UNIVERSITY OF WATERLOO

FACULTY OF MATHEMATICS
WATERLOO, ONTARIO N2L 3G1

CENTRE FOR EDUCATION IN
MATHEMATICS AND COMPUTING

Grade 7/8 Math Circles
November 11, 2020
Algorithms

What is an Algorithm?

Algorithms are everywhere. Whether you know it or not, you use algorithms on a daily basis. Do you follow the same routine every morning to get ready for school? Then you are following an algorithm. If you've ever solved a Rubik's Cube, made a paper airplane, baked a cookie or cooked a specific recipe, you followed an algorithm. **An algorithm is a set of steps to accomplish a task.** In computer science, we tell computers what to do by creating algorithms for them to follow. Consider the following example.

Example: Adding Two-digit Numbers

Let's think about how we add two-digit numbers together. When we add two numbers, usually we place one number above the other and add each of their columns together. Recall that these columns are called place values and have names such as the ones column, tens column, hundreds column, and so on. Thus to add two-digit numbers, we add the ones column and then the tens column. We can use this information to create an algorithm.

Algorithm for Adding Two-digit Numbers:

1. Add the digits in the ones column. Write this sum in the ones column of your total sum.

2. Add the digits in the tens column. Write this sum in the tens column of your total sum.

Now adding two-digit numbers is easy! We just need to follow our algorithm to get the correct answer. Let's test our algorithm on a few examples:

(a)		9	(b)		1	0	(c)		1	8
	+	9		+	1	8		+	2	7
		18			2	8			3	15

Wait, $18 + 27$ is not equal to 315. But I followed each step of our algorithm perfectly! Why did our algorithm produce a wrong answer?

In this case, we forgot to include the carry over rule and now our algorithm has an error! Sometimes algorithms will produce errors and that's why we should always test a few examples. Even if we believe our algorithms are perfect there could be a single example that breaks it! Let's fix our algorithm.

Algorithm for Adding Two-digit Numbers:

1. Add the digits in the ones column.

2. If their sum is greater than 9, then write the ones digit of this sum in the ones column of the total sum and then carry the tens digit of this sum to the next column.

3. Add the digits in the tens column and the amount we carried. Write this sum in the tens column of your total sum.

Try to convince yourself that this algorithm will work for any example where we add two-digit numbers.

Exercise: The word generalize means to make something more widespread. Our algorithm only adds two-digit numbers with a ones and tens column. Can you generalize our algorithm to something that works for more than just two-digit numbers? Try three digits? What about any number of digits?

Sorting

Sorting is used every day by almost everyone in one way or another. Books in a library are sorted by genre and then by author. At school, you may sort your notes and homework by subject. Your clothes at home may be sorted by type with your socks in one drawer and your pants in another drawer.

Consider the following list and **find the smallest number**:

$$8 \quad 1 \quad 5 \quad 13$$

When you looked for the smallest number, did it feel like you took in all four numbers at once rather than looking at them one at a time? Our brains do an amazing job of processing small groups of data like this and quickly coming up with an answer. Computers on the

other hand have to look through data one unit at a time and often have to deal with more data than humans can handle!

Because of this, computer scientists have come up with many ways to sort large amounts of data quickly and efficiently. There are many different **sorting algorithms** and we will look at some today.

Insertion Sort

The first sorting algorithm we will look at is called **insertion sort**. This algorithm is used to sort a list of numbers into ascending (increasing) order. It involves taking one number (we will call it an **element**) at a time from the front of an unsorted list and creating a new sorted list. As we move elements over from the unsorted list, we compare them to each element in the sorted list one by one until their proper place is found.

Insertion Sort Algorithm:

1. Create an empty list for the sorted numbers.

2. Move the first element from the unsorted list to the beginning of the sorted list.

3. If there is more than 1 element in the sorted list, compare that element with the next element in the sorted list. If the next element is smaller, then swap the two elements.

4. Repeat step 3 until the newly inserted element cannot be swapped with the next element (in other words, the next element is greater or equal to it).

5. Repeat steps 2-4 until the unsorted list is empty.

Example: Use insertion sort to sort the following list of numbers:

$$9 \qquad 4 \qquad 3 \qquad 6 \qquad 7$$

Video Solution: *https://youtu.be/TcW6KjAoNC4*

To practice the insertion sort algorithm try out this activity: *https://www.geogebra.org/m/k36vvw4c*.

How come we have to go through all those steps? Why can't we just drop a number in the right place in the sorted list?

While we can easily look at a small list and find where a number fits, computers cannot do this and have to go through a list one by one. Even for us, it is not easy to quickly insert a number into its correct place if a list has millions of elements.

Selection Sort

Selection sort uses the following algorithm to sort a list of numbers into ascending (increasing) order. Just like insertion sort this algorithm works by maintaining two lists - the original unsorted list, and a new sorted list.

Selection Sort Algorithm:

1. Create an empty list for the sorted numbers.

2. Let the first number in the unsorted list be the current smallest number.

3. Find the smallest number in the unsorted list by comparing it to the current smallest number. Update the current smallest number each time a smaller number is found.

4. Move the current smallest number to the end of the sorted list.

5. Repeat steps 2-4 until the unsorted list is empty.

Example: Use selection sort to sort the following list of numbers:

$$14 \quad 6 \quad 11 \quad 5 \quad 20$$

Video Solution: *https://youtu.be/kjLT91U0uKE*

To practice the selection sort algorithm try out this activity: *https://www.geogebra.org/m/gcxxcmcc*.

What happens if a number is repeated in our unsorted lists? Will this algorithm sort it into ascending order correctly? Yes! - can you see why?

Bubble Sort

Unlike the sorting algorithms mentioned above, **bubble sort** does not move elements one at a time from an unsorted list to a sorted list. The algorithm repeatedly goes through the list and compares and then possibly swaps each pair until the largest number is moved to the

end of the list. Then it repeats these steps untill the list is sorted into ascending (increasing) order.

Bubble Sort Algorithm:

1. Look at the first pair of numbers.

2. If they are not in increasing order, swap them.

3. Now look at the next pair. The last element of the previous pair should be the first element of the new pair.

4. Repeat steps 2 and 3 until you reach the end of the list.

5. Repeat steps 1-4 until you can go through the list without any swaps being made.

Example: Use bubble sort to sort the following list of numbers:

$$\textbf{3} \qquad \textbf{9} \qquad \textbf{1} \qquad \textbf{5}$$

Video Solution: *https://youtu.be/tGCCkmRVEmM*

Notice that in the last 5 steps, the list was already in sorted order! The reason bubble sort must go over them again is because the algorithm doesn't have a way to recognize that the list is sorted until it compares all the pairs in the list to make sure they are in the correct order.

To practice the bubble sort algorithm try out this activity: *https://www.geogebra.org/m/ ssqzmy6k*.

Example: Use bubble sort to sort the following letters in alphabetically ascending order:

$$\textbf{A} \qquad \textbf{E} \qquad \textbf{D} \qquad \textbf{C} \qquad \textbf{B}$$

Video Solution: *https://youtu.be/8c-pB9fmpZ4*

Another common approach to sorting (and problem solving in general) is to break the problem down into smaller parts. Often these smaller parts are easier to solve. We can then combine the smaller solutions to solve the whole problem. We'll see this in the sorting algorithm below.

Merge Sort

Merge sort takes an unsorted list and breaks that list in half. It keeps breaking the resulting lists in half until there is only one element in each list. These lists are very easy to sort! The single element lists are then paired up, with each 2 element list getting sorted. This merging process continues until we have one list again. This is sometimes called the **Divide and Conquer strategy.**

Merge Sort Algorithm:

1. Split the list in half.

2. Repeat step 1 on each sublist until each sublist has one element.

3. Merge and sort all pairs of sublists on the same level.

4. Repeat step 3 on the next levels until all sublists are merged into one.

Example: Use merge sort to sort the following list of numbers:

$$\textbf{7} \quad \textbf{1} \quad \textbf{5} \quad \textbf{3} \quad \textbf{6} \quad \textbf{4} \quad \textbf{0} \quad \textbf{9}$$

Video Solution: *https://youtu.be/G5tzgzb-MEo*

To practice the merge sort algorithm try out this activity: *https://www.geogebra.org/m/mgufcpar*.

While the idea of breaking the list into smaller and smaller pieces seems to make sense, how does the merging process actually work?

At each merging step the algorithm uses the fact that each list it's merging together, is already sorted. This means it only has to compare the first element in each list (i.e. if the smallest element in list A is smaller than the smallest element in list B, then it must be the smallest in both).

To see exactly how this works consider the following example:

Video Solution: *https://youtu.be/61izwg-NsOA*