Unit 6 - Algorithms

Unit Overview

Students learn to design and analyze algorithms to understand how they work and why some algorithms are considered more efficient than others. This short unit is entirely unplugged, and features hands-on activities that help students get an intuitive sense of how quickly different algorithms run and the pros and cons of different algorithms. Later in the unit, students explore concepts like undecidable problems and parallel and distributed computing.

Unit Philosophy and Pedagogy

- **A Break from Programming:** This unit is intentionally designed as a short respite from programming. After three units and a major hackathon project, it's a great opportunity to get away from screens for a while before the final programming push that leads to the Create PT.
- **Just Enough Math:** This unit includes some mathematical concepts which enrich students' understanding of how algorithms are analyzed, which might at first be a little intimidating to some students (and teachers!). The mathematical topics included in this unit are only those necessary to provide a solid foundation in algorithmic analysis to the depth described in the CS Principles framework. If you're a teacher with a strong mathematical background, check carefully that you don't needlessly add complexity to a unit which might already prove challenging for some students. All teachers should keep an eye out for the ways visuals, hands-on examples, and patterns in presentation style are used to ensure a consistent presentation of these mathematical topics.

Major Assessment and Projects

This unit does not conclude with a major project. Students will complete an end-of-unit assessment that is aligned with CS Principles framework objectives covered in this unit.

AP Connections

This unit helps build towards the enduring understandings listed below. For a detailed mapping of units to Learning Objectives and EKs please see the "Standards" page for this unit.

- AAP-2: The way statements are sequenced and combined in a program determines the computed result. Programs incorporate iteration and selection constructs to represent repetition and make decisions to handle varied input values.
- AAP-4: There exist problems that computers cannot solve, and even when a computer can solve a problem, it may not be able to do so in a reasonable amount of time.
- CSN-2: Parallel and distributed computing leverage multiple computers to more quickly solve complex problems or process large data sets.

This unit includes content from the following topics from the AP CS Principles Framework. For more detailed information on topic coverage in the course review **Code.org CSP Topic Coverage**.

- 3.9 Developing Algorithms
- 3.11 Binary Search
- 3.17 Algorithmic Efficiency
- 3.18 Undecidable Problems
- 4.3 Parallel and Distributed Computing

Week 1

Lesson 1: Algorithms Solve Problems

Learn how computer scientists think about problems and the way different algorithms can be designed to solve them.

Lesson 2: Algorithm Efficiency

Learn how computers describe how fast different algorithms are and the fact that different algorithms that solve the same problem may be faster than others.

Lesson 3: Unreasonable Time

Explore two different types of algorithms that seem similar but take fundamentally different amounts of computing power to run.

Lesson 4: The Limits of Algorithms

Widget

Explore two different kinds of problems at the limits of what a computer is capable of solving and learn the difference in how computer scientists respond.

Lesson 5: Parallel and Distributed Algorithms

Learn how algorithms are designed to run on many computers and the benefits and challenges that result.

Week 2

Lesson 6: Assessment Day

Project

Assessment day to conclude the unit.

Lesson 1: Algorithms Solve Problems

Overview

Students will complete two exploratory activities that introduce the concept of a problem and an algorithm. In the first students answer a series of questions about birthdates and names of their classmates. They then discuss the similarities and differences between the problems. In the second activity students are given six different algorithms and must analyze them to determine which they think are the same or different. At the end of the lesson they are introduced to the formal definitions of a problem and an algorithm.

Purpose

This lesson is an approachable and interactive introduction to the main concepts of this short unit. Students have been writing a lot of code, and now they are ready to think on a high level about the patterns that make two different problems, or two different algorithms, similar or different. This mindset will be important as they tackle the more challenging material later in the unit where students will learn to compare different algorithms that address the same problem.

Agenda

Warm Up (5 mins)

Activity (30 mins)

- **Comparing Problems**
- **Comparing Algorithms**

Wrap Up (10 mins)

- **Assessment: Check For Understanding**

View on Code Studio

Objectives

Students will be able to:

- Explain the formal definitions of a problem, an algorithm, sequencing, selection, and iteration.
- Explain that some problems may look different but be similar or look similar but be different.
- Explain that some algorithms may look or operate differently but still solve the same problem.

Preparation

☐ Review the algorithms students will be comparing in the second activity to make sure you are prepared to support students in trying them out.

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teachers

- **CSP Unit 6 - Algorithms** - Presentation

Teaching Guide

Warm Up (5 mins)

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?

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Discussion Goal

Goal: This is an optional prompt. If you are able to move directly to the main activity you should do so. This prompt should get students thinking about the themes of the lesson.

There are no "wrong answers" here though you should expect answers that focus on the fact that often there's lots of ways to write code that does the same thing.

Activity (30 mins)

Remarks

> Today we are going to kick off a short unit about how computer scientists think about problem solving. A really important skill will be recognizing patterns and similarities. The same thing that

Comparing Problems

Distribute: Ask students to take out their journals or give them some blank paper for working on the following problems

Display: Show the slides for the ten problems students will need to solve.

Circulate: Ask students to review the problems for one minute, and then let them move around the room collecting information needed to solve the problems. This may take them several minutes.

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

Discuss: Have students discuss the prompt with a neighbor before asking them to share with the room. Lead a discussion on their experiences.

Discussion Goal

Goal: This discussion should focus on what made the problems that students solved similar to one another. You likely will want to put the problems back on the screen to make it easier to talk through. Here are some connections you may pull out though there are more students may make.

- Problems 1 and 2 are very similar. As soon as you find one person who meets the criteria you know you're done.
- Problem 3 and 4 are very similar but you need to talk to every other student to answer it. You only need to keep track of the closest birthday you've heard so far, however.
- Problem 5 is easy to solve as soon as you've solved problems 3 and 4.
- Problem 6 - 10 require you to have written down everyone's birthday, likely in order.
- Problems 7 - 9 are the same problem but for different numbers of people. Whatever strategy you use for one of those would be helpful to solve the others
- Problem 10 is a different version of problem 7 but instead of finding the smallest gap you're finding the largest.

Remarks

> The first thing that we need to think about as computer scientists is what is a "problem". We just looked at 10 problems, but as we discussed, a lot of them are similar. If we solve one problem we may actually solve another, or at least have a good idea for how to start solving another. As computer scientists it's important to ask "have I seen this problem before" or "how is this problem similar to others I've solved?"

Comparing Algorithms

Remarks

> We just thought about whether problems are similar. Now we're going to look at whether we're actually solving the same problem.

Prompt: Which of these algorithms are “the same” as one another?

Circulate: Ask students to review the algorithms with a partner and group them into categories. Move around the room making sure students are not getting stuck. If they finish early push them to see if they can think about the problem in a different way.

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

Discuss: Have students discuss the prompt with another before asking them to share with the room. Lead a discussion on their experiences using tips from the discussion goal at the side.

Discussion Goal

Goal: This discussion should focus on what made the algorithms the same. While they are designed to fall into two categories, ideally a number of points should come out of this discussion.

- Algorithms 1, 3, and 4 draw a square while 2, 5, and 6 draw a rectangle
- Algorithms 4 and 6 are written using a flowchart while 1, 2, 3, and 5 are written in the AP Pseudocode
- Some of these algorithms turn the robot right but turning left three times. It's debateable whether we can really call these algorithms "the same"
- Some of these algorithms create lists or variables to store information. Depending on the context we may not be able to call these algorithms "the same"

Wrap Up (10 mins)

Journal: Students add the following vocabulary words and definitions to their journals: problem, algorithm, sequencing, selection, iteration.

Prompt: How did today’s activities change the way you think about algorithms and problems?

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Discussion Goal

Goal: Use this discussion to reinforce vocabulary introduced in the slides and check in on whether students have begun the transition towards thinking on a higher level about algorithms and problems.

Remarks

> Computer scientists don't just think about "code", they think about problems and the algorithms that solve them. In this unit we're going to explore what makes two problems, or two algorithms, similar or different from one another, and the way computer scientists talk about them. Not only will you be a better programmer, but you'll get to work on some really interesting problems along the way.

Assessment: Check For Understanding

Check For Understanding Question(s) and solutions can be found in each lesson on Code Studio. These questions can be used for an exit ticket.

Question: In your own words explain the difference between a problem and an algorithm.

Standards Alignment

CSP2021

- **AAP-2** - The way statements are sequenced and combined in a program determines the computed result

Lesson 2: Algorithm Efficiency

Overview

In this lesson students follow a demonstration of Linear Search before writing their own search algorithms. Following this, students are introduced to Binary Search after which they compare graphs of the search algorithms to determine which is most efficient.

Purpose

In the previous lesson students learned that there are many different ways to write an algorithm to solve a problem. However, not all solutions are good options, and some algorithms require many fewer steps to run than others. This lesson focuses on comparing Linear Search to Binary Search. The the length of the list being searched, the more efficient Binary Search is compared to Linear Search. That sai,d Binary Search has a constraint: the list must first be sorted. This lesson more broadly introduces students to thinking about the efficiency of an algorithm, an idea that will be explored more in future lessons.

Agenda

Warm Up (5 mins)
Activity (30 mins)
Wrap Up (10 mins)
Assessment: Check For Understanding

View on Code Studio

Objectives

Students will be able to:

- Use Linear Search to determine if a number is in a list
- Use Binary Search to determine if a number is in a list
- Compare the efficiency of Linear Search and Binary Search

Preparation

☐Preview the slides and click through all animations
☐Practice running Binary Search yourself

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teachers

- **CSP Unit 6 - Algorithms** - Presentation

Teaching Guide

Warm Up (5 mins)

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

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Discussion Goal

Goal: Use this discussion to highlight the following points

- Students may have as systematic approach (search through every pocket in order) or a random approach.
- Some students may have a system where they place their books and materials in their backpack to be retrieved in the order of their class schedule. If they look in their bag at a certain spot and don't see the pencil, they have a pretty good idea where to search next.
- There are no right answers here - the idea is to start thinking about the efficiency of different search methods.

Remarks

> There are many different ways to achieve the same goal for this problem: finding your pencil. Today we are going to explore different search methods that computers use.

Activity (30 mins)

Distribute: Students should have sticky notes and their journals

Do This: Call for three volunteers. These students use the Ticket Generator on Code Studio to produce raffle tickets numbers (one per volunteer). Students write the number on a sticky note and come to the front of the room.

Remarks

> Thank you, volunteers! Let's figure out if any of you have the winning number for this instance of the problem. I'm going to ask you one by one to show your number and we will compare it with the winning number.

Teaching Tip

Before class, choose your own winning number (a number between 0-999), or alternatively use the Ticket Generator yourself!

Do This: Each volunteer reveals their number and compares it to the winning number.

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?

Display: Look at the chart as a class and examine the graph. The graph visualizes the pattern the class discussed.

Discussion Goal

Answer:

- The greatest possible number of steps for this instance is 3.
- With six volunteers, the greates number of steps is 6.
- The patterns is a 1:1 relationship. The number of tickets in the raffle exactly equals the number of steps it would take to check for a number in the worst case.

Remarks

> Now let's try a different way to search.

Group: Place students in groups of two

Do This: With a partner, students navigate to Code Studio and use the Ticket Generator to create seven random numbers that are copied to sticky notes. Students organize these sticky notes in numerical order. Students choose one extra number as the "winning number" and write that number down on a separate sticky note.

Challenge: Students work with their partner to write an algorithm to search for their winning number. Make sure the rules are displayed.

- 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?

Discuss: Students share their algorithms with another group and determine which one runs faster, depending on the number of steps it takes to find a number in the list.

Do This: Students return to their partner and now run a given algorithm, displayed on the board, step by step (click through for animation). As they are doing this, they should think about whether or not this algorithm runs faster (has less steps) than their own algorithms.

Teaching Tip

It's important for students to think about this problem as looking for the number in the worst possible situation. In other words, what is the least amount of steps it would take to definitevely cross off every option?

Remarks

This search algorithm is known as Binary Search. Let's see it in action.

Do This: Click through the animations to see Binary Search in action on the activity slide. Discuss each step as it is happening.

- Find the number in the middle and compare it with the given number. The given number is greater than the middle number, so remove the middle number and all to the left.
- Find the number in the middle of the numbers left, and compare it with the given number. The given number is less than the middle number, so remove the middle number and all to the right.
- Find the number in the middle of the numbers left (there is only one!), and compare it with the given number. The given number is equal to the middle number. We have found our number!

Remarks

Binary search only works with a sorted list of numbers. This allows us to remove options from the search after we've made a decision. In other words, if we know the number is greater than 410, we can remove all numbers less than or equal to 410 and we don't have to manually check those numbers one by one.

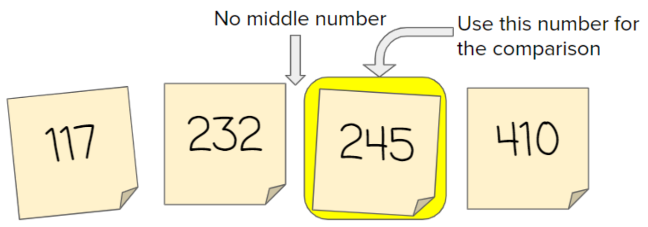
Do This: Display the graph. Students copy the chart into their journals and fill out the missing parts. They can use their sticky note to try out each instance of the problem. When the class has mostly finished, click through the animations to see the answers.

Display: Display the chart on the next slide. Click through the animations. This slide shows another way to fill out the chart, that does not require multiple sticky notes. Instead, we can use the flippy do! For Binary Search, for each instance the number of steps it take to run is equal to the number of bits required to represent the input.

Display: Discuss the graph and click through the animation to see Binary Search graphed.

Teaching Tip

What if there is no middle number while performing a step in Binary Search? Students can come up with a protocol for this like: if there is no middle number, compare the number on the right.



Remarks

Both Binary Search and Linear Search find the answer to our search problem. However, one option is much faster than the other: Binary Search, although it requires that the numbers are sorted first.

Teaching Tip

This is a tricky concept! It's ok if students don't fully understand. What's most important is what is revealed next: Binary Search is more efficient than Linear Search.

Wrap Up (10 mins)

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?

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Journal: Students add the following vocabulary words and definitions to their journals: efficiency, linear search, binary search.

Discussion Goal

Answers:

- For one input, either Binary Search or Linear Search would be appropriate. Students may argue for Linear as it does not need to first be sorted.
- For five inputs, it's clear that Binary Search is fastest, although it only saves two steps over Linear Search. This is all the more true with one hundred inputs, which only takes seven steps (Binary number: 1100100) compared to Linear Search's one hundred steps.

Remarks

> Well done! It's clear that there are many algorithms we can use to search for an item in a list. Some are faster, or more efficient, than others. In the case of Binary Search, the larger the list we are searching through, the greater the efficiency in using this algorithm instead of Linear Search.

Assessment: Check For Understanding

Check For Understanding Question(s) and solutions can be found in each lesson on Code Studio. These questions can be used for an exit ticket.

Question: What is the third step using Binary Search to look for the number 32 in this list?

Question: Which of the following is true of two algorithms designed to solve the same problem?

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

CSP2021

- ▶ **AAP-2** - The way statements are sequenced and combined in a program determines the computed result
- ▶ **AAP-4** - There exist problems that the computer cannot solve

Lesson 3: Unreasonable Time

Overview

Students investigate two different types of raffles that highlight the way seemingly small problems can quickly grow large. The first raffle asks students to hunt for pairs of tickets that add to a target value. The second raffle asks students to hunt for any group of tickets that adds to a target value. After trying out each raffle live students will try to figure out the patterns for how many total combinations need to be checked in each. At the end they discuss the difference between reasonable and unreasonable algorithms based on their experiences.

Purpose

This lesson explores some tricky mathematical concepts in a hands on and interactive way. Building off the raffle analogy used in the previous lesson, it gives students an experience with two types of problems that grow quickly in size, though as they'll see one grows much faster than the other. This lesson should give students a sense of how computer scientists use mathematics to think about problems without relying too heavily on mathematical background that not all students may have.

Agenda

Warm Up (5 mins)
Activity (30 mins)
- **Running The Two Raffles**
- **Activity Guide - Unreasonable Time**

Wrap Up (10 mins)
- **Assessment: Check For Understanding**

View on Code Studio

Objectives

Students will be able to:

- Explain the difference between problems that run in a reasonable time and those that do not
- Explain how both formal mathematical reasoning and informal measurement can be used to determine an algorithms efficiency

Preparation

☐ Review the slides to make sure you are prepared to lead the different discussions in this lesson.

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teachers

- **CSP Unit 6 - Algorithms** - Presentation

For the Students

- **Unreasonable Time** - Activity Guide
  Make a Copy ▾

Teaching Guide

Warm Up (5 mins)

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

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Remarks

> Yesterday we started looking at how computer scientists might compare two algorithms that solve the same problem. Today we're going to look at two different problems. They may seem similar but as we'll see they actually are much harder to solve than either of the two we saw yesterday. The question will be "how much harder"? Let's get to it!

Discussion Goal

Goal: This prompt is a review from the previous lesson. Students should be free to refer to notes or their journals. You should might hear the following points.

- One algorithm requires fewer steps to complete than another
- "The line for one algorithm curves below the other"
- More efficient algorithms are much more helpful as input size grows. The amount of work grows more slowly.

You may wish to refer to slides from previous lessons.

Activity (30 mins)

Running The Two Raffles

Distribute: Send students to the ticket generator widget on Code Studio.

Display: Show slides that explain the pair raffle.

Circulate: Give students a couple of minutes to walk around the room seeing if they can find a partner with which they can win the raffle. After a couple of minutes have them return to their desks. If there are any winners feel free to celebrate.

Remarks

> Alright, that was interesting. Let's try out a different version of the raffle. Last time I made you work silently so that we got a better sense of how many checks needed to happen. This time we're going to run a group raffle, but I'll let you talk out loud if you want.

Display: Show slides that explain the group raffle.

Circulate: Give students a couple of minutes to walk around the room seeing if they can find a group with which they can win the raffle. After a couple of minutes have them return to their desks. If there are any winners feel free to celebrate.

Teaching Tip

Running the Raffles: Both because it helps give students a sense of the problem space and because it's fun to run a raffle, you should run both the pair raffle and the group raffle in your classroom. Here's some helpful tips.

- Insist that students check for the pair raffle silently. For this one it's very easy to just yell out what number you would pair with and lose the opportuntity to see how many total checks are necessary if they can just all yell from their seats.
- The group raffle doesn't necessarily need to be silent. As students will see, it's an incredibly difficult problem and they're going to need to do a lot of checking even if they're able to talk out loud.
- Think ahead about whether you actually want to award winners of the raffles. It's pretty unlikely that there will be a winner.
- There's a pretty good chance that no one will win either raffle.

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

Discuss: Have students discuss this prompt with a neighbor. Then have a few students share out their reflections.

Remarks

> As computer scientists we're getting better at thinking about problems and algorithms. We also saw last time that we can use a little mathematical reasoning to decide if one algorithm is more efficient than the other. Let's do a little work on these two raffles to see if our intuitions are correct.

Distribute: Distribute copies of **Unreasonable Time - Activity Guide**

Activity Guide - Unreasonable Time

Pair Raffle: Ask students to fill out the first page where they must figure out the total number of pairs for different sized "pair raffles". This may take several minutes and require students to draw pictures.

Group Raffle: Ask students to fill out the second page where they must figure out the total number of groups for different sized "group raffles". This may take several minutes and require students to draw pictures.

Share Responses: Ask students to share their responses with another group.

Display: Show the slides summarizing the correct solutions for each as well as providing language to describe these two kinds of curves.

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

Discuss: Briefly ask students for their ideas why before showing them the following slides. Exponential curves grow extremely quickly. You simply can't build a computer fast enough even for relatively small raffle sizes.

Discussion Goal

Goal: This discussion is primarily designed to get quick reactions from students to motivate the second big activity in this lesson. Students will likely note that the group raffle felt a lot harder to check, even with the ability to talk. That said, there's no wrong answers at this point. You're about to check their intutions mathematically.

Teaching Tip

Encouraging Drawing: Students will likely be able to answer these questions much more easily if they draw on a sheet of paper. The question for the pair raffle becomes "if I draw some dots on a sheet of paper, how many lines can I draw between them?" since each line connects two points. For the group raffle it is a little more tricky but it's "how many ways can I combine the dots?"

8 is a Challenge: The activity guide indicates that 8 is a challenge for both activities. Unless students are starting to see that patterns for each raffle size they will likely find this extremely challenging, especially for the group raffle. Reassure students it's a challenge and they don't need to complete it.

Ending Early: It's more important that students get experience with how these two problems are different in how quickly they grow. If students are getting stuck with one encourage them to move on to the other.

Wrap Up (10 mins)

Journal: Students add the following vocabulary words and definitions to their journals: unreasonable time, reasonable 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.

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Discussion Goal

Goal: Use this discussion to reinforce vocabulary introduced in the slides and make sure they have understood the main concepts of the day. Students should be able to explain with the term reasonable, unreasonable, and exponential, why running a group raffle in a school of most sizes is impossible.

Remarks

The group raffle is just one example of an algorithm that is "unreasonable". It grows exponentially and so we'd never want to run it at our school. Next time we'll talk more about what to do when we encounter unreasonable problems.

Assessment: Check For Understanding

Check For Understanding Question(s) and solutions can be found in each lesson on Code Studio. These questions can be used for an exit ticket.

Question: Which of the follow efficiencies would be considered unreasonable?

Question: Based on the data provided, does this algorithm run in a reasonable or unreasonable time? Explain your answer

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

CSP2021

- ▶ **AAP-4** - There exist problems that the computer cannot solve

Lesson 4: The Limits of Algorithms

Overview

Students explore the limits of what algorithms are able to compute. First they use a widget that exposes students to the Traveling Salesman Problem. After recognizing this problem can only be solved using an unreasonable time algorithm the develop their own good enough solutions that run more efficiently. Later in the lesson students watch a video about undecidable problems for which no algorithm can ever be developed to solve them.

Purpose

Prior to this lesson students have learned about the efficiencies of different algorithms and have considered the difference between reasonable and unreasonable algorithms. In this lesson they explore what happens when you reach that limit. In one instance the response is to accept good enough solutions that run more reasonably. In the other case the problem simply can't be solved at all.

Agenda

View on Code Studio

Objectives

Students will be able to:

- Determine if an algorithm runs in unreasonable time.
- Develop a heuristic to solve a problem.
- Distinguish between decision problems and optimization problems.
- Explain the existence of undecidable problems

Preparation

- ☐ Read through the slides
- ☐ Preview the Traveling Salesman widgets

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teachers

- **CSP Unit 6 - Algorithms** - Presentation

Teaching Guide

Warm Up (5 mins)

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

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Discussion Goal

Goal: Use this discussion to quickly review topics brought up in the previous lesson.

- Reasonable algorithms grow at a polynomial rate or slower. Unreasonable algorithms grow exponentially.
- The time to solve an unreasonable algorithm grows very quickly even for relatively small problem sizes.

Remarks

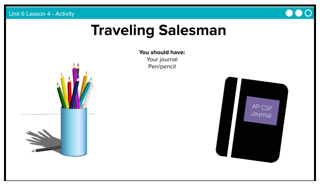
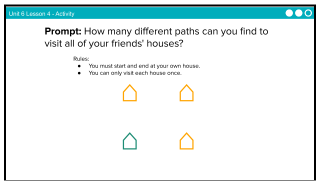
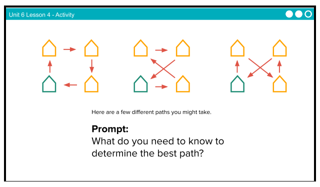
> This was a good reminder of what we learned last time. Unreasonable time algorithms just don't make sense to run. Today we're going to be thinking more about what happens when we reach the limits of algorithms, and how sometimes we can make compromises.

Activity (30 mins)

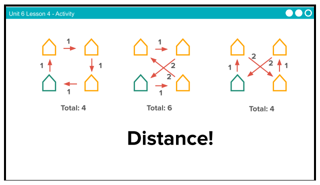
Traveling Salesman

Set Up: Each student needs to have their journals and a pen or pencil.

Display: Use the activity slides for this lesson to guide the activity on the limits of algorithms.

Slides	Speaker Notes
 Unit 6 Lesson 4 - Activity **Traveling Salesman** **You should have:** Your journal Pen/pencil	**Say:** Today we are going to explore the Traveling Salesman Problem.
 Unit 6 Lesson 4 - Activity **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.	**Prompt:** How many different paths can you find to visit all of your friends' houses? **Do This:** Students should sketch out different paths in their journals. Read through the rules together as a class. **Rules:** • You must start and end at your own house. • You can only visit each house once. **Discuss:** Students share their paths with a partner before discussing as a class.
 Unit 6 Lesson 4 - Activity Here are a few different paths you might take. **Prompt:** What do you need to know to determine the best path?	**Say:** Take a look at some of these paths on the screen. You may have similar ones in your journal. **Prompt:** What do you need to know to determine the best path?

Slides | **Speaker Notes**


Unit 6 Lesson 4 - Activity

Distance!

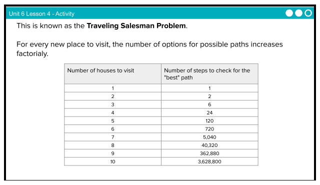
Say: Distance! If we know the distance between each house, we can make a better decision about which path to take. In this example, the first option is shortest, so that's the path we would choose.

Unit 6 Lesson 4 - Activity

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

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

Discussion Goal: The goal here is for students to come to the realization that they would need to explore all possible paths in order to determine which one is the best.

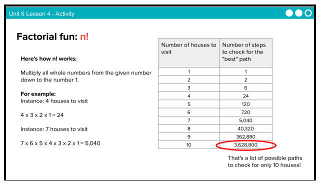

Unit 6 Lesson 4 - Activity

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

Say: This is known as the Traveling Salesman Problem. For each new place to visit, the number of options for possible paths increases factorialy.


Unit 6 Lesson 4 - Activity

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

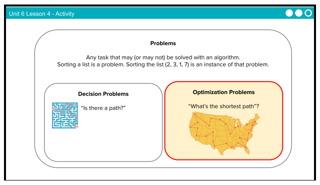
7 x 6 x 5 x 4 x 3 x 2 x 1 = 5,040

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!

Say: A factorial function is written with the letter "n" followed by an explanation point. Here's how n! works: Multiply all whole numbers from the given number down to the number 1.

Note: Talk through the examples and the table. The goal here isn't for students to memorize the math, but to understand that with each new house added, the number of potential paths that have to be checked expands very quickly.


Unit 6 Lesson 4 - Activity

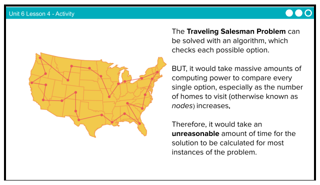
Problems

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"?

Say: The Traveling Salesman is an Optimization Problem. It's not a Decision Problem. We know there is a path. Now we must determine which is the shortest or most efficient path.


Unit 6 Lesson 4 - Activity

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

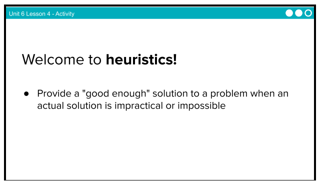
BUT, it would take massive amounts of computing power to compare every single option, especially as the number of homes to visit (otherwise known as *nodes*) increases.

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

Say: 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.


Unit 6 Lesson 4 - Activity

Welcome to **heuristics!**

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

Say: Welcome to **heuristics**! Heuristics provide a good enough solution to a problem when an actual solution is impractical or impossible.

Do This: Students now navigate to Level 2 on Code Studio where they will use the Traveling Salesman Widget to find the "best" path to visit all nodes.

- In their journal, students should write down a plan or **heuristic** for choosing a good path.

Discuss: In pairs, students should share their heuristics and make adjustments to their own as needed.

Do This: Now students navigate to Level 3 and use the Widget: Traveling Salesman with Random Nodes. Students test their heuristic on this widget, keeping a log in their journal of the distance for the path their heuristic finds and the best distance the student finds not using the stated heuristic (i.e. clicking around, or "brute force").

- Note: Click reset to try different paths on the same level. Click "New Level" to generate at new random assortment of nodes.

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

Discussion Goal

Goal: Tease out what factors students used to create their heuristics.

Discuss: Call on several students to share their heuristics. As a class, discuss which heuristic we think is best, and will give us a "good enough" result for most cases.

Display: Show the sample heuristic. This may be one that students already came up with. Discuss whether or not this heuristic is "good enough".

Discussion Goal

Goal: There are times when the Next-Closest heuristic will not provide the best option, but on average it's a good option.

- Students can reflect back on the paper route problem from earlier in the class. This may have been the option they first suggested.

Remarks

> The Traveling Salesman Problem is an optimization problem. We are attempting to find the best path.
>
> It's 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 wtih a solution that is "good enough" for most instances of the problem.

Undecidable Problems

Remarks

> We've learned how we address unreasonable problems. Before we finish our study of problems let's learn about one more type, problems that no algorithm will ever be able to solve. This video is a little tricky, but it gives you a good sense for the way this problem shows up.

Teaching Tip

Do I Need to Understand the Halting Problem?: Students don't actually need to understand the Halting Problem or the proof in this video. The main ideas are covered in the takeaways and are fairly simple. There are a few problems, most notably "will this code run?" that we've proven there is no algorithm that will ever be able to determine the correct answer in every case. The proof is interesting but if you are short on time you may opt to skip the video.

Video: Show the video on the halting problem, located on the slide

Display: Review the main takeaways about undecidable problems.

Wrap Up (10 mins)

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

Discussion Goal

Answer: A heuristic is used when it's impossible or impractical to use an algorithm to solve a problem. Therefore, we need something that is good enough on average for most instances. This is where a heuristic shines.

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Remarks

> Great! Heuristics are handy in every day life. Think about using mapping software to find the best route to a destination!

Assessment: Check For Understanding

Check For Understanding Question(s) and solutions can be found in each lesson on Code Studio. These questions can be used for an exit ticket.

Question: In which of the following situations is it most appropriate to use a heuristic solution?

Question: In your own words, explain the difference between undecidable problems and unreasonable time algorithms.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

- ▶ **AP** - Algorithms & Programming

CSP2021

- ▶ **AAP-4** - There exist problems that the computer cannot solve

Lesson 5: Parallel and Distributed Algorithms

Overview

In this lesson students explore the benefits and limitations of parallel and distributed computing. First they discuss the way human problem solving changes when additional people lend a hand. Then they run a series of demonstrations that show how simple tasks like sorting cards get faster when more people help, but there is a limitation to the efficiency gains. Later in the lesson students watch a video about distributed computing that highlights the ways distributed computing can help tackle new kinds of problems. To conclude the lesson students record new vocabulary in their journals and discuss any open questions.

Purpose

This lesson is a quick tour of the challenges and benefits of parallel and distributed computing. It caps off the unit to showcase ways these techniques are being used to push the boundaries of how efficiently computer can solve problems.

Agenda

Warm Up (5 mins)
Activity (30 mins)
- **Card Sorting Challenge**
- **Debriefing the Challenge**
- **Distributed Computing in Real World Settings**

Wrap Up (10 mins)
- **Assessment: Check For Understanding**

View on Code Studio

Objectives

Students will be able to:

- Explain the difference between sequential, parallel, and distributed computing.
- Calculate the speedup of a parallel solution to a problem
- Describe the benefits and challenges of parallel and distributed computing.

Preparation

☐ Collect the manipulatives you will use for the main activity. While decks of cards are suggested, other manipulatives are possible. See the teaching tip in the main activity for suggestions.

Links

Heads Up! Please make a copy of any documents you plan to share with students.

For the Teachers

- **CSP Unit 6 - Algorithms** - Presentation

Teaching Guide

Warm Up (5 mins)

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?

Discuss: Have students think about their answers on their own, then share with a partner, and then finally discuss responses with the entire room.

Discussion Goal

Goal: This should be a quick prompt to foreshadow the main ideas of the lesson. Students should brainstorm many potential tasks. When they start mentioning the maximum number of people they'd want to help them direct attention towards why that's the case. You might hear that:

- Adding extra people makes it more complicated
- Sometimes extra people doesn't really speed things up
- If you're working with lots of people then if one person is slower the whole group is slowed down

Remarks

> As we've explored in this unit, computer scientists are always looking for more efficient ways to run programs. One way to do this is to develop faster algorithms that run on a single computer. Another idea we'll explore today, is figuring out ways to run programs on many computers at the same time. We just talked about some benefits and challenges when many people help with a task. As we'll see, the same is true with running programs on multiple computers. It can lead to some improvements, but also some new challenges.

Activity (30 mins)

Card Sorting Challenge

Group: Place students in groups of three or four.

Distribute: Give one member of each group a deck of cards.

Teaching Tip

Choosing Manipulatives: This activity can easily be done with many different types of manipulatives, not just cards. For example, students could sort pennies by even / odd year, sort coins into piles of different denominations, sort blocks by color / size, or sort any other readily available item. Pick whatever makes the most sense for your context.

Challenge 1 - One Person Sort: At the front of the room display the directions for the first sort as well as the clock. Run the clock, and have students put the cards in order. Have students record their time. Then let the another partner repeat the process.

Challenge 2 - Two Person Sort: At the front of the room display the directions for the second. Run the clock, and have students put the cards in order. Have students record their time. If students are in groups of four offer to let the other two students try the challenge.

Challenge 3 - Full Group Sort: At the front of the room display the directions for the challenge. Run the clock, and have students put the cards in order. Have students record their time.

Debriefing the Challenge

Display: Show the slides explaining the difference between parallel and sequential computing models. Talk through the different models.

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?

Discuss: Have groups discuss responses at their tables before sharing with the room.

Remarks

> A lot of the challenges you just encountered show up when you try to run a program on multiple computers as well.

- Sometimes you need to wait because one computer finished before another
- It can be complicated to split up work and recombine it when moving in and out of parallel portions
- They're faster, but not always as much faster as you think.

Prompt: What was your group's speedup in Challenge 2? What about in Challenge 3? Are you surprised?

Discuss: Have groups calculate their speedup and share with the room.

Display: Cover the primary points of speed in the real world.

- Students probably noticed their speedup is lower than the number of people helping sort. Sorting with two people doesn't give a speedup of 2. Sorting with 3 people doesn't give a speedup of 3.
- Because some portions are always still sequential, the benefits of adding more processors will go down and eventually the speedup reaches a limit.

Discussion Goal

Goal: This discussion has two goals. The first is to reinforce the difference between parallel and sequential portions of an algorithm. Any time multiple processes are happening at once (for example multiple people are sorting cards), an algorithm is parallel. The second goal is to bring up some common challenges that come up when running parallel algorithms. The remarks cover some of the most important ones but the main point is that while parallel algorithms are faster, they still present challenges.

Discussion Goal

Goal: Use this discussion to reinforce how speedup is calculated, but also to prime students to realize that adding additional parallel processes doesn't always lead to the same amount of speedup. During the parallel portion things are in fact moving faster, but sequential portions still take a long time (e.g. collecting individual piles once each group member has sorted their cards). Therefore speedup is rarely your original time divided by the number of computers running the process. Eventually it will level off.

Distributed Computing in Real World Settings

Remarks

We've just explored some of the core and theoretical ideas of parallel computing. It can speed things up, but not infinitely, and it adds complications and many more resources. That said, parallel computing can help tackle some big problems.

Prompt: Before showing the video share the two prompts.

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

Display: Show the video on Folding at home

Discuss: Have students share their responses to the two prompts:

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

Remarks

Distributed computing is very similar to parallel computing. The main idea is that programs can be run on lots and lots of computers. Distributed and parallel computing are helpful for solving really big problems that you couldn't normally solve on a single computer.

Wrap Up (10 mins)

Remarks

Let's sum up what we learned: Parallel computing consists of both a parallel portion that is shared and a sequential portion.

A sequential solution's efficiency is measured as the sum of all of its steps, but a parallel solution takes as along as its sequential tasks plus the longest of its paralell tasks. Often times a parallel solution will be the fastest option, but there is a limit.

> Solutions that use parallel computing can scale more effectively than solutions that use sequential computing. Why is this so? If we continue to add tasks, a sequential solution would continue to get larger and larger. However, with a parallel system, those tasks can be balanced.

Journal: Students add the following vocabulary words and definitions to their journals: sequential computing, parallel computing, distributed computing, speedup

Prompt: Based on what we saw here today, create a list of pros and cons for distributed and parallel computing. Share it with a partner.

Discuss: Have students write their list, then share with a partner, and then finally discuss responses with the entire room.

Discussion Goal

Goal: This lesson covers a lot of ground so this is a good chance to review some of the main points of the lesson. Here's some big ones to cover.

- Parallel programs typically are faster
- Parallel programs don't get faster forever. At some point adding more processors doesn't really help
- Parallel programs can be more complicated.
- Parallel programs can be slowed down if only one of many devices is slow.
- Distributed programs can be run on thousands or even millions of computers which allows you to take on enormous problems

Remarks

Assessment: Check For Understanding

Check For Understanding Question(s) and solutions can be found in each lesson on Code Studio. These questions can be used for an exit ticket.

Question: What is the speedup of this parallel solution?

Question: In your own words, explain why the speedup of a parallel algorithm will eventually reach some limit.

Standards Alignment

CSTA K-12 Computer Science Standards (2017)

▶ **AP** - Algorithms & Programming

CSP2021

▶ **CSN-2** - Parallel and distributed computing leverages multiple computers to more quickly solve complex problems or process large data sets

Lesson 6: Assessment Day

Overview

Students complete a multiple choice assessment which covers the unit topics.

Agenda

Assessment (25 mins)

- Topic Coverage

Assessment Review (20 mins)

View on Code Studio

Preparation

- ☐ Preview the assessment questions

Teaching Guide

Assessment (25 mins)

Administer the Unit 6 Assessment, found on Code Studio. Make sure to unlock the assessment following instructions **here**.

Assessment Review (20 mins)

Review the answers to the assessment with the class. Discuss any questions that come up and take note of topics where students may need extra review.

Teaching Tip

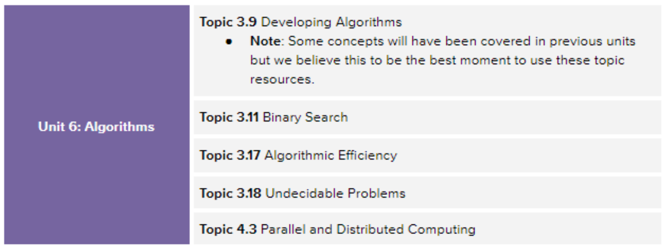
Topic Coverage

The College Board has provided a bank of questions to help formatively assess student understanding of the content in the framework. These questions are mapped to topics with each topic having a handful of questions available.

The College Board has a few strict guidelines about how topic questions can be used. In particular, students may not receive a grade based on performance on topic questions nor can they be used for teacher evaluation. Beyond these requirements, however, they are primarily intended to formatively assess student progress and learning as they prepare for the end of course exam.

Within our own course we recommend that you use them in a variety of ways:

- Throughout the unit assign topic questions to students related to the topics students are learning about that day or that week
- Prior to the unit assessment assign topic questions to help students practice and prepare for the summative assessment
- After the unit assessment use these topic questions to help students track their progress towards preparation for the AP assessment



Unit 6: Algorithms	**Topic 3.9** Developing Algorithms • **Note**: Some concepts will have been covered in previous units but we believe this to be the best moment to use these topic resources.
	Topic 3.11 Binary Search
	Topic 3.17 Algorithmic Efficiency
	Topic 3.18 Undecidable Problems
	Topic 4.3 Parallel and Distributed Computing

Click for more info: **Code.org CSP Topic Coverage**