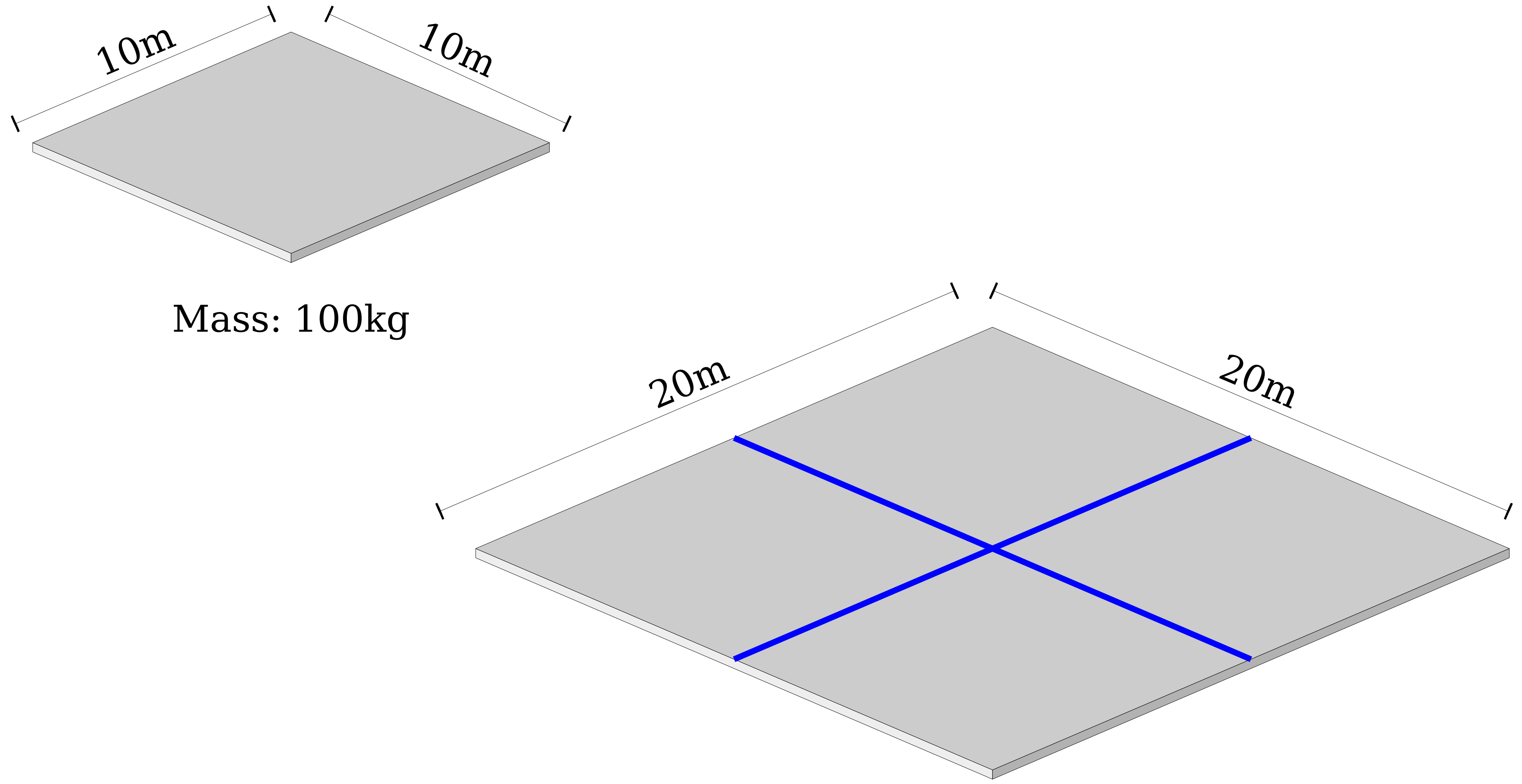
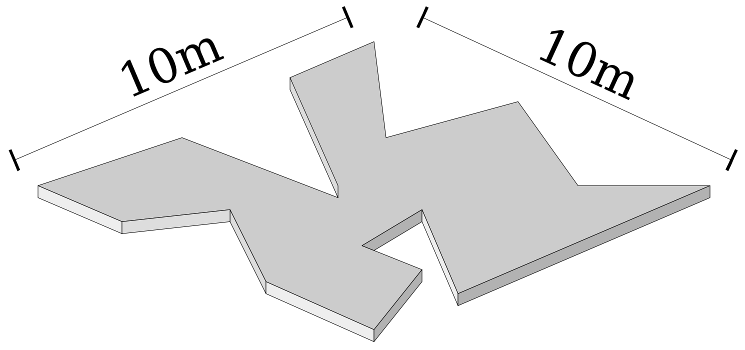
Big-O Notation

Estimating Quantities

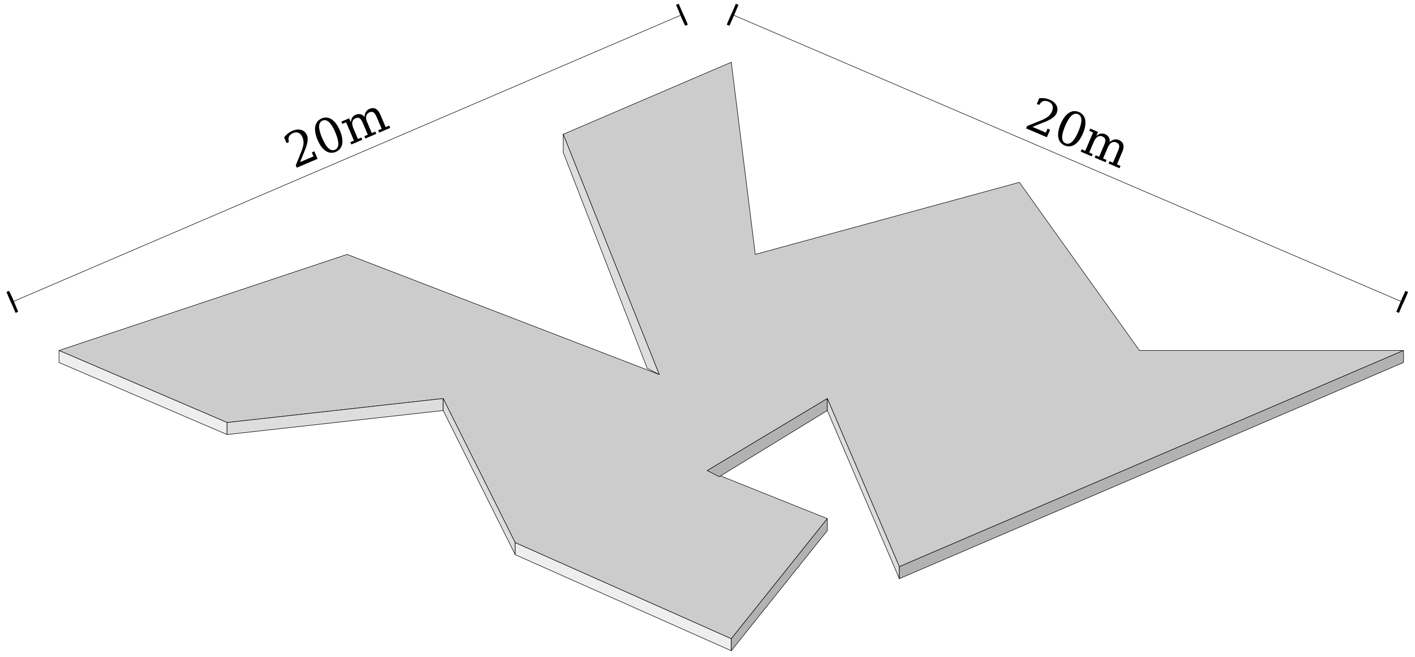


These two square plates are made of the same material.
They have the same thickness.

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

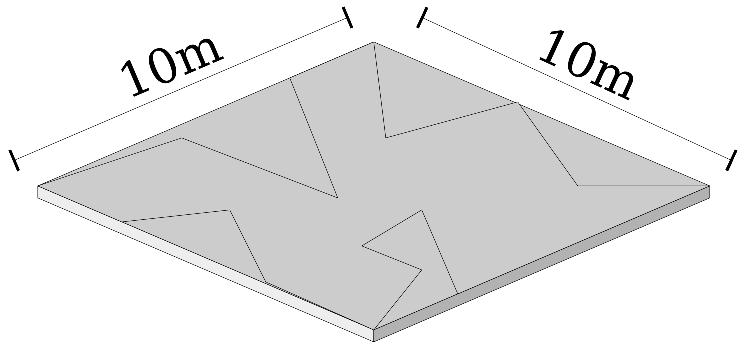


Mass: 60kg

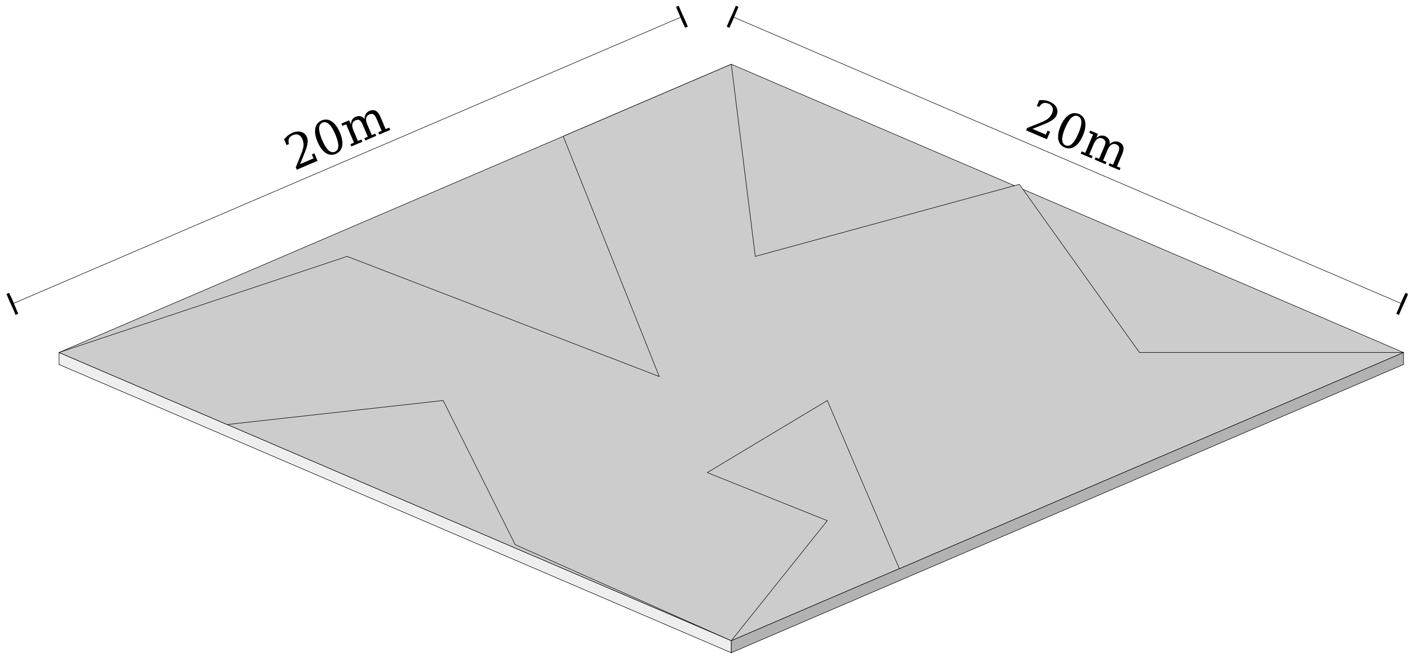


These two figures are made of the same material.
They have the same thickness.

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

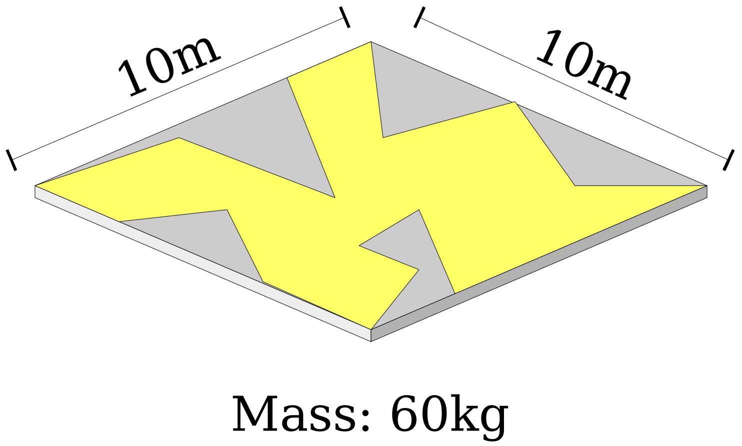


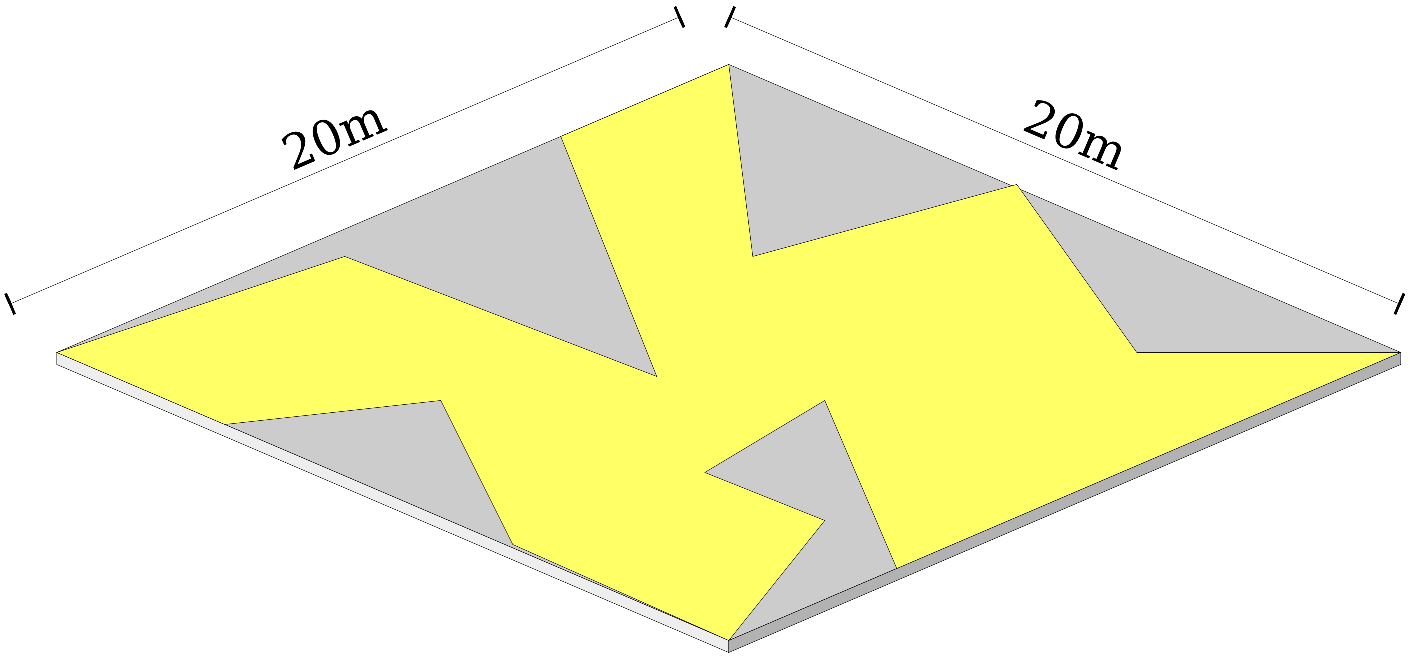
Mass: 60kg



These two figures are made of the same material.
They have the same thickness.

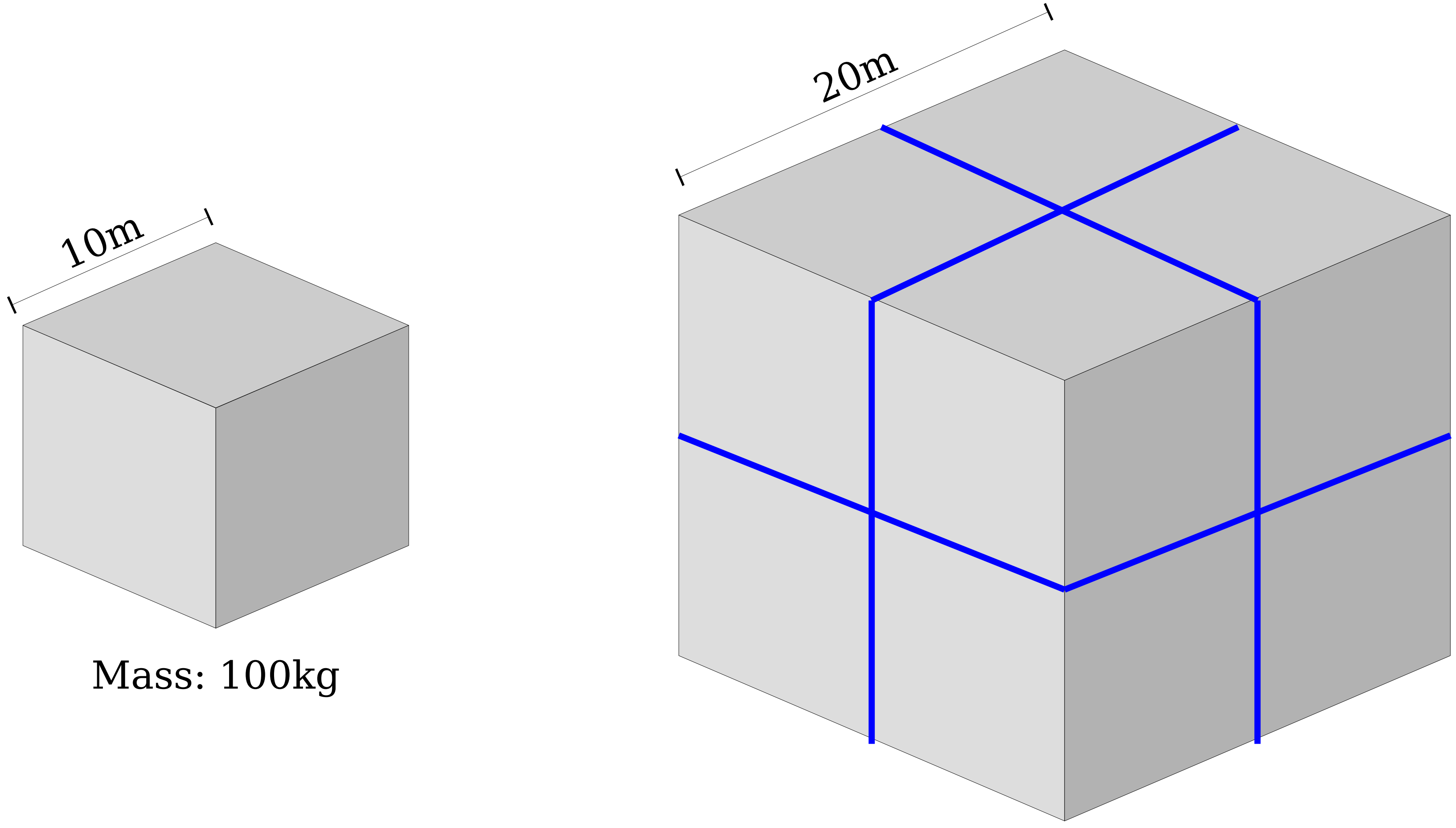
What’s your best guess for the mass of the second figure?





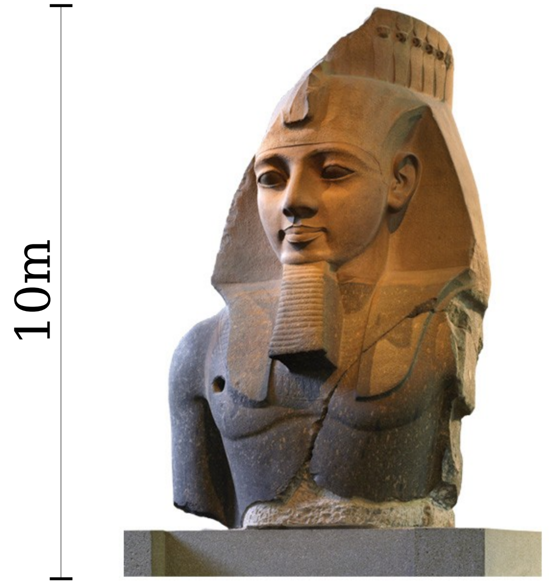
These two figures are made of the same material.
They have the same thickness.

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

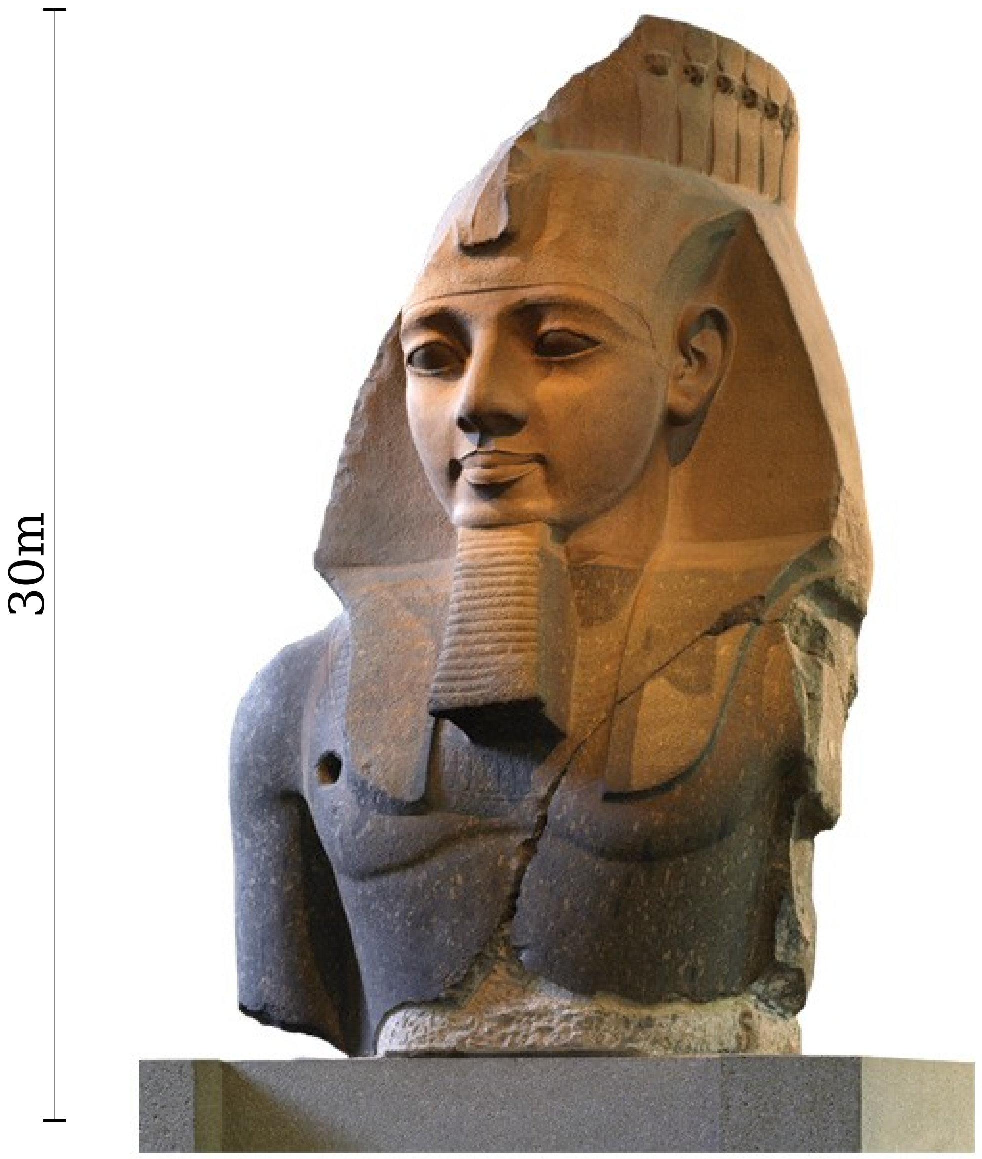


These two cubes are made of the same material.

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



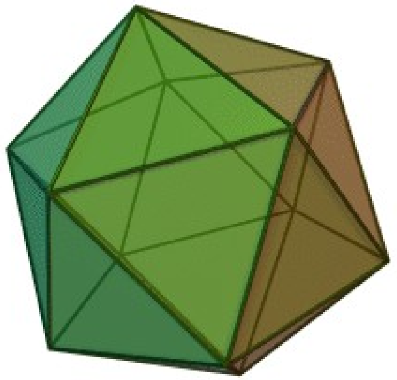
Mass: 1,000kg



These two statues are made of the same material.

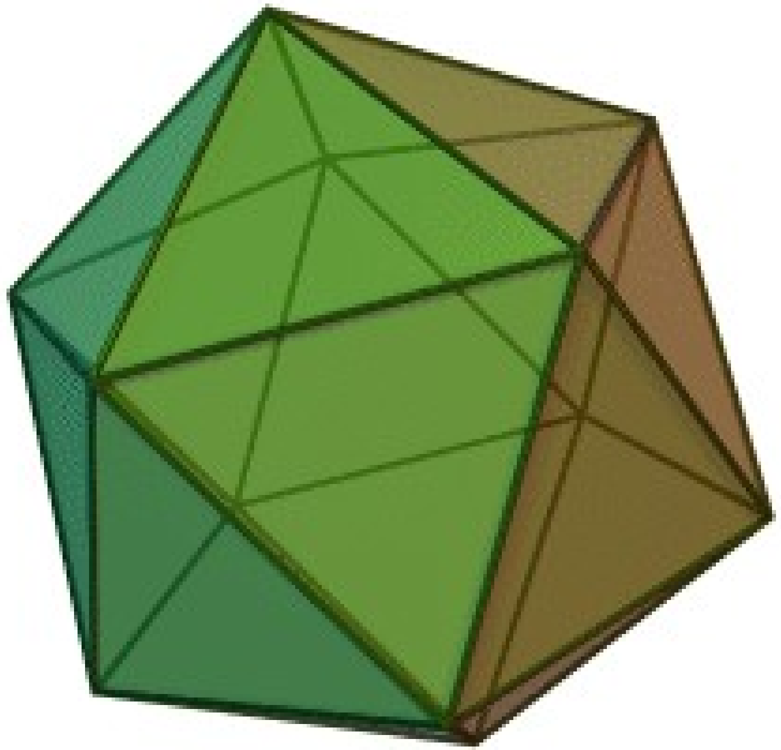
What’s your best guess for the mass of the second statue?

Image Credit



All sides of each triangle are 10*m* long.

Paint required: 90L



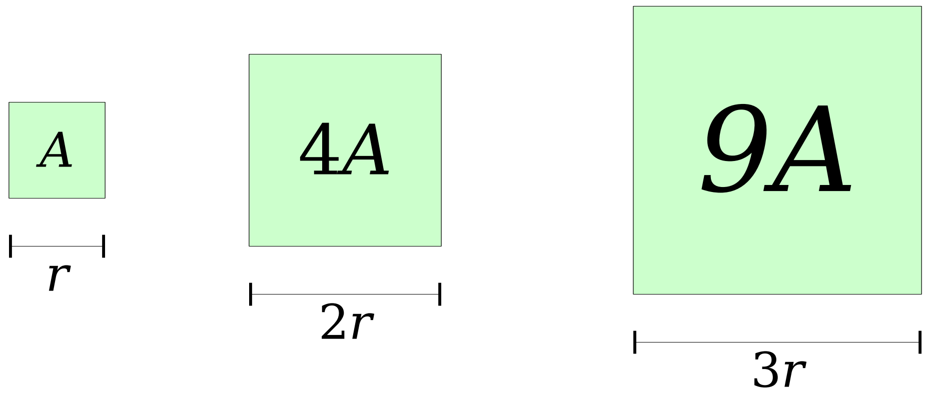
All sides of each triangle are 40*m* long.

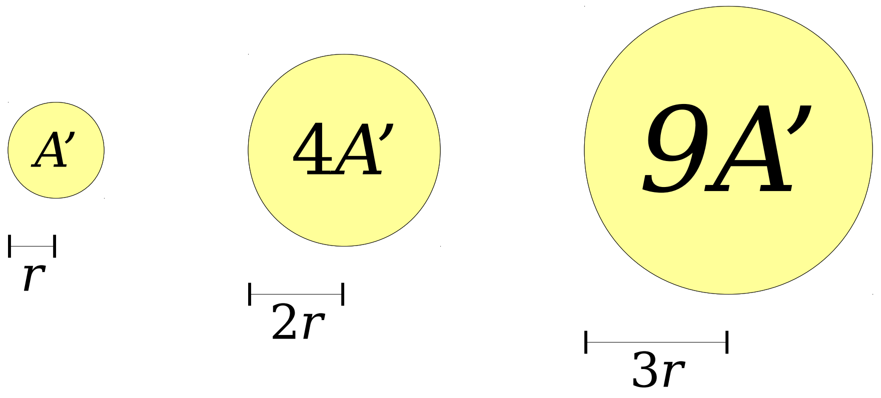
How much paint is needed to paint the surface of the larger icosahedron?

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×.

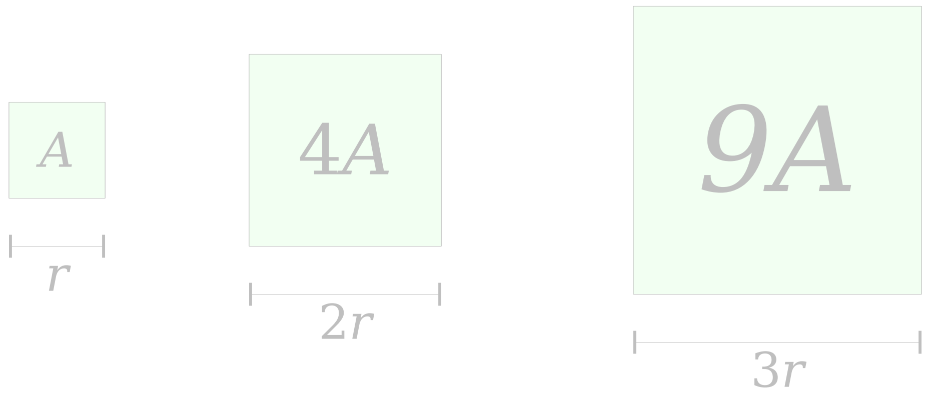
Doubling r increases area 4×.
Tripling r increases area 9×.

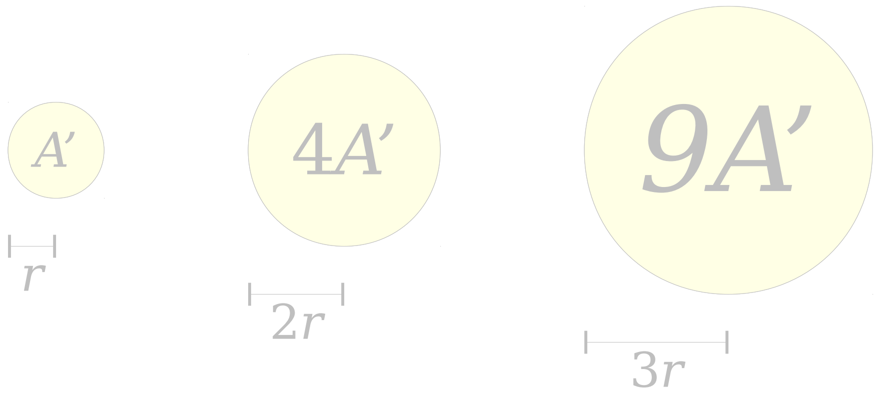
Big-O Notation

- ***Big-O notation*** is a w
  rate at which some qu

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: Network Value

- ***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: Biomechanics

- ***Kleiber's Law*** says that

  An animal of mass *m* has
  metabolic rate $O(m^{3/4})$.

- Assume a 50kg human has a metabolic rate of 100W. Estimate the metabolic rate of a 5000kg elephant.
- ***Reasonable guess:*** We've increased our scale by a factor of 100, so the metabolic rate should scale by $100^{3/4} \approx 31.62$. A good guess is 3,162W.

Example: Biology

- ***Question:*** Why are cells tiny?
- Assume, for now, that 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 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.

Example: Manufacturing

- You're working at a company producing widgets. It costs you some amount of money to produce a widget, 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 widgets?

$$\boxed{\begin{aligned}\text{Cost}(n) &= \overset{\substack{\textit{This term grows as a}\\ \textit{function of } n.\\ \downarrow}}{n \times \mathit{costPerWidget}} + \overset{\substack{\textit{This term does}\\ \textit{not grow.}\\ \downarrow}}{\mathit{startupCost}} \\ &= \mathbf{O}(\boldsymbol{n}).\end{aligned}}$$

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 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 powerful tool for predicting behavior.

Time-Out for Announcements!

Midterm Exam Logistics

- Our midterm exam is next ***Tuesday, February 11th*** from ***7:00PM – 10:00PM***. Locations are divvied up by last (family) name:
  - `A – L`: Go to Cubberley Auditorium.
  - `M – V`: Go to Bishop Auditorium.
  - `W – Z`: Go to 320-105.
- You're responsible for Lectures 00 – 09 and topics covered in Assignments 0 – 3. The topic coverage goes up through but not including recursive backtracking.
- The exam is closed-book, closed-computer, and limited-note. You can bring a double-sided, 8.5" × 11" sheet of notes with you to the exam, decorated however you'd like.
- Students with OAE accommodations: Please contact us ***immediately*** if we don't yet have your OAE letter.

Midterm Exam

- ***We want you to do well on this exam.*** We're not trying to weed out weak students. We're not trying to enforce a curve where there isn't one. We want you to show what you've learned up to this point so that you get a sense for where you stand and where you can improve.
- The purpose of this midterm is to give you a chance to show what you've learned in the past few weeks. It is not designed to assess your “programming potential” or “innate coding ability.”

Preparing for the Exam

- We've released a handout (Handout 16 containing advice about how to prepare for the exam. In particular:
  - There’s advice about how to prepare for exams that require writing code on paper.
  - There’s advice about how to best study for the exam.
  - There’s advice about what you do and don’t need to write on a coding exam.
- Please take the time to read over this handout – it’s there for a reason!

Practice Exams

- Up on the course website, you’ll find three practice midterm exams, each of which contains questions really used in past CS106B midterms.
- Take the time to work through some of these problems. This is, perhaps, the best way to study.

fg

(The Unix command that takes a paused program and starts it up again.)

What does big-O notation have to do with computer science?

Fundamental Question:

How do we measure efficiency?

One Idea: ***Runtime***

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***.

```
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) {
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
}
```

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: **$O(n)$**.

```cpp
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) 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.

```
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.

```
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.

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

If we time this code on input n, how much longer will it take to run on the input $2n$?

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

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

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

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

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

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

Answer: **O(1)**.

Next Time

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