UNIVERSITY OF WATERLOO

Faculty of Mathematics
Waterloo, Ontario N2L 3G1

Centre for Education in
Mathematics and Computing

Grade 6 Math Circles

14/15 November, 2017

Algorithms

Introduction

Algorithms are an important area of study in both mathematics and computer science. Everyday devices such as your computer and smartphone follow multiple algorithms to do what you want them to do.

When talking about algorithms in math, we might be interested in how long an algorithm takes to finish a task or how *efficient* it is when compared to other algorithms. Since it's such a wide topic, this week we'll be looking at some simple sorting algorithms and why they're interesting. We'll also be looking at a very interesting game that arises when talking about algorithms.

Sorting

Now you may think this is a weird way to start learning about algorithms. But sorting algorithms are some of the most fundamental algorithms you can study. Basic sorting algorithms are easy to understand and have interesting properties, making them an ideal starting point for us.

Algorithms exist to solve a problem or complete a task. Without a problem/task, it's unclear what our algorithm should do - a taxi driver can't go anywhere until you tell him/her where you want to go.

Another important fact is that algorithms are written in such a way that the person/thing carrying out the algorithm is assumed to have no capability of thinking for themselves. This is important because this is what separates algorithms in math from everyday algorithms such as cooking recipes.

In the case of a cooking recipe, you're allowed to make slight changes to the recipe as you see fit. This isn't allowed when we talk about algorithms in math - every step must be followed strictly.

If you're given a number of things (such as names, numbers, cars etc.), the fundamental problem of sorting is to order the items in a particular way (alphabetically, by color etc.).

In other words, say you're given a list of things:

$$\{A, E, D, C, B\}$$

A sorting algorithm should tell you how to go from this **unordered list**, to an **ordered** one like the one below:

$$\{A, B, C, D, E\}$$

The above list is just one example of an ordered list where the items were sorted alphabetically ascending order. You could sort them in any way you wish.

Bubble Sort

The bubble sort is probably the simplest sorting algorithm. Say you're given an unordered list of items (as before):

$$\{A, E, D, C, B\}$$

You'd like to order this list in alphabetically ascending order. The bubble sort algorithm tells you:

1. If the list has only 1 item, do nothing (the list is already sorted).
2. Pick the first two items (from the left) in the list. This is called a **sub-list**.
3. Compare the two items in your sub-list. If they aren't in alphabetical order, swap them. Otherwise leave them unchanged.
4. Repeat Steps 2 and 3 on the next sub-list consisting of the 2^{nd} and 3^{rd} items in the list. Continue this for sub-lists made up of 3^{rd} and 4^{th}, 4^{th} and 5^{th} etc. items, up to and including the last and second last items in the list.
5. When you get to the end of the list, go back to the start and repeat Steps 2-4 until you reach the end again.
6. Repeat Step 5 until the list is sorted.

Example 1

Use a bubble sort algorithm to sort the following list in order of size (ascending):

$$\{\textit{Elephant, Beetle, Human, Mouse}\}$$

How many times did you have to *pass through* the list? How many *comparisons* did you have to make? How many *swaps* did you have to make?

Hint: Passing through a list means that you've compared all adjacent pairs of items in the list exactly once (starting from the left).

Notice from the above Example, that a bubble sort can often be time consuming. This is because only two items in a list are considered at a time.

Additionally, as you might have seen from Example 1, the bubble sort algorithm often requires you to pass through the list more than once.

Selection Sort

The selection sort algorithm might seem more intuitive to you as it is usually the closest to how humans typically sort things. It's a slightly modified bubble sort where you make only one swap per pass through the list. Suppose you're given a list of items:

$$\{A, E, D, C, B\}$$

You're asked to sort the list in alphabetically ascending order. The selection sort algorithm tells you to do the following:

1. If the list has only 1 item, do nothing (the list is already sorted).
2. Find the item in the list that comes alphabetically last and swap it with the right most item in the list.
3. Find the next alphabetically highest item in the list and swap it with the second right most item in the list.
4. Repeat Step 3 until the entire list is sorted.

The above algorithm is incomplete because it doesn't tell you how to find the alphabetically highest item in your list. To do this, we use the following **sub-algorithm** or **subroutine**:

1. If the list has only 1 item, do nothing.
2. Start by making a sub-list of the first two items in the unsorted list (from the left).
3. Compare the two items in your sub-list and keep the item that comes alphabetically later than the other.
4. Make a new sub-list consisting of the item you kept in Step 3 and the 3^{rd} item in your unsorted list.
5. Repeat Step 3.
6. Repeat Step 4 but with the item you kept and the 4^{th} item in your unsorted list.
7. Repeat Steps 3-4, up to and including the sub-list with the item you've kept and the last item in your unsorted list.
8. The last item you've kept will be the alphabetically highest item in your unsorted list.

So when Step 2 in the selection sort algorithm asks you to find the alphabetically highest item in your unsorted list, you're actually doing Steps 1-8 of the above subroutine. There is an interesting thing to notice here about the selection sort. The following example should illustrate this.

Example 2

Use a selection sort algorithm to sort the following list in order of size (ascending):

{*Elephant, Beetle, Human, Mouse*}

How many times did you have to pass through the list in this case? How many comparisons did you have to make? How many swaps did you have to make?

Efficiency of Sorting Algorithms

Although we've only covered two of the simplest sorting algorithms, you might've noticed from the earlier examples that the number of comparisons and exchanges were different between the algorithms.

This is where we generally talk about how quick or **efficient** an algorithm is. Mathematically speaking, how quickly a sorting algorithm can finish running depends on the size of the list it has to sort.

This makes sense intuitively - the more things you have in your room, the longer it's going to take to clean it up.

Figuring out how different algorithms compare against each other in terms of how long they take to finish the same task, is called estimating the **Time Complexity** of the algorithms. Strictly speaking, doing this requires some very tricky algebra, so we'll stay away from it for now.

However, we can still roughly figure out which of the two algorithms we learned is more efficient. Remember that algorithms are very important when talking about computers. Suppose you had two identical computers and you had an unsorted list (like the one from our examples). You decide to sort the list using a bubble sort algorithm on the first computer and a selection sort algorithm on the second computer.

Example 3

If computer A is running the bubble sort algorithm and computer B is running the selection sort algorithm on the following list:

$$\{A, E, D, C, B\}$$

which one finishes sorting first? Justify your answer.

Hint: A swap operation typically takes twice as much time as a comparison.

Conway's Game of Life

Algorithms are lists of rules that tell you how to complete a task. We're going to look at something closely related - **Cellular Automata** (singular: Cellular Automaton).

You can think of cellular automatons as a *game* of sorts, with a **grid** of **cells** and a set of definite rules. The cellular automaton that we're going to be focussing on is called **Conway's Game of Life**, named after the mathematician John Conway (1937 -) (Figure 1).

Figure 1: John Conway

Conway came up with his game as an attempt to simplify a very complicated set of rules that another famous mathematician, John von Neumann (1903 - 1957) (Figure 2), came up with in the 1940s.

Figure 2: John von Neumann

The Game of Life consists of a grid of cells (as shown in Figure 3) which can be of any size you like. The grid is initially empty but cells can be shaded in as you wish. A shaded cell is called *live* whereas an empty cell is called *dead*.

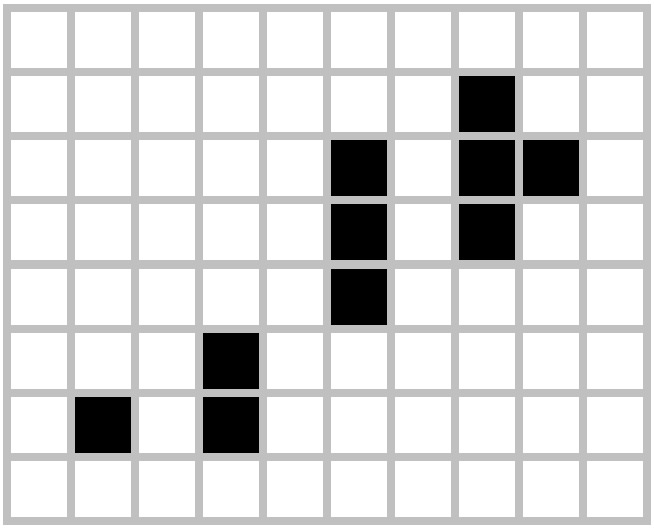


Figure 3: The Game of Life grid showing *live* and *dead* cells

At the start of the game, you can start with any number of live cells you wish and in whichever configuration you wish. This configuration is called the **initial state** or **initial generation**. Each cell on the grid has **8 neighbors**.

The game works by applying the following rules to each and every cell on the grid to get newer generations of cells (just like a population):

1. Any live cell with 0 or 1 live neighbor dies (becomes empty) in the next generation.
2. Any live cell with 2 or 3 live neighbors lives (stays the same) in the next generation.
3. Any live cell with more than 3 live neighbors dies (becomes empty) in the next generation.
4. Any dead (empty) cell with *exactly* 3 live neighbors turns live (becomes shaded) in the next generation.

In other words, you start the game with any particular configuration of live (shaded) cells you like. You then follow Rules 1-4 to progress the game.

It's important to realize that there are no winners or losers in this game. Instead think of it like your own ant farm or tiny universe, where you can watch stuff happening if you set it up a particular way.

Since the game has only 4 rules which must always be followed, you can answer some pretty interesting questions about the Game of Life.

Example 4

If you're given the following initial state (Figure 4), what is the **final state** after 3 generations?

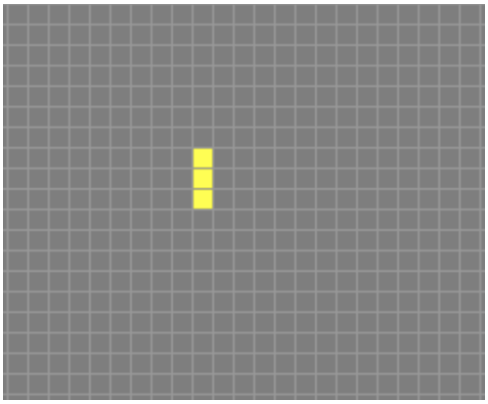


Figure 4: Example 4

Example 5 What is the final state of the following initial state after 1 generation? What is the final state after 2 generations?

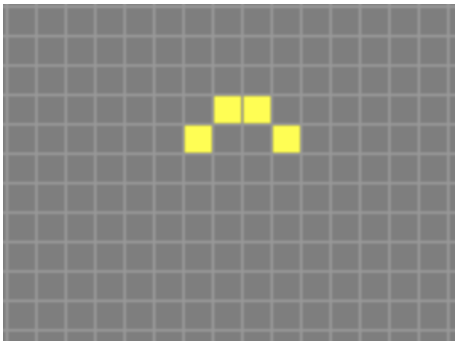


Figure 5: Example 5

Since the Game of Life has strict rules, if we were to walk into a room where a game was in progress, we would be able to predict *all* the states of the game in the future. To do this, all we have to do is follow the rules.

The Game of Life can get pretty complicated when starting with a large number of live cells. For this reason, we usually use a computer to run it. You can try it out at (https://bitstorm.org/gameoflife).

If you play around with it a bit, you will notice that most games settle into groups of cells that *do nothing* (called **still lifes**), groups of cells that wiggle around (called **oscillators**) and groups of cells that move across the grid (called **spaceships**).

Try going back to Examples 4 and 5 and figure out if the final states are still lifes, oscillators or spaceships.

Simulations Within Simulations

The Game of Life is interesting in that it leads to a wide variety of interesting things from a small number of very simple rules.

One such interesting thing is the concept of **decidability**. Suppose you are given an initial and a final state in the game. Having learned about algorithms, you might wonder if there's an algorithm that you can follow so that you can tell if you can get the final state from the initial state.

For example, if your initial state was as shown in Figure 4 and your final state was as shown in Figure 5, you might wonder if you can find an algorithm that will tell you if you could ever get from the initial to the final state using Rules 1-4.

It turns out that such an algorithm is impossible.

Another very interesting fact about the Game of Life is that you can run the Game of Life within itself! In other words, you can make a version of the Game of Life, within the Game of Life.

Additionally, you can make the Game of Life do anything that a normal computer can (although it would be significantly more difficult).

Finally, although you can predict future states from a given initial state, you can't go backwards i.e. figure out the initial state from the final state. To see this, imagine a final state where all the cells on the grid are *dead*. Clearly you could've started out with 1 live cell or 2 live cells etc (see Figures 6, 7 and 8).

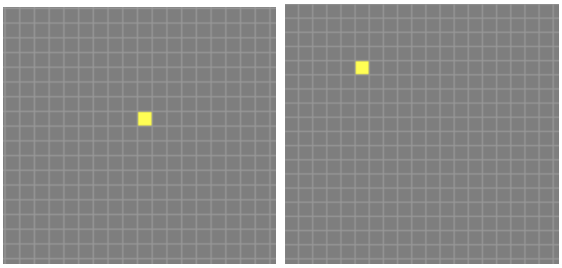


Figure 6: Both these initial states could lead to the final state shown in Figure 7

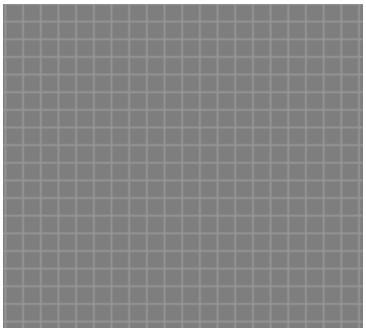


Figure 7: Final state

Problem Set

1. Sort the following list using a bubble sort algorithm in ascending order:

$$\{5, 2, 4, 1, 3\}$$

2. Sort the following list using a selection sort algorithm in ascending order:

$$\{2, \tfrac{1}{2}, \tfrac{3}{7}, \tfrac{1}{100}, 7\}$$

3. You're asked to sort the following list of items in alphabetical order:

$$\{A, B, D, C, E\}$$

   How will you decide which of the two sorting algorithms you should use?

   Hint: Try counting the number of comparisons and swaps each algorithm would need to make.

4. The Fibonacci Sequence is a very famous number sequence. Numbers from the sequence and the sequence itself are often found in nature used, we think, for specific reasons. Perhaps this is because the sequence can be generated algorithmically. Here are the first six numbers: $1, 1, 2, 3, 5, 8, \ldots$

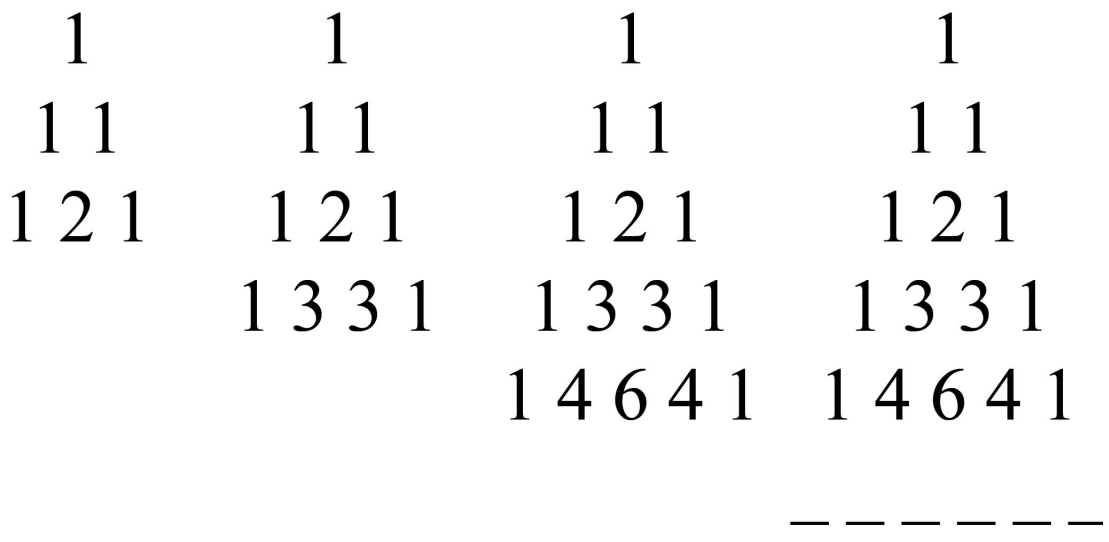
   (a) Determine the next three numbers in the sequence.

   (b) What is the pattern of this sequence? Explain.

   (c) Write an algorithm with precise steps to generate this sequence.

   (d) Would your algorithm work if you were to start with 1 and 3 instead of 1 and 1?

5. Pascal's Triangle is a number pyramid that grows by continuing to add rows below. Here are three iterations:



   (a) Find the next iteration.

   (b) Write down a list of steps that someone could follow to build the next triangle in this pattern.

   (c) * Write down a list of steps that someone could follow to build any iteration of the triangle from the previous iteration.

6. The following figure shows the initial state for a *Glider*. What is the final state after 4 generations?

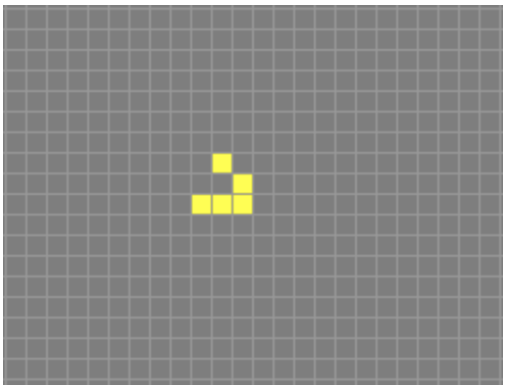


Figure 8: A glider

7. The following initial state is called a **Toad**.

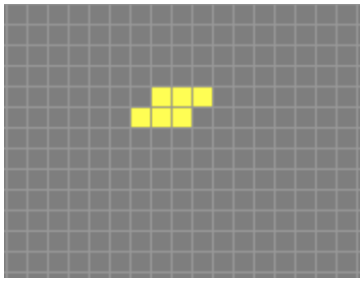


Figure 9: A Toad

   Find the final state after 3 generations. Is it a still life, oscillator or a spaceship?

8. Suppose Rule 1 of the Game of Life were to be deleted. Would the initial state shown in Example 4 still be an oscillator? Why or why not?