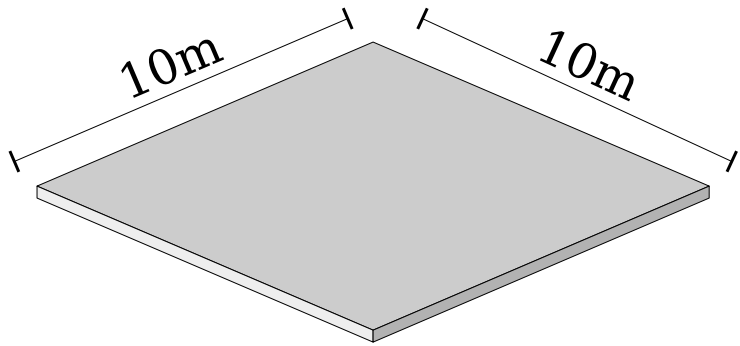


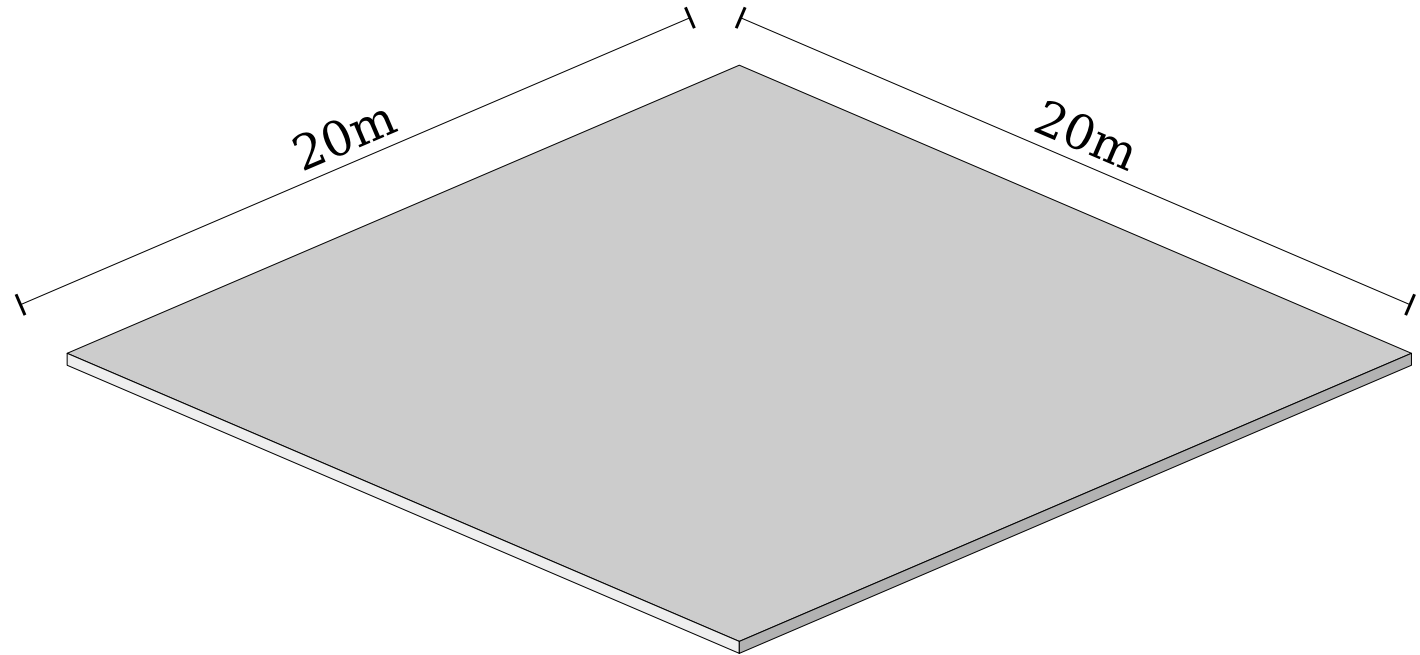
Big-O Notation

Estimating Quantities



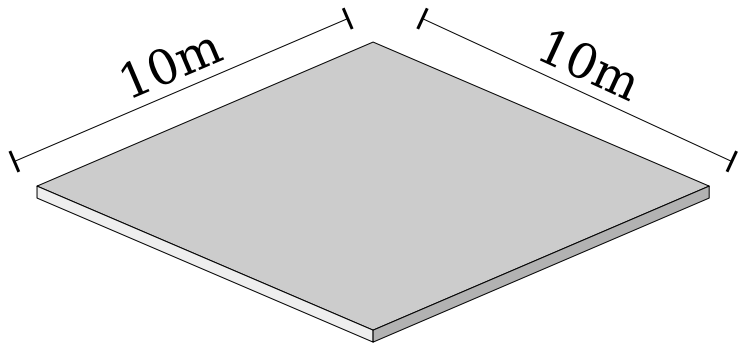
Mass: 100kg

Formulate a hypothesis!



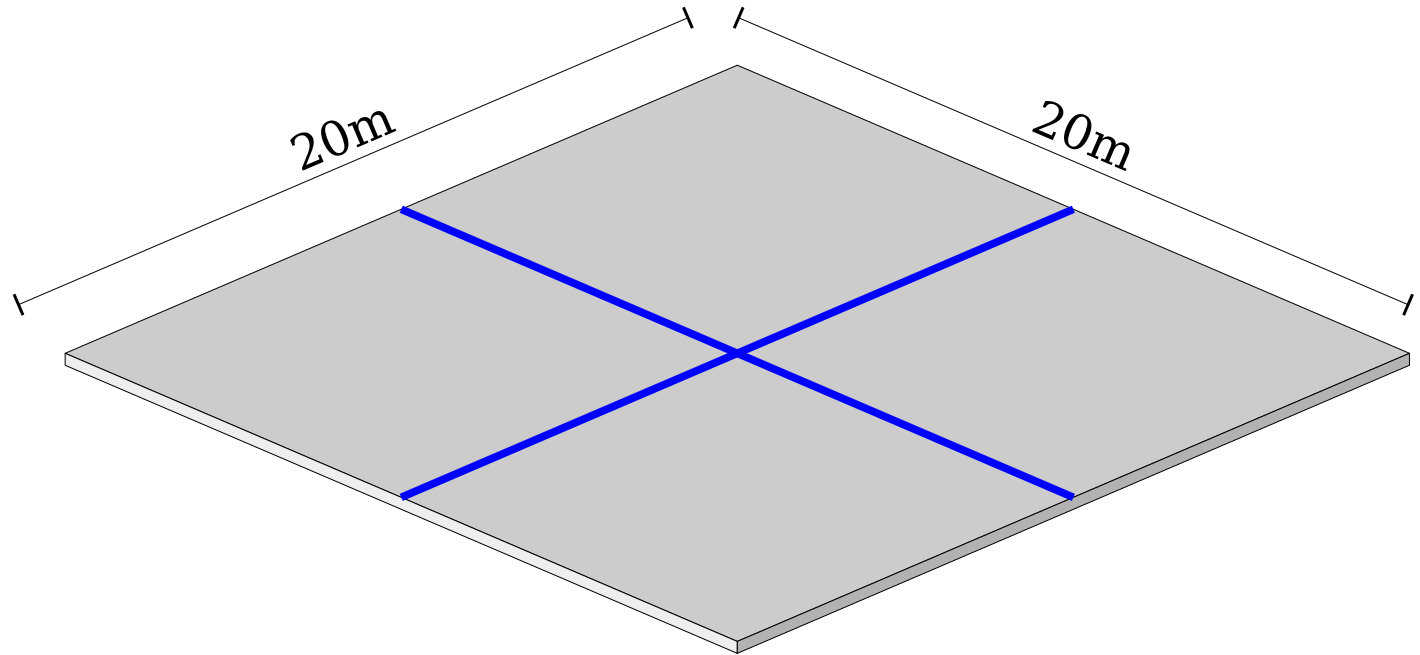
These two square plates are made of the same material.
They have the same thickness.

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



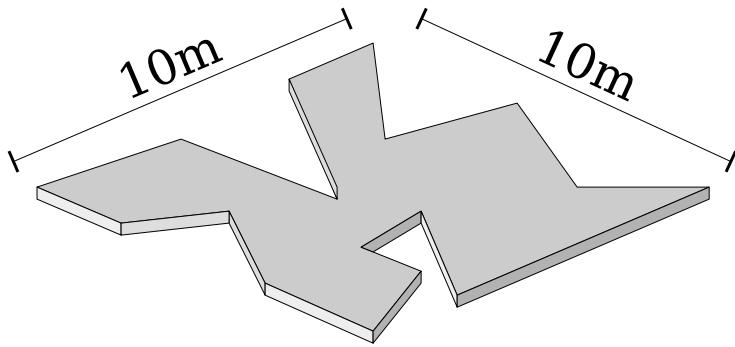
Mass: 100kg

Turn to your neighbor
and discuss!

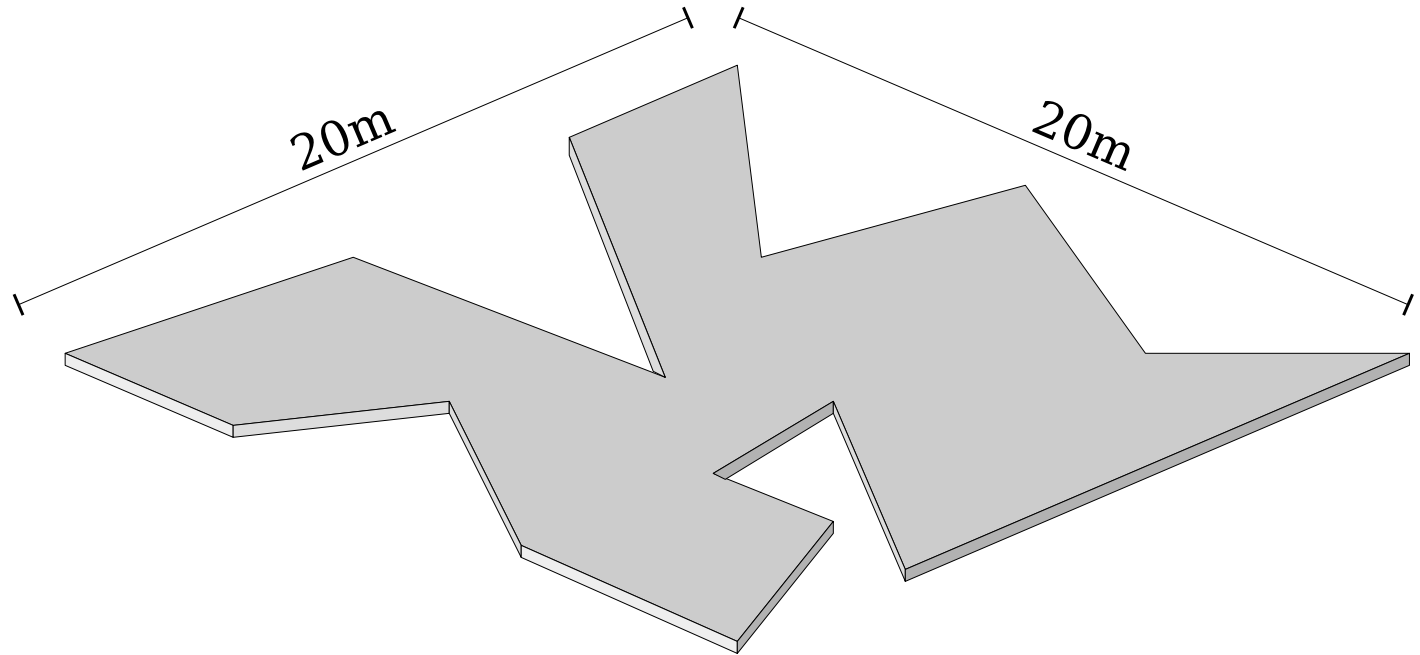


These two square plates are made of the same material.
They have the same thickness.

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

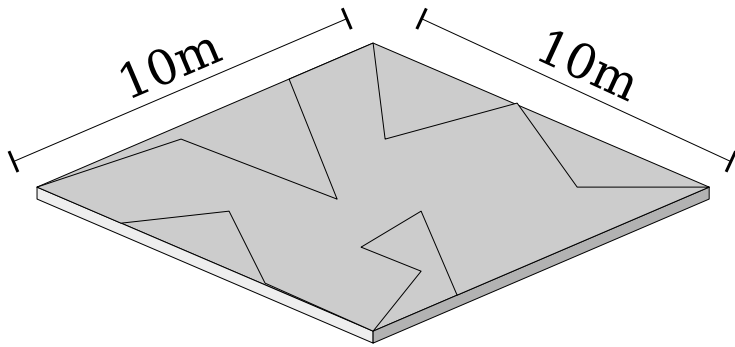


Mass: 60kg

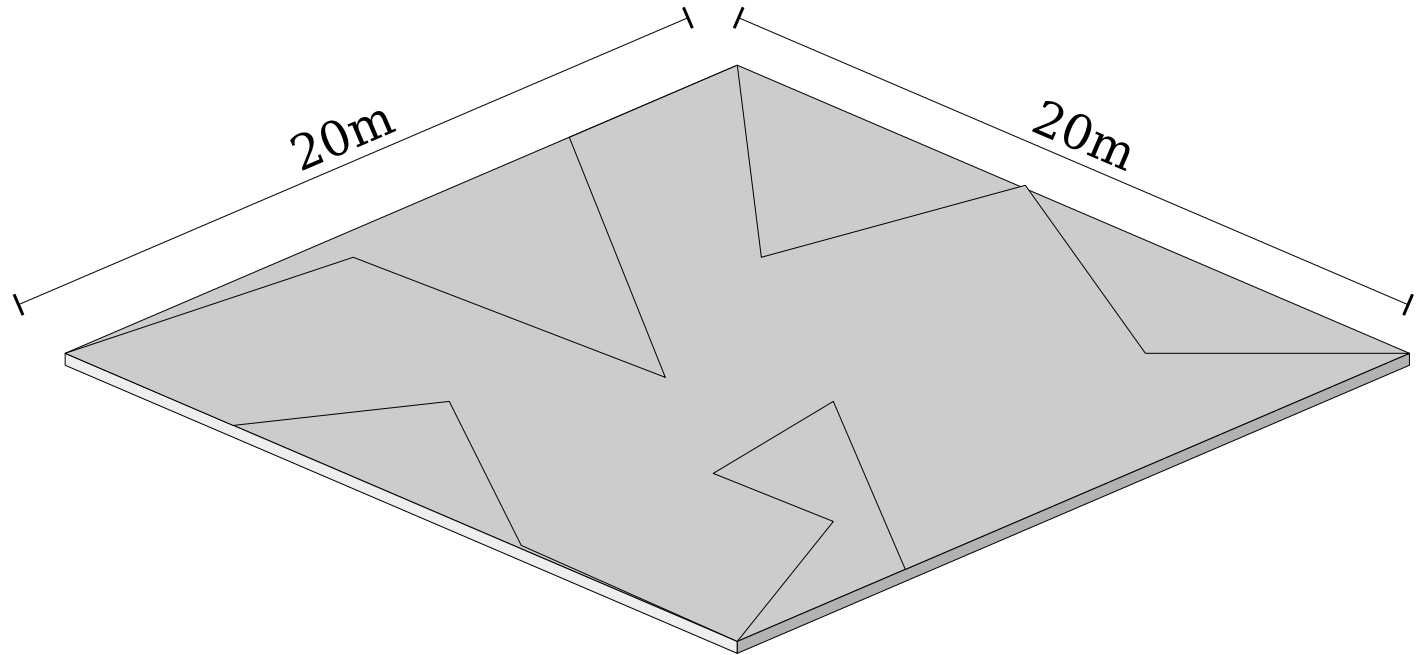


These two figures are made of the same material.
They have the same thickness.

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

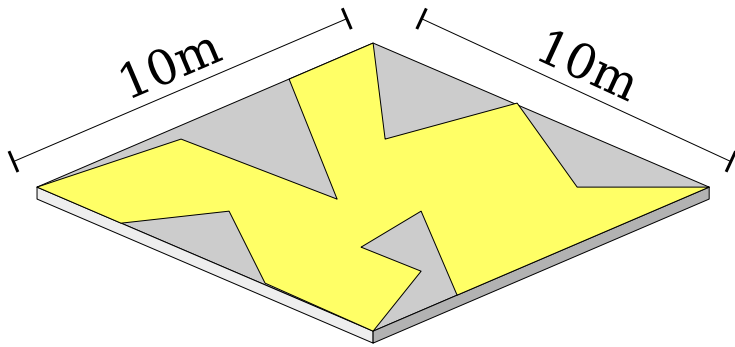


Mass: 60kg

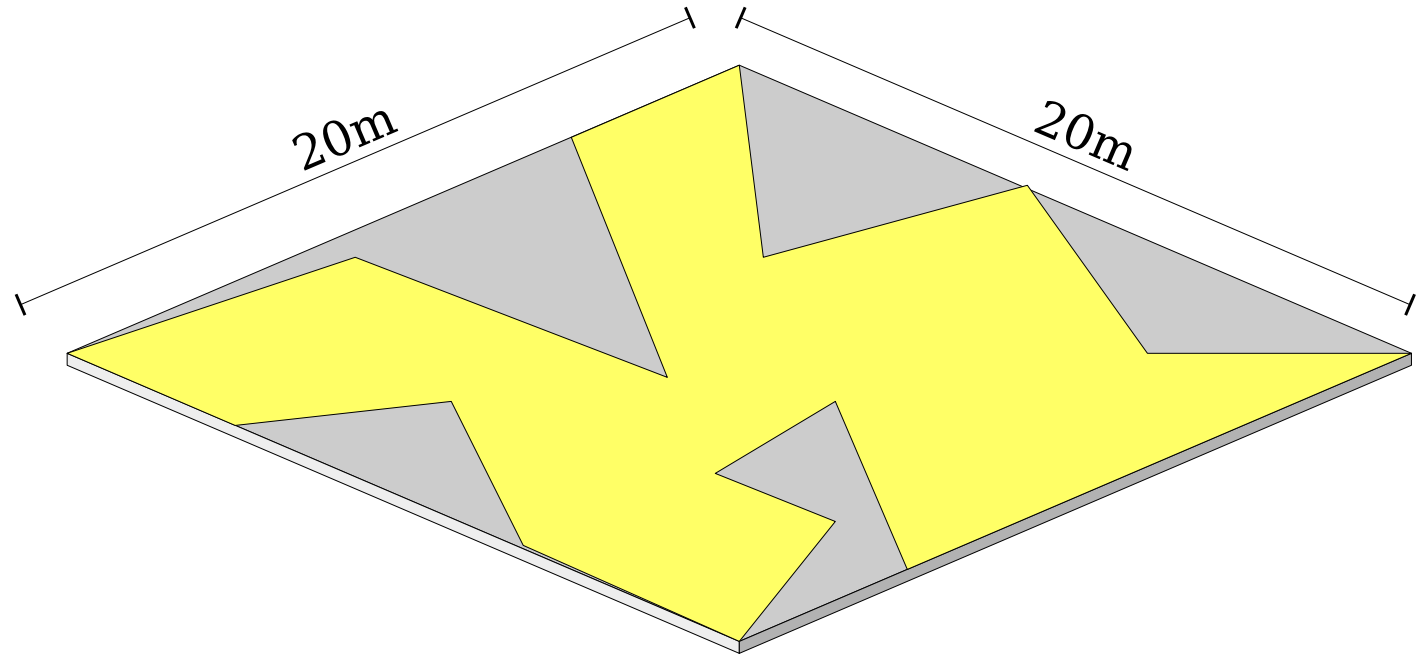


These two figures are made of the same material.
They have the same thickness.

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

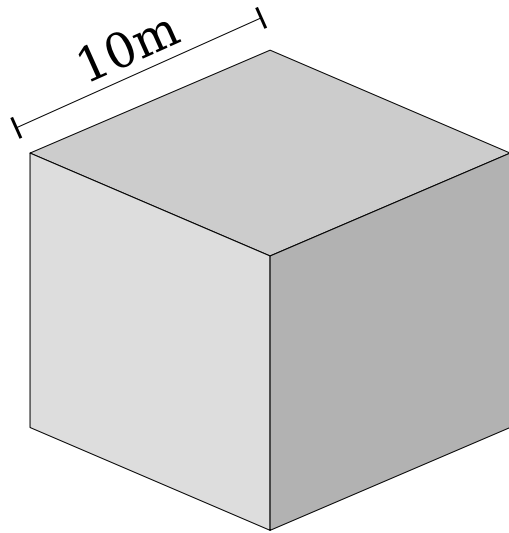


Mass: 60kg

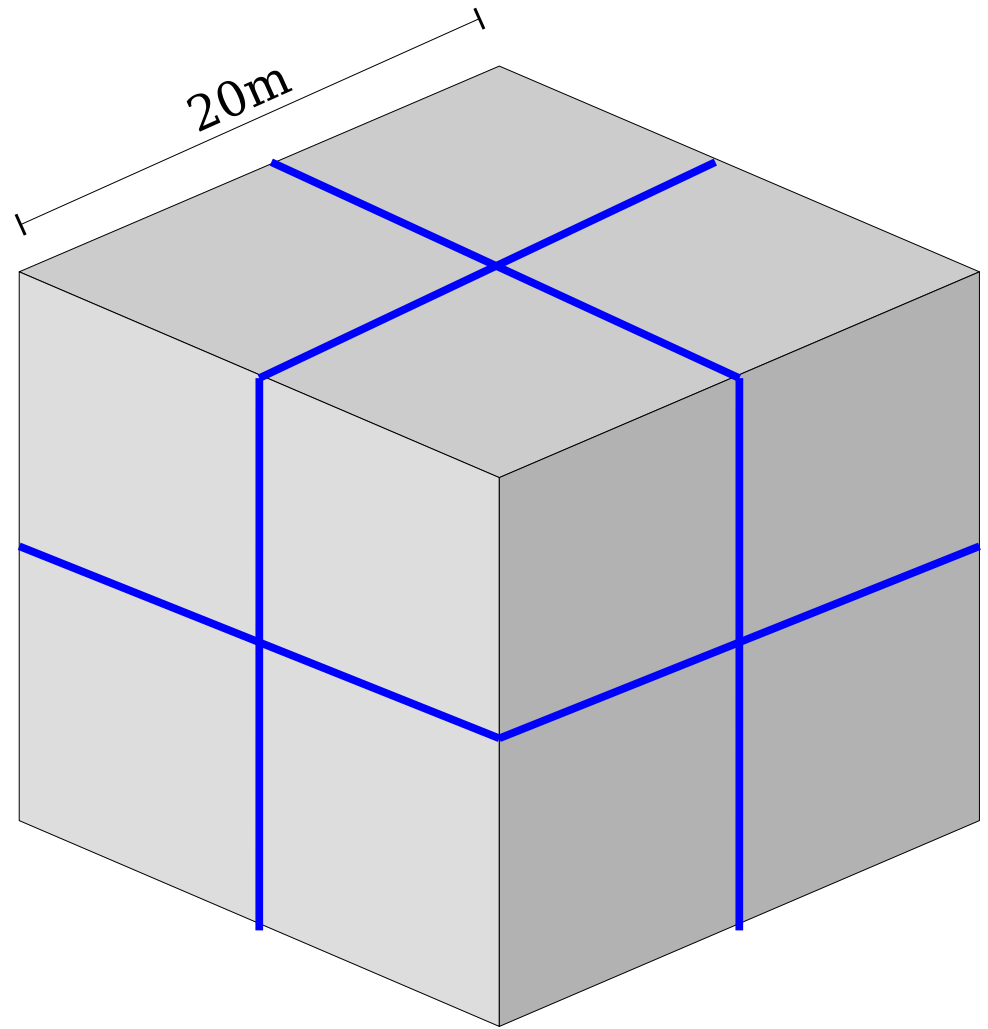


These two figures are made of the same material.
They have the same thickness.

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



Mass: 100kg



These two cubes are made of the same material.
What's your best guess for the mass of the second cube?



Mass: 1,000kg

30m



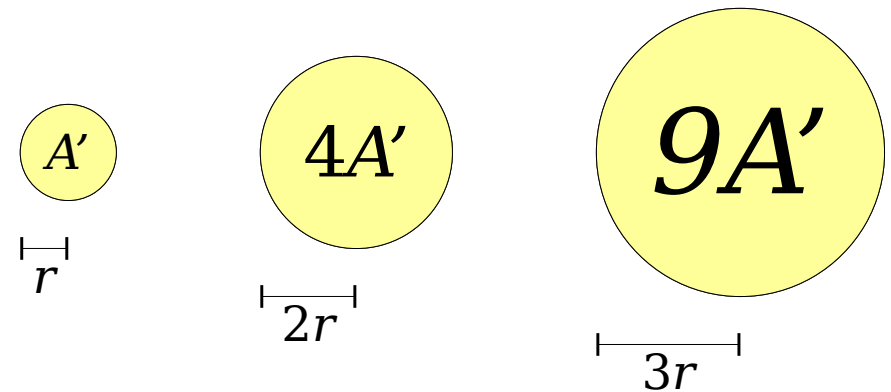
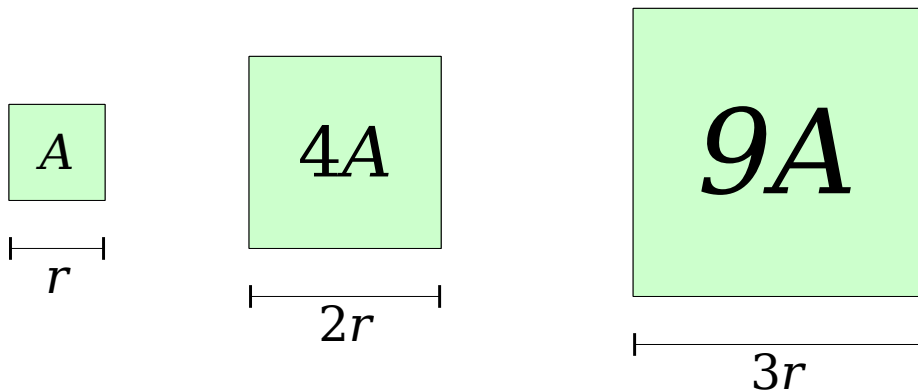
These two statues are made of the same material.

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

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 $O(r^2)$.

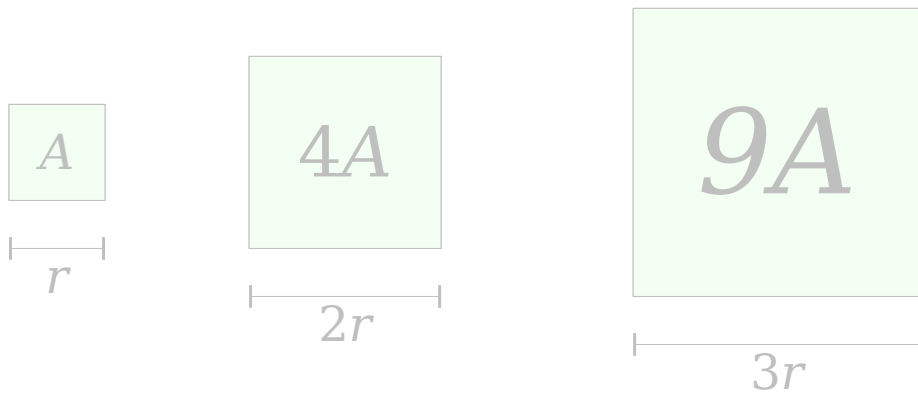


Doubling r increases area $4\times$.
Tripling r increases area $9\times$.

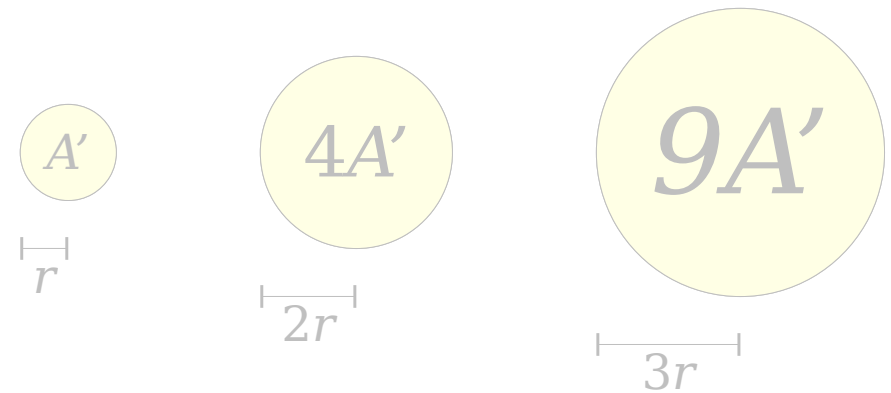
Doubling r increases area $4\times$.
Tripling r increases area $9\times$.

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 $4\times$.
Tripling r increases area $9\times$.

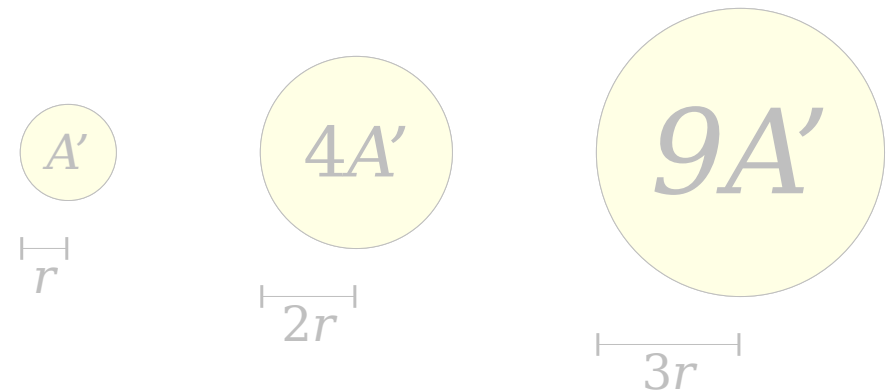
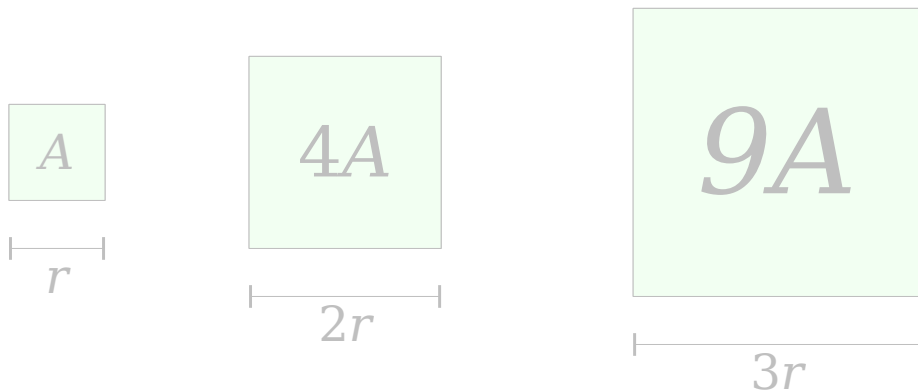


Doubling r increases area $4\times$.
Tripling r increases area $9\times$.

Big-O Notation

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

- **Big-O notation** is a way to describe 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 $4\times$.
Tripling r increases area $9\times$.

Doubling r increases area $4\times$.
Tripling r increases area $9\times$.

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 $O(n^2)$.

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.

Formulate a hypothesis!

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.

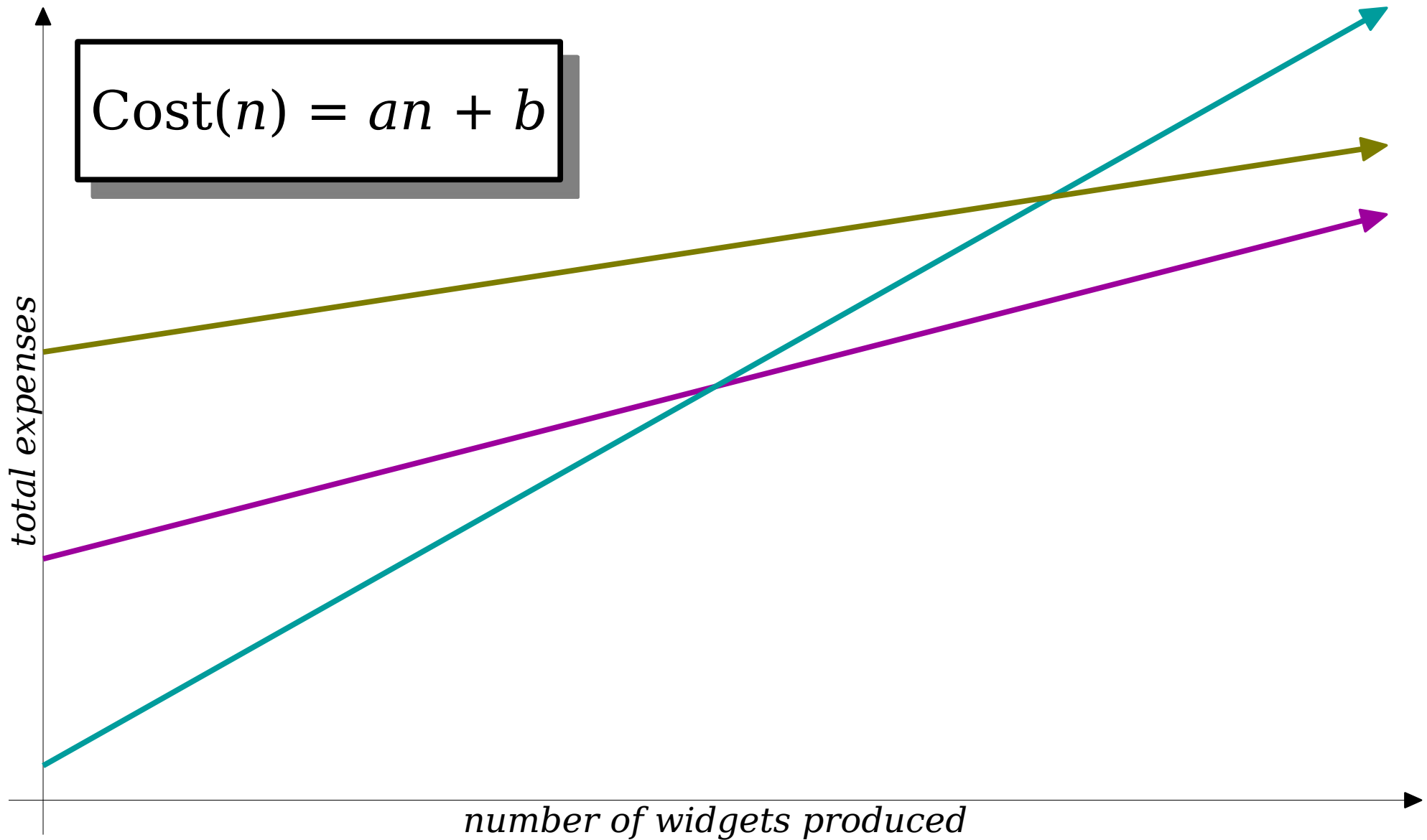
Now, discuss your best guess with your neighbor.

Example: Network Value

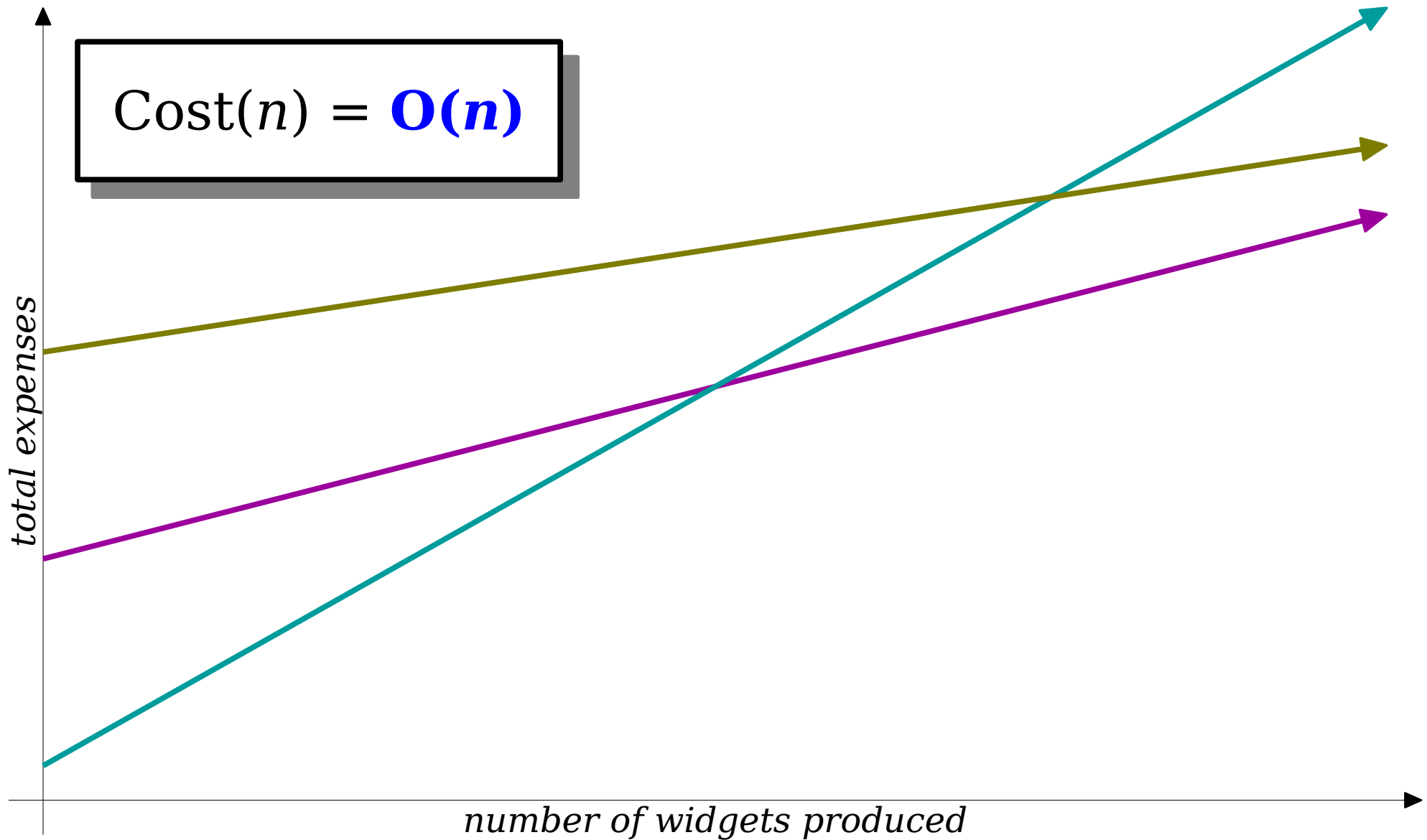
- ***Metcalfe's Law*** says that
The value of a communications network with n users is $O(n^2)$.
- Imagine a social network has 10,000,000 users and is worth \$10,000,000. Estimate how many users it needs to have to be worth \$1,000,000,000.
- ***Reasonable guess:*** The network needs to grow its value 100×. Since value grows quadratically with size, it needs to grow its user base 10×, requiring 100,000,000 users.

A Messier Example: Manufacturing

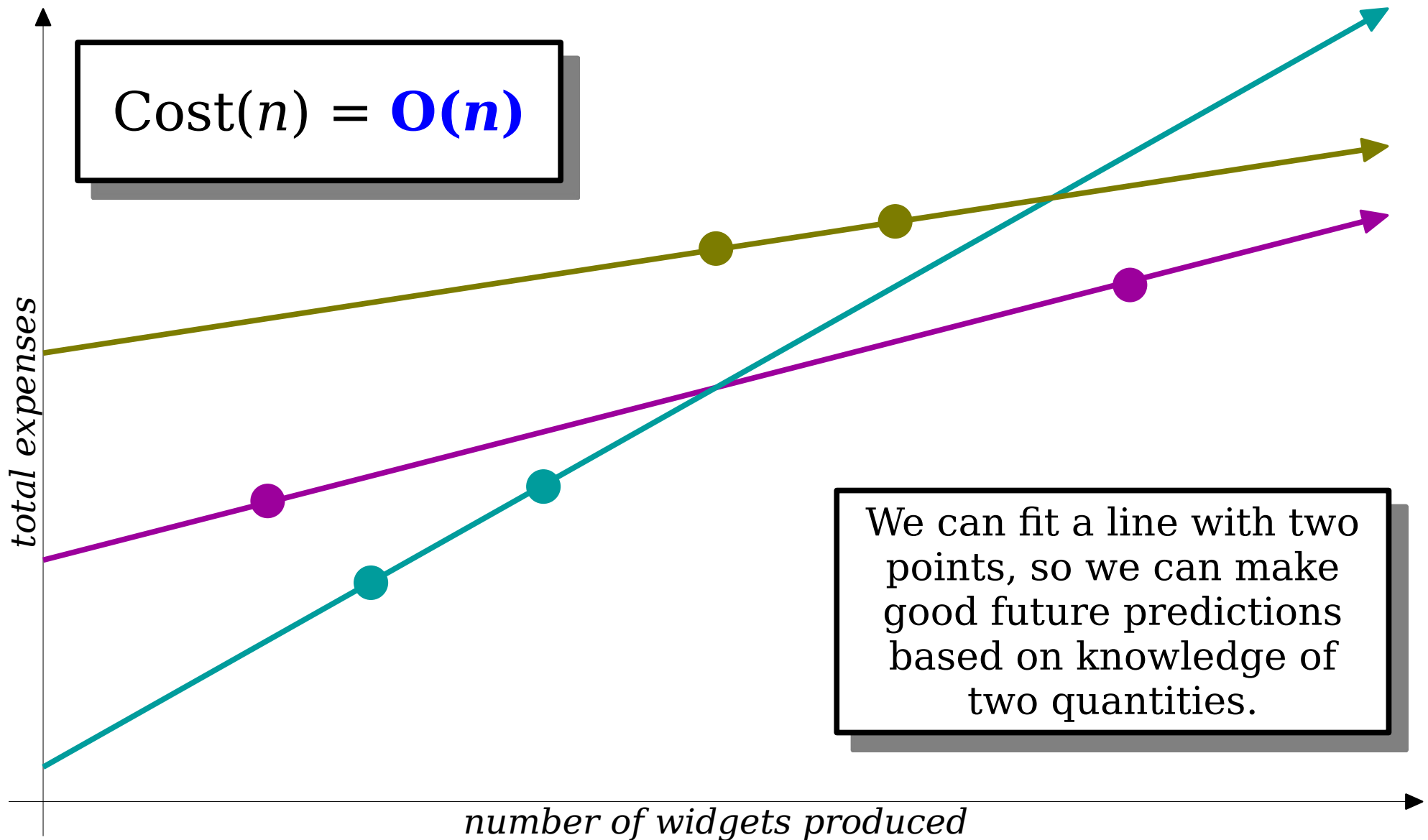
Making Widgets



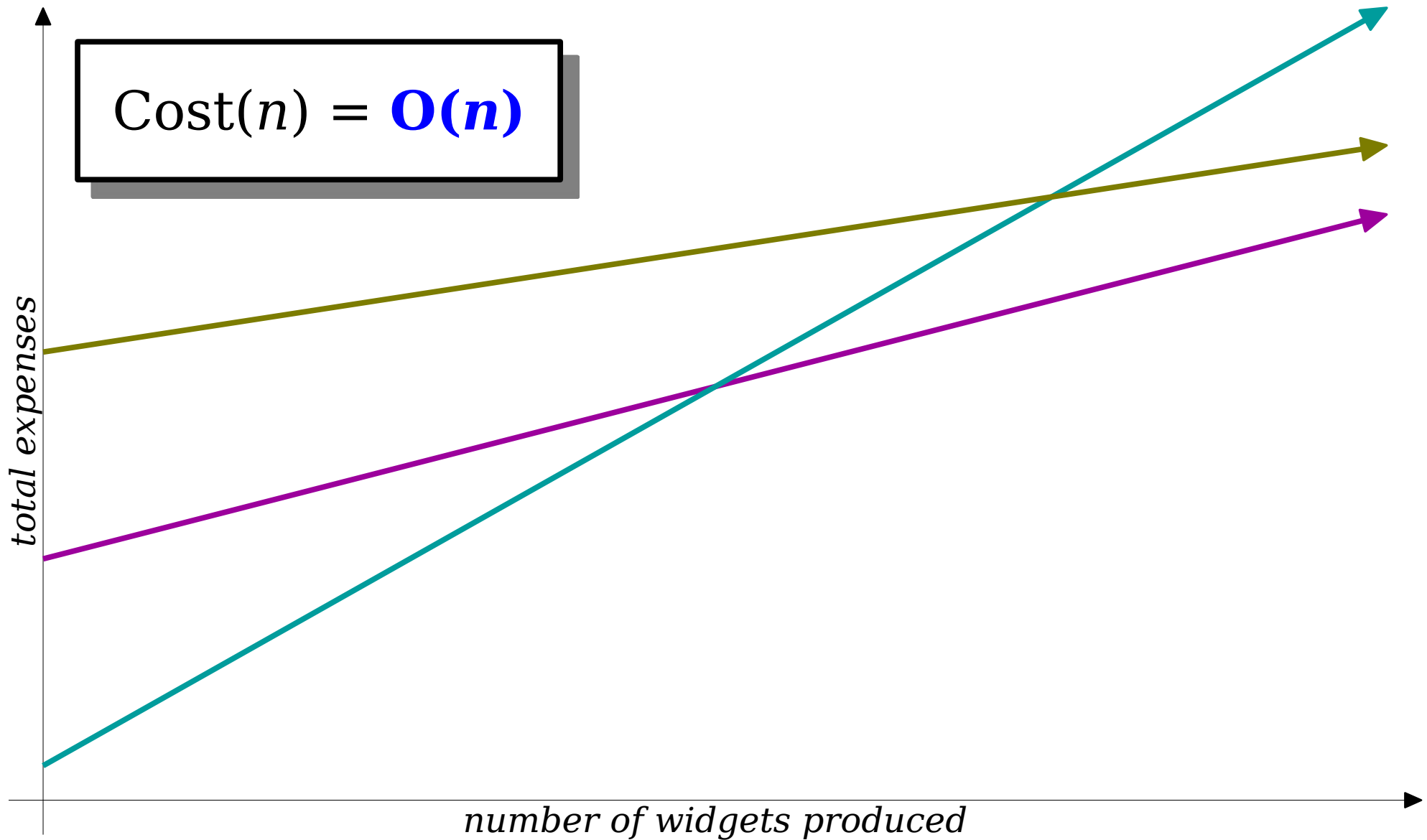
Making Widgets



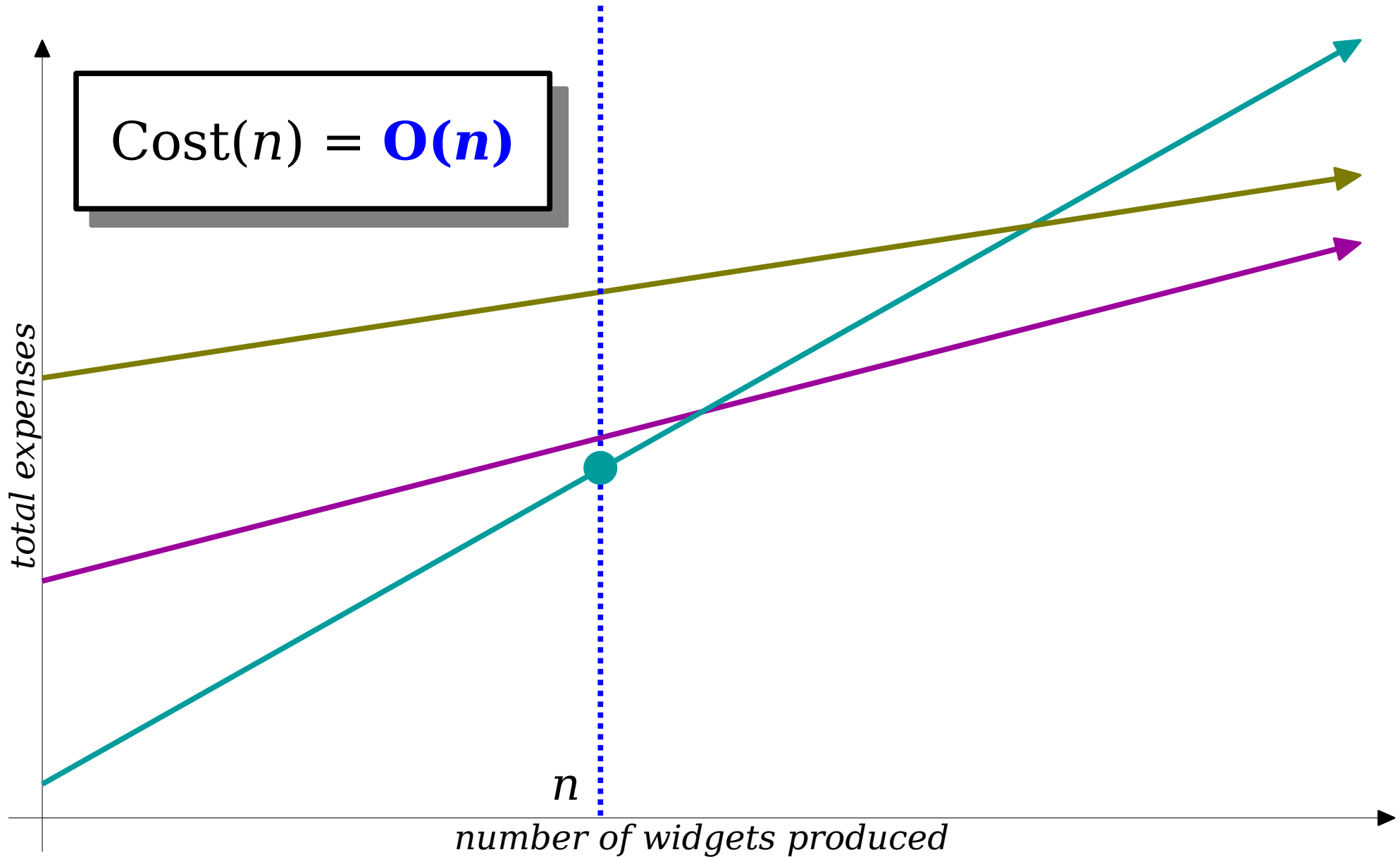
Making Widgets



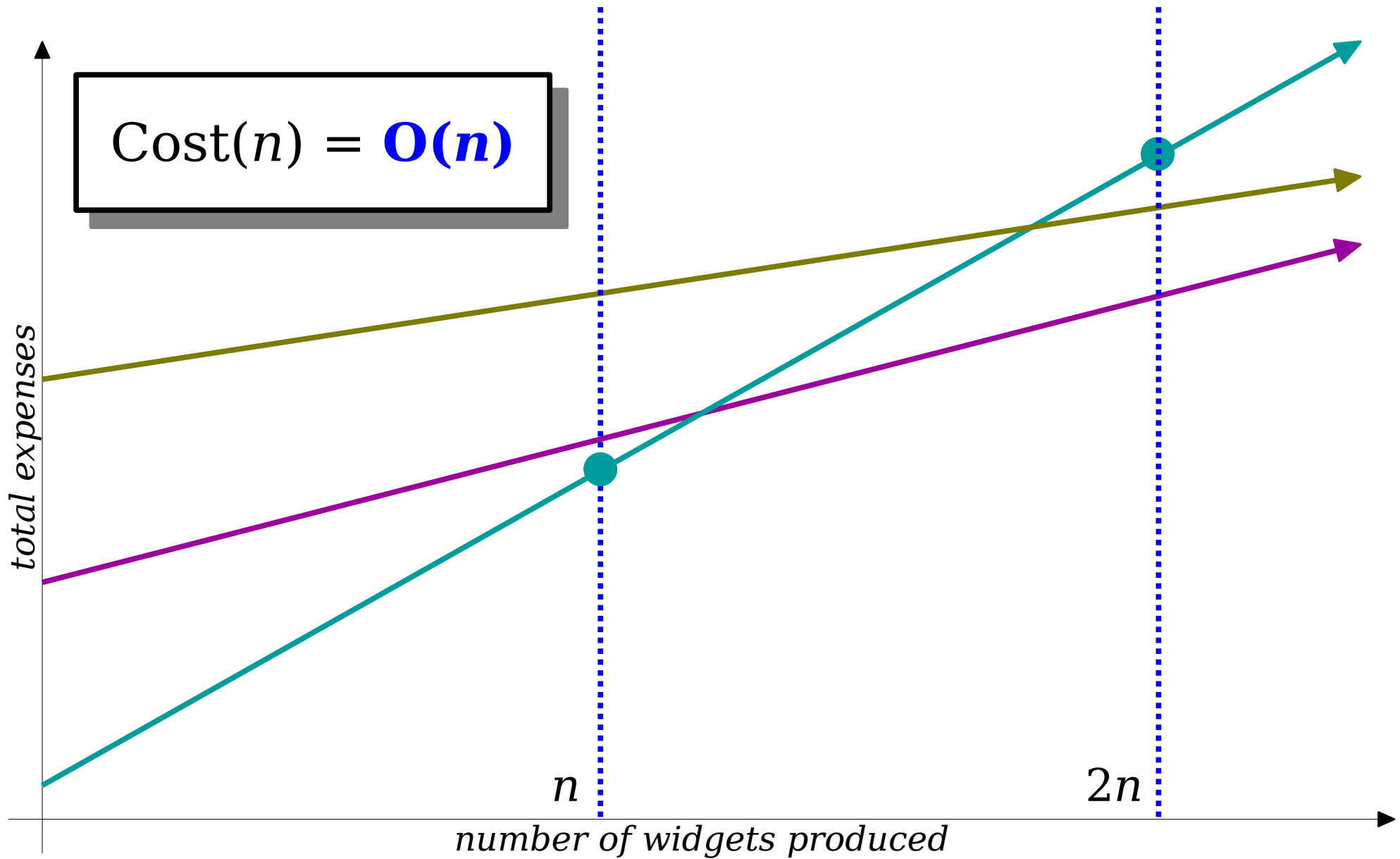
Making Widgets



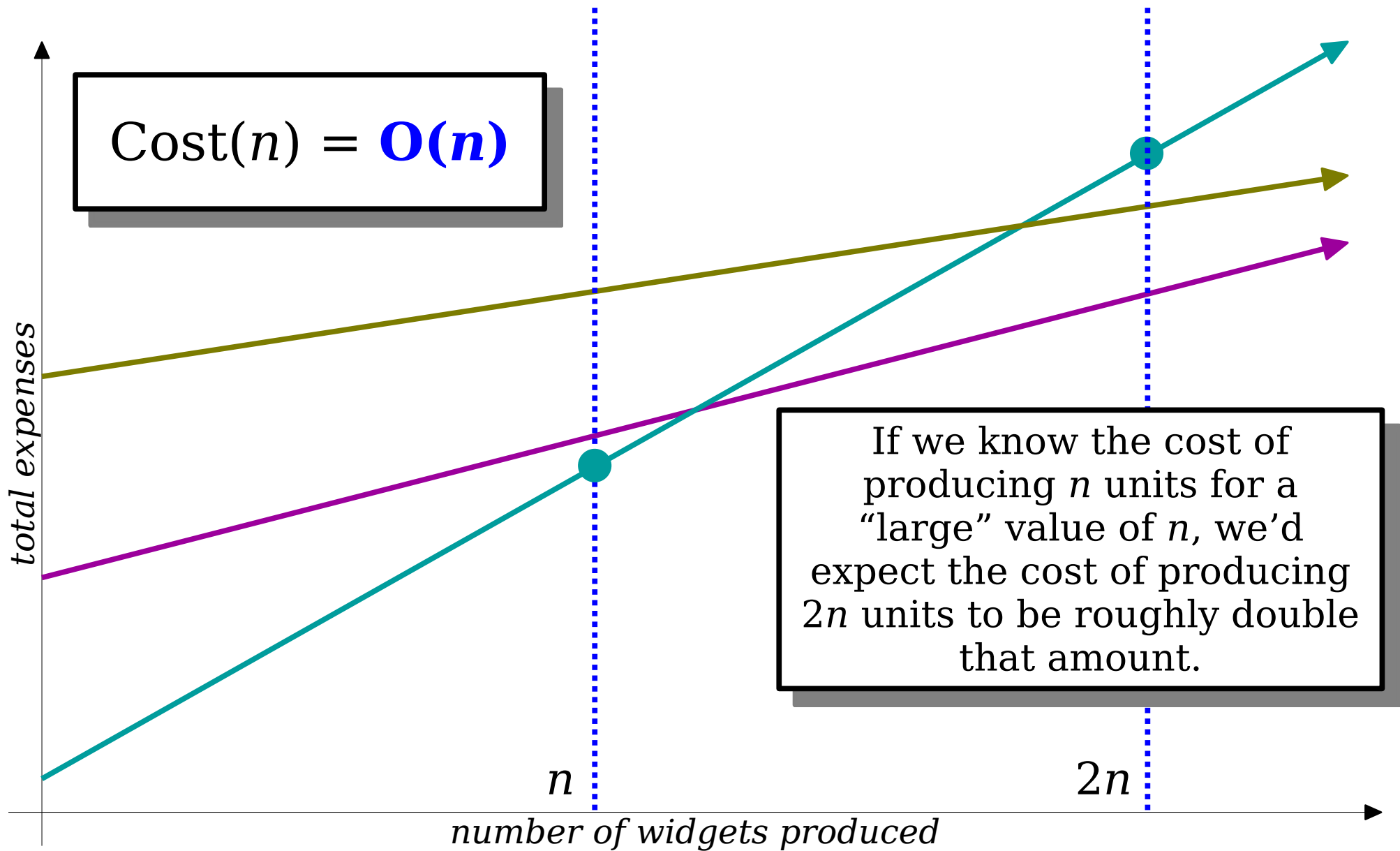
Making Widgets



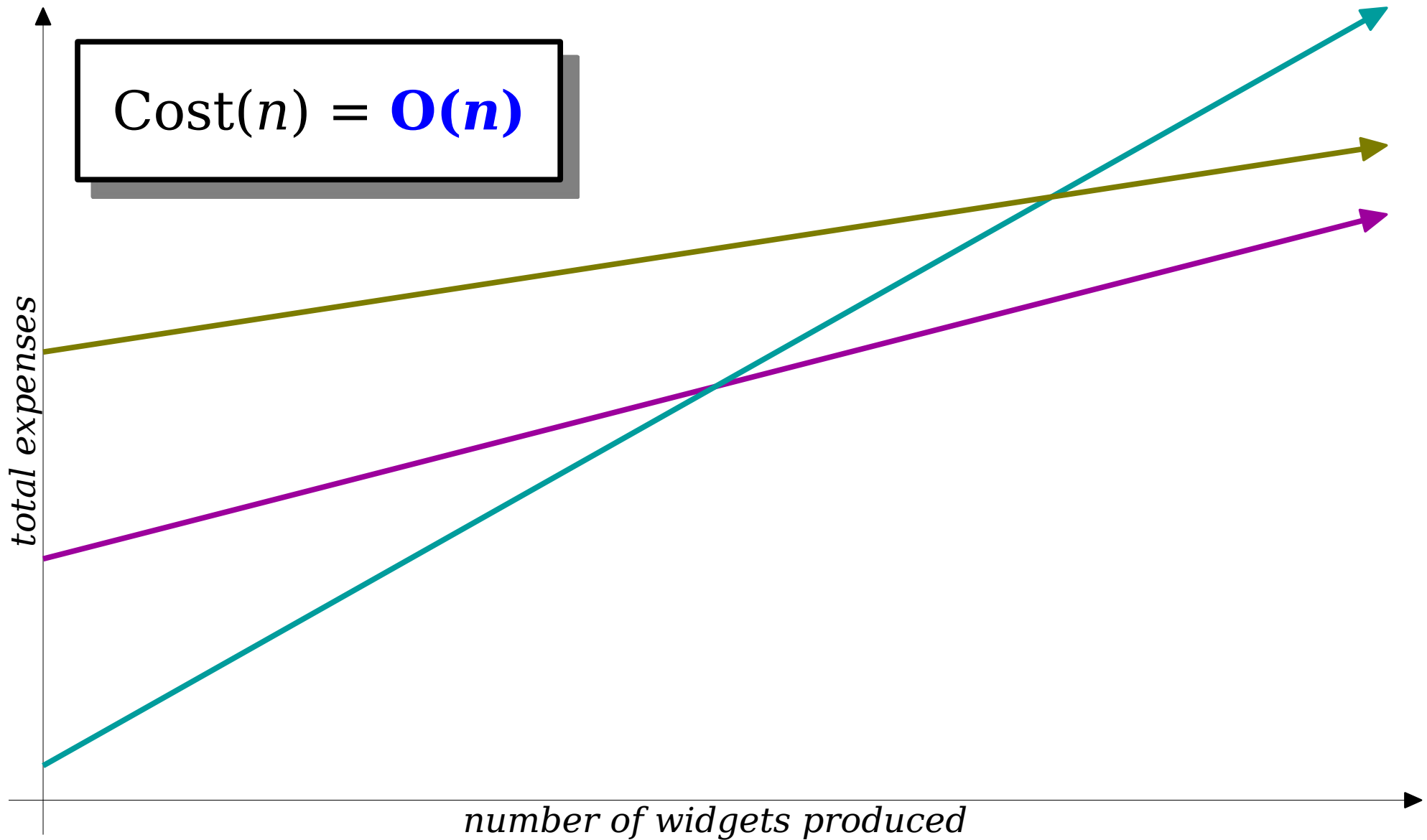
Making Widgets



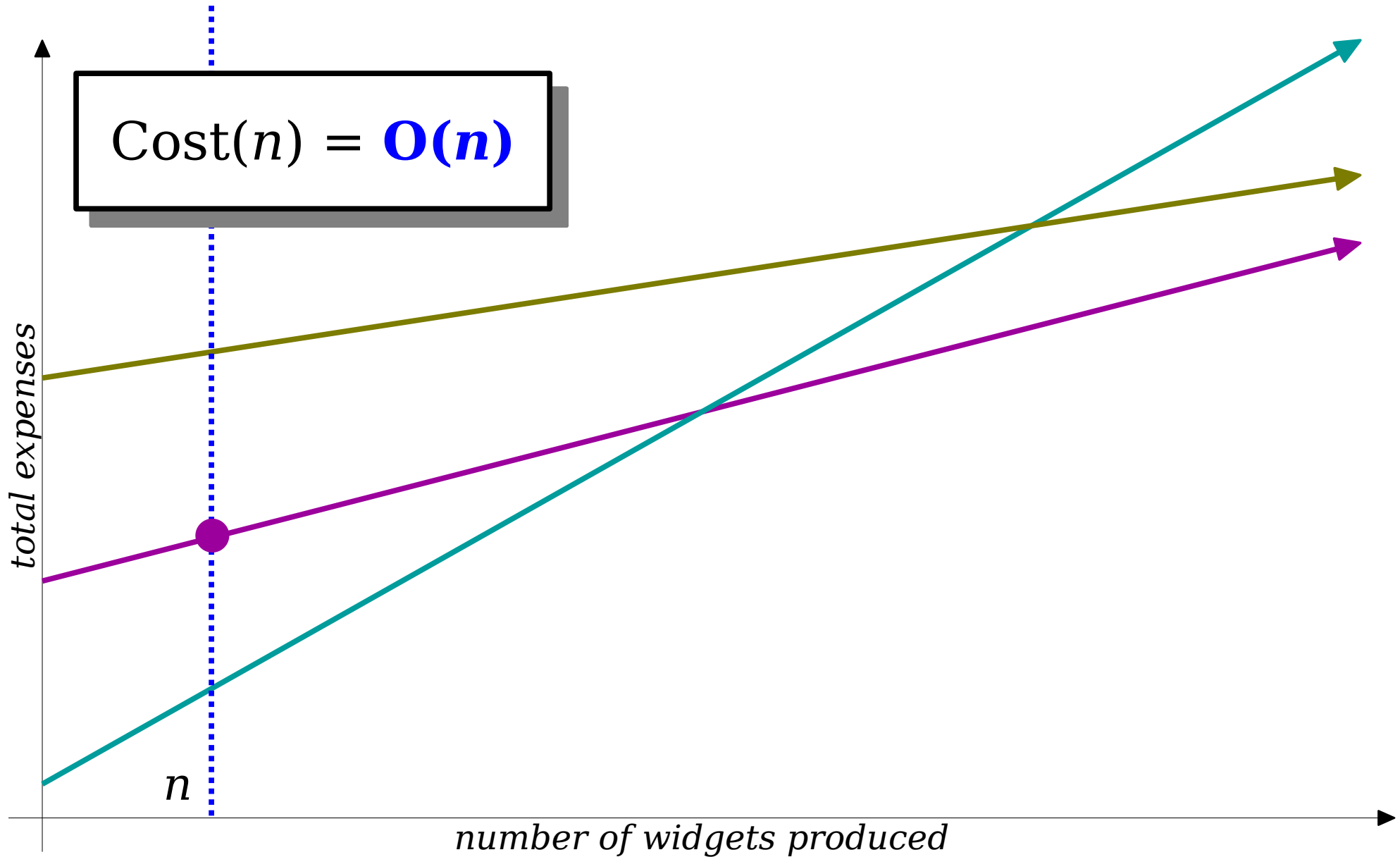
Making Widgets



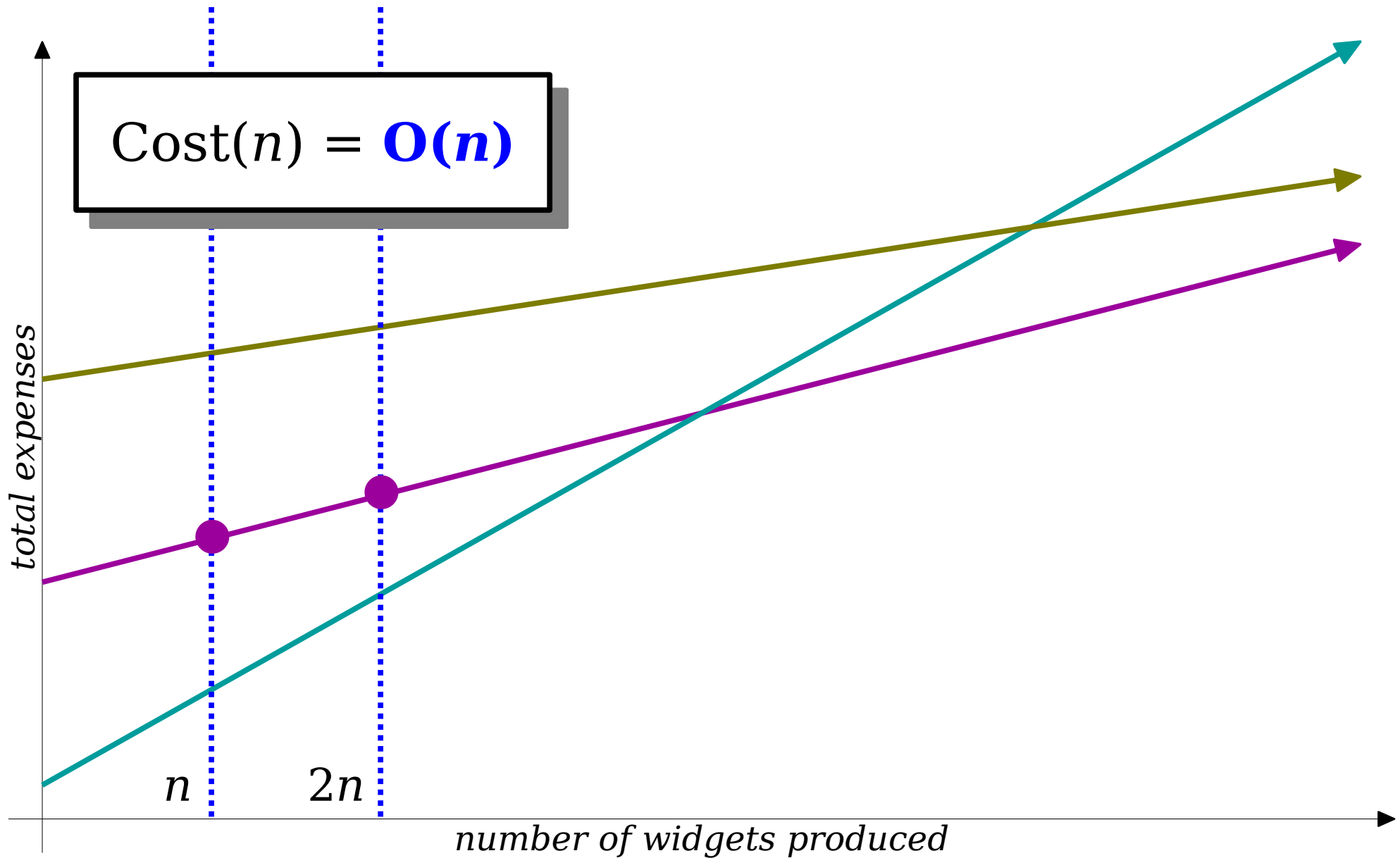
Making Widgets



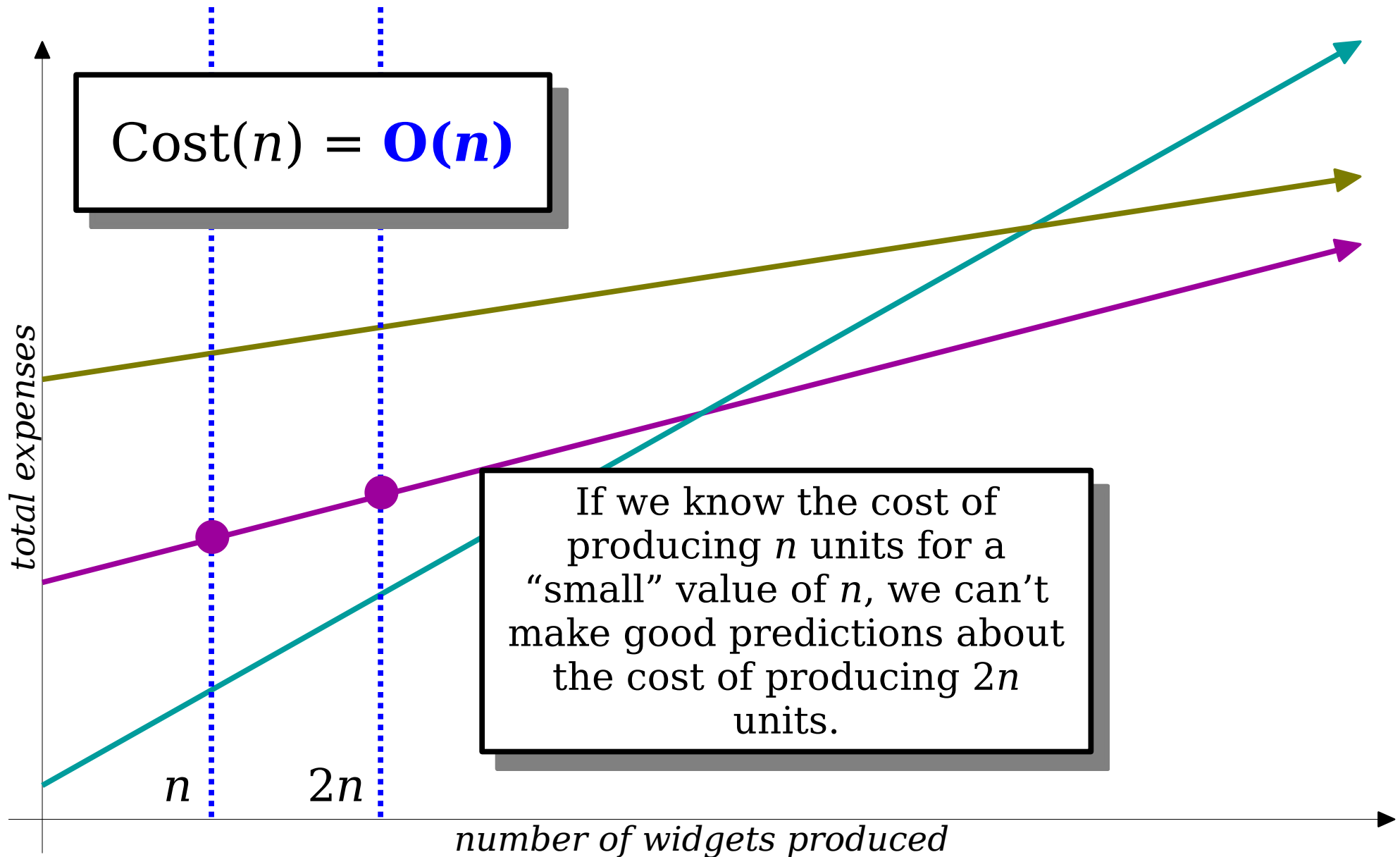
Making Widgets



Making Widgets



Making Widgets



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.

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

Time-Out for Announcements!

Assignment 4

- Assignment 4 (***Recursion to the Rescue!***) is due this Friday.
 - If you're following our timetable, you should have finished the debugging warmup and should be working on Matchmaker right now.
- As always, feel free to ask for help when you need it! Ping us on EdStem, stop by the LaIR, call into our office hours, or email your section leader!

Midterm Exam Logistics

- Our first midterm exam runs from 10:30AM Friday, February 4th to 10:30AM Sunday, February 6th, Pacific time.
- It's a take-home coding exam. It will be designed to take three hours to complete, though you'll have the full 48-hour window to complete it.
- We'll post starter files and instructions along the lines of what we've done for the programming assignments.
- Topic coverage is as follows:
 - Primary focus is on material and concepts from Lectures 00 - 10 and Assignments 0 - 3.
 - Secondary focus is on material and concepts from Lectures 11 - 13 and Assignment 4.
- Neel is holding a review session here in Hewlett 200 from 7:30PM - 9:30PM this Wednesday, February 2nd.

fg

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

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,
 - 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;
```

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

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

```
}
```

Is this useful?
What does that
tell us?

One possible answer: $3n + 4$.

```
double averageOf(const Vector<int>& vec) {
```

```
1 double total = 0.0;
```

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

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

Formulate hypotheses!

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;
    }
}

```

Formulate hypotheses!

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;  
    }  
}
```

Now, share your best guesses with your neighbor.

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;  
    }  
}
```

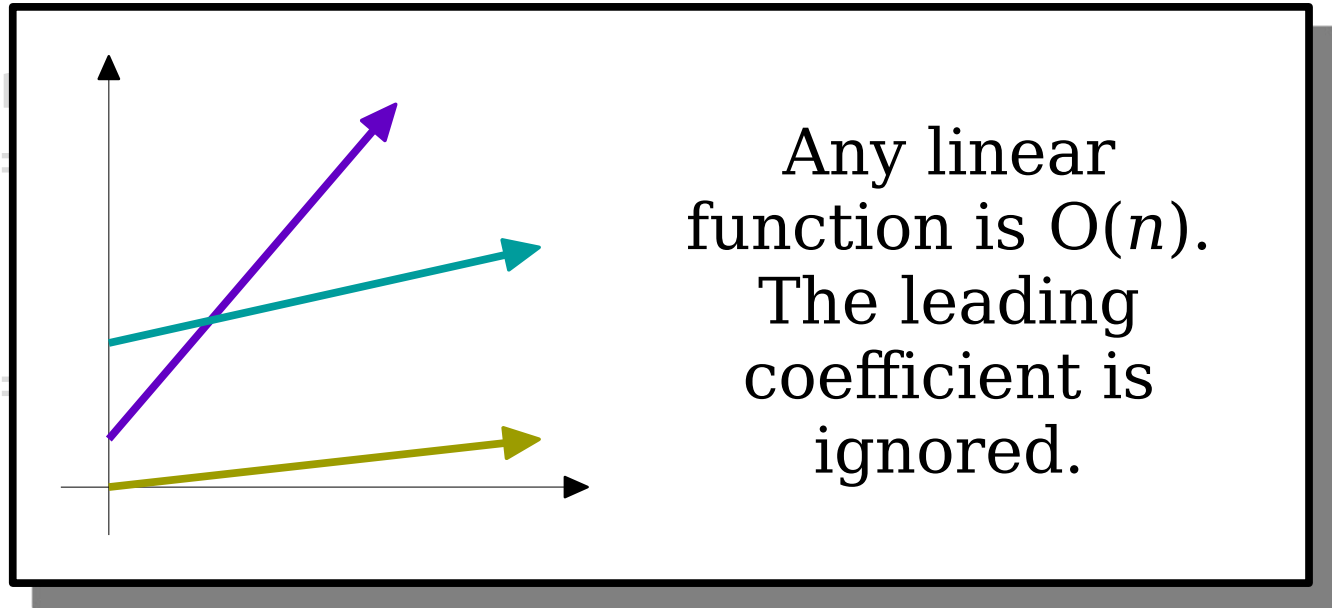
How much time will it take for these functions to run, as a function of n ? Answer using big-O notation.

```
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 5n units of work;  
  
    }  
}
```

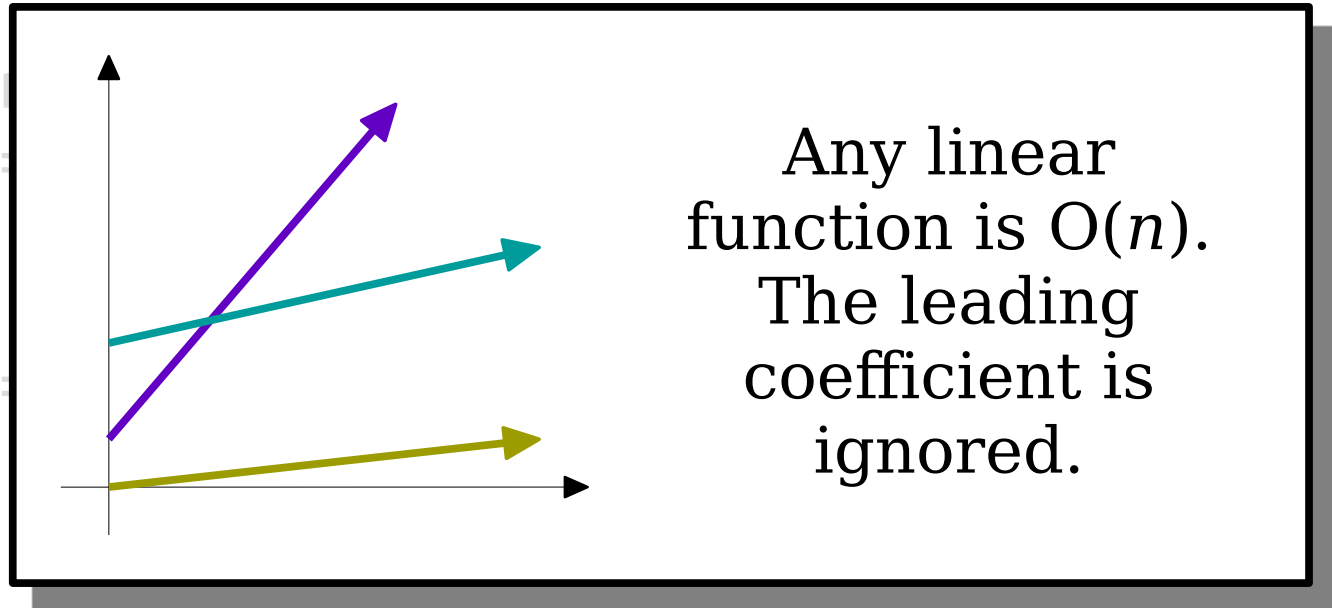
```
void pando(int n) {  
    for (int i = 0; i < n; i++)  
        cout << " ";  
}  
for (int i = 0; i < n; i++)  
    cout << " ";  
}
```



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 0(n) work;  
  
    }  
}
```

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



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  $O(n^2)$  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) {  
    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.

```
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.

```
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.

```
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.


```
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 0(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 0(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 0(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 0(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 0(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 0(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.

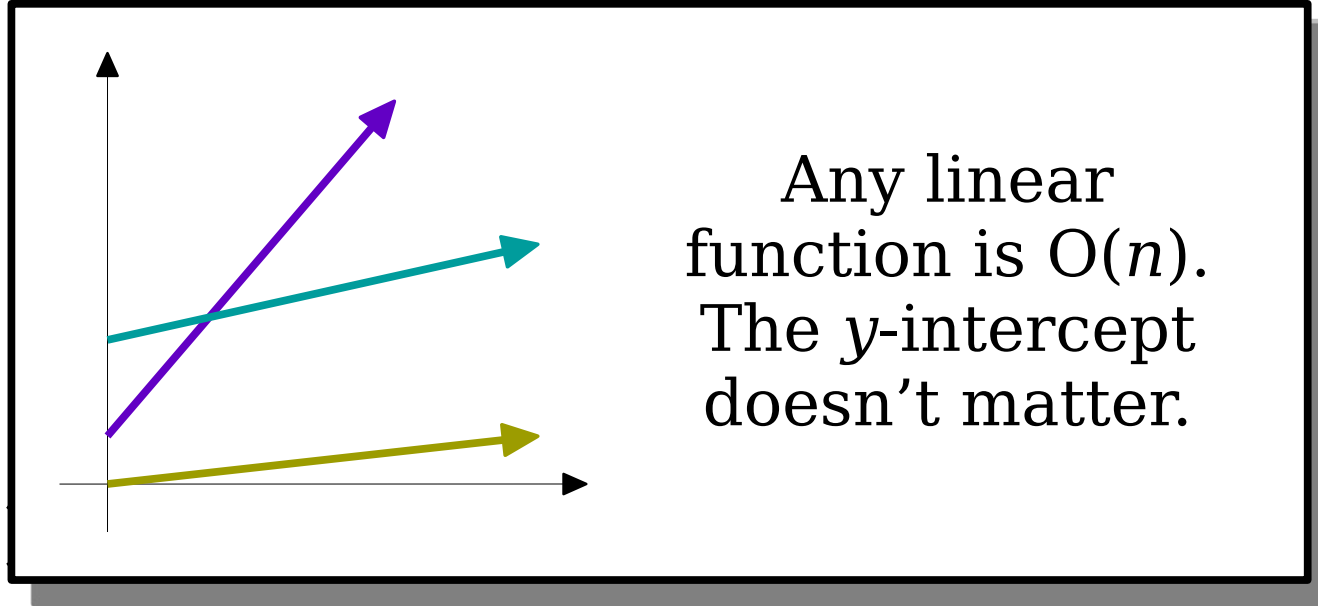
```
void beni(int n) {  
    for (int i = 0; i < n; i++)  
        for (int j = 0; j < n; j++)  
            cout << " ";  
}
```

```
void pando(int n)
```

```
    do 0(n) units of work;
```

```
    do 8 units of work;
```

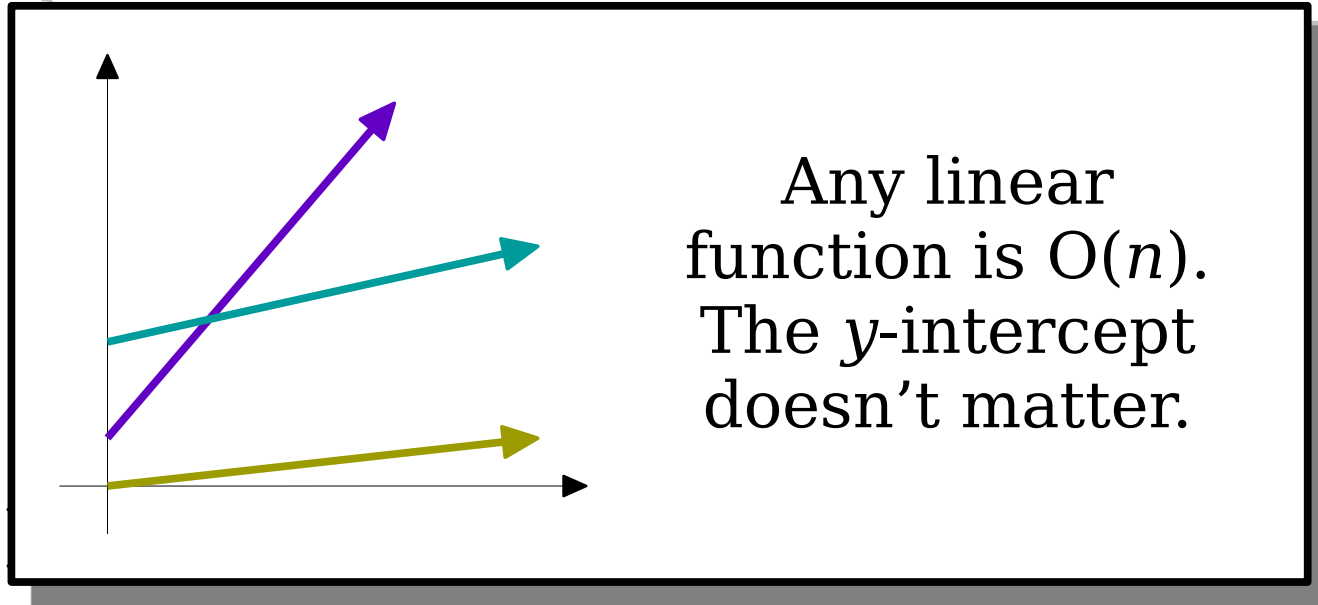
```
}
```



Any linear
function is $O(n)$.
The y -intercept
doesn't matter.

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 < n; i++)  
        for (int j = 0; j < n; j++)  
            cout << " ";  
}  
}  
  
void pando(int n)
```



do $O(n)$ 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) {  
    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.

```
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.

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.
- ***Keep Working on Assignment 4.***
 - If you want to follow our suggested timetable, aim to complete the debugging exercise and Matchmaker by next time and start working on Disaster Planning.

Next Time

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