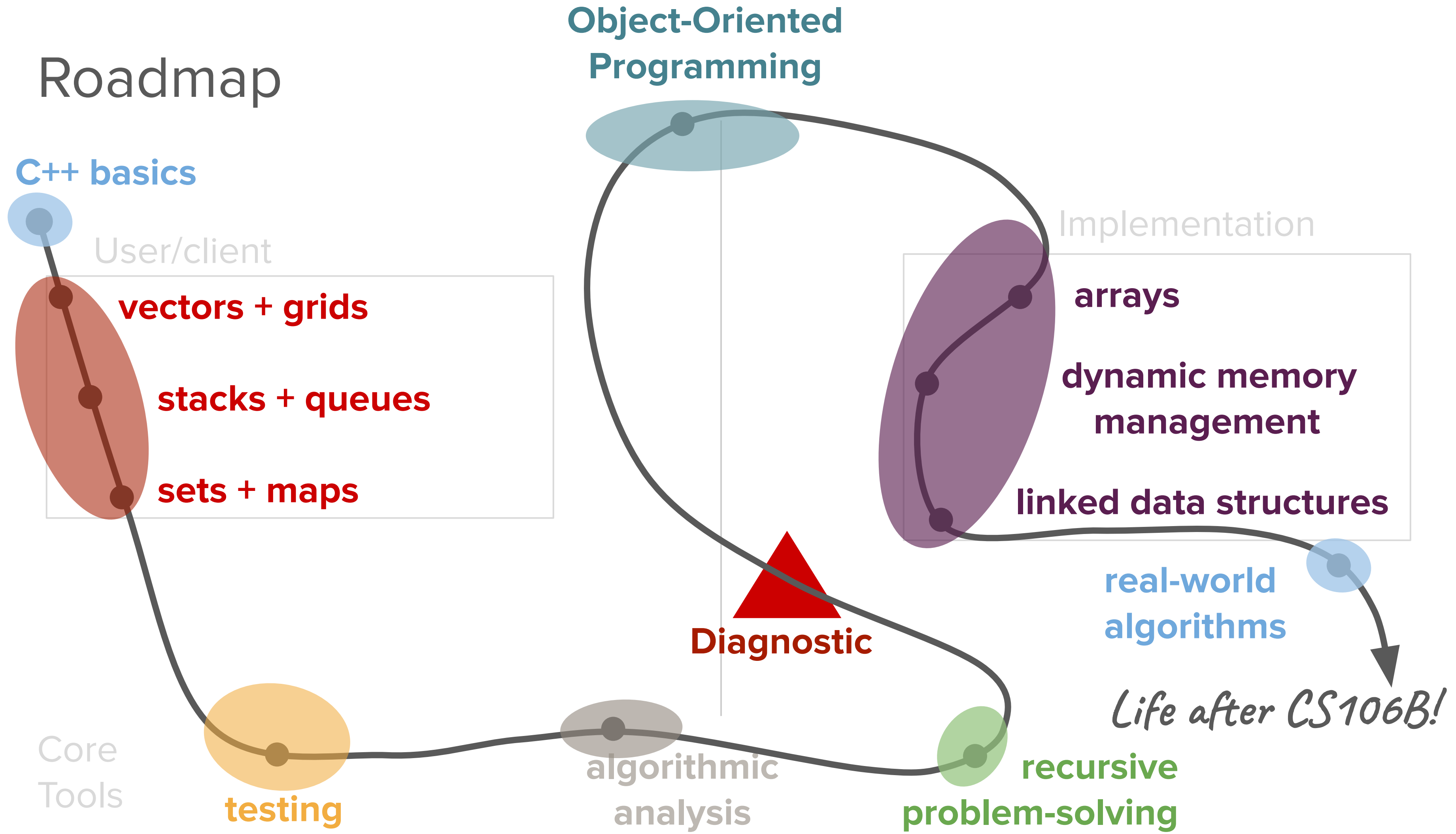
Big-O Notation and Algorithmic Analysis

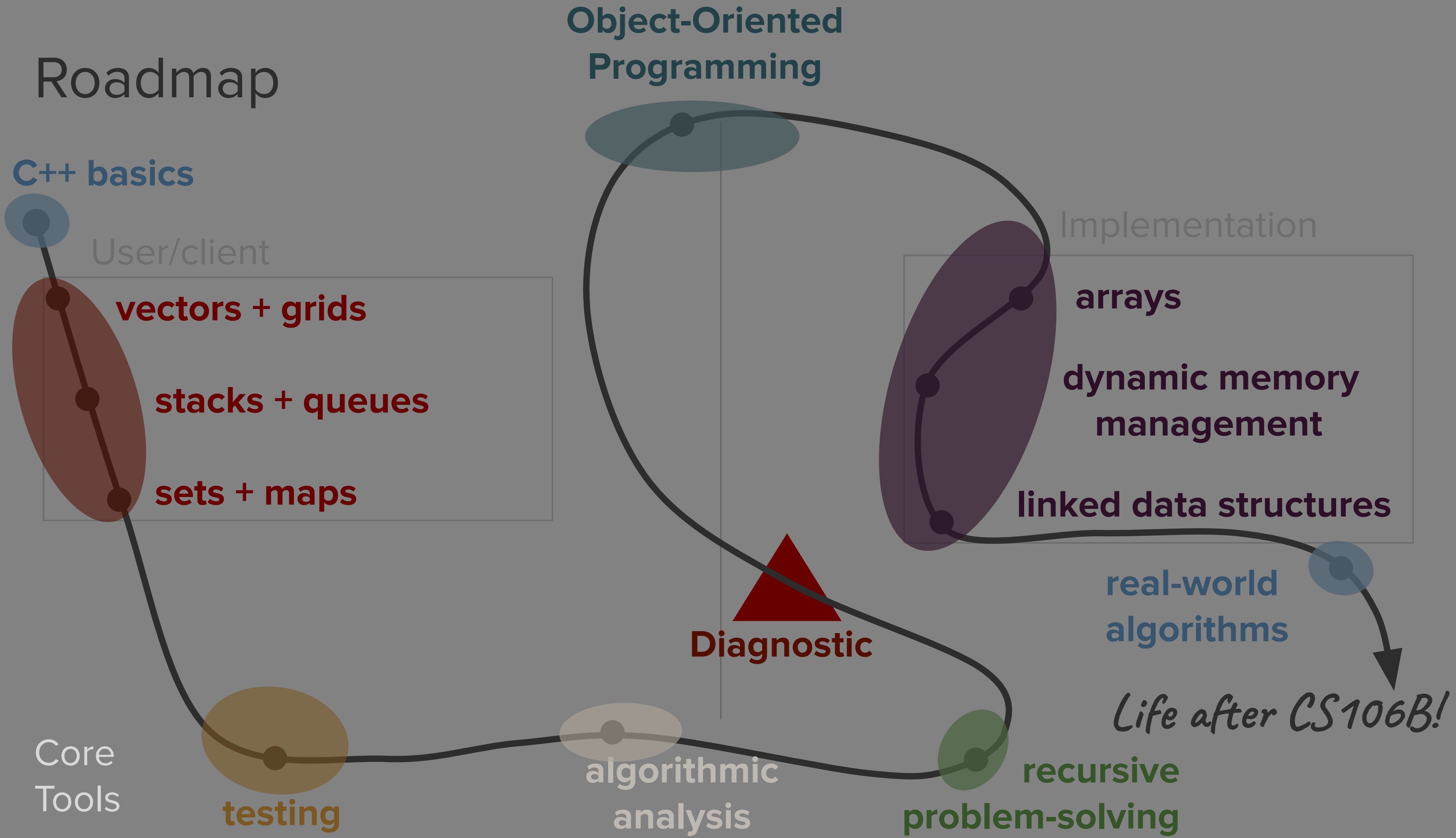
What do you think makes some algorithms "faster" or "better" than others?

(put your answers in the chat)

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Today's question

How can we formalize the notion of efficiency for algorithms?

Today's topics

1. Review
2. Big-O Notation
3. Algorithmic Analysis
4. Beyond Algorithmic Analysis

Review

Ordered ADTs

Elements accessible by indices:

- Vectors (1D)
- Grids (2D)

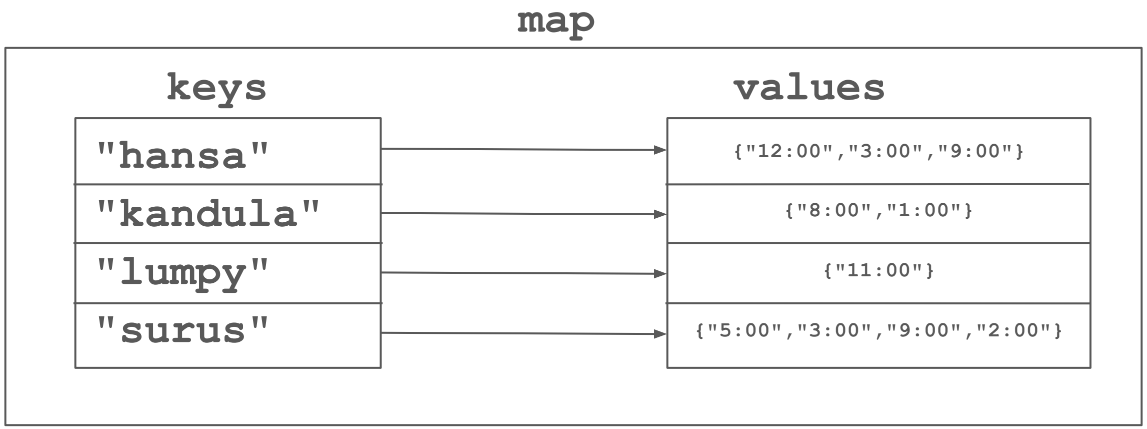
Elements not accessible by indices:

- Queues (FIFO)
- Stacks (LIFO)

Unordered ADTs

- Sets (elements unique)
- Maps (keys unique)

Nested Data Structures



Wonderful diagram and animal naming borrowed from Sonja Johnson-Yu

Breadth-First Search Data Structures

- A data structure to represent (partial word) ladders
  - **`Stack<string>`**
- A data structure to store all the partial word ladders that we have generated so far and have yet to explore
  - **`Queue<Stack<string>>`**
- A data structure to keep track of all the words that we've explored so far, so that we avoid getting stuck in loops
  - **`Set<string>`**

Nested ADTs Summary

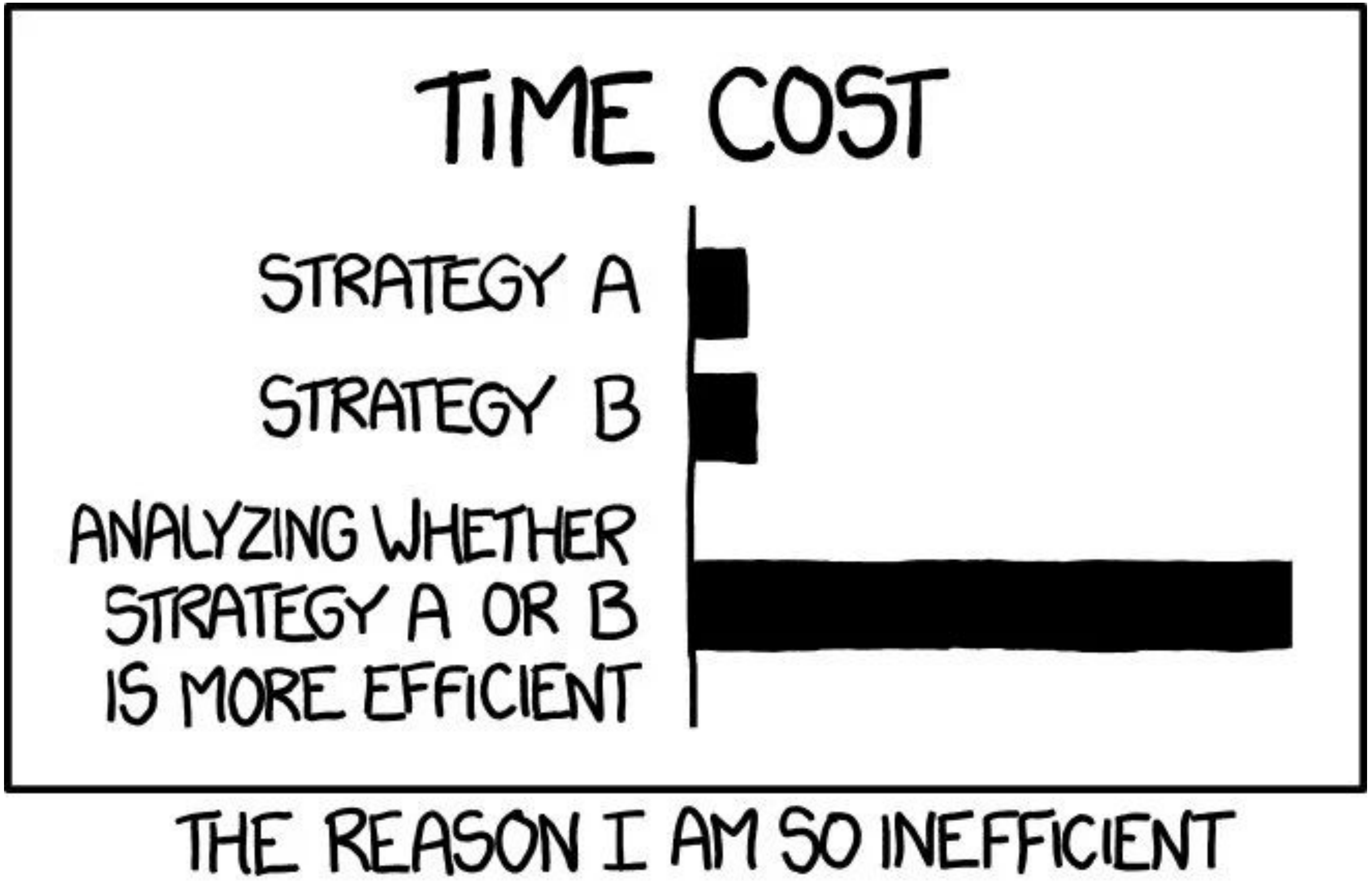
- Powerful
  - Can express highly structured and complex data
  - Used in many real-world systems

- Tricky
  - With increased complexity comes increased cognitive load in differentiating between the levels of information stored at each level of the nesting
  - Specifically in C++, working with nested data structures can be tricky due to the fact that references and copies show up at different points in time. Follow the correct paradigms presented earlier to stay on track!

Note on ADTs (and learning overall)

- We covered six different data structures and their applications over the span of a week, and concluded by implementing BFS using many of these ADTs.
  - This is a lot to take in!
- As Kylie mentioned at the beginning of the quarter, we want to normalize struggle in this class.
  - We cover content very quickly in this class!
  - If you leave lecture feeling you don't understand the algorithm/concept covered that day, don't worry.
  - Lecture is always your first exposure to content – very few people can build deep understanding upon the first exposure
  - The assignments (and section and office hours and LaIR) are your chance to revisit lecture, practice, and really nail down the concepts!
  - Struggling along the way means that you are really ***learning***.

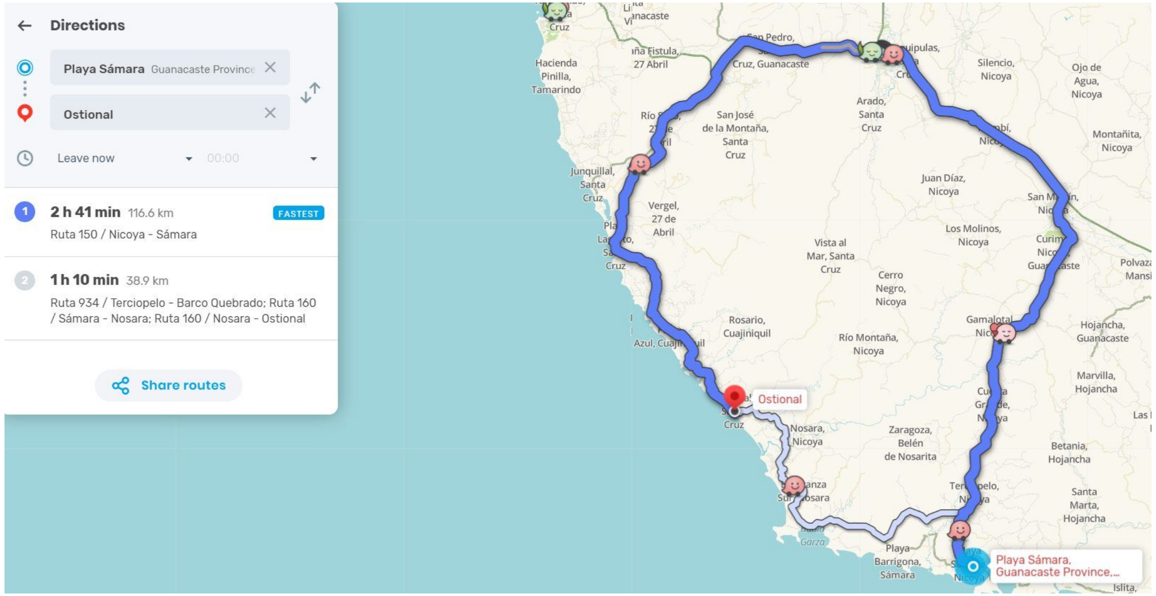
How can we formalize the notion of efficiency for algorithms?


TIME COST
STRATEGY A
STRATEGY B
ANALYZING WHETHER STRATEGY A OR B IS MORE EFFICIENT
THE REASON I AM SO INEFFICIENT

Source

Why do we care about efficiency?

- Implementing inefficient algorithms may make solving certain tasks impossible, even with unlimited resources



Directions

Playa Sámara Guanacaste Province

Ostional

Leave now 00:00

1 **2 h 41 min** 116.6 km FASTEST
Ruta 150 / Nicoya - Sámara

2 **1 h 10 min** 38.9 km
Ruta 934 / Terciopelo - Barco Quebrado; Ruta 160 / Sámara - Nosara; Ruta 160 / Nosara - Ostional

Share routes

Why do we care about efficiency?

- Implementing inefficient algorithms may make solving certain tasks impossible, even with unlimited resources

- Implementing efficient algorithms allows us to solve important problems, often with limited resources available

Why do we care about efficiency?

- Implementing inefficient algorithms may make solving certain tasks impossible, even with unlimited resources

- Implementing efficient algorithms allows us to solve important problems, often with limited resources available

- If we can quantify the efficiency of an algorithm, we can understand and predict its behavior when we apply it to unseen problems

Assignment 1 Redux

- In Assignment 1, you implemented three different algorithms for finding perfect numbers

Assignment 1 Redux

- In Assignment 1, you implemented three different algorithms for finding perfect numbers
  - Exhaustive Search
    - Runtime predictions to find 5th perfect number: Anywhere from 25-100+ days

Assignment 1 Redux

- In Assignment 1, you implemented three different algorithms for finding perfect numbers
    - Exhaustive Search
        - Runtime predictions to find 5th perfect number: Anywhere from 25-100+ days
    - Smarter Search
        - Runtime predictions to find 5th perfect number: Anywhere from a couple minutes to 1 hour

Assignment 1 Redux

- In Assignment 1, you implemented three different algorithms for finding perfect numbers
    - Exhaustive Search
        - Runtime predictions to find 5th perfect number: Anywhere from 25-100+ days
    - Smarter Search
        - Runtime predictions to find 5th perfect number: Anywhere from a couple minutes to 1 hour
    - Euclid's Algorithm
        - Actual runtime to find 5th perfect number: Less than a second!

Assignment 1 Redux

- In Assignment 1, you implemented three different algorithms for finding perfect numbers
  - Exhaustive Search
    - Runtime predictions to find 5th perfect number: Anywhere from 25-100+ days
  - Smarter Search
    - Runtime predictions to find 5th perfect number: Anywhere from a couple minutes to 1 hour
  - Euclid's Algorithm
    - Actual runtime to find 5th perfect number: Less than a second!
- Core idea: Although each individual experienced dramatically different real runtimes for these three algorithms, there is a clear distinction here between "fast"/"efficient" and "slow"/"inefficient" algorithms

Estimating Quantities

[polls]

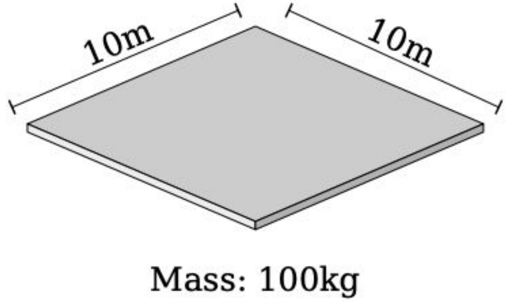
Leveraging Intuition

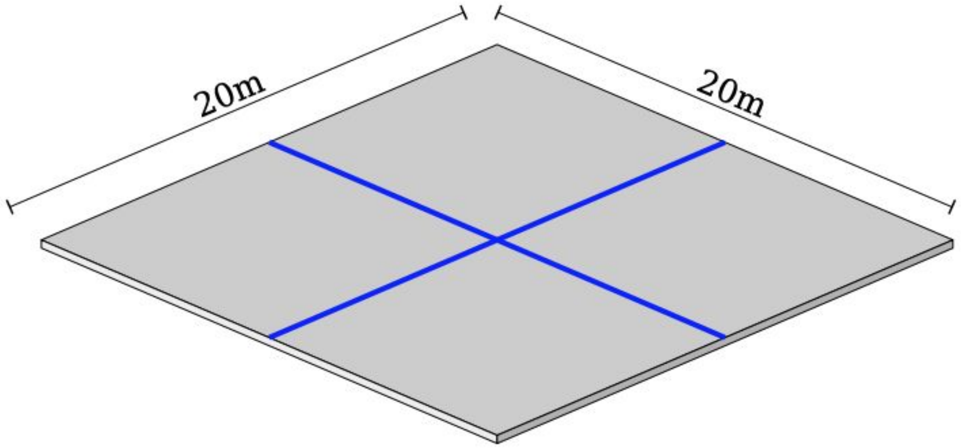
- Today's activity is going to look a little bit different than usual. There's no code, no pseudocode, and nothing that resembles C++.

- Instead, you're going to be presented with a set of 4 scenarios, where you have two similar items of different magnitudes, one small and one larger. You know the exact magnitude of the smaller item – can you predict what the magnitude of the larger item will be based on the intuitive visual relationship?

- We'll collect your response to all 4 polls first, and then we'll walk through the answers to the exercises. Remember that these are ***guesses*** based on your intuition – don't try to do any complex calculations!

Example 1



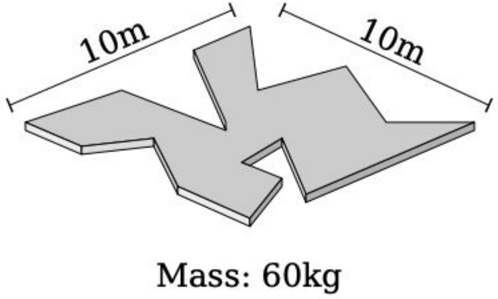


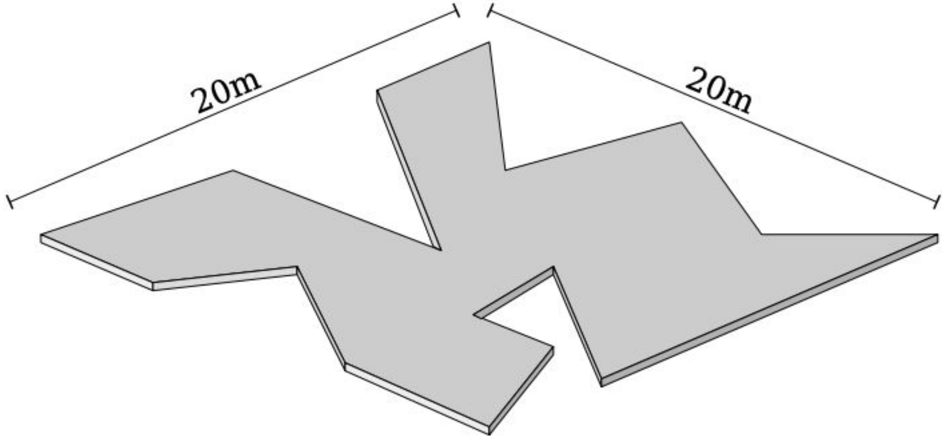
These two square plates are made of the same material.

They have the same thickness.

What's your best guess for the mass of the second square?

Example 2



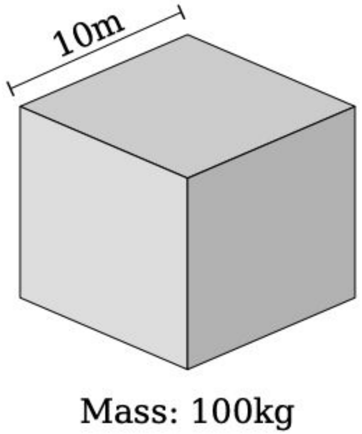


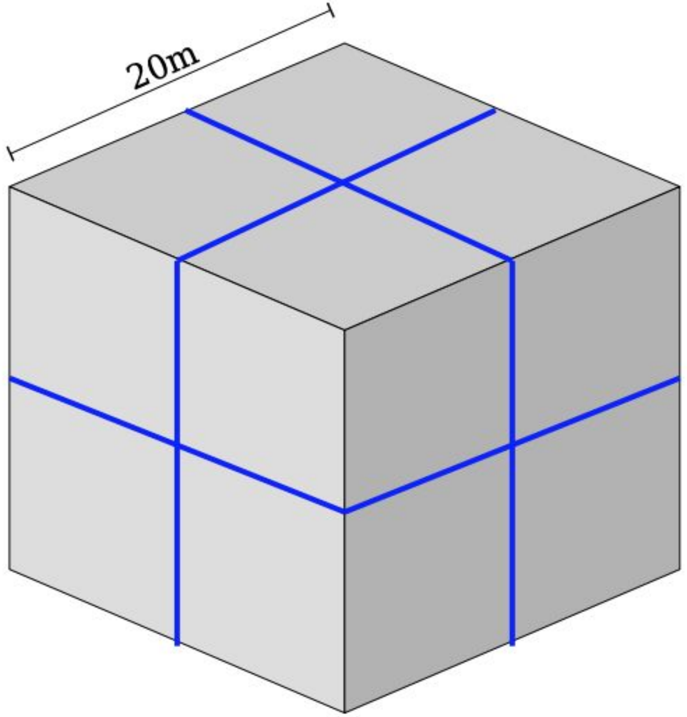
These two square plates are made of the same material.

They have the same thickness.

What's your best guess for the mass of the second square?

Example 3

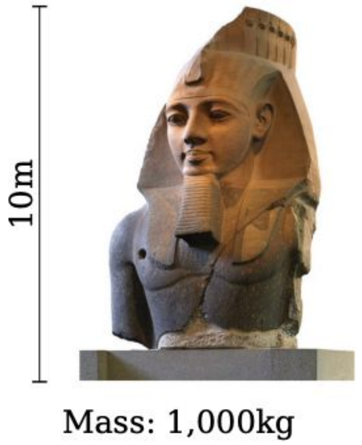


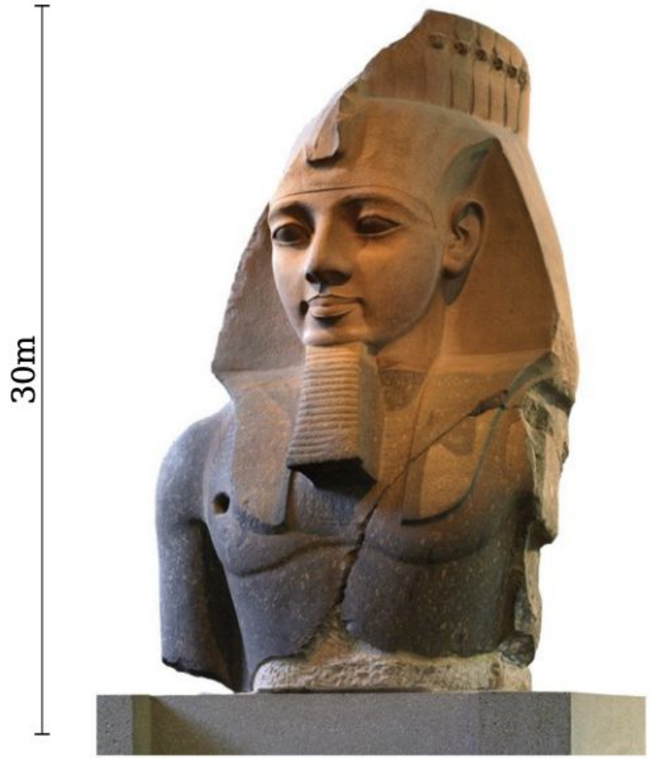


These two cubes are made of the same material.

What's your best guess for the mass of the second cube?

Example 4





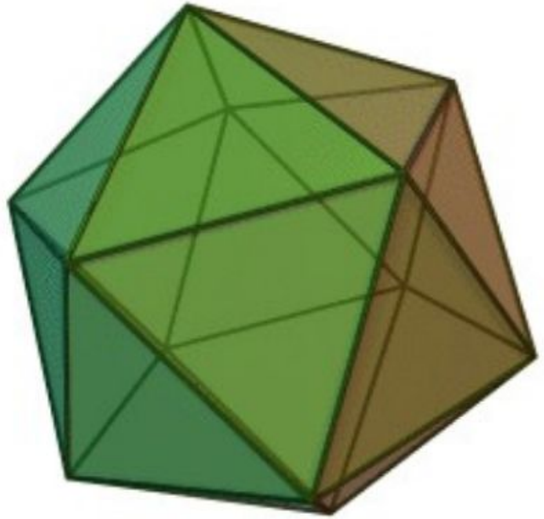
These two statues are made of the same material.

What’s your best guess for the mass of the second statue?

Example 5

All sides of each triangle are 10m long.

Paint required: 90L

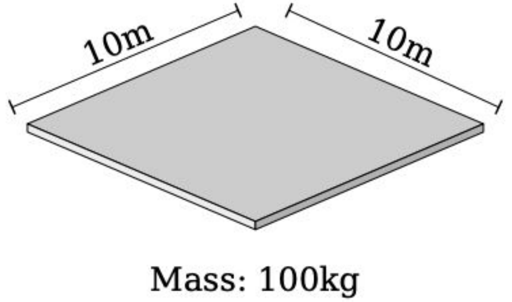


All sides of each triangle are 40m long.

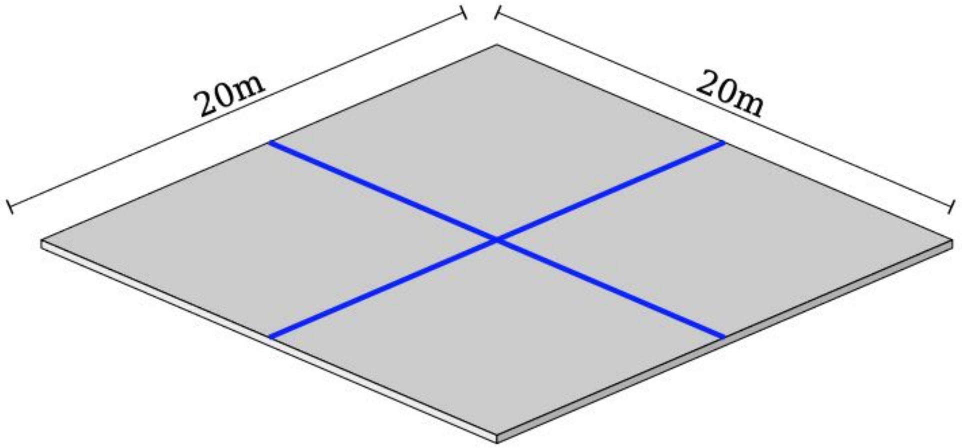
How much paint is needed to paint the surface of the larger icosahedron?

Answers

Example 1



Mass is about 400kg (4 smaller squares make up the larger square)

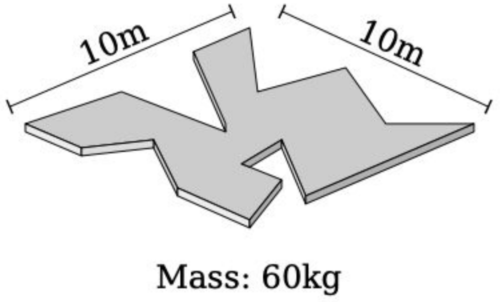


These two square plates are made of the same material.

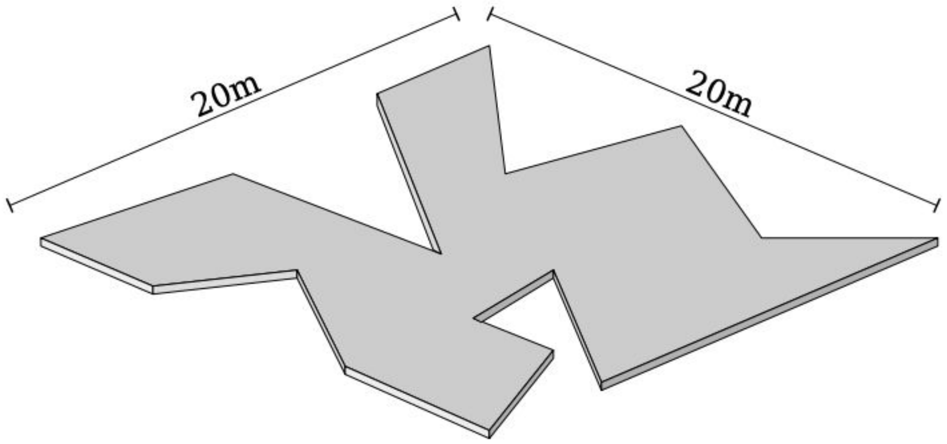
They have the same thickness.

What's your best guess for the mass of the second square?

Example 2



Mass is about 240kg (side length is doubled, overall are increases by factor of 4)



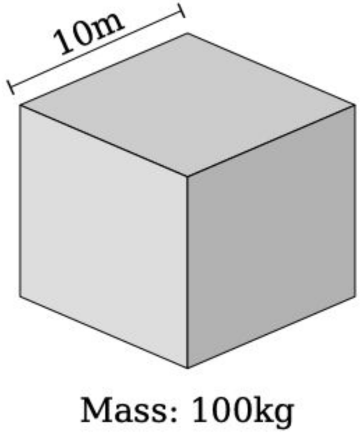
These two square plates are made of the same material.

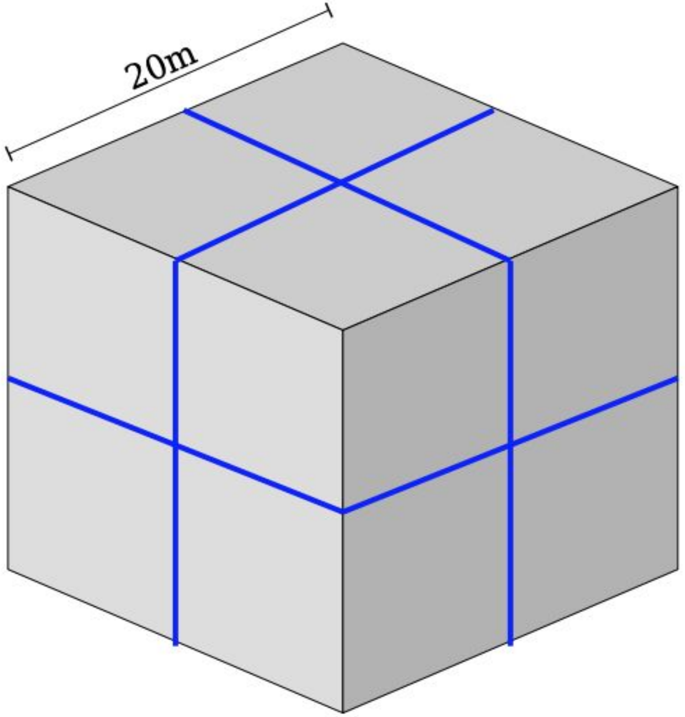
They have the same thickness.

What's your best guess for the mass of the second square?

Example 3

Mass is about 800kg (8 smaller cubes make up the larger cube)



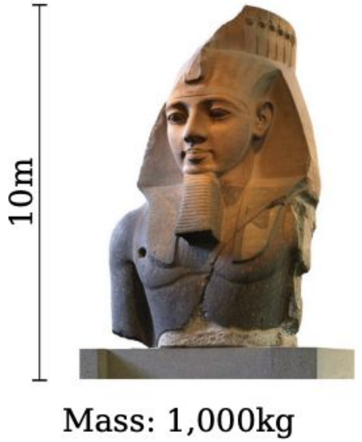


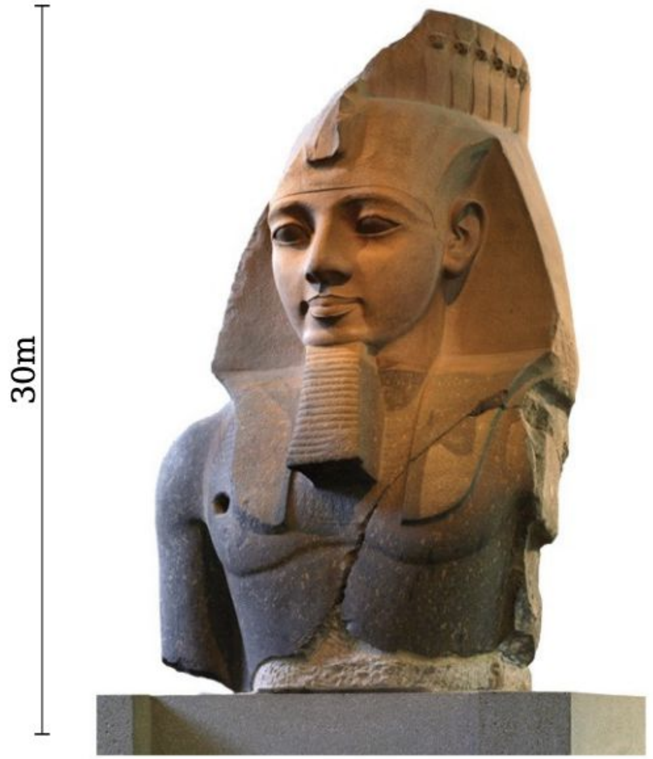
These two cubes are made of the same material.

What's your best guess for the mass of the second cube?

Example 4

Mass is about 27000kg (statue dimensions increased by factor of 3, and volume increases by factor of 27)





These two statues are made of the same material.

What's your best guess for the mass of the second statue?

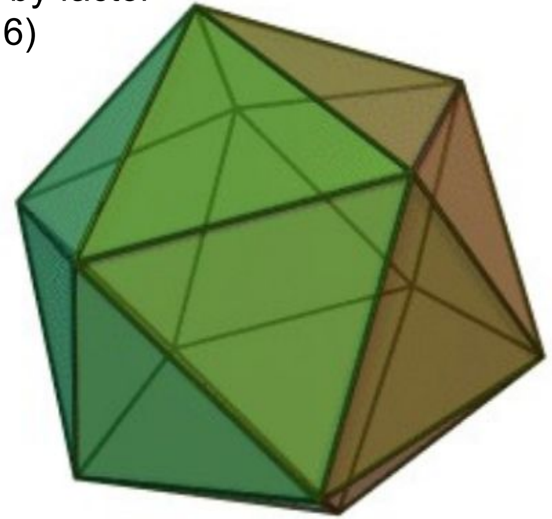
Example 5

Paint Required is about 1440L (side length grows by factor of 4, area increases by factor of 4^2 = 16)

How much paint is needed to paint the surface of the larger icosahedron?

All sides of each triangle are 10m long.

Paint required: 90L



All sides of each triangle are 40m long.

Key Takeaway

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

- **Big-O notation** is a way of quantifying the rate at which some quantity grows.

Big-O Notation

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

Big-O Notation

- **Big-O notation** is a way of quantifying the rate at which some quantity grows.
- Example:
  - A square of side length `r` has area `O(r²)`.

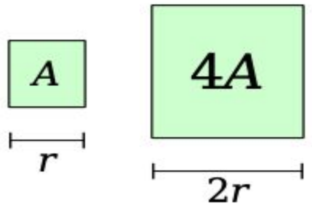
The "O" stands for "on the order of", which is a growth prediction, not an exact formula

Big-O Notation

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

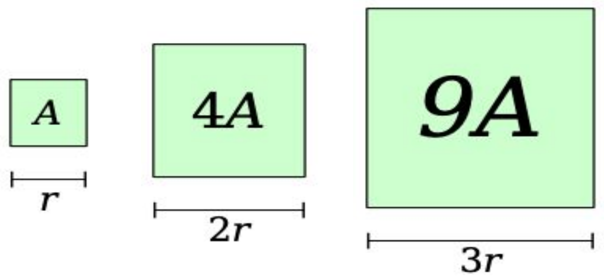
Big-O Notation

- **Big-O notation** is a way of quantifying the rate at which some quantity grows.
- Example:
  - A square of side length `r` has area `O(r²)`.



Big-O Notation

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

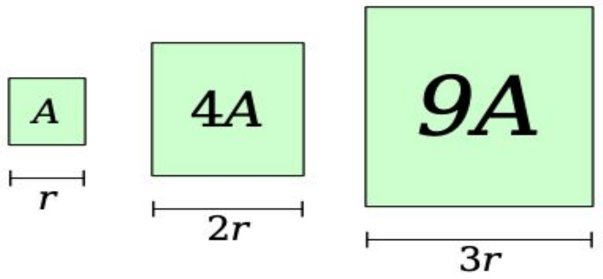


Doubling r increases area 4x

Tripling r increases area 9x

Big-O Notation

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

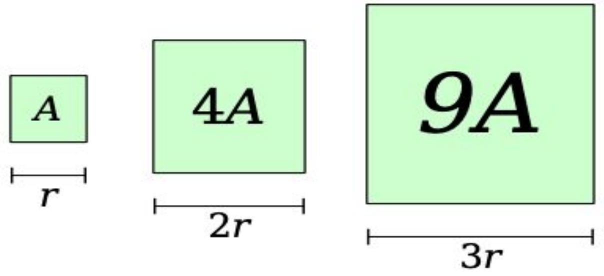


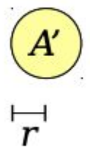
Doubling r increases area 4x

Tripling r increases area 9x

Big-O Notation

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



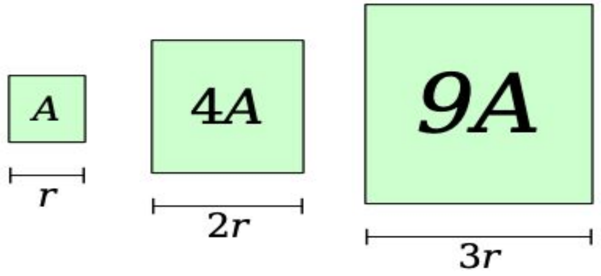


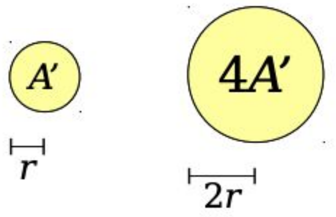
Doubling r increases area 4x

Tripling r increases area 9x

Big-O Notation

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



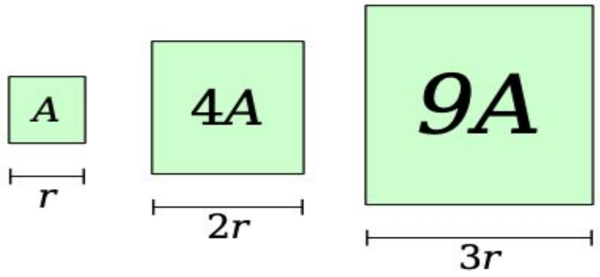


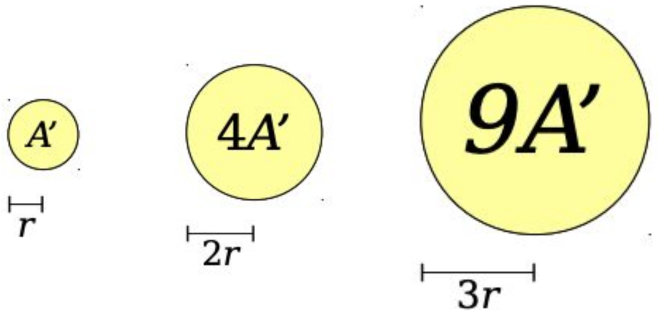
Doubling r increases area 4x

Tripling r increases area 9x

Big-O Notation

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



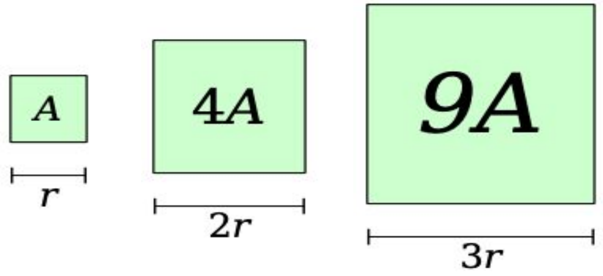


Doubling r increases area 4x

Tripling r increases area 9x

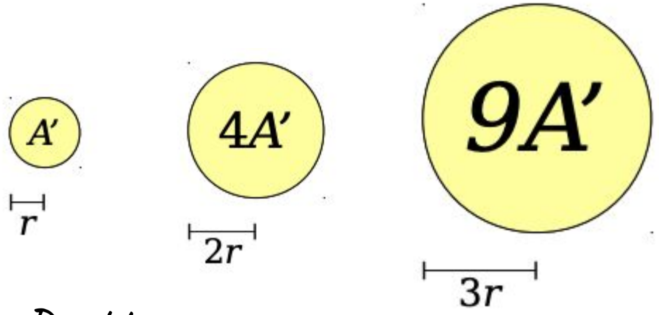
Big-O Notation

- **Big-O notation** is a way of quantifying the rate at which some quantity grows.
- 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 4x

Tripling r increases area 9x



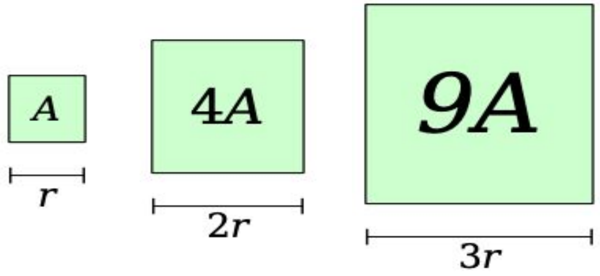
Doubling r increases area 4x

Tripling r increases area 9x

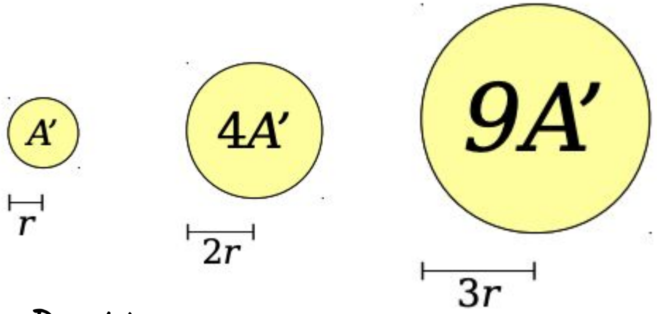
Big-O Notation

- **Big-O notation** is a way of quantifying the rate at which some quantity grows.
- Example:
  - A square of side length `r` has area `O(r²)`.
  - A circle of radius `r` has area `O(r²)`.

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



Doubling r increases area 4x
Tripling r increases area 9x



Doubling r increases area 4x
Tripling r increases area 9x

Big-O in the Real World

Big-O Example: Network Value

- Metcalfe’s Law
    - The value of a communications network with **`n`** users is **`O(n`**^{2}**`)`**.

Big-O Example: Network Value

- Metcalfe's Law
  - The value of a communications network with `n` users is `O(n²)`.
- 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.

1. 10,000,000
2. 50,000,000
3. 100,000,000
4. 1,000,000,000

Big-O Example: Network Value

- Metcalfe's Law
  - 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.

Big-O Example: Cell Size

- Question: Why are cells tiny?

Big-O Example: Cell Size

- Question: Why are cells tiny?
- Assumption: Cells are spheres

Big-O Example: Cell Size

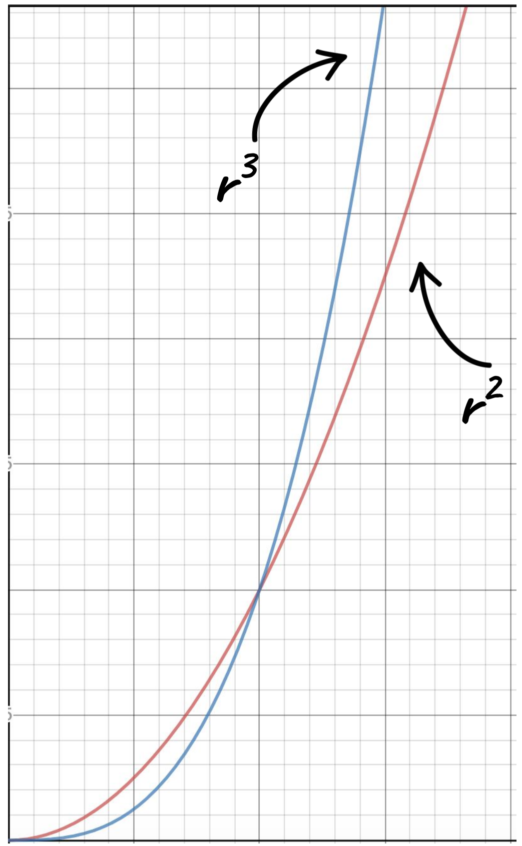
- Question: Why are cells tiny?
- Assumption: Cells are spheres
- A cell absorbs nutrients from its environment through its surface area.
    - Surface area of the cell: **$O(r^2)$**

Big-O Example: Cell Size

- Question: Why are cells tiny?
- Assumption: Cells are spheres
- A cell absorbs nutrients from its environment through its surface area.
  - Surface area of the cell: $O(r^2)$
- A cell needs to provide nutrients all throughout its volume
  - Volume of the cell: $O(r^3)$

Big-O Example: Cell Size

- Question: Why are cells tiny?
- Assumption: Cells are spheres
- A cell absorbs nutrients from its environment through its surface area.
    - Surface area of the cell: `O(r`^{2}`)`
- A cell needs to provide nutrients all throughout its volume
    - Volume of the cell: `O(r`^{3}`)`
- As a cell gets bigger, its resource *intake* grows slower than its resource *consumption*, so each part of the cell gets less energy.



Big-O Example: Manufacturing

- You’re working at a company producing cat toys. It costs you some amount of money to produce a cat toy, and there was some one-time cost to set up the factory.
- What data would you need to gather to estimate the cost of producing ten million cat toys?

Big-O Example: Manufacturing

- You're working at a company producing cat toys. It costs you some amount of money to produce a cat toy, and there was some one-time cost to set up the factory.
- What data would you need to gather to estimate the cost of producing ten million cat toys?

```
Cost(n) = n × costPerToy + startupCost
```

Big-O Example: Manufacturing

- You’re working at a company producing cat toys. It costs you some amount of money to produce a cat toy, and there was some one-time cost to set up the factory.
- What data would you need to gather to estimate the cost of producing ten million cat toys?

This term grows as a function of n

$$\texttt{Cost(n)} = \overset{\downarrow}{\texttt{n} \times \texttt{costPerToy}} + \texttt{startupCost}$$

Big-O Example: Manufacturing

- You’re working at a company producing cat toys. It costs you some amount of money to produce a cat toy, and there was some one-time cost to set up the factory.
- What data would you need to gather to estimate the cost of producing ten million cat toys?

This term grows as a function of n

This term does not grow

$$\mathtt{Cost(n)} = \underbrace{\mathtt{n} \times \mathtt{costPerToy}} + \underbrace{\mathtt{startupCost}}$$

Big-O Example: Manufacturing

- You’re working at a company producing cat toys. It costs you some amount of money to produce a cat toy, and there was some one-time cost to set up the factory.
- What data would you need to gather to estimate the cost of producing ten million cat toys?

$$\begin{aligned}\texttt{Cost(n)} &= \overbrace{\texttt{n} \times \texttt{costPerToy}}^{\text{This term grows as a function of } n} + \overbrace{\texttt{startupCost}}^{\text{This term does not grow}} \\ &= \texttt{O(n)}\end{aligned}$$

Nuances of Big-O

- 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 and a circle are both O(r^2).
  - lower-order terms: there may be other factors contributing to growth that get glossed over.

- However, it's still a **very powerful tool for predicting behavior.**

Announcements

Announcements

- Assignment 2 was released this past Friday afternoon. It will be due on **Friday, July 9 at 11:59pm PDT**.
  - YEAH hours: Happened yesterday, recorded and posted on Canvas.
  - This assignment is a step-up in complexity compared to A1 – make sure to get started ASAP if you haven't already.
- If you haven't already, come visit Kylie and me at our office hours!
- Fill out the End-of-Quarter Ethics Lecture Survey (more info coming at the end of this lecture).
- Please remember to prioritize Ed for conceptual questions and limit private posts with code!
- Revisions for Assignment 1 available until end-of-day Monday.

Analyzing Code

Analyzing Code

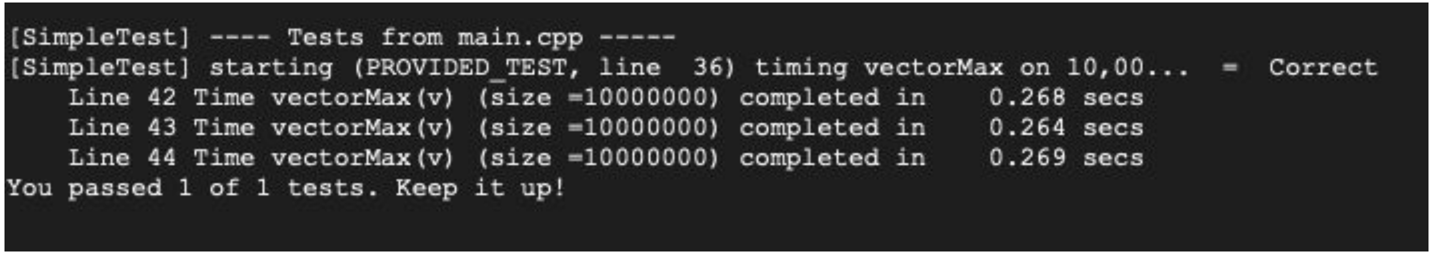
How can we apply Big-O to computer science?

Why runtime isn't enough

- What is runtime?
  - Runtime is simply the amount of real time it takes for a program to run

Why runtime isn’t enough

- What is runtime?
    - Runtime is simply the amount of real time it takes for a program to run

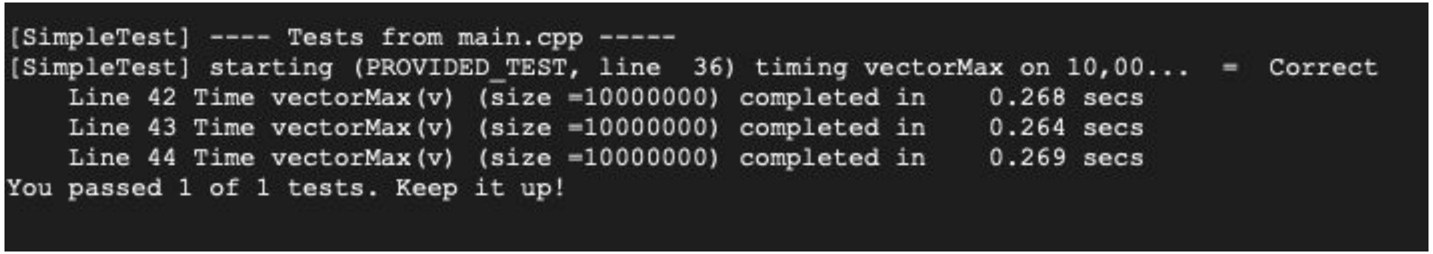


```
[SimpleTest] ---- Tests from main.cpp -----
[SimpleTest] starting (PROVIDED_TEST, line  36) timing vectorMax on 10,00...  =  Correct
    Line 42 Time vectorMax(v) (size =10000000) completed in    0.268 secs
    Line 43 Time vectorMax(v) (size =10000000) completed in    0.264 secs
    Line 44 Time vectorMax(v) (size =10000000) completed in    0.269 secs
You passed 1 of 1 tests. Keep it up!
```

Nick's 2012
MacBook

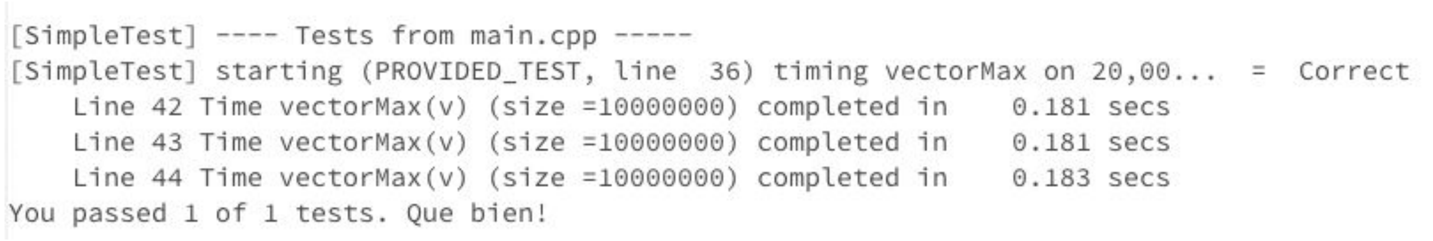
Why runtime isn't enough

- What is runtime?
    - Runtime is simply the amount of real time it takes for a program to run



```
[SimpleTest] ---- Tests from main.cpp -----
[SimpleTest] starting (PROVIDED_TEST, line  36) timing vectorMax on 10,00...  =  Correct
    Line 42 Time vectorMax(v) (size =10000000) completed in    0.268 secs
    Line 43 Time vectorMax(v) (size =10000000) completed in    0.264 secs
    Line 44 Time vectorMax(v) (size =10000000) completed in    0.269 secs
You passed 1 of 1 tests. Keep it up!
```

Nick's 2012 MacBook



```
[SimpleTest] ---- Tests from main.cpp -----
[SimpleTest] starting (PROVIDED_TEST, line  36) timing vectorMax on 20,00...  =  Correct
    Line 42 Time vectorMax(v) (size =10000000) completed in    0.181 secs
    Line 43 Time vectorMax(v) (size =10000000) completed in    0.181 secs
    Line 44 Time vectorMax(v) (size =10000000) completed in    0.183 secs
You passed 1 of 1 tests. Que bien!
```

Ed's powerful computers

Why runtime isn’t enough

- Measuring wall-clock runtime is less than ideal, since
    - It depends on what computer you’re using,
    - What else is running on that computer,
    - Whether that computer is conserving power,
    - Etc.

Why runtime isn’t enough

- Measuring wall-clock runtime is less than ideal, since
  - It depends on what computer you’re using,
  - What else is running on that computer,
  - Whether that computer is conserving power,
  - Etc.

- Worse, **individual runtimes can’t predict future runtimes**.

Why runtime isn’t enough

- Measuring wall-clock runtime is less than ideal, since
  - It depends on what computer you’re using,
  - What else is running on that computer,
  - Whether that computer is conserving power,
  - Etc.

- Worse, **individual runtimes can’t predict future runtimes**.

- Let's develop a computer-independent efficiency metric using big-O!

Analyzing Code: **`vectorMax()`**

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
 }
```

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
 }
```

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?*

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
}
```

Total time based on # of repetitions

1 time unit

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
}
```

Total time based on # of repetitions

1 time unit

1 time unit

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
}
```

Total time based on # of repetitions

1 time unit

1 time unit

1 time unit

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
}
```

Total time based on # of repetitions

1 time unit

1 time unit

1 time unit

N time units

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
}
```

Total time based on # of repetitions

1 time unit
1 time unit
1 time unit
N time units
N-1 time units

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
 }
```

Total time based on # of repetitions

1 time unit
1 time unit
1 time unit
N time units
N-1 time units
N-1 time units

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
}
```

Total time based on # of repetitions

1 time unit
1 time unit
1 time unit
N time units
N-1 time units
N-1 time units
(up to) N-1 time units

vectorMax()

Total time based on # of repetitions

`int vectorMax(Vector<int> &v) {`	
`  int currentMax = v[0];`	1 time unit
`  int n = v.size();`	1 time unit
`  for (int i = 1; i < n; i++) {`	1 time unit
	N time units
`    if (currentMax < v[i]) {`	N-1 time units
`      currentMax = v[i];`	N-1 time units
`    }`	(up to) N-1 time units
`  }`	
`  return currentMax;`	1 time unit
`}`	

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
 }
```

Total amount of time

$4N + 1$

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
 }
```

Total amount of time

$$4N + 1$$

Is this useful?

What does this tell us?

vectorMax()

```
int vectorMax(Vector<int> &v) {
  int currentMax = v[0];
  int n = v.size();
  for (int i = 1; i < n; i++) {
      if (currentMax < v[i]) {
          currentMax = v[i];
      }
  }
  return currentMax;
}
```

Total amount of time

O(n)

More practical: Doubling the size of the input roughly doubles the runtime. Therefore, the input and runtime have a linear (O(n)) relationship.

Analyzing Code: **`printStars()`**

printStars()

```
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.

printStars()

```
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.

printStars()

```
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.

printStars()

```
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.

printStars()

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

        // do O(n) time units of work

    }
}
```

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

printStars()

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

        // do O(n) time units of work

    }
}
```

How much time will it take for this code to run, as a function of n?

Answer using big-O notation.

printStars()

```
void printStars(int n) {

    // do O(n²) time units of work

}
```

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

printStars()

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

$O(n^2)$

A final analyzing code example

hmmThatsStrange()

```
void hmmThatsStrange(int n) {
    cout << "Mirth and Whimsy" << n << endl;
}
```

The runtime is completely independent of the value **n**.

hmmThatsStrange()

```
void hmmThatsStrange(int n) {
    cout << "Mirth and Whimsy" << n << endl;
}
```

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

hmmThatsStrange()

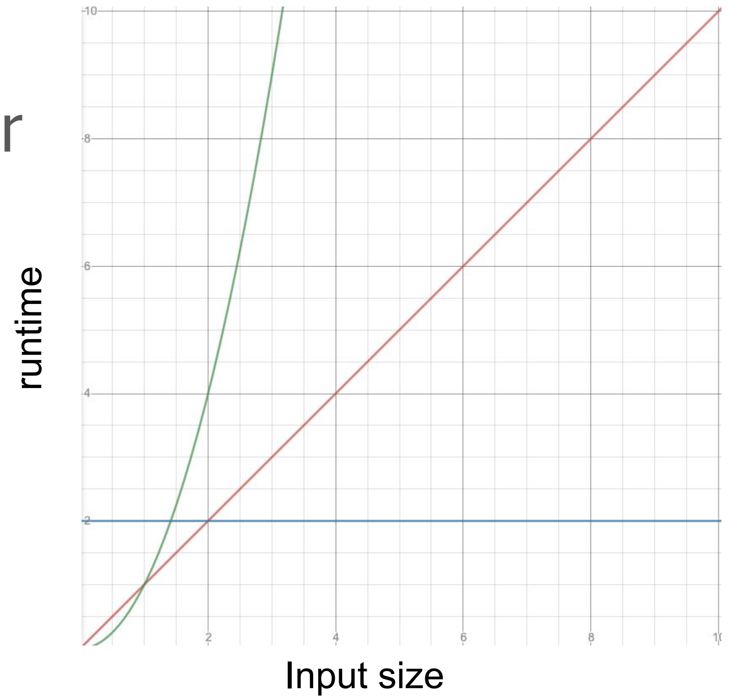
```
void hmmThatsStrange(int n) {
    cout << "Mirth and Whimsy" << n << endl;
}
```

O(1)

Applying Big-O to ADTs

Efficiency Categorizations So Far

- Constant Time – O(1)
  - Super fast, this is the best we can hope for!
  - Euclid's Algorithm for Perfect Numbers
- Linear Time – O(n)
  - This is okay, we can live with this
- Quadratic Time – O(n^2)
  - This can start to slow down really quickly
  - Exhaustive Search for Perfect Numbers
- How do all the ADT operations we've seen so far fall into these categories?



ADT Big-O Matrix

ADT Big-O Matrix

- Vectors
  - `.size() - O(1)`
  - `.add() - O(1)`
  - `v[i] - O(1)`
  - `.insert() - O(n)`
  - `.remove() - O(n)`
  - `.sublist() - O(n)`
  - `traversal - O(n)`
- Grids
  - `.numRows()/.numCols() - O(1)`
  - `g[i][j] - O(1)`
  - `.inBounds() - O(1)`
  - **traversal - O(n^2)**

ADT Big-O Matrix

- Vectors
  - `.size() - O(1)`
  - `.add() - O(1)`
  - `v[i] - O(1)`
  - `.insert() - O(n)`
  - `.remove() - O(n)`
  - `.sublist() - O(n)`
  - `traversal - O(n)`
- Grids
  - `.numRows()/.numCols() - O(1)`
  - `g[i][j] - O(1)`
  - `.inBounds() - O(1)`
  - **traversal - O(n^2)**
- Queues
  - `.size() - O(1)`
  - `.peek() - O(1)`
  - `.enqueue() - O(1)`
  - `.dequeue() - O(1)`
  - `.isEmpty() - O(1)`
  - `traversal - O(n)`
- Stacks
  - `.size() - O(1)`
  - `.peek() - O(1)`
  - `.push() - O(1)`
  - `.pop() - O(1)`
  - `.isEmpty() - O(1)`
  - `traversal - O(n)`

ADT Big-O Matrix

- Vectors
  - `.size() – O(1)`
  - `.add() – O(1)`
  - `v[i] – O(1)`
  - `.insert() – O(n)`
  - `.remove() – O(n)`
  - `.sublist() – O(n)`
  - `traversal – O(n)`
- Grids
  - `.numRows()/.numCols() – O(1)`
  - `g[i][j] – O(1)`
  - `.inBounds() – O(1)`
  - `traversal` – $O(n^2)$
- Queues
  - `.size() – O(1)`
  - `.peek() – O(1)`
  - `.enqueue() – O(1)`
  - `.dequeue() – O(1)`
  - `.isEmpty() – O(1)`
  - `traversal – O(n)`
- Stacks
  - `.size() – O(1)`
  - `.peek() – O(1)`
  - `.push() – O(1)`
  - `.pop() – O(1)`
  - `.isEmpty() – O(1)`
  - `traversal – O(n)`
- Sets
  - `.size() – O(1)`
  - `.isEmpty() – O(1)`
  - `.add() – ???`
  - `.remove() – ???`
  - `.contains() – ???`
  - `traversal – O(n)`
- Maps
  - `.size() – O(1)`
  - `.isEmpty() – O(1)`
  - `m[key] – ???`
  - `.contains() – ???`
  - `traversal – O(n)`

ADT Big-O Matrix

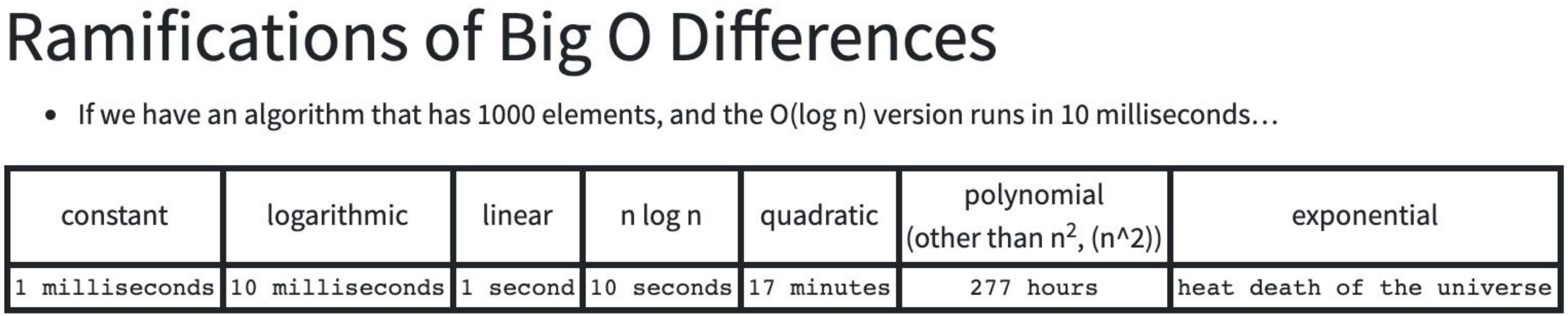
- Vectors
  - `.size()` – O(1)
  - `.add()` – O(1)
  - `v[i]` – O(1)
  - `.insert()` – O(n)
  - `.remove()` – O(n)
  - `.clear()` – O(n)
  - traversal – O(n)
- Grids
  - `.numRows()/.numC`
    – O(1)
  - `g[i][j]` – O(1)
  - `.inBounds()` – O(1)
  - traversal – O(n^2)

- Queues
  - `.pop()` – O(1)
  - `.isEmpty()` – O(1)
  - traversal – O(n)

- Sets
  - `size()` – O(1)
  - `Empty()` – O(1)
  - `d()` – ???
  - `move()` – ???
  - `ntains()` – ???
  - `versal` – O(n)
- aps
  - `ze()` – O(1)
  - `Empty()` – O(1)
  - `key]` – ???
  - `.contains()` – ???
  - traversal – O(n)

How can we achieve faster than O(n) runtime when searching/storing n elements?

Beyond Algorithmic Analysis (Katie Creel)

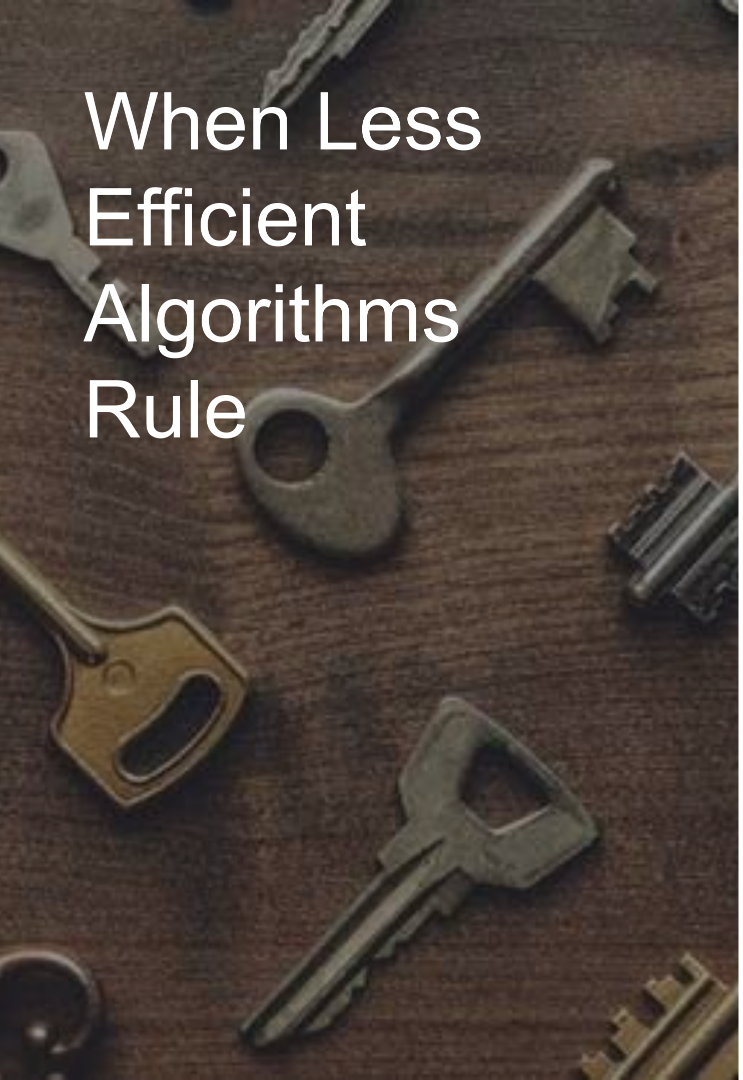


Ramifications of Big O Differences

- If we have an algorithm that has 1000 elements, and the O(log n) version runs in 10 milliseconds…

constant	logarithmic	linear	n log n	quadratic	polynomial (other than n^2, (n^2))	exponential
1 milliseconds	10 milliseconds	1 second	10 seconds	17 minutes	277 hours	heat death of the universe

Algorithmic complexity analysis can be the difference between a program that runs in a few seconds and one that won't finish before the heat death of the universe.   It is often necessary.



When Less Efficient Algorithms Rule

Passwords are often encrypted with a hash.

What prevents a hacker from guessing randomly, perhaps millions of times per minute, until the password is discovered?

~ Algorithmic Inefficiency ~

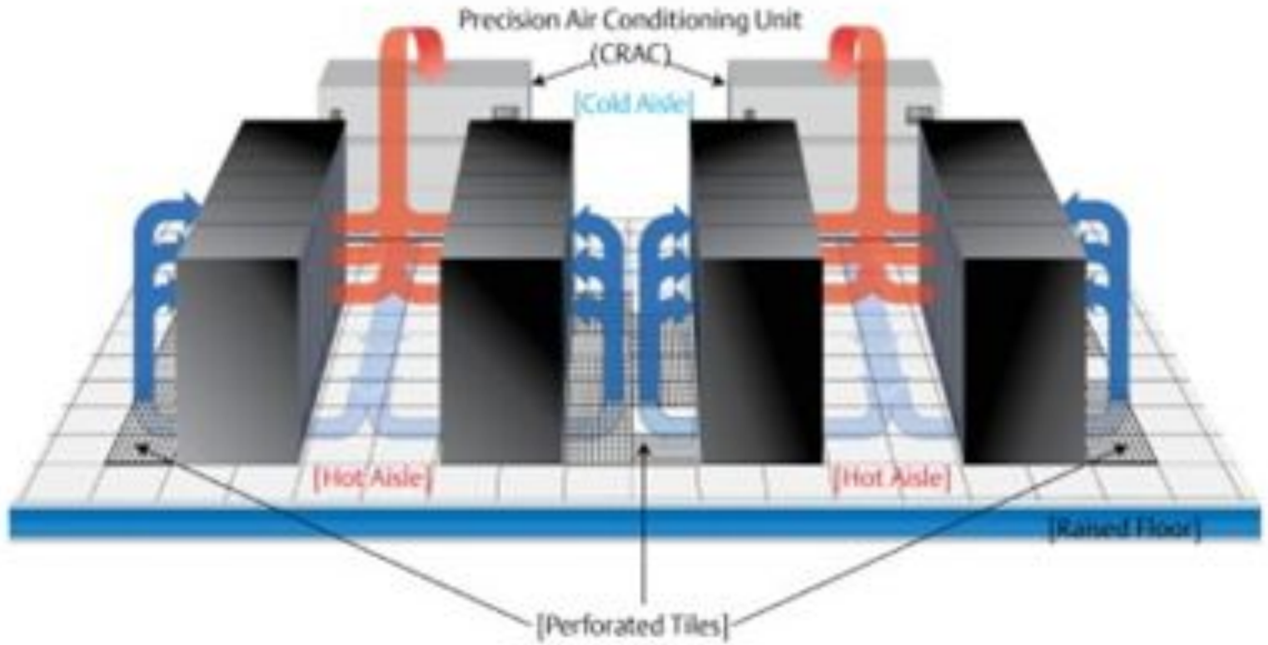
bcrypt and other popular encryption functions are intentionally designed to be slow, memory intensive, or both, making guessing more costly.

Green Computing

Big-O analysis can also be part of a “green computing”: a commitment to decreasing the environmental impact of computing.

Green Computing

Big-O analysis can also be part of a “green computing”: a commitment to decreasing the environmental impact of computing.



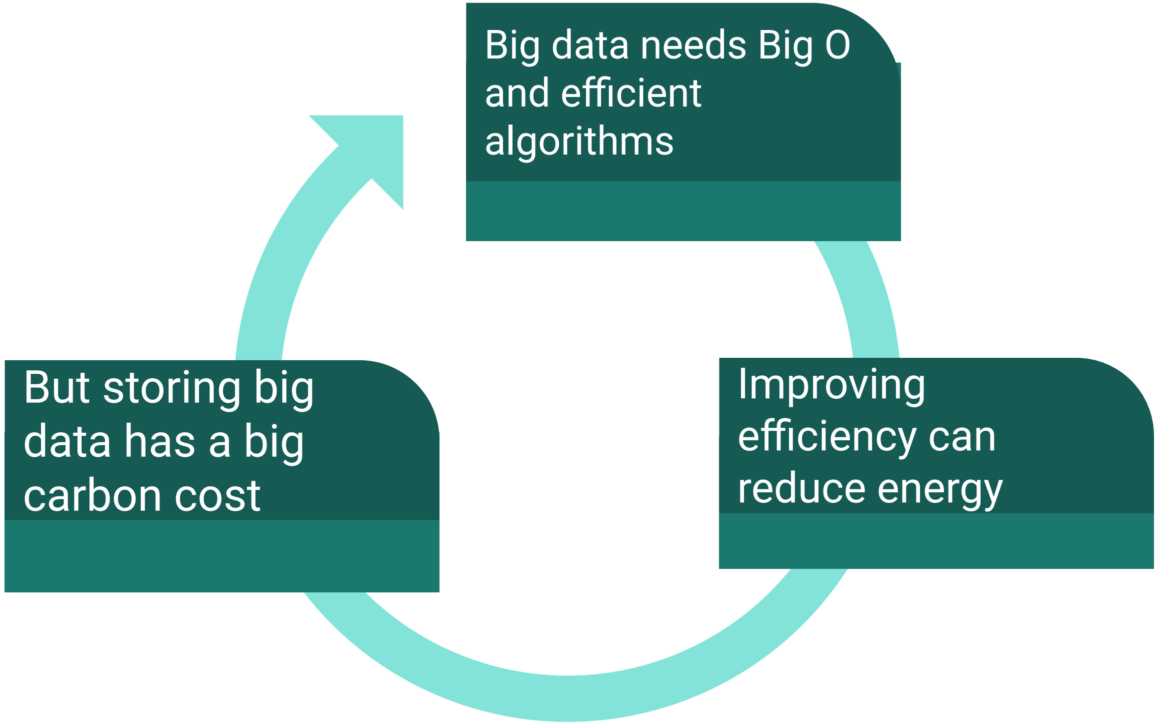
Green Computing Includes ...

Selecting sources of electricity with lower greenhouse gas emissions, including by moving computing to the cloud when necessary

Decreasing carbon footprint of datacenters

Recycling and reducing use of raw materials during manufacturing, especially metals that are socially and environmentally damaging to mine

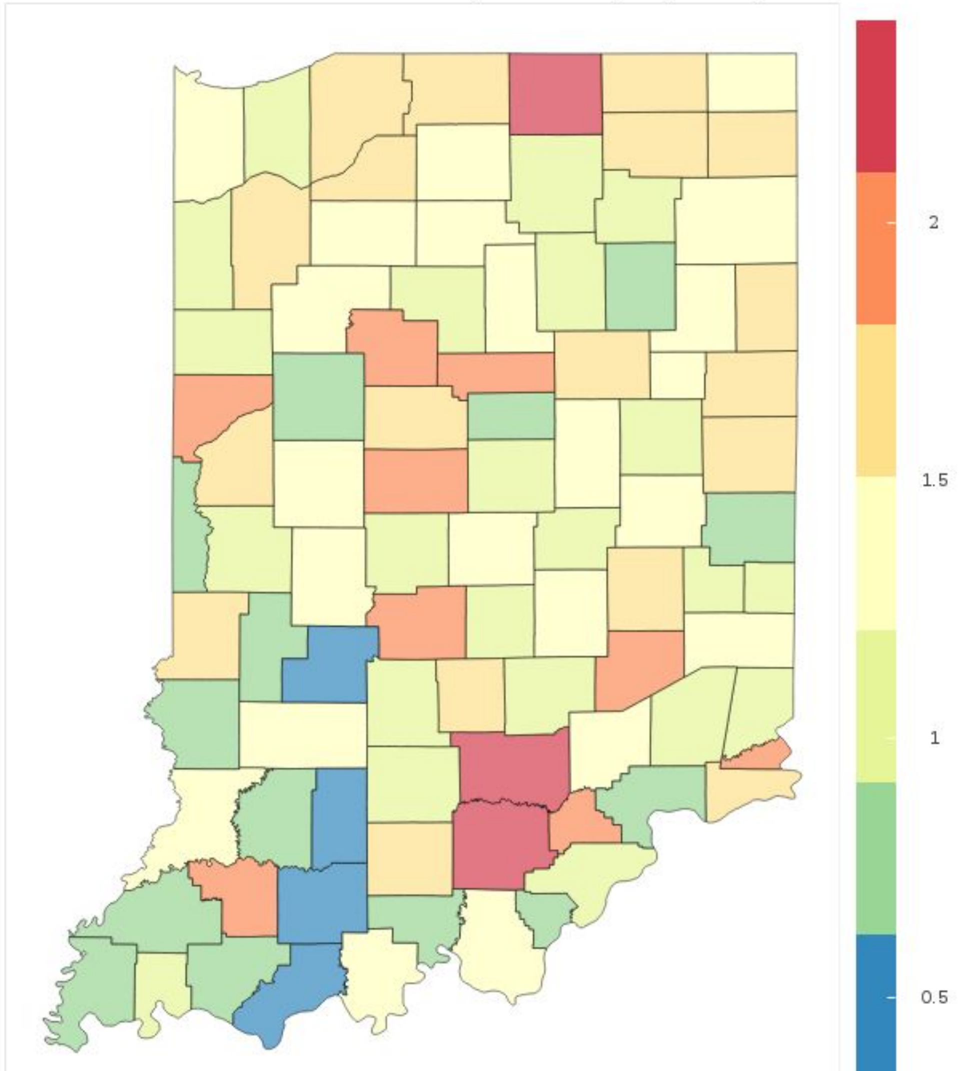
Reducing energy consumption of computation itself, including by *increasing algorithmic efficiency!*


Big data needs Big O and efficient algorithms
Improving efficiency can reduce energy
But storing big data has a big carbon cost

Green Computing's Efficiency Tradeoffs

- When is the answer to increase efficiency?
- When is the answer to choose smaller data?

WHAT ELSE IS IN THE BIG PICTURE VIEW OF AN ALGORITHM?



Case Study:

In 2006, the State of Indiana awarded IBM/ACS a contract for >$1 billion to “modernize” Indiana’s welfare case management system, including applications for food stamps and Medicaid.

After only 19 months, it was clearly not going as planned.

Case Study

Some "lowlights" of the system's failures:

- "Applicants waited 20 or 30 minutes on hold, only to be denied benefits for “failure to cooperate in establishing eligibility” if they were unable to receive a callback after having burned through their limited cellphone minutes.”
- "By February 2008, the number of households receiving food stamps in Delaware County, which includes Muncie, Indiana, dropped more than 7 percent, though requests for food assistance had climbed 4 percent in Indiana overall.”

Case Study

In light of these failures, the State of Indiana cancelled its contract with IBM and sued the company for breach of contract.

In court, IBM argued that they were *not responsible* for issues related to wait times, appeals, wrongful denials, lost documents, etc. as the contract only stated that a successful system would *succeed by increasing efficiency and reducing costs and fraud*. IBM's system did reduce costs, but did so by denying people the benefits they needed.

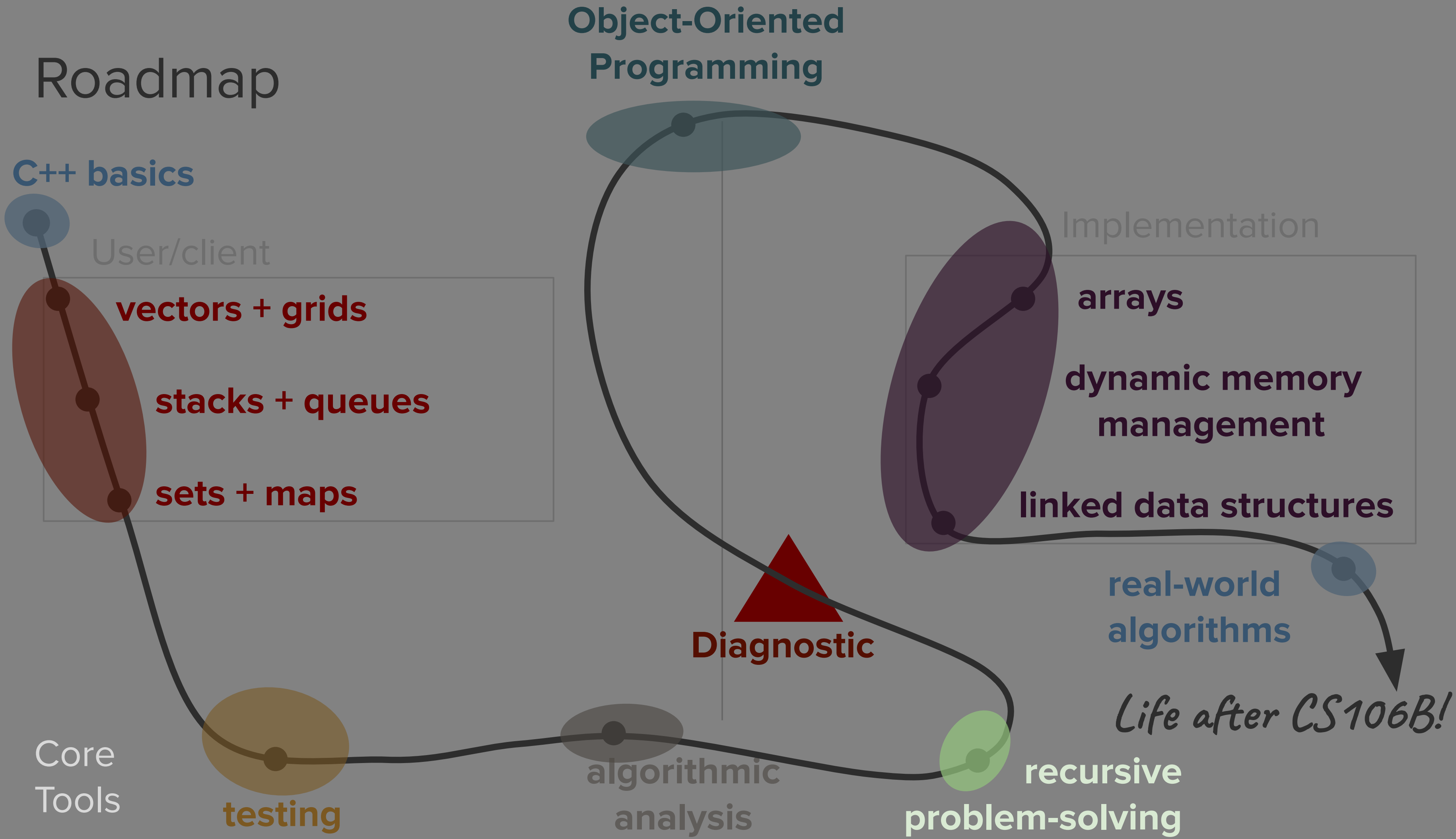
EFFICIENCY & OPTIMIZATION CAN ONLY BE AS GOOD AS GOOD AS WHAT IS OPTIMIZED...

Thank you!

Come visit any time:
calendly.com/kathleencreel
or by email kcreel@stanford.edu

What’s next?

Roadmap


Object-Oriented Programming
C++ basics
User/client
vectors + grids
stacks + queues
sets + maps
Implementation
arrays
dynamic memory management
linked data structures
real-world algorithms
Diagnostic
Life after CS106B!
Core Tools
testing
algorithmic analysis
recursive problem-solving

Recursion