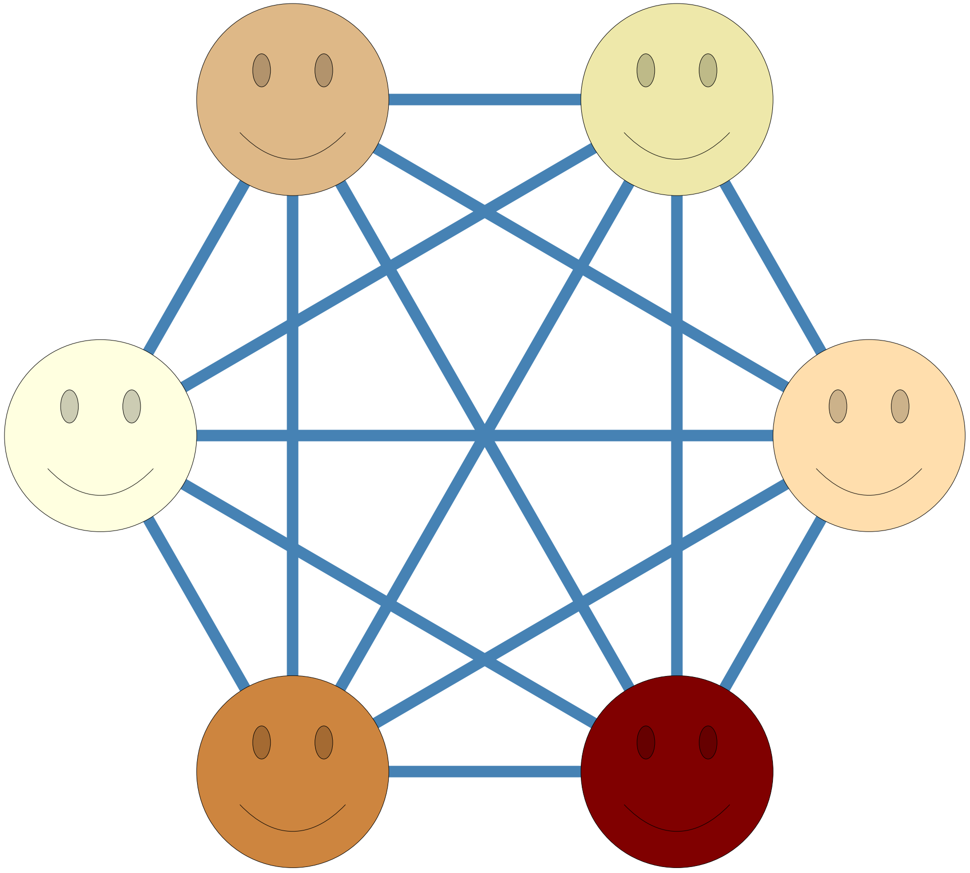
Example: Metcalfe’s Law

- ***Metcalfe’s Law*** says that

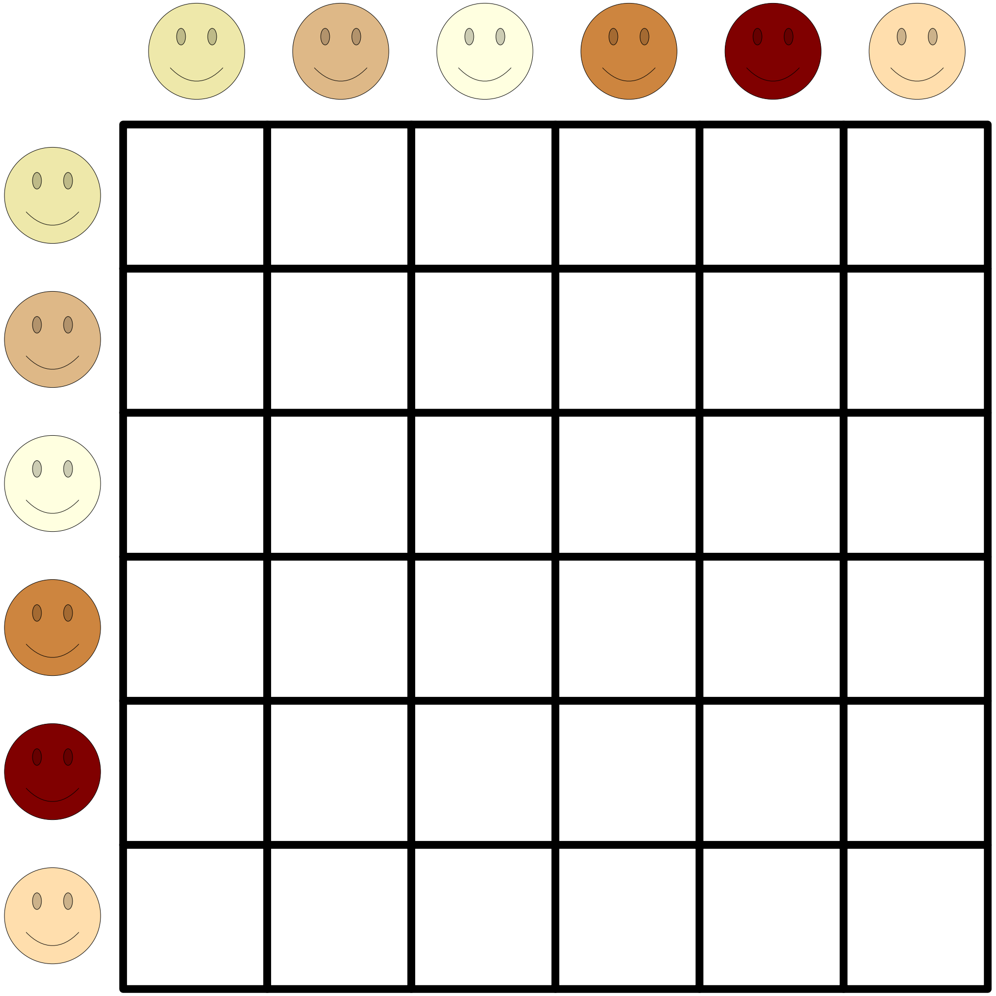
  The value of a communications network with n users is $O(n^2)$.

- Imagine a social network has 10,000,000 users and is worth \$10,000,000. Estimate how many users it needs to have to be worth \$1,000,000,000.
- ***Reasonable guess:*** The network needs to grow its value 100×. Since value grows quadratically with size, it needs to grow its user base 10×, requiring 100,000,000 users.

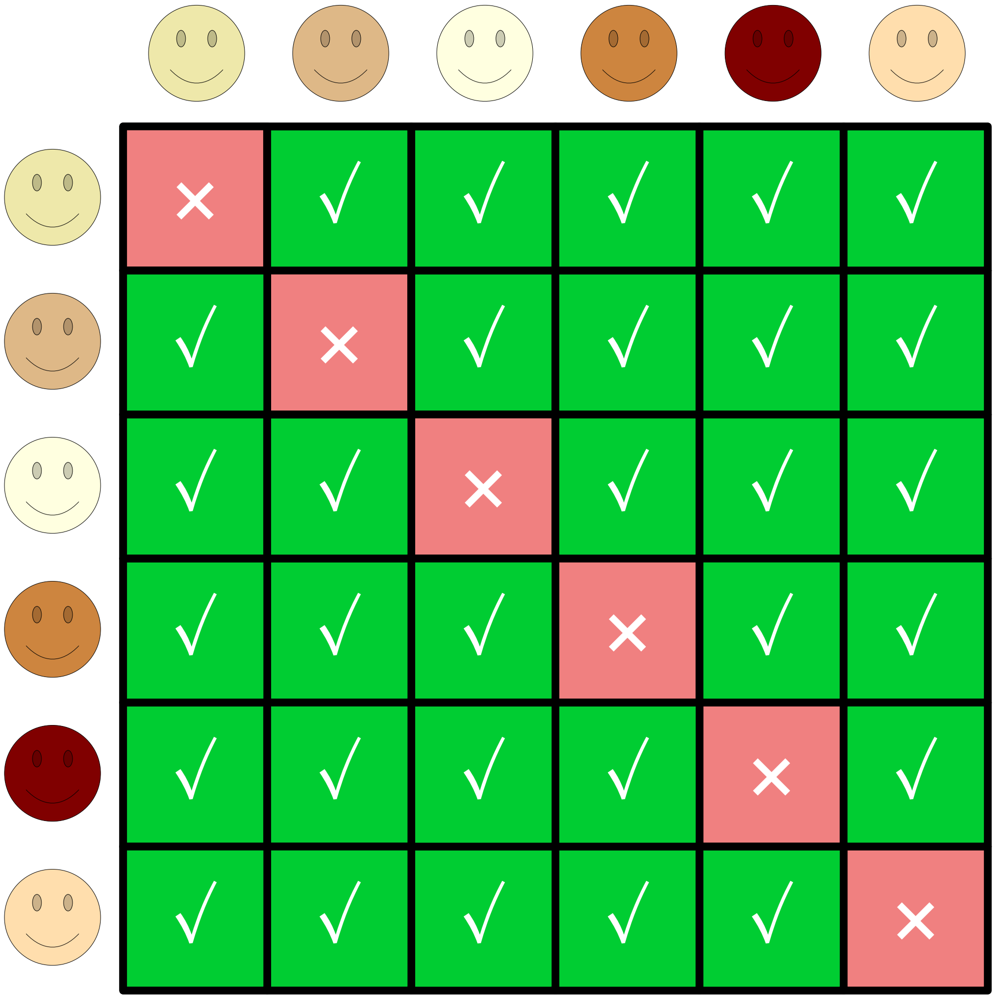
Example: Metcalfe’s Law



Example: Metcalfe's Law

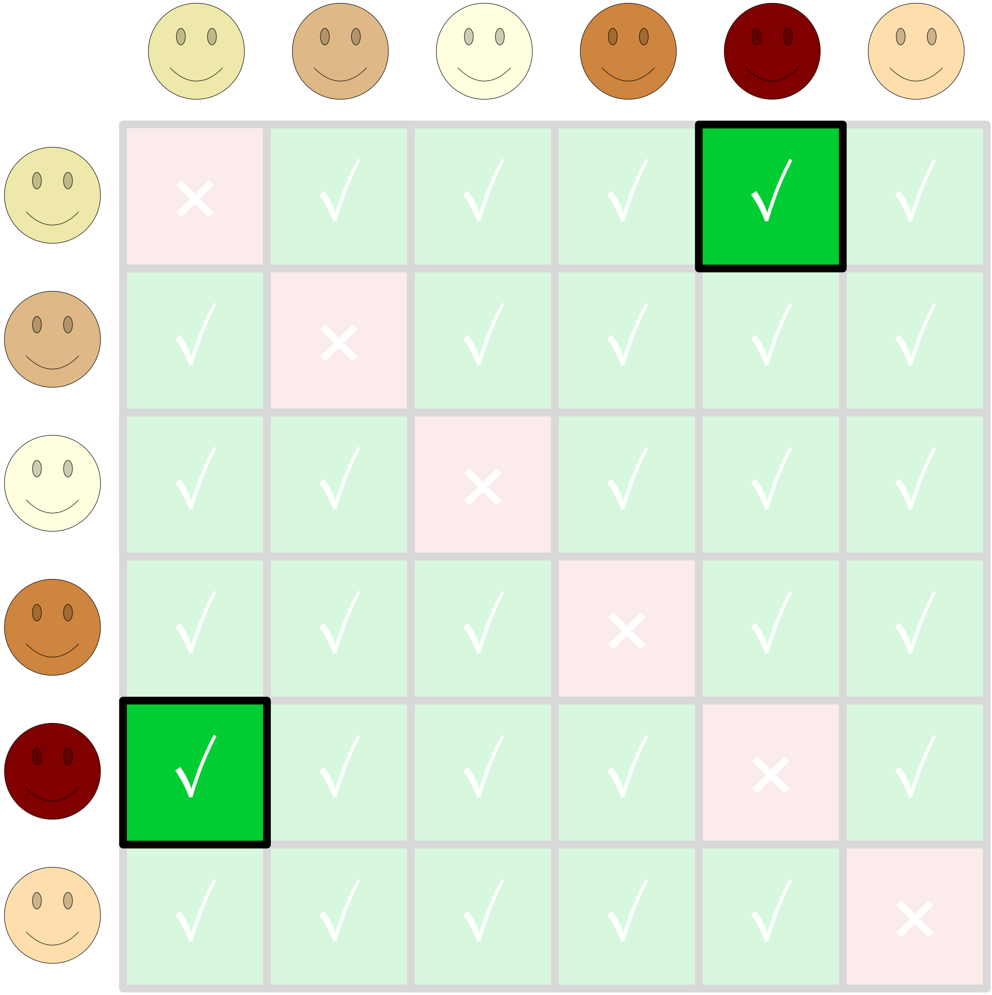


Example: Metcalfe's Law



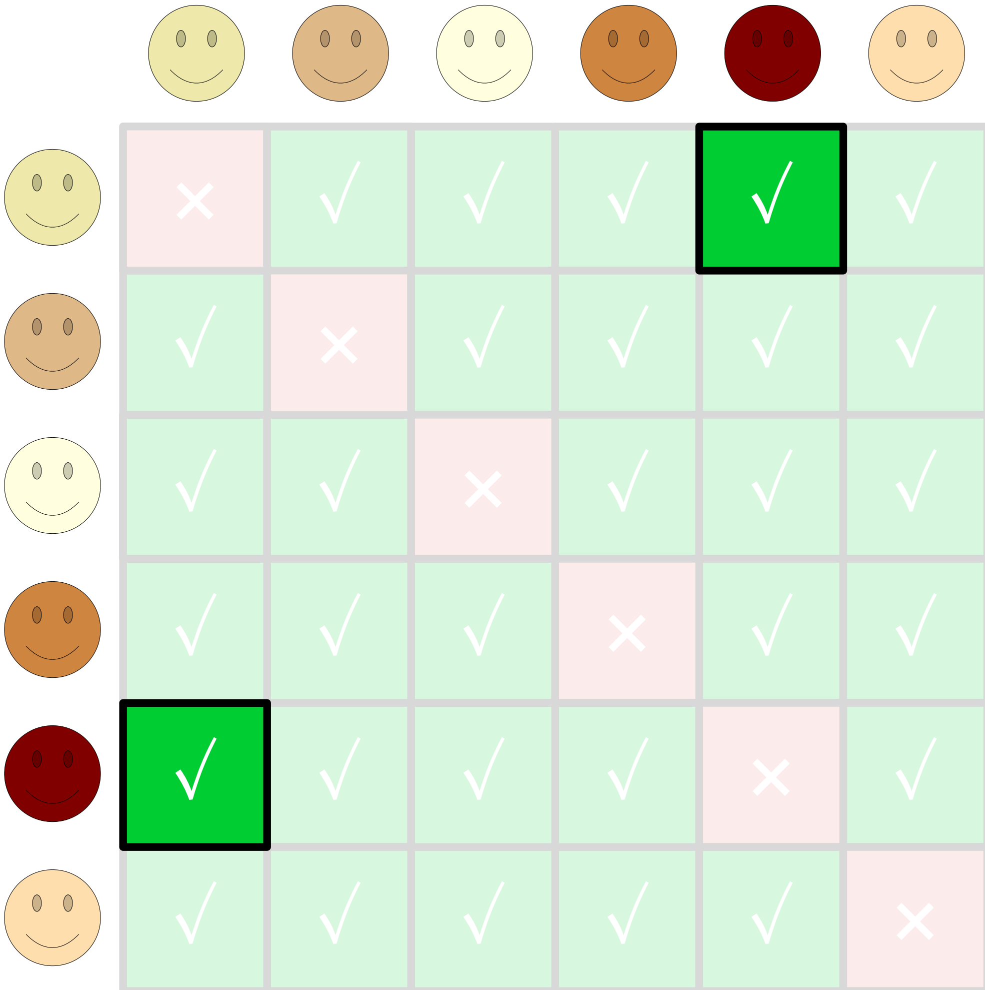
$$n^2 - n$$

Example: Metcalfe’s Law



$$n^2 - n$$

Example: Metcalfe's Law



$$\frac{n^2 - n}{2}$$

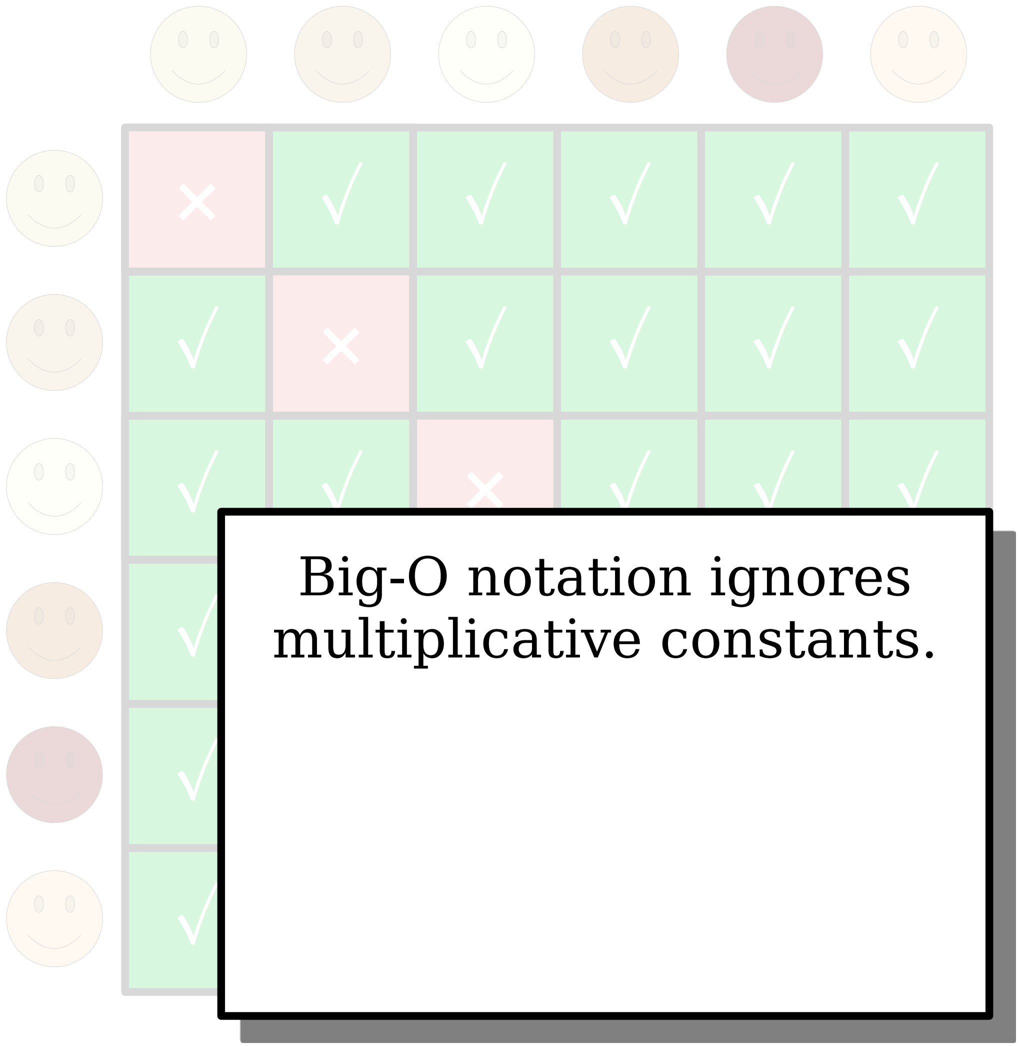
Example: Metcalfe's Law

$$\frac{n^2 - n}{2}$$

Example: Metcalfe's Law

$$\frac{n^2}{2} - \frac{n}{2}$$

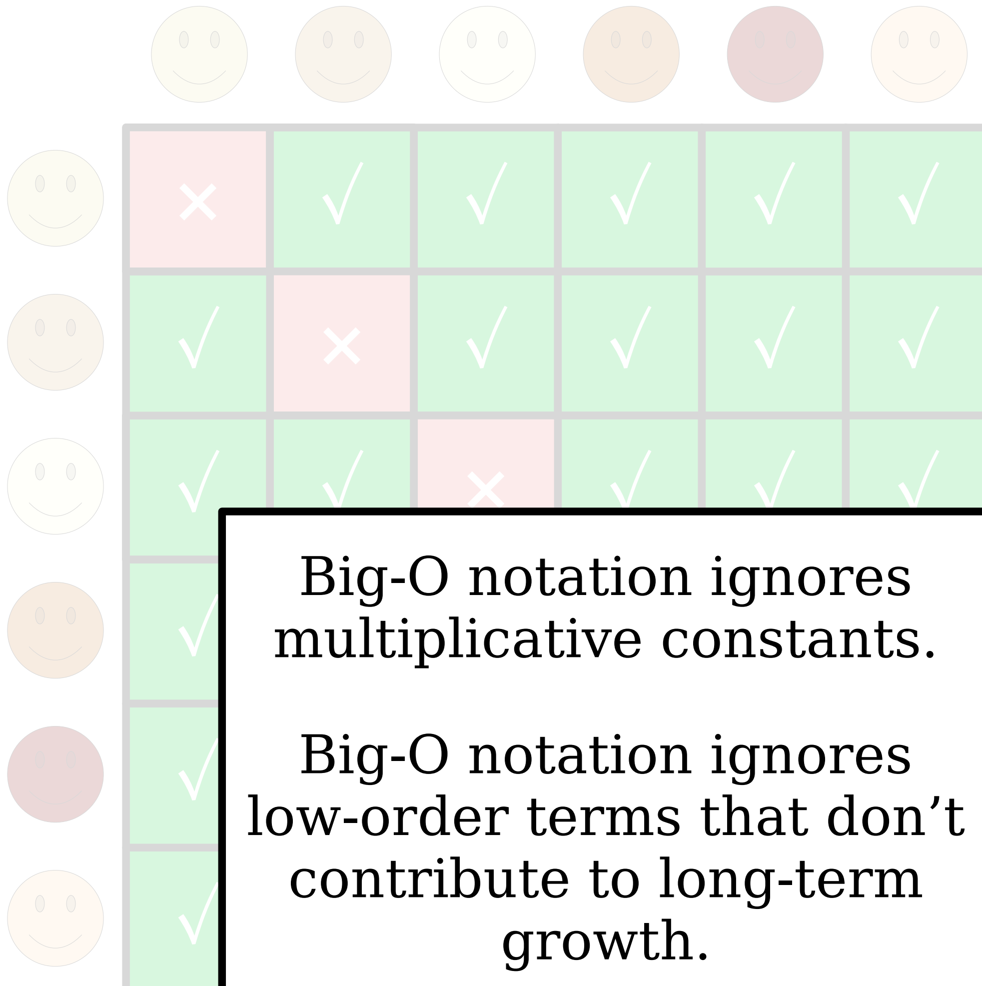
Example: Metcalfe's Law



$$O\left(\frac{n^2}{2} - \frac{n}{2}\right)$$

Big-O notation ignores multiplicative constants.

Example: Metcalfe's Law

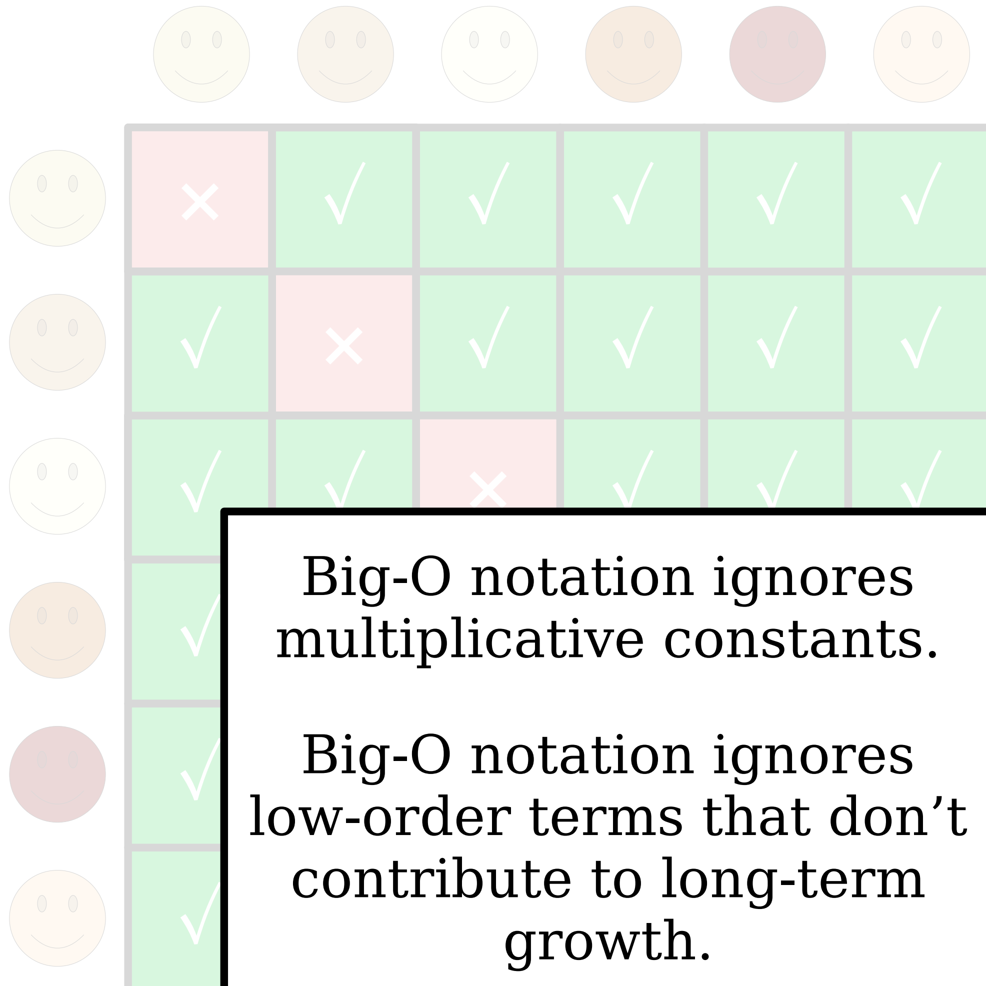


$O(n^2 - n)$

Big-O notation ignores multiplicative constants.

Big-O notation ignores low-order terms that don't contribute to long-term growth.

Example: Metcalfe’s Law



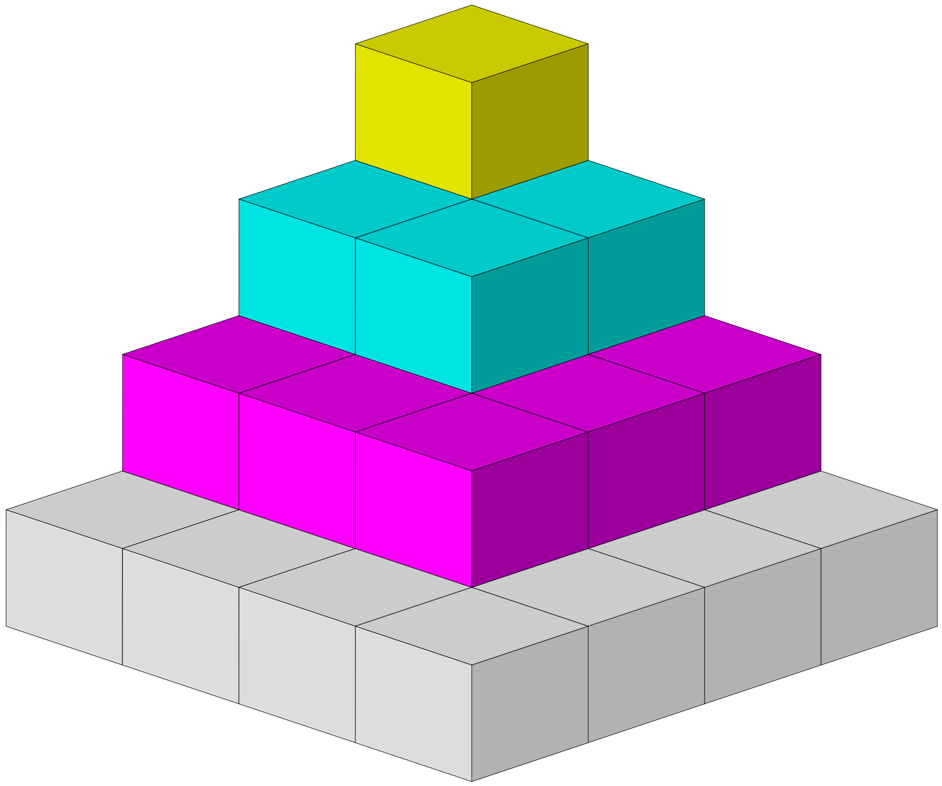
$$O(n^2)$$

Big-O notation ignores multiplicative constants.

Big-O notation ignores low-order terms that don’t contribute to long-term growth.

Counting Cubes

- Suppose we make a pyramid of cubes like the one shown to the right.
- The top layer is a 1×1 square of cubes, the next is a 2×2 square of cubes, then a 3×3 square of cubes, etc.
- How many cubes are there if the pyramid is *n* layers deep?

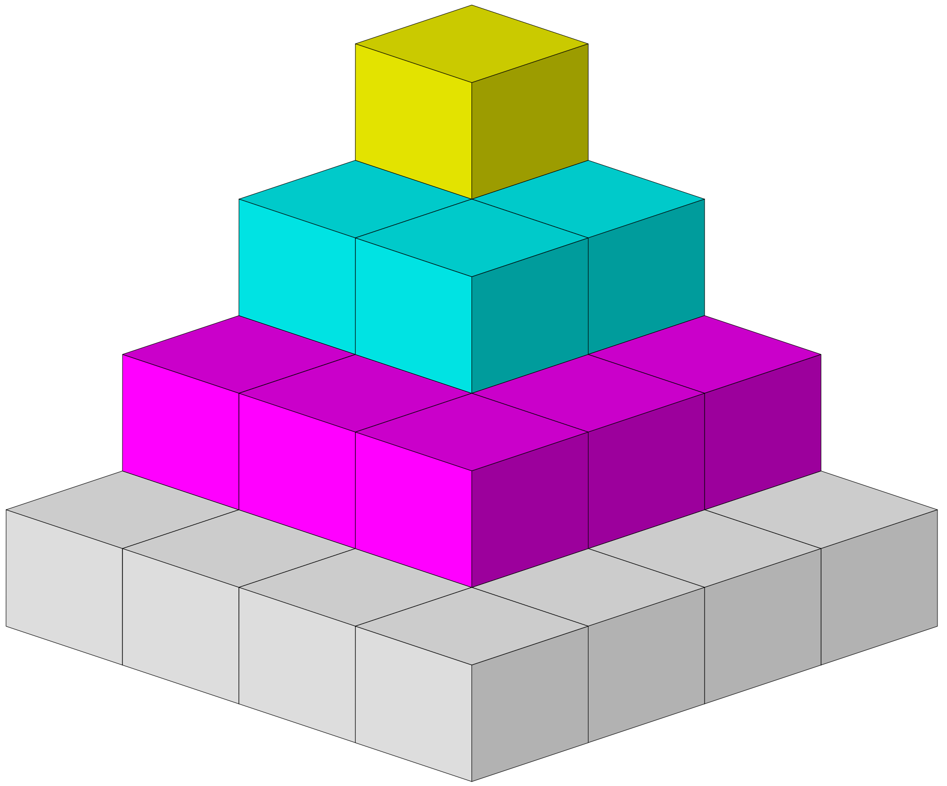


Counting Cubes

- Here's some numbers:

10 layers:	385 cubes
20 layers:	2,870 cubes
30 layers:	9,455 cubes
40 layers:	22,140 cubes
50 layers:	42,925 cubes

- ***Question:*** Can we roughly estimate how many cubes will be in a 60-layer stack, even if we don't have an exact formula?

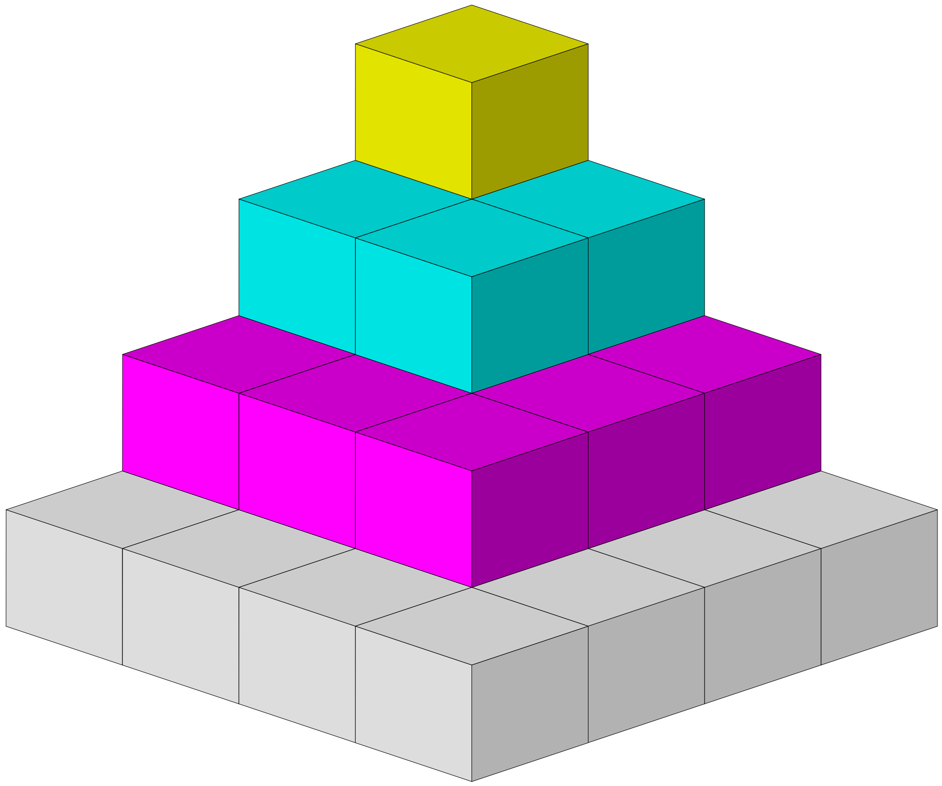


Counting Cubes

Suppose we form a tower of cubes like this that is n layers deep. How many cubes are there in total?

A. $O(n)$
B. $O(n^2)$
C. $O(n^3)$
D. $O(n^4)$

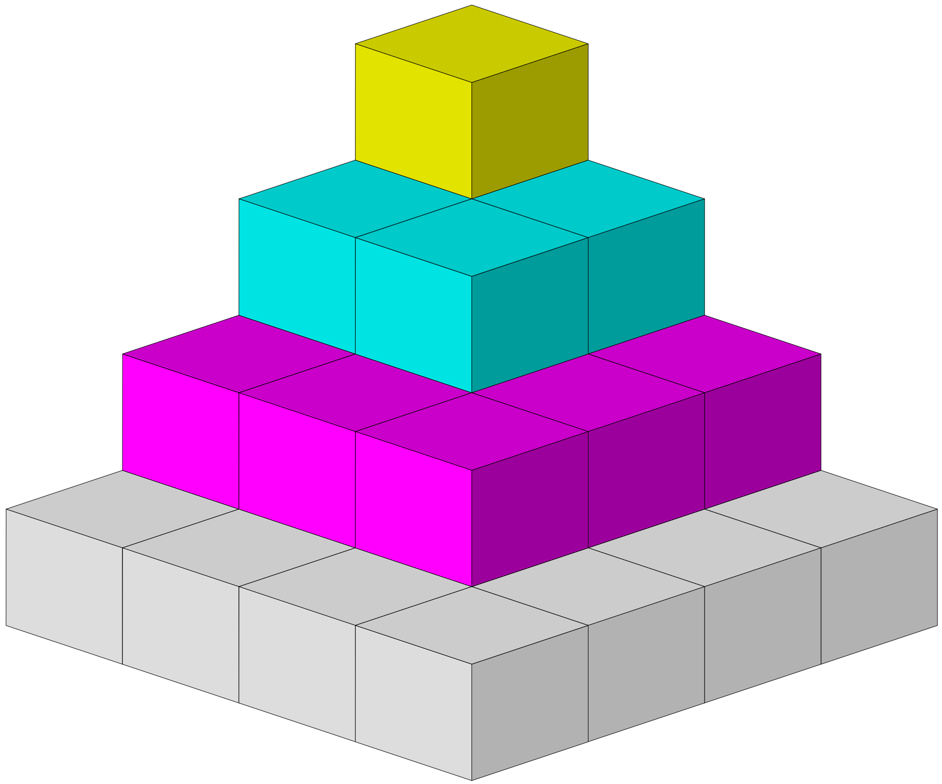
Answer at
https://cs106b.stanford.edu/pollev



Counting Cubes

- Here's some numbers:

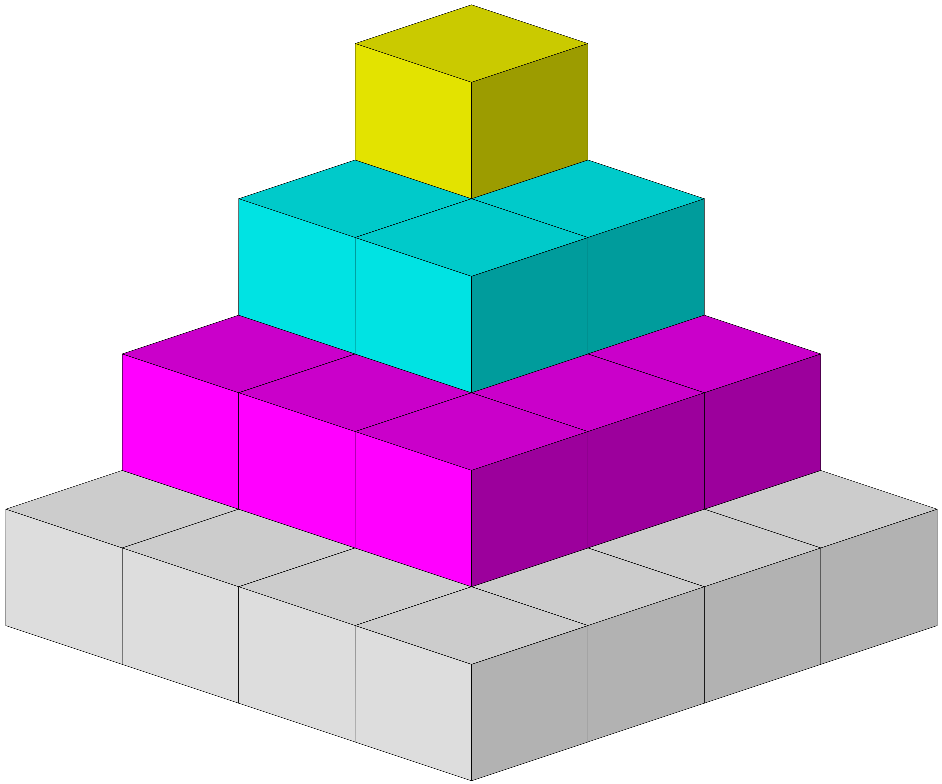
10 layers:	385 cubes
20 layers:	2,870 cubes
30 layers:	9,455 cubes
40 layers:	22,140 cubes
50 layers:	42,925 cubes



Counting Cubes

Here’s some numbers:

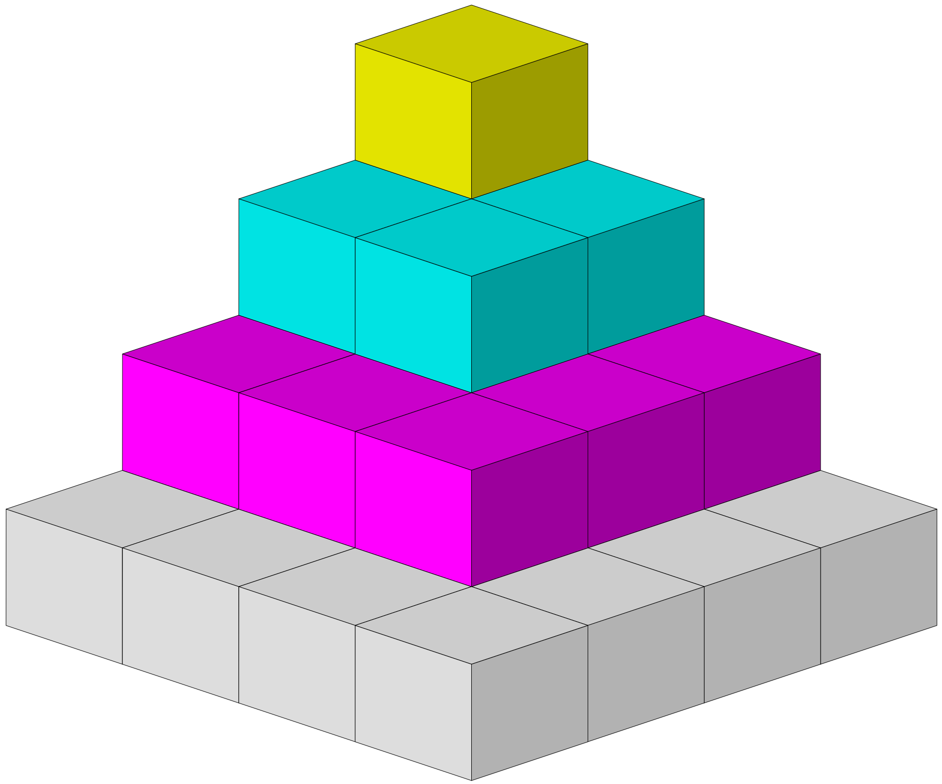
10 layers:	385 cubes
20 layers:	2,870 cubes
30 layers:	9,455 cubes
40 layers:	22,140 cubes
50 layers:	42,925 cubes



Counting Cubes

Here's some numbers:

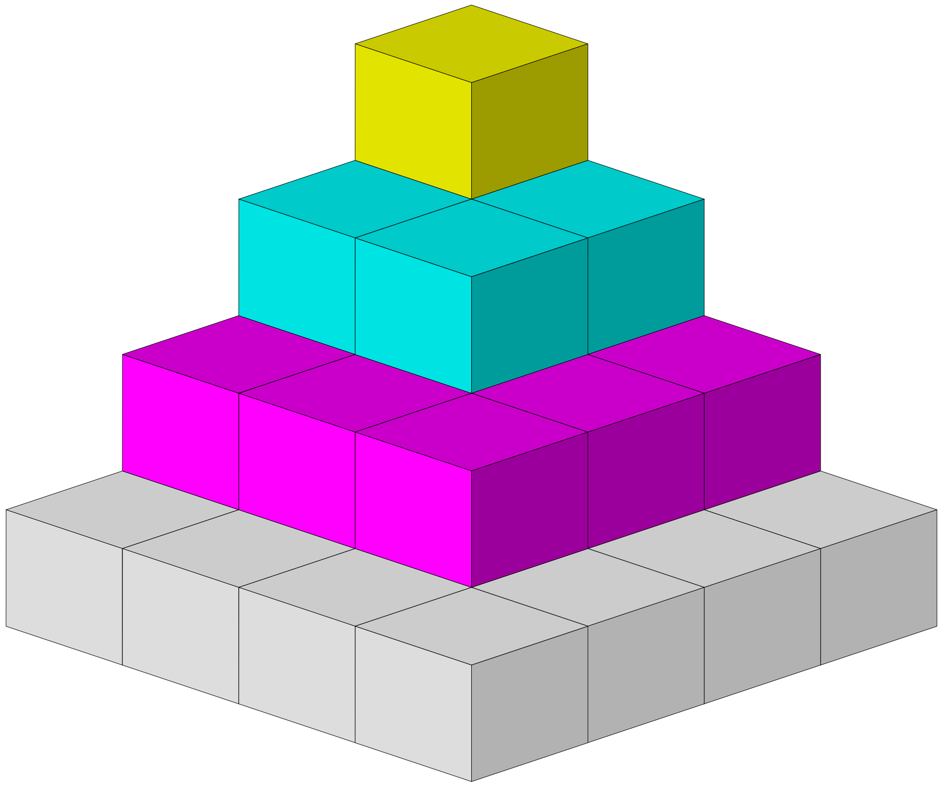
10 layers:	385 cubes
20 layers:	2,870 cubes
30 layers:	9,455 cubes
40 layers:	22,140 cubes
50 layers:	42,925 cubes
60 layers: ≈ 8× as many	



Counting Cubes

Here's some numbers:

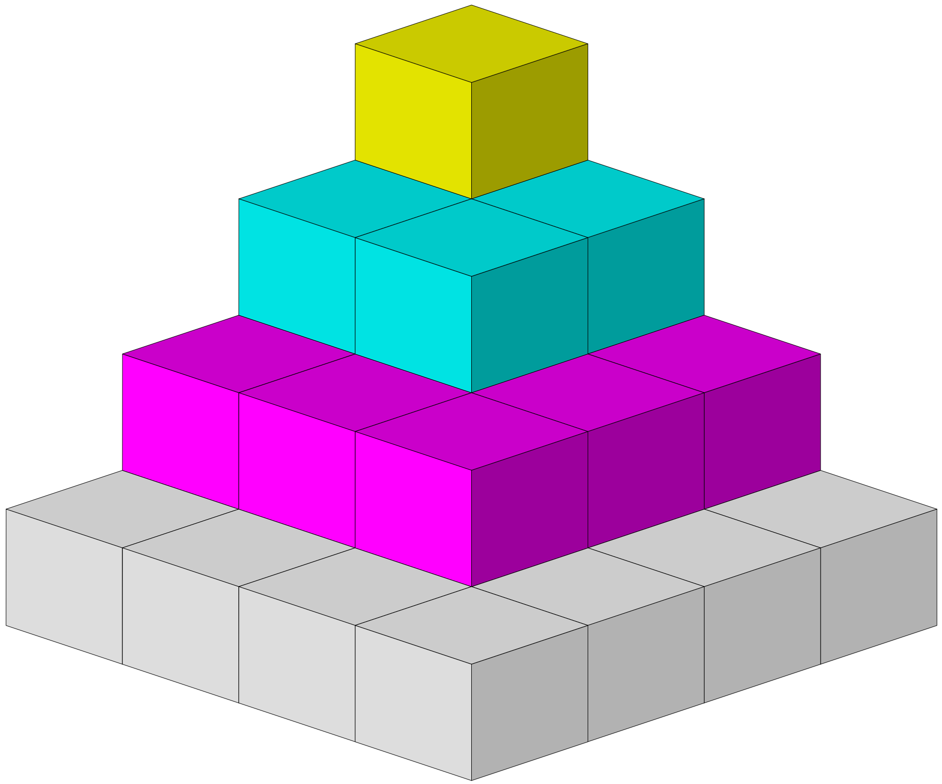
10 layers:	385 cubes
20 layers:	2,870 cubes
30 layers:	9,455 cubes
40 layers:	22,140 cubes
50 layers:	42,925 cubes
60 layers:	≈75,640 cubes



Counting Cubes

- Here's some numbers:

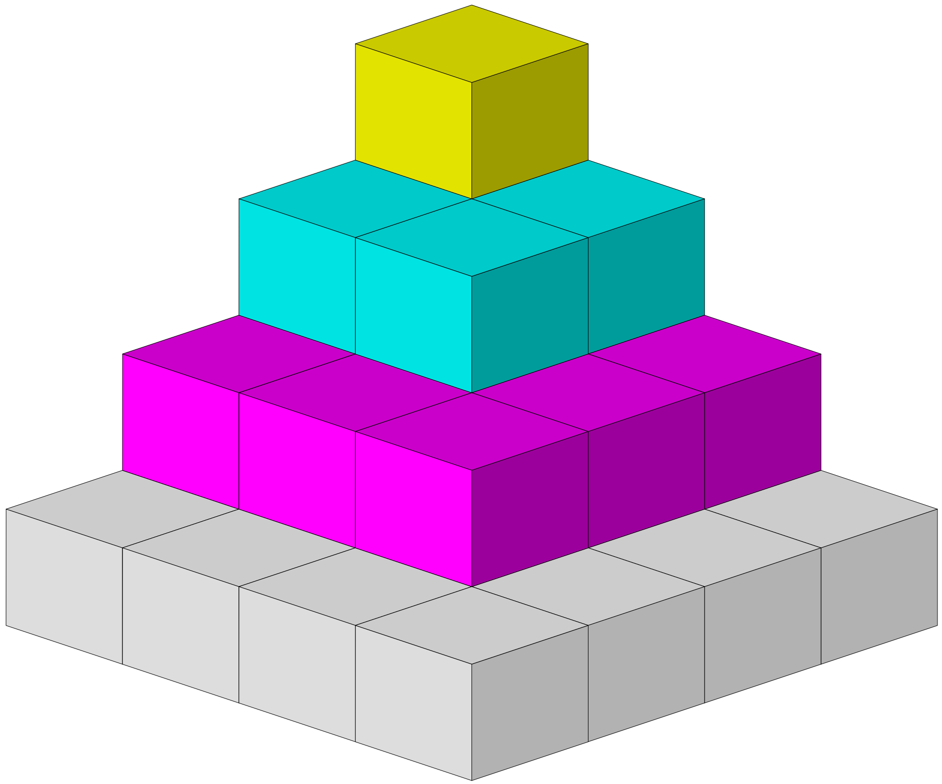
10 layers:	385 cubes
20 layers:	2,870 cubes
30 layers:	9,455 cubes
40 layers:	22,140 cubes
50 layers:	42,925 cubes
60 layers:	≈75,640 cubes



Counting Cubes

- Here's some numbers:

10 layers:	385 cubes
20 layers:	2,870 cubes
30 layers:	9,455 cubes
40 layers:	22,140 cubes
50 layers:	42,925 cubes
60 layers:	73,810 cubes

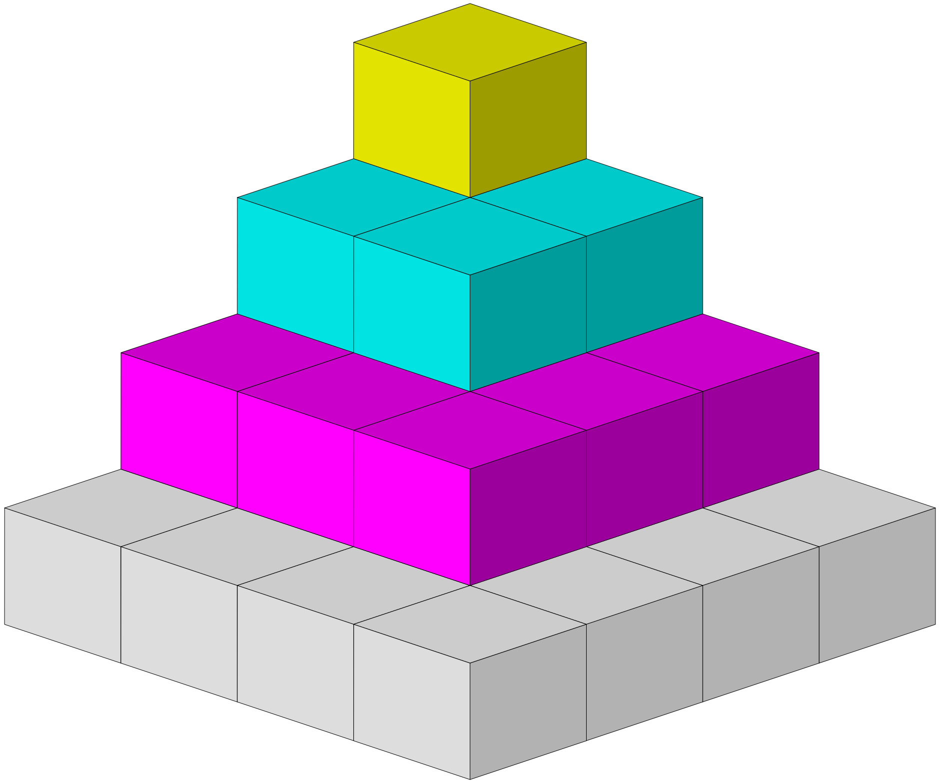


Counting Cubes

- Here's some numbers:

10 layers:	385 cubes
20 layers:	2,870 cubes
30 layers:	9,455 cubes
40 layers:	22,140 cubes
50 layers:	42,925 cubes
60 layers:	73,810 cubes

- Even without an exact formula, we've made a remarkably good prediction!

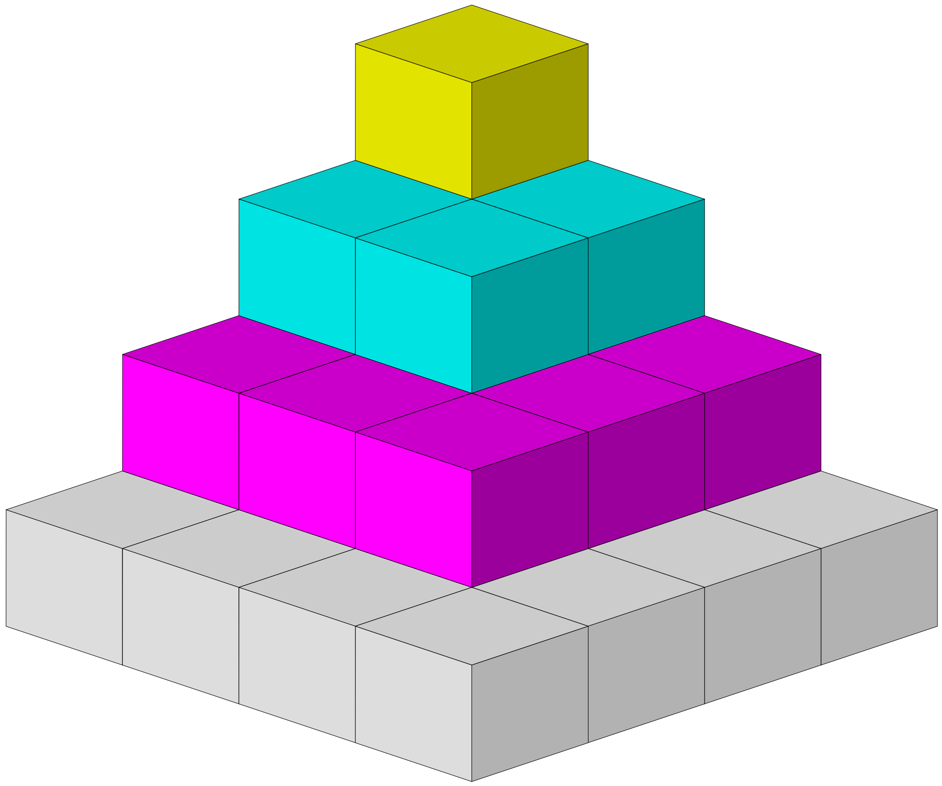


Counting Cubes

- In case you're curious, the exact formula is

$$\frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}.$$

- This quantity is $O(n^3)$ because
  - big-O notation ignores constant factors, and
  - big-O notation ignores lower-order terms.
- We still worked out the big-O growth rate without the exact formula!



Nuances of Big-O Notation

- Big-O notation is designed to capture the ***rate at which a quantity grows***.
- It does not capture information about
  - leading coefficients: the area of a square of side length r and a circle of radius r are each $O(r^2)$.
  - lower-order terms: the functions n, $5n$, and $137n + 42$ are all $O(n)$.
- However, it's still a powerful tool for predicting behavior.

Time-Out for Announcements!

Assignment 4

- Assignment 3 was due today at 1:00PM.
  - Need more time? You have four free “late days” to use over the quarter. You can use up to two of them here.
- Assignment 4 (***Recursion to the Rescue!***) goes out today. It’s due next Friday at 1:00PM and must be completed individually.
  - Play around with recursive problem-solving in realistic situations.
  - Explore the power – and potential pitfalls – of recursive optimization.
- YEAH Hours run today from 4:30PM – 5:30PM just around the corner in Hewlett 101.
- As always, feel free to ask for help when you need it! Ping us on EdStem, stop by the LaIR, or visit our office hours in the Shiny New CoDa Building.

Lecture Participation Opt-Out

- The deadline to opt out of lecture participation and shift the weight to your final exam is ***tonight at 11:59PM***.
- Link is available
  - on the course website in the announcements section,
  - on EdStem on the pinned post, and
  - ***right here***!
- Make sure to get this in by tonight!

Midterm Exam Reminder

- Our midterm exam will be on ***Monday, February 10^th^*** from ***7:00PM – 10:00PM***.
- We will go over more exam logistics this upcoming Monday. Briefly:
  - The exam covers L00 – L09 (basic C++ up through but not including recursive backtracking) and A0 – A3 (debugging through recursion).
  - It’s a traditional sit-down, pencil-and-paper exam.
  - It’s closed-book, closed-computer, and limited-note. You can bring an 8.5” × 11” sheet of notes with you to the exam.
- We’ve posted a huge searchable bank of practice problems to the course website, along with three practice exams made from questions selected from that bank.
- Students with OAE accommodations: You should already have heard from us with details of your alternate exam arrangements. Contact us ***immediately*** if you haven’t.

fg

(The Unix command to resume a program that was paused)

```
void printTrigrams_v1(const string& str) {
    for (int i = 0; i + 3 <= str.length(); i++) {
        string trigram = str.substr(i, 3);
        cout << trigram << endl;
    }
}
```

```
void printTrigrams_v2(const string& str) {
    string s = str;
    while (s.length() >= 3) {
        cout << s[0] << s[1] << s[2] << endl;
        s = s.substr(1);
    }
}
```

Applying Big-O Notation to Code

```
double averageOf(const Vector<int>& vec) {
  double total = 0.0;

  for (int i = 0; i < vec.size(); i++) {
      total += vec[i];
  }

  return total / vec.size();
}
```

Assume any individual statement takes one unit of time to execute. If the input `Vector` has n elements, how many time units will this code take to run?

```
double averageOf(const Vector<int>& vec) {
    double total = 0.0;

    for (int i = 0; i < vec.size(); i++) {
        total += vec[i];
    }

    return total / vec.size();
}
```

1 (double total = 0.0;)

1 (int i = 0;) $n+1$ (i < vec.size();) n (i++)

n (total += vec[i];)

1 (return total / vec.size();)

Assume any individual statement takes one unit of time to execute. If the input `Vector` has n elements, how many time units will this code take to run?

```
double averageOf(const Vector<int>& vec) {
1   double total = 0.0;

                 1              n+1           n
    for (int i = 0; i < vec.size(); i++) {
        total += vec[i];
    }
                   n

    return total / vec.size();  1
}
```

Is this useful?

What does that tell us?

One possible answer: $3n + 4$.

```
double averageOf(const Vector<int>& vec) {
1  double total = 0.0;

              1               n+1          n
   for (int i = 0; i < vec.size(); i++) {
       total += vec[i];
   }          n

   return total / vec.size();
}
```

Doubling the size of the input roughly doubles the runtime.

If we get some data points, we can extrapolate runtimes to good precision.

~~One possible answer: $3n + 4$.~~

More useful answer: **$O(n)$**.

```
void printStars(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << '*' << endl;
        }
    }
}
```

How much time will it take for this code to run, as a function of n? Answer using big-O notation.

```
void printStars(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << '*' << endl;
        }
    }
}
```

How much time will it take for this code to run, as a function of *n*? Answer using big-O notation.

```
void printStars(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            do a fixed amount of work;
        }
    }
}
```

How much time will it take for this code to run, as a function of *n*? Answer using big-O notation.

```
void printStars(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            do a fixed amount of work;
        }
    }
}
```

How much time will it take for this code to run, as a function of *n*? Answer using big-O notation.

```
void printStars(int n) {
    for (int i = 0; i < n; i++) {

        do O(n) units of work;

    }
}
```

How much time will it take for this code to run, as a function of *n*? Answer using big-O notation.

```
void printStars(int n) {
    for (int i = 0; i < n; i++) {

        do O(n) units of work;

    }
}
```

How much time will it take for this code to run, as a function of *n*? Answer using big-O notation.

```
void printStars(int n) {
    do O(n²) units of work;
}
```

How much time will it take for this code to run, as a function of *n*? Answer using big-O notation.

```
void printStars(int n) {
    do O(n²) units of work;
}
```

Answer: **O(n^2)**.

```cpp
void printStars(int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << '*' << endl;
        }
    }
}
```

Answer: **$O(n^2)$**.

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

Answer at
https://cs106b.stanford.edu/pollev

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            do one unit of work;
        }
    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            do one unit of work;
        }
    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {

        do 5n units of work;

    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {

        do O(n) work;

    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {

        do O(n) work;

    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {

    do 2n * O(n) work;

}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {


    do 2n * O(n) work;


}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

As before, big-O ignores any leading coefficients.

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {

    do O(n²) work;

}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

As before, big-O ignores any leading coefficients.

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```cpp
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```cpp
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```cpp
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```cpp
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        do one unit of work;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        do one unit of work;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {

    do 3n units of work;

    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```cpp
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```cpp
void pando(int n) {

    do O(n) units of work;

    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {

    do O(n) units of work;

    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {

    do O(n) units of work;

    for (int i = 0; i < 8; i++) {
        do one unit of work;
    }
}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {

    do O(n) units of work;

    for (int i = 0; i < 8; i++) {
        do one unit of work;
    }
}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {

    do O(n) units of work;


    do 8 units of work;

}
```

How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {

    do O(n) units of work;


    do 8 units of work;

}
```

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
void pando(int n) {

    do O(n) units of work;

}
```

$O(n^2)$

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
```

$O(n^2)$

```
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

$O(n)$

How much time will it take for these functions to run, as a function of n? Answer using big-O notation.

```cpp
void beni(int n) {
    for (int i = 0; i < 2 * n; i++) {
        for (int j = 0; j < 5 * n; j++) {
            cout << '*' << endl;
        }
    }
}
void pando(int n) {
    for (int i = 0; i < 3 * n; i++) {
        cout << "*" << endl;
    }
    for (int i = 0; i < 8; i++) {
        cout << "*" << endl;
    }
}
```

$O(n^2)$

$O(n)$

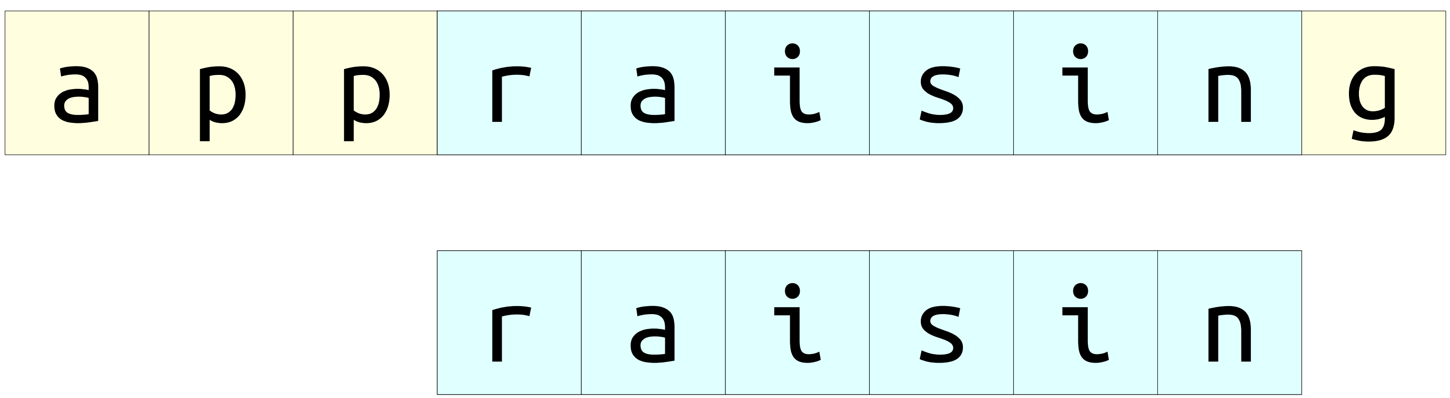
How much time will it take for these functions to run, as a function of *n*? Answer using big-O notation.

```cpp
void printTrigrams_v1(const string& str) {
    for (int i = 0; i + 3 <= str.length(); i++) {
        string trigram = str.substr(i, 3);
        cout << trigram << endl;
    }
}
```

```cpp
void printTrigrams_v2(const string& str) {
    string s = str;
    while (s.length() >= 3) {
        cout << s[0] << s[1] << s[2] << endl;
        s = s.substr(1);
    }
}
```

Computing Substrings

- Constructing a substring of length k takes time $O(k)$.
- Why?



- We need to copy each of the k characters that make up the substring.

```cpp
void printTrigrams_v1(const string& str) {
    for (int i = 0; i + 3 <= str.length(); i++) {
        string trigram = str.substr(i, 3);
        cout << trigram << endl;
    }
}
```

O(n)

```cpp
void printTrigrams_v2(const string& str) {
    string s = str;
    while (s.length() >= 3) {
        cout << s[0] << s[1] << s[2] << endl;
        s = s.substr(1);
    }
}
```

O(n^2)

Recap from Today

- Big-O notation captures the rate at which a quantity grows or scales as the input size increases.
- Big-O notation ignores low-order terms and constant factors.
- "When in doubt, work inside out!" When you see loops, work from the inside out to determine the big-O complexity.

Your Action Items

- ***Read Chapter 10.1 – 10.2.***
  - It’s all about big-O and efficiency, and it’s a great complement to what we covered today.
- ***Read the Guide to Big-O Notation.***
  - It includes a bunch of useful tips that expand upon what we did in lecture today.
- ***Start Assignment 4.***
  - If you want to follow our suggested timetable, aim to complete Win Sum, Lose Sum and Shift Scheduling by this Monday.

Next Time

- ***Sorting Algorithms***
  - How do we get things in order?
- ***Designing Better Algorithms***
  - Using predictions from big-O notation.