Unit 6: Algorithms

Unit 6 - Lesson 1
Algorithms Solve Problems

Warm Up

● ○ ○

Prompt:

What makes two pieces of code “the same”?

Could there ever be two pieces of code that you consider to be “the same” even if they aren’t identical?

Activity

●●○

Go around the room and write down your answers to these problems.

Problems

1. Find a person whose birthday is before yours
2. Find a person whose birthday is after yours
3. Find the person whose birthday is the closest before yours
4. Find the person whose birthday is the closest after yours
5. Find the person whose birthday is closest to yours
6. Find the person with an equal number of birthdays before and after theirs
7. Find the two people with the closest birthdays in the room
8. Find the shortest period of time in which three people have birthdays
9. Find the shortest period of time in which four people have birthdays
10. Find the longest period of time in which no one has a birthday

Prompt:

Share with someone how you went about solving each of these problems.

Which problems did you need to do something similar in order to solve them?

Unit 6 Lesson 1 - Activity

With your partner decide which of these programs are “the same” as one another.

Algorithm 1

```
MOVE_FORWARD()
TURN_RIGHT()
MOVE_FORWARD()
TURN_RIGHT()
MOVE_FORWARD()
TURN_RIGHT()
MOVE_FORWARD()
TURN_RIGHT()
```

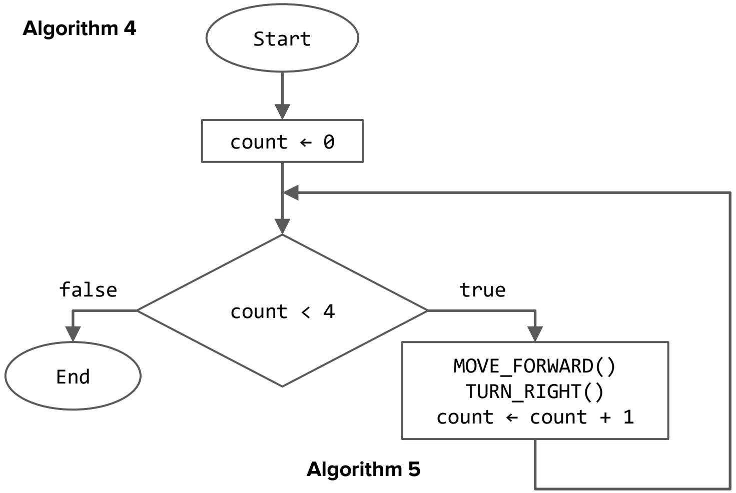
Algorithm 2

```
REPEAT 2 TIMES
{
    MOVE_FORWARD()
    MOVE_FORWARD()
    TURN_RIGHT()
    MOVE_FORWARD()
    TURN_RIGHT()
}
```

Algorithm 3

```
moves ← [“F”, “R”, “F”, “R”, “F”, “R”, “F”, “R”]
FOR EACH move IN moves
{
    IF (move = “F”)
    {
        MOVE_FORWARD()
    }
    ELSE
    {
        TURN_RIGHT()
    }
}
```

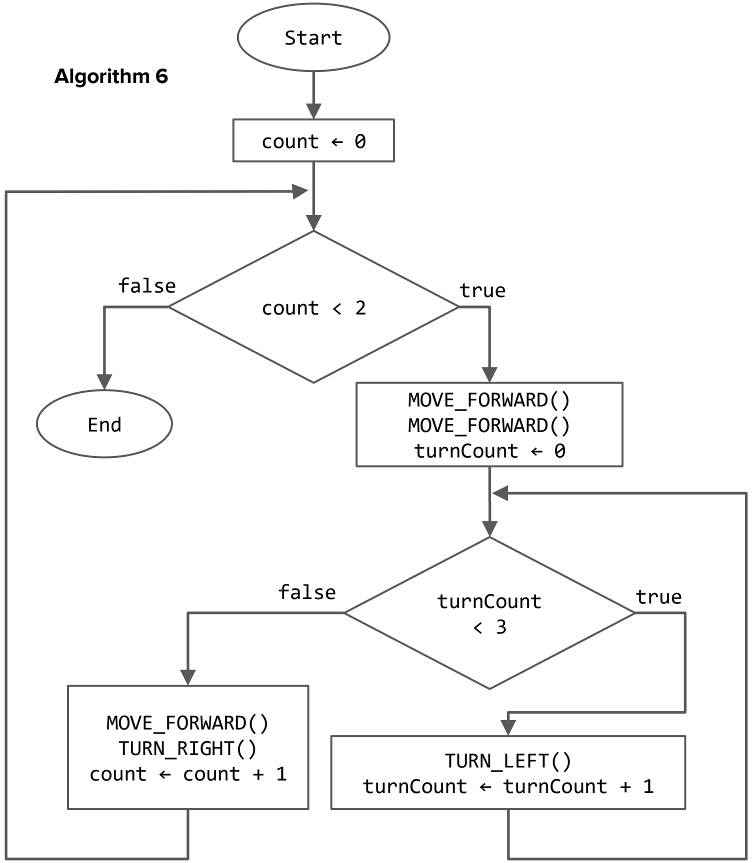
Algorithm 4



Algorithm 5

```
REPEAT 2 TIMES
{
    REPEAT 2 TIMES
    {
        MOVE_FORWARD()
    }
    REPEAT 3 TIMES
    {
        TURN_LEFT()
    }
    MOVE_FORWARD()
    REPEAT 3 TIMES
    {
        TURN_LEFT()
    }
}
```

Algorithm 6



Prompt:

Discuss with another group which of these algorithms are “the same” as one another?

How did you decide that?

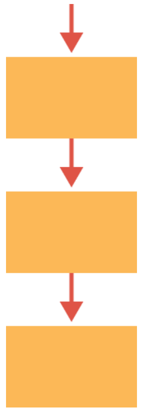
Wrap Up

Unit 6 Lesson 1 - Wrap Up

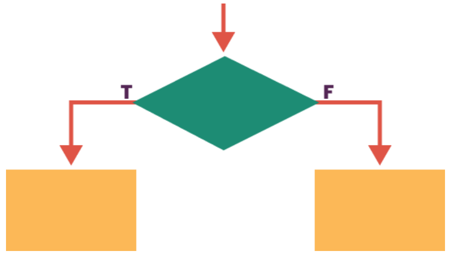
Problem: a general description of a task that can (or cannot) be solved with an algorithm
Algorithm: a finite set of instructions that accomplish a task.

There are usually many algorithms to solve the same problem, and many ways to write or express one algorithm including natural language, psuedocode, diagrams, and are implemented using programming code. All algorithms can be created by combining steps in three different ways.

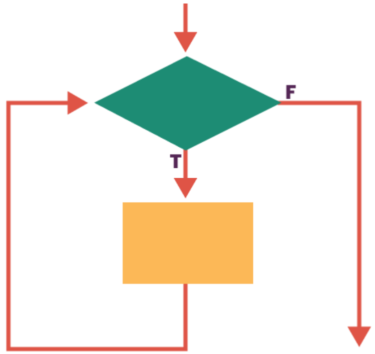
Sequencing
Putting steps in an order



Selection
Deciding which steps to do next



Iteration
Doing some steps over and over



Prompt:

How did today's activities change the way you think about algorithms and problems?

Unit 6 - Lesson 2
Algorithm Efficiency

Warm Up

● ○ ○

Prompt:

Have you ever lost a pencil in a backpack? What are the steps you take to find the pencil?

Activity

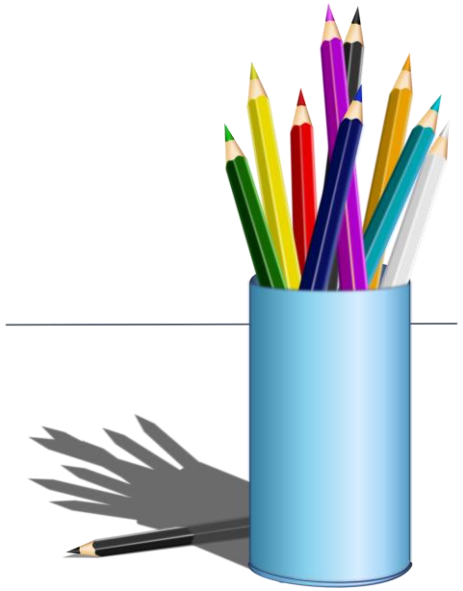
● ● ○

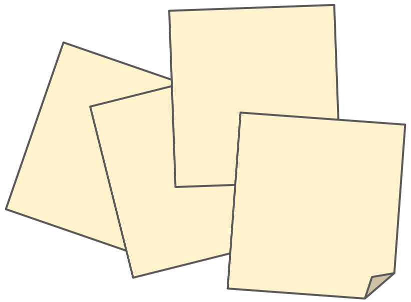
Algorithm Efficiency

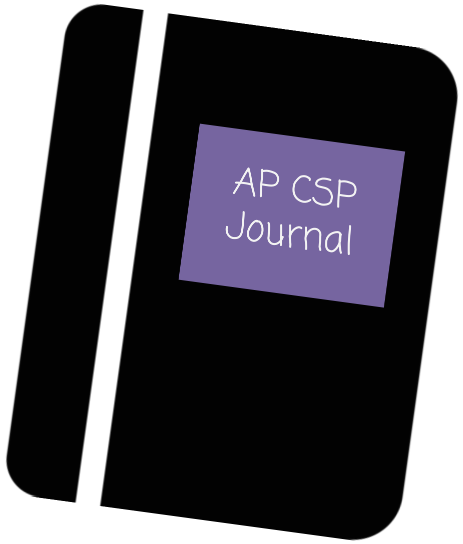
You and your partner should have:

Sticky Notes

Your Journal



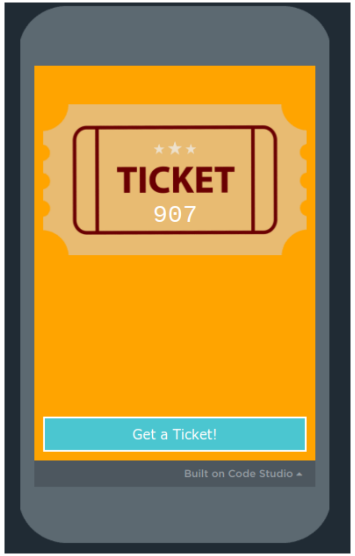


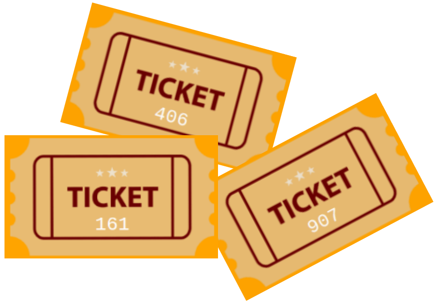


Unit 6 Lesson 2 - Activity

3 Volunteers:

- Navigate to the second level on Code Studio.
- Click to generate your raffle ticket number.
- Write the number on a sticky note.
- Come up to the front of the room.



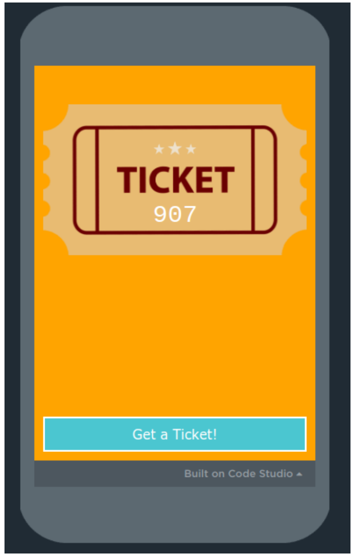


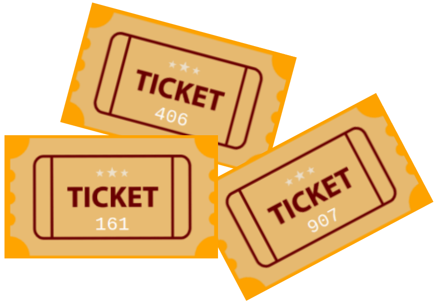
Problem: Figure out if anyone has the winning raffle ticket

Instance: A list of tickets plus the winning number

Let's check if anyone has the winning number!

Do This: Check for the winning number, by revealing your numbers, one by one.





Prompt:

How many steps did it take to find out if anyone had the winning ticket? What is the greatest possible number of steps it could take for this instance?

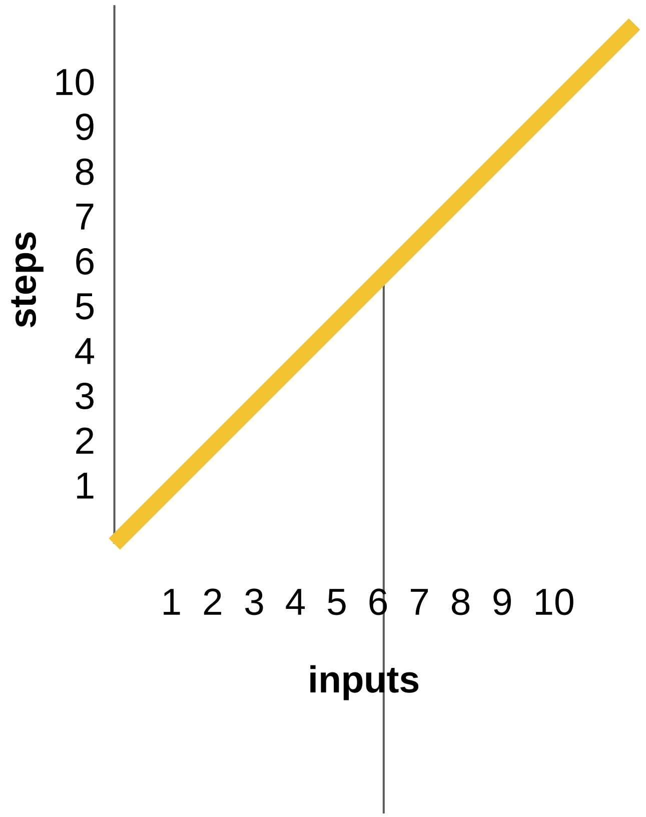
Prompt:

What if we had six volunteers? The whole class? The whole school?

What is the pattern here?

Unit 6 Lesson 2 - Activity

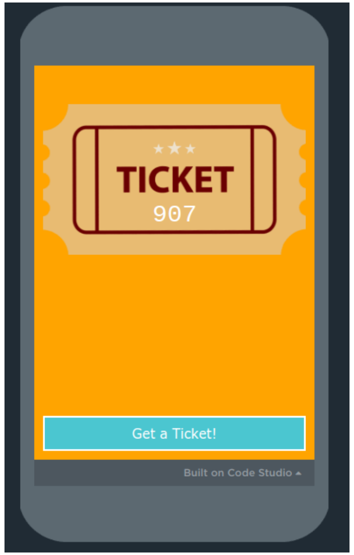
	inputs	steps
Instance	3	3
Instance	6	6
Instance	10	10
Instance	100	100

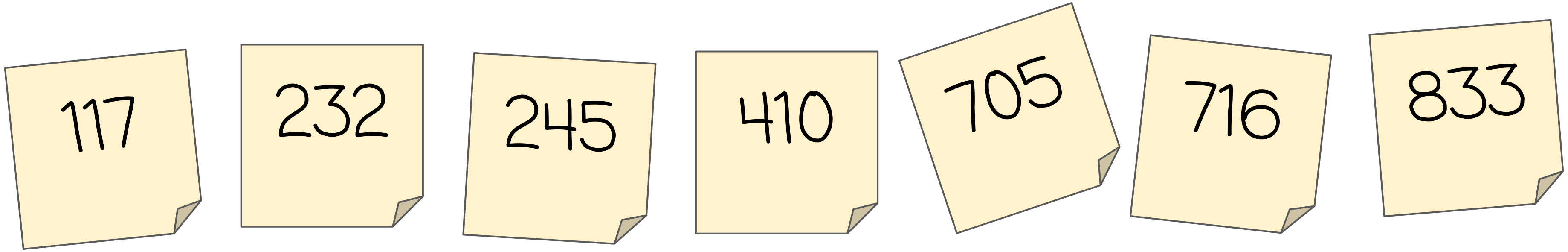


With a partner...

Do This:

- Use the ticket generator to generate seven tickets. Write down the numbers on separate sticky notes.
- Organize the sticky notes in numerical order.
- Copy one of the numbers on to a separate sticky note. This is the number you are searching for.





Challenge: Create an algorithm to determine if a given number is in a list of sorted numbers.

- The search can start at any of the sticky notes
- You can "jump" over sticky notes. In other words, you don't need to search the stickies in order.
- You can determine which sticky notes to search next based on the current sticky note you are checking.
- The goal is to make the determination in the least steps possible, but don't forget your number could be anywhere in the list - what is the worst possible case? What is the greatest number of comparison steps it would take to find any number in your list using your current algorithm?

Share & Compare: Partner up with another group. Share your algorithms and practice running both fully. Determine which one is "faster" or takes the least amount of comparison steps.

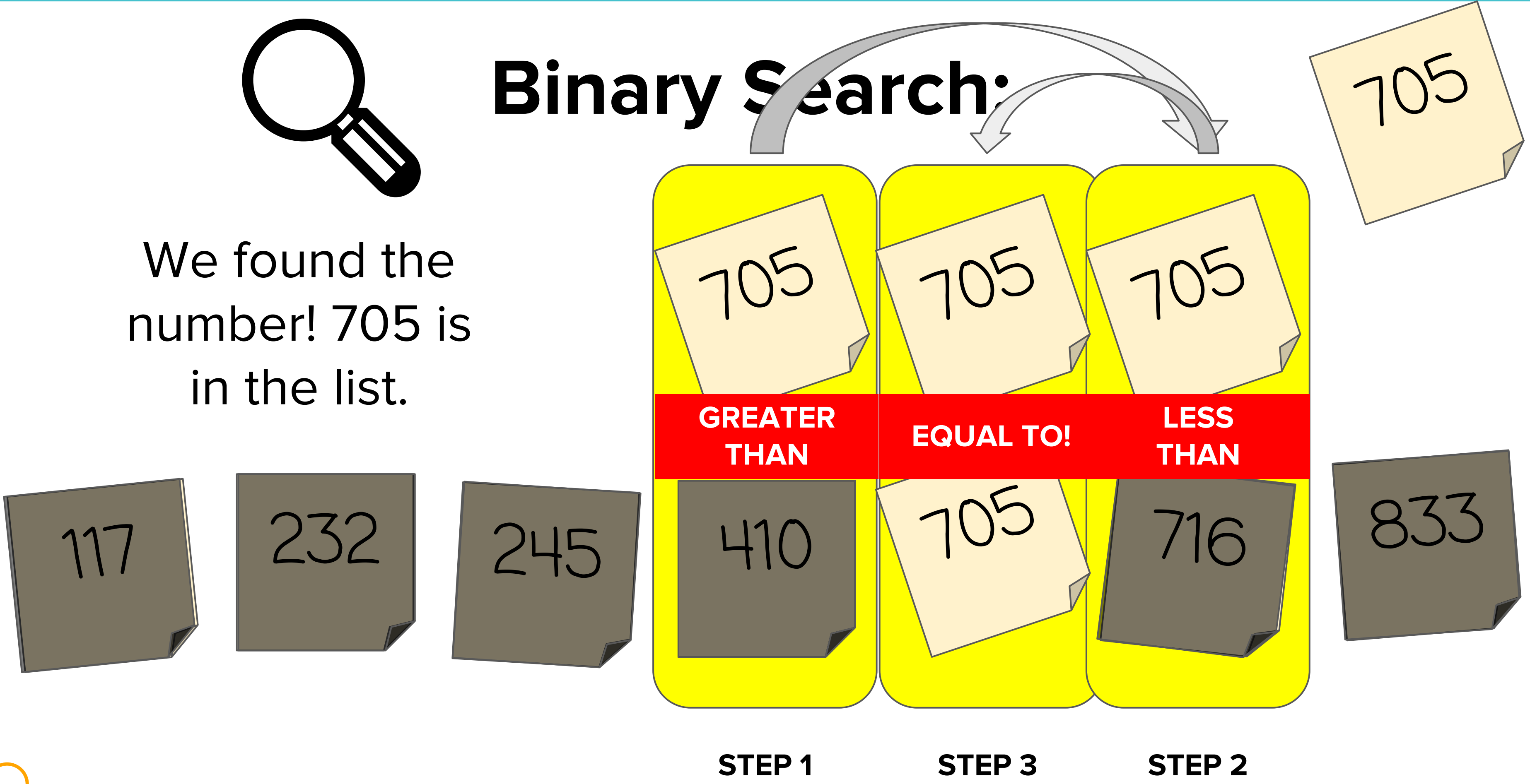
Do This: Now try this algorithm. See how it compares to your own.

1. Find the middle number in the list. Compare that number to the given number. If the number is less than the given number, remove all of the cards to the right (including the middle number). If the number is more than the given number, remove all of the cards to the left (including the middle number).
2. Find the middle number in this shorter list. Follow the instructions in Step #1 for comparing.
3. Find the middle number in this new shorter list. Follow the instructions in Step #1 for comparing.

You have found your number!

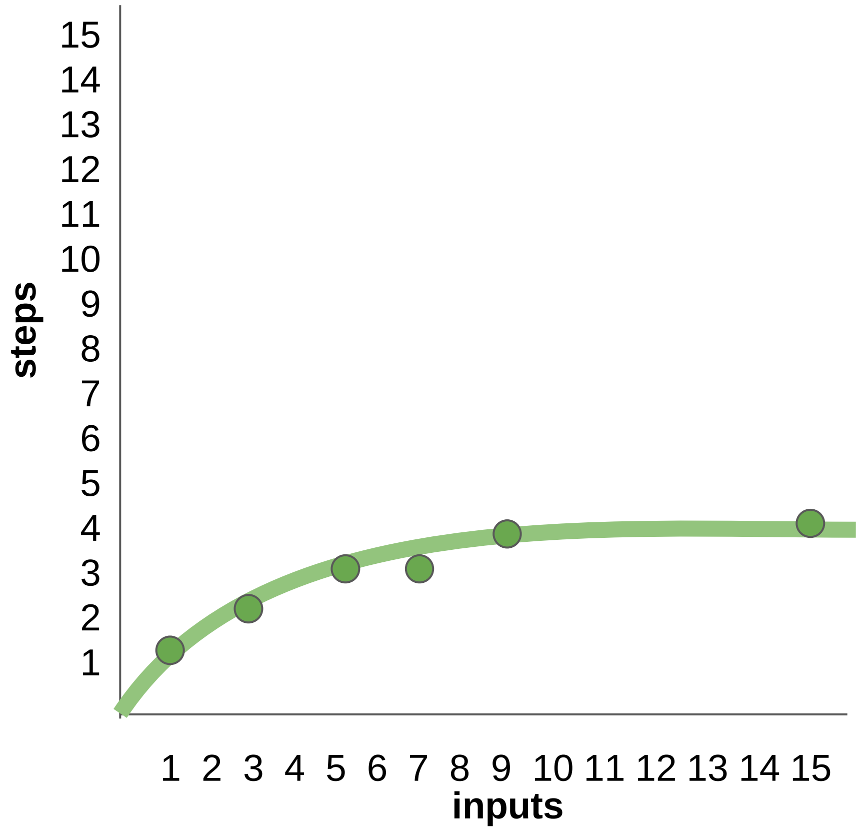
Binary Search:

We found the number! 705 is in the list.



Do This: Now try the Binary Search algorithm for different instances. We've done a few for you! Copy the instance table to your journal and plot the points on the graph.

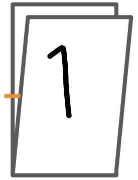
	inputs	steps
Instance	1	1
Instance	3	2
Instance	5	3
Instance	7	3
Instance	9	4
Instance	15	4



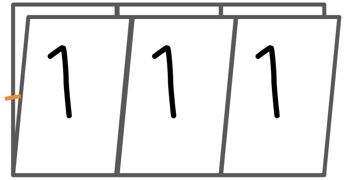
There's another way of thinking about this. Can you see the pattern?

	inputs	steps
Instance	1	1
Instance	3	2
Instance	5	3
Instance	7	3
Instance	9	4
Instance	15	4

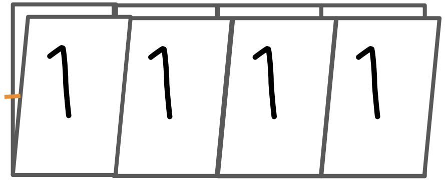
How many bits does it take to represent 1 in Binary?

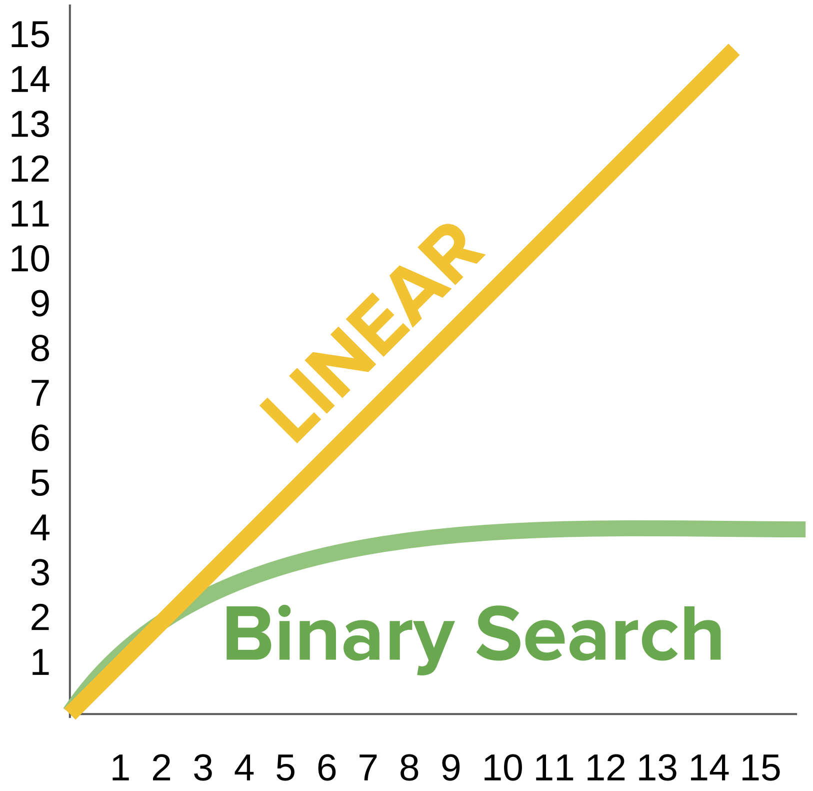


What about 7?



And 15?





Here's our two search algorithms we've explored. The first is **linear**. As we add more inputs, the number of steps grows at the same rate.

The second represents what happens with Binary Search. Notice how it grows at a much slower rate! Binary Search is faster than Linear search, BUT the data must be sorted.

Wrap Up

Prompt:

If I had one input, which algorithm would I use to get my answer with the fewest amount of steps?

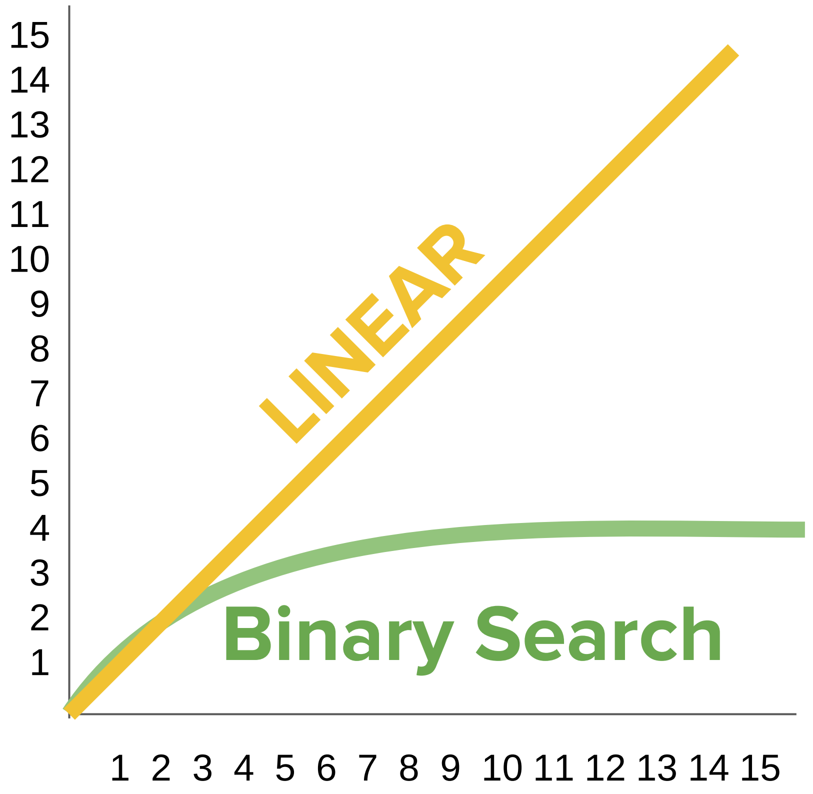
What if I had five?

What about one hundred?

Efficiency: a measure of how many steps are needed to complete an algorithm

Linear Search: a search algorithm which checks each element of a list, in order, until the desired value is found or all elements in the list have been checked.

Binary Search: a search algorithm that starts at the middle of a sorted set of numbers and removes half of the data; this process repeats until the desired value is found or all elements have been eliminated.



Unit 6 - Lesson 3
Unreasonable Time

Warm Up

● ○ ○

Prompt:

What does it mean to say one algorithm is “more efficient” than another?

Activity

● ● ○

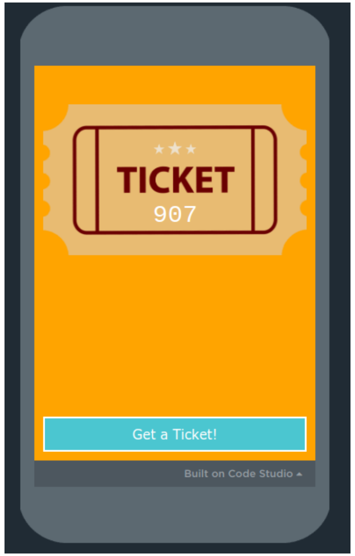
The Pair Raffle

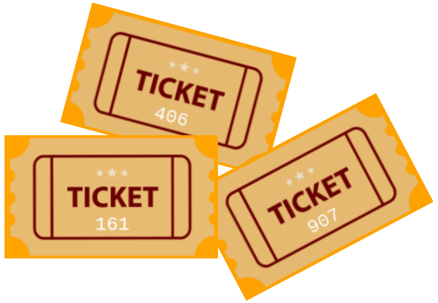
The winners are any two tickets that adds to the winning number.

The winning number is 1000.

Do This

Generate a ticket
Silently move around the room.
See if you're a part of a winning pair!





The Group Raffle

The winners are any group of (from one ticket up to all of them) that adds to the winning number.

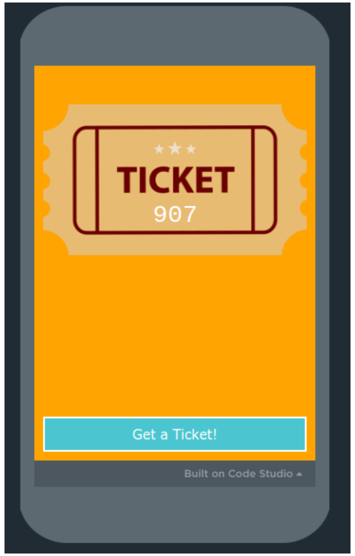
The winning number is 2500.

Do This

Generate a ticket

Move around the room (you can talk this time

See if you're part of a winning group!



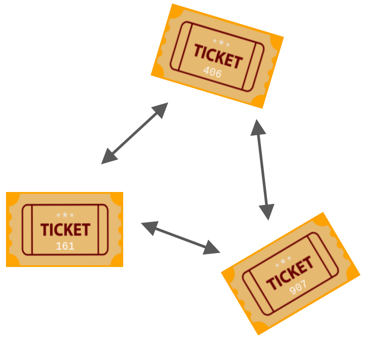
Prompt

Which raffle felt like it was more difficult to check? Why?

We could write an algorithm that goes through every possible “check” for the pair raffle or the group raffle.

Let’s see how many checks there are!

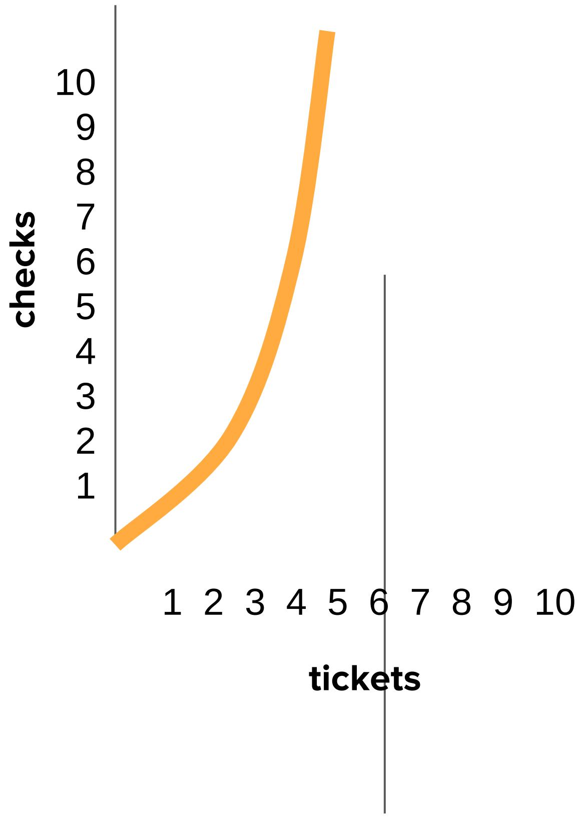
With your partner fill in the two tables on the activity guide.



Share your responses with another group!

Unit 6 Lesson 3 - Activity

tickets	checks
2	1
3	3
4	6
5	10
8	28



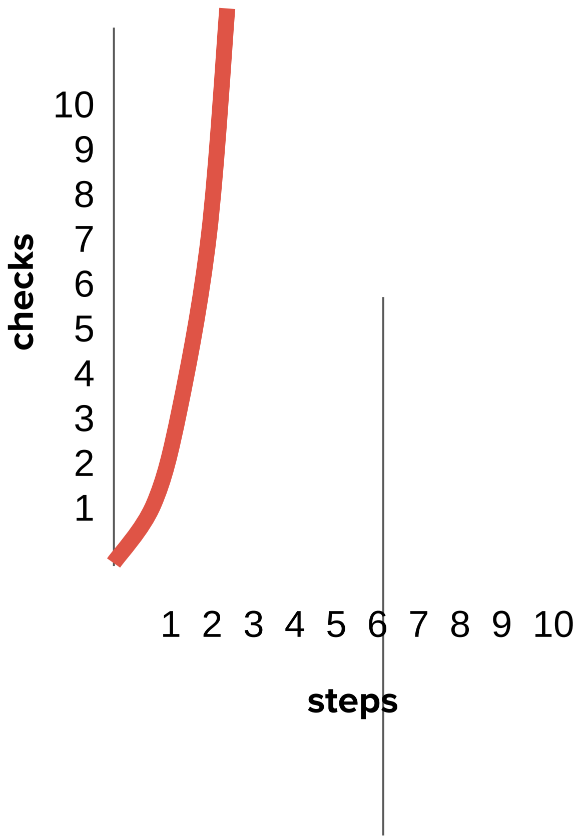
The exact formula for this relationship is

$$(n^2 - n)/2$$

You don't need to know that formula, but you should know that because of the "n-squared" term the graph curves up.

Any algorithm whose efficiency includes an n^2, n^3, n^4 … is called **polynomial.**

tickets	checks
2	3
3	7
4	15
5	31
8	255



The exact formula for this relationship is

$$(2^n) - 1$$

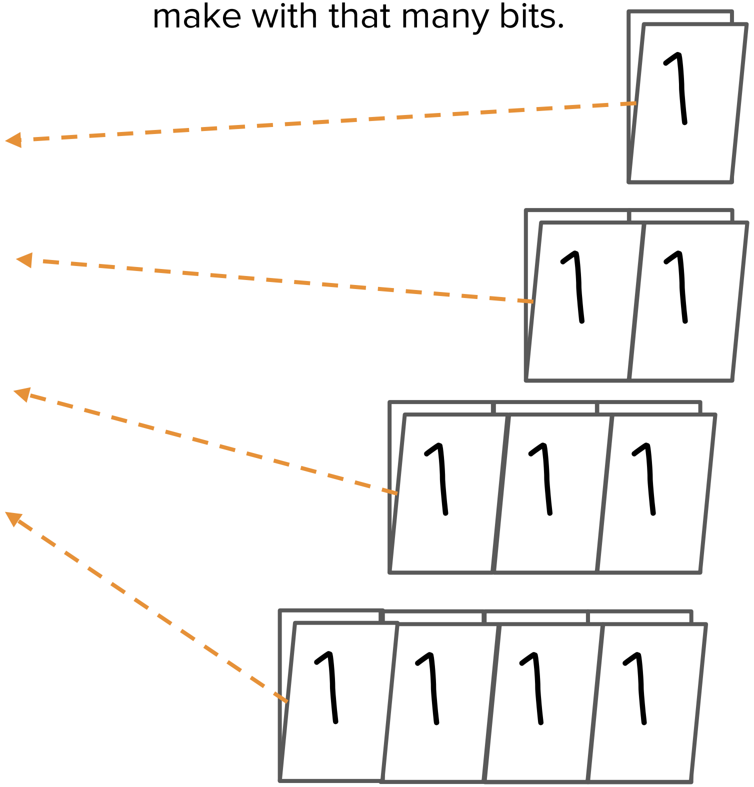
You don't need to know that formula, but you should know that because of the "2 to the n" term the graph curves up very quickly.

Any algorithm whose efficiency includes an 2^n, 3^n, 4^n ... is called **exponential.**

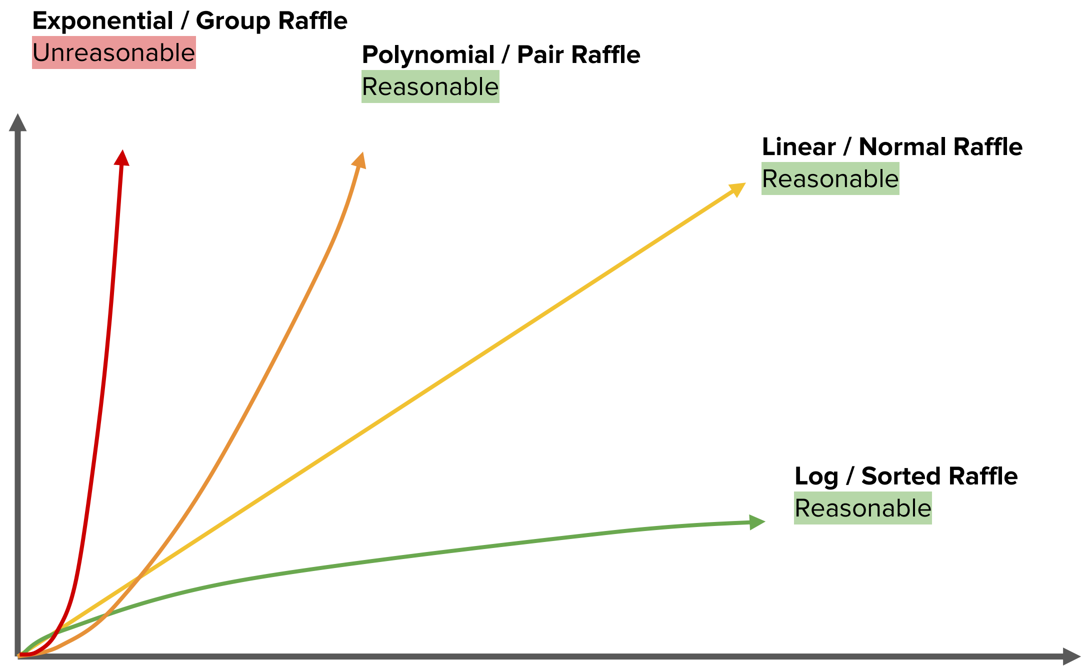
There's another way of thinking about this.

tickets	checks
1	1
2	3
3	7
4	15
5	31
8	255

The number of checks is the largest number you can make with that many bits.



Polynomial and Exponential both curve up. Why do you think only exponential is considered “unreasonable”?



Unit 6 Lesson 3 - Activity

Tickets	Sorted Raffle log	Normal Raffle linear	Pair Raffle polynomial	Group Raffle exponential
10	4 Checks	10 checks	100 checks	1,024 checks
20	5 checks	20 checks	400 checks	1,048,576 checks
100	7 checks	100 checks	10,000 checks	$1.26 * 10^{30}$ checks
1000	10 checks	1,000 checks	1,000,000 checks	$1.07 * 10^{301}$ checks
10,000	14 checks	10,000 checks	100,000,000 checks	$2.00 * 10^{3010}$ checks
100,000	17 checks	100,000 checks	10,000,000,000 checks	$9.99 * 10^{30102}$ checks

At this point there are more checks than atoms in the universe

Polynomial is bad but exponential gets unreasonably large extremely quickly.

Wrap Up

Reasonable Time: Algorithms with a polynomial efficiency or lower (constant, linear, square, cube, etc.) are said to run in a reasonable amount of time.

Unreasonable Time: Algorithms with exponential or factorial efficiencies are examples of algorithms that run in an unreasonable amount of time.

Prompt:

Your school is considering running the group raffle at an upcoming assembly to give away a prize.

Write a brief explanation of what advice you would give them.

Unit 6 - Lesson 4
Traveling Salesman

Warm Up

● ○ ○

Prompt:

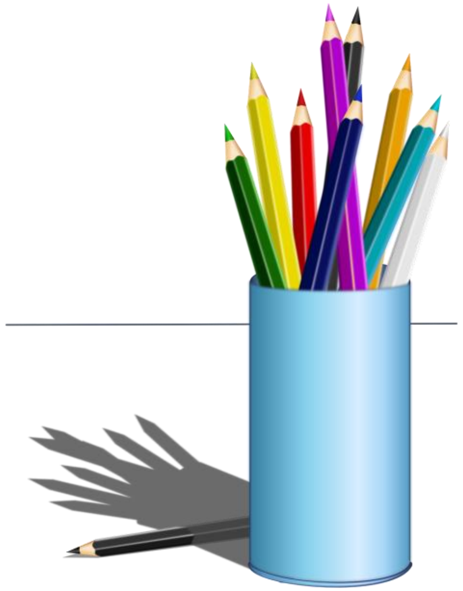
What is the difference between a reasonable and unreasonable time algorithm?

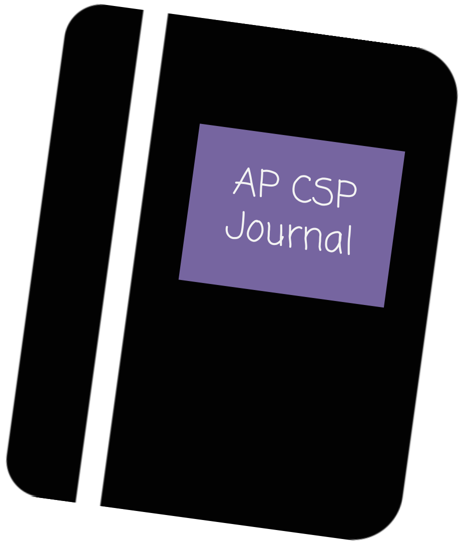
Activity

● ● ○

Traveling Salesman

You should have:
Your journal
Pen/pencil

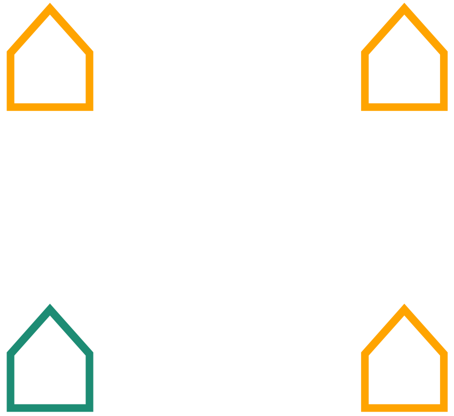


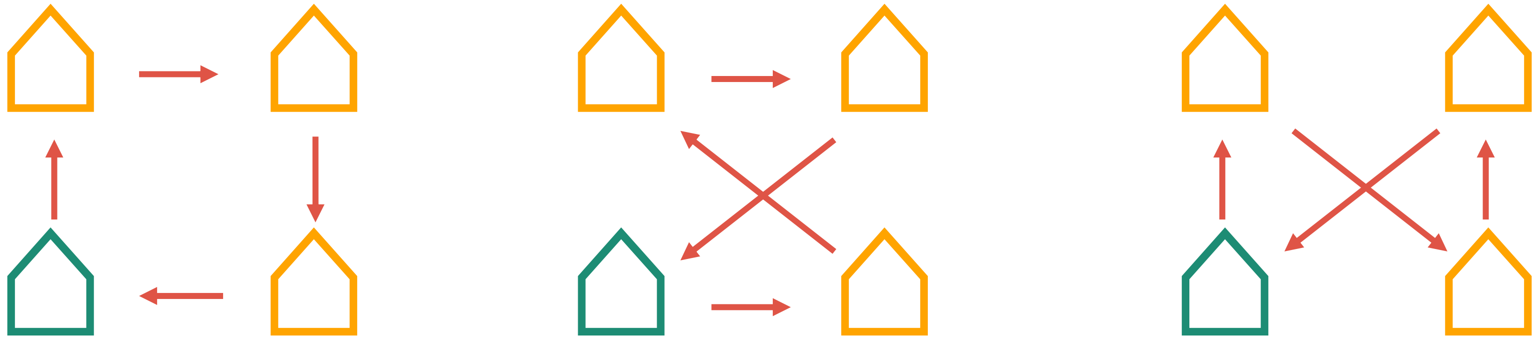


Prompt: How many different paths can you find to visit all of your friends' houses?

Rules:

- You must start and end at your own house.
- You can only visit each house once.

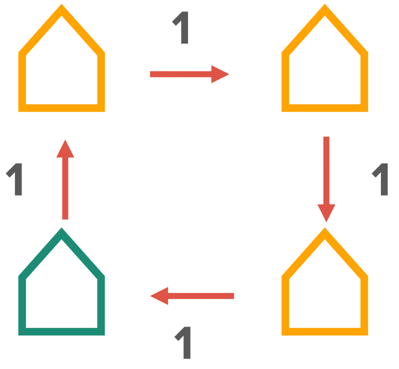




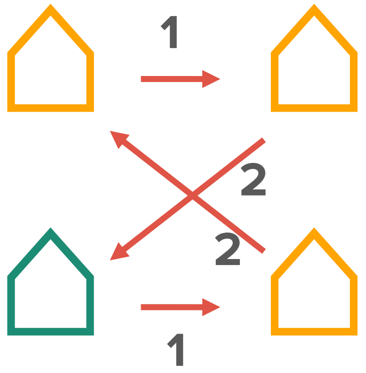
Here are a few different paths you might take.

Prompt:

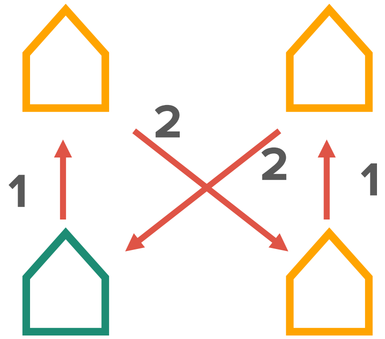
What do you need to know to determine the best path?



Total: 4



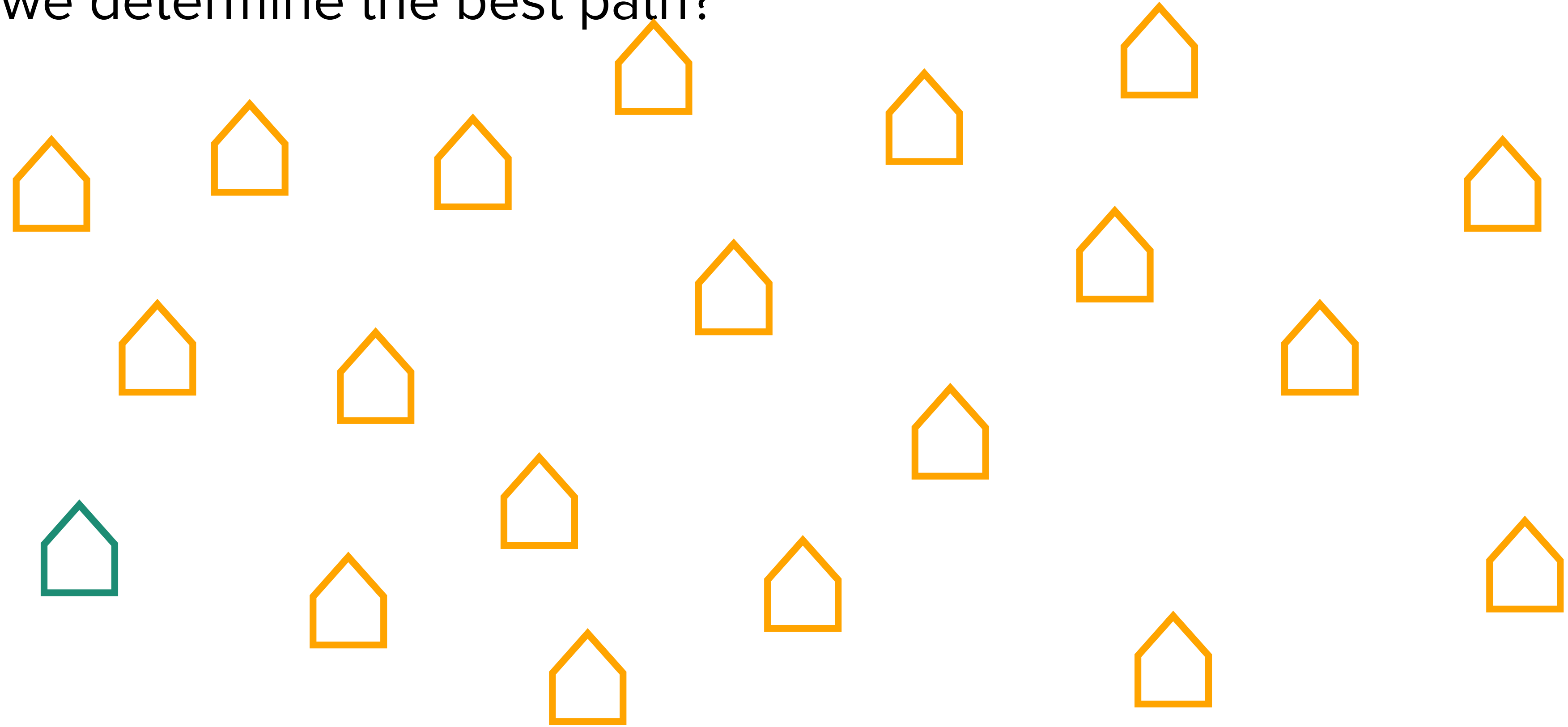
Total: 6



Total: 6

Distance!

Prompt: What if we had a lot more places to visit? How would we determine the best path?



This is known as the **Traveling Salesman Problem**.

For every new place to visit, the number of options for possible paths increases factorialy.

Number of houses to visit	Number of steps to check for the "best" path
1	1
2	2
3	6
4	24
5	120
6	720
7	5,040
8	40,320
9	362,880
10	3,628,800

Factorial fun: n!

Here's how n! works:

Multiply all whole numbers from the given number down to the number 1.

For example:
Instance: 4 houses to visit

4 x 3 x 2 x 1 = 24

Instance: 7 houses to visit

7 x 6 x 5 x 4 x 3 x 2 x 1 = 5,040

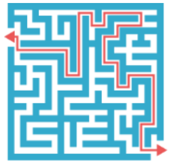
Number of houses to visit	Number of steps to check for the "best" path
1	1
2	2
3	6
4	24
5	120
6	720
7	5,040
8	40,320
9	362,880
10	3,628,800

That's a lot of possible paths to check for only 10 houses!

Problems

Any task that may (or may not) be solved with an algorithm.
Sorting a list is a problem. Sorting the list (2, 3, 1, 7) is an instance of that problem.

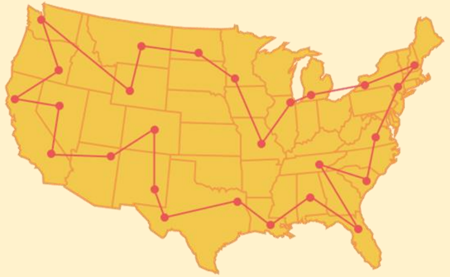
Decision Problems

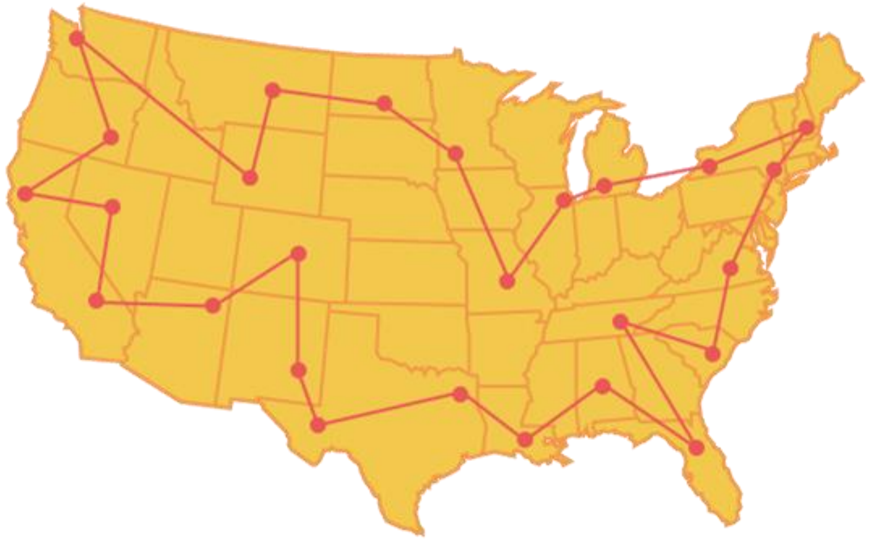


“Is there a path?”

Optimization Problems

“What’s the shortest path”?





The **Traveling Salesman Problem** can be solved with an algorithm, which checks each possible option.

BUT, it would take massive amounts of computing power to compare every single option, especially as the number of homes to visit (otherwise known as *nodes*) increases.

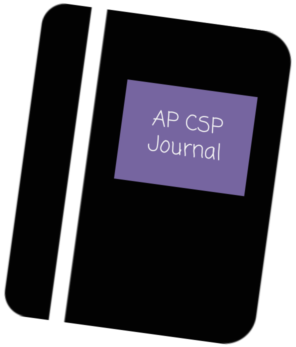
Therefore, it would take an **unreasonable** amount of time for the solution to be calculated for most instances of the problem.

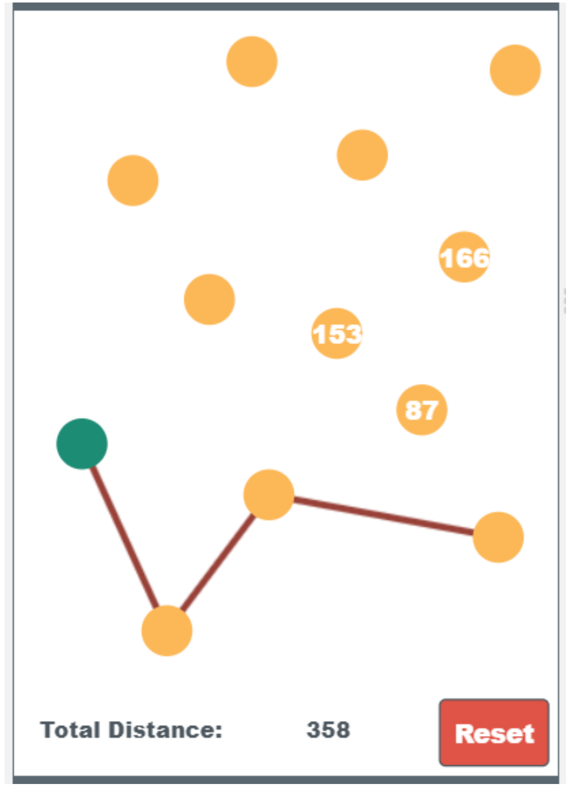
Welcome to **heuristics!**

- Provide a "good enough" solution to a problem when an actual solution is impractical or impossible

Do This:

- Navigate to Level 2 on Code Studio
- Try to find the "best" path to visit all nodes.
- Write down a plan or **heuristic** for choosing a good path.
    - Note: your **heuristic** may not always find the best path, but it should be close enough

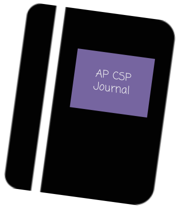




Do This:

- Navigate to Level 3 on Code Studio
- Test your heuristic on three different levels.
- Write down the distance for the path your heuristic finds.
- Try to find the best version not using the heuristic (brute force). Can you find a better path? Is your heuristic on average pretty good? Should you update your heuristic?

Distance (Heuristic)	Distance (Brute Force)
1225	1215



Prompt:

- How did you create your heuristic?
- Did you change your heuristic after testing it out?

Share Out:

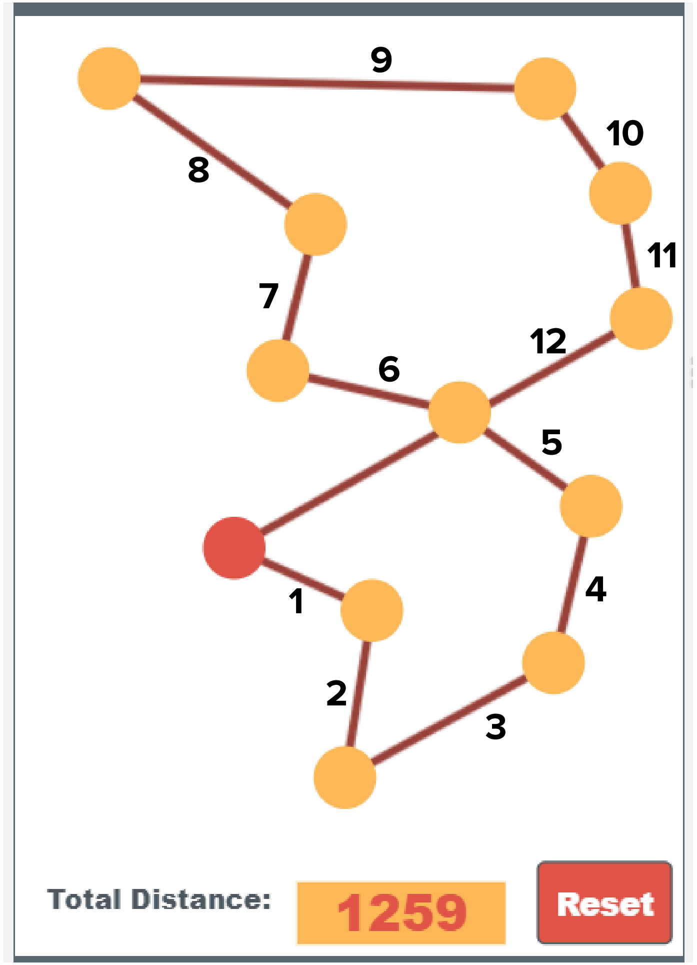
Explain your heuristic.

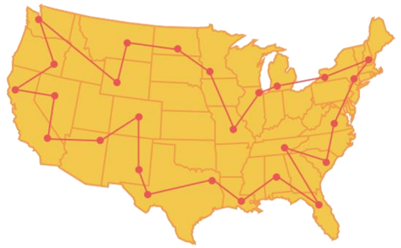
As a class, which do we think is best?

Sample Heuristic:

At each node, travel to the next closest node

Is this "good enough"?





Takeaways:

The Traveling Salesman Problem is an **optimization** problem. We are attempting to find the best path.

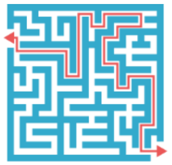
It is also **unreasonable** because there is not an algorithm that can solve the problem in a reasonable amount of time.

We need to use a **heuristic** to come up with a solution that is "good enough" for most instances of the problem.

Problems

Any task that may (or may not) be solved with an algorithm.
Sorting a list is a problem. Sorting the list (2, 3, 1, 7) is an instance of that problem.

Decision Problems



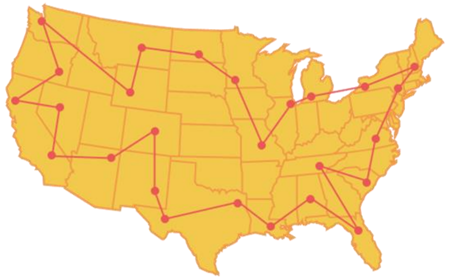
"Is there a path?"

Undecidable Problems

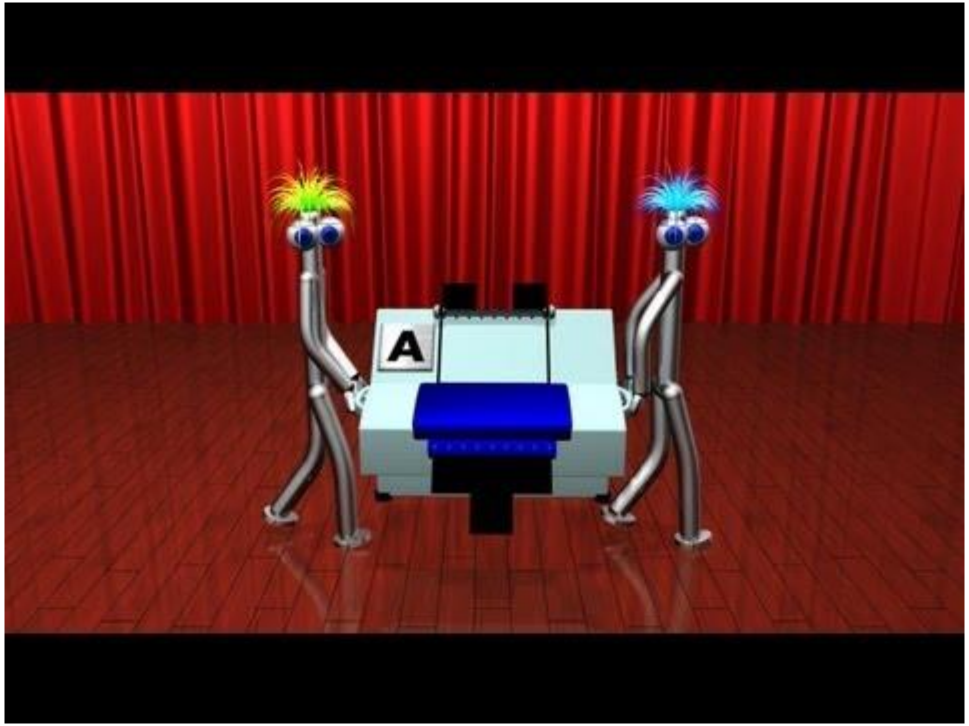
"Will this code work?"

Optimization Problems

"What's the shortest path"?



Unit 6 Lesson 4 - Activity



Takeaways:

There are some problems we’ve proven that no computer will ever be able to solve. The Halting Problem is a very famous example and in general we call these problems **undecidable**.

Wrap Up

Prompt:

Why is a heuristic acceptable when it doesn't always produce the "best" result?

Heuristic: provides a "good enough" solution to a problem when an actual solution is impractical or impossible

Undecidable Problem: a problem for which no algorithm can be constructed that is always capable of providing a correct yes-or-no answer

Unit 6 - Lesson 5
Distributed Algorithms

Warm Up

● ○ ○

Prompt:

Brainstorm a task that you can complete faster if you get other people to help.

What's the most number of people you'd want to help you and why?

Activity

● ● ○

Parallel Algorithms and Speedup

Groups: Get into groups of 3 or 4
Each group should have a deck of cards

Challenge One - One Person Sort

Shuffle the cards
Put them in a neat stack, face down
As quickly as you can, get the cards sorted so all the red cards are at the bottom and all the black cards are at the top.
Time stops when you have the cards sorted and back in a neat stack.

Record the best time in your group

Challenge Two - Two Person Sort

Shuffle the cards
Put them in a neat stack, face down
As quickly as you can, get the cards sorted so all the red cards are at the bottom and all the black cards are at the top.
Time stops when you have the cards sorted and back in a neat stack.

This time two people can sort the cards.
Record the best time in your group

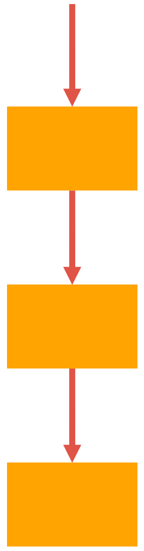
Challenge Three - Full Group Sort

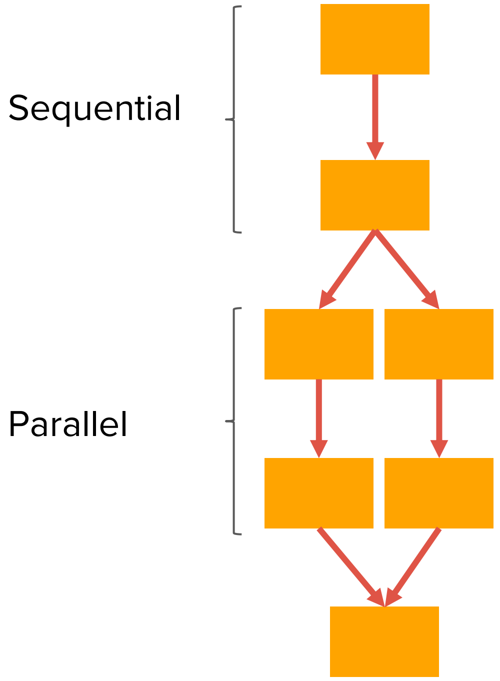
Shuffle the cards
Put them in a neat stack, face down
As quickly as you can, get the cards sorted so all the red cards are at the bottom and all the black cards are at the top.
Time stops when you have the cards sorted and back in a neat stack.

This time your entire group (three or four people) can sort the cards.
Record the best time in your group

Sequential
Steps are performed in order, one at a time.

Parallel
Some steps are performed at the same time.





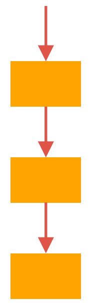
Prompt

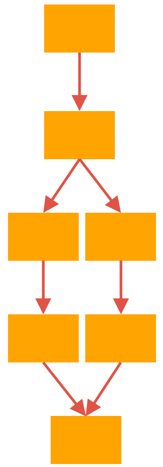
What portions of your algorithms for Challenges 2 and 3 were parallel?

What makes things complicated or slows you down during parallel portions of your algorithm?

Speedup

Sequential time divided by parallel time





60 seconds

40 seconds

60 seconds / 40 seconds = 1.5
The speedup of this parallel solution is 1.5

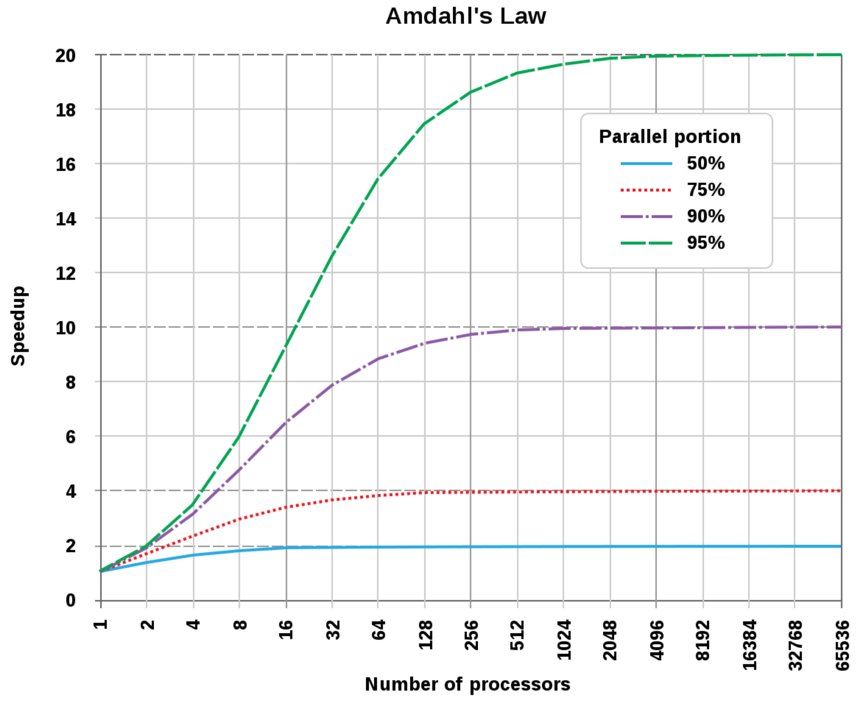
Prompt: What was your group's speedup in Challenge 2?

What about in Challenge 3?

Are you surprised?

It's not just you! Speed-up is never equal to the number of processors.

Some portions of your algorithm can't be made parallel. Each additional processor helps a little less. Eventually the speedup reaches a limit.



Prompt

As you watch this video write down

- Why is the type of computing presented “distributed”?
- Why is distributed computing used to solve the problem?

Unit 6 Lesson 6 - Activity

Prompt

As you watch this video write down

- Why is the type of computing presented “distributed”?
- Why is distributed computing used to solve the problem?

Wrap Up

Sequential Computing: programs run in order, one command at a time.

Parallel Computing: programs are broken into small pieces, some of which are run simultaneously

Distributed Computing: programs are run by multiple devices

Speedup: the time used to complete a task sequentially divided by the time to complete a task in parallel

Prompt:

Based on today's activities, what are the pros and cons of parallel and distributed computing?

Unit 6 - Lesson 6
Assessment Day

Activity

●●○

Unit Assessment