



Grade 7/8 Math Circles

November 19th/20th/21st

Intro to Computer Science Solutions

How Computers Work

Everywhere you look, computers are changing the world. Whether they're on our desktops, in our homes, our pockets, or just about anywhere else. While most of us use computers and technology in our daily lives, many of us don't really know *how computers work*.

Over time, humans have created many tools to help make their lives simpler. Certain tools such as the wheelbarrow, the hammer, the printing press and many others, help humans with **manual** work. Eventually humans wondered "*Is there a tool that can help us with the thinking work we do?*" As humans worked to create such a machine, they realized it had to perform four different tasks:

1. Take input
2. Store information
3. Process information
4. Output the results

Computers started out as basic calculators which at the time was very useful. Now humans use computers to talk to each other, to play games, to control robots and many more things.



The Four Tasks

1. Input

To put it simply, **input** is the stuff that the world does, or stuff that humans do that makes the computer do stuff. You can tell a computer what to do with the keyboard, the mouse, the camera or the microphone. If you wear a computer on your wrist, such as an Apple Watch, your heartbeat may send commands to the computer. Touchscreens on cellphones sense your finger and use that movement as input.

2. Storage

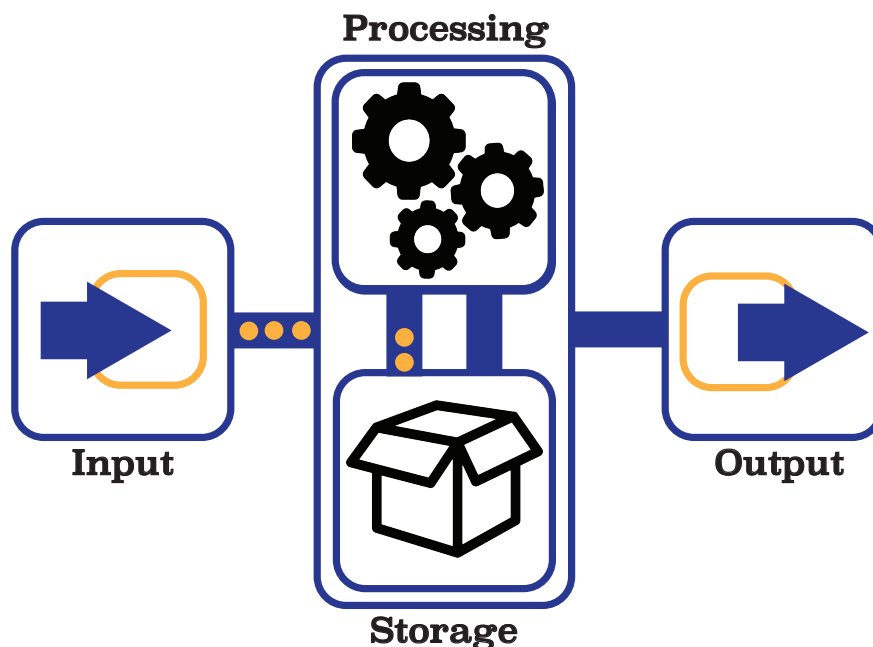
Different inputs give the computer information that is *stored* in the computer's memory.

3. Processing

A computer's processor takes information from memory, it manipulates or changes it using a series of commands and then it sends the processed information back to be stored in memory.

4. Output

How a computer outputs information depends on what the computer is designed to do. A computer display can show texts, photos, videos or interactive games. The output may also be commands to control a robot.



Binary Number System

You may have heard that computer work on ones and zeroes. Almost nobody today actually deals with these ones and zeroes but they play a big role in how computers work on the inside. Information taken from input is stored and processed using these ones and zeroes and eventually converted back into something humans understand.

In the decimal number system, we have ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

In the binary number system, we only have two digits, 0 and 1.

In the decimal number system, we have the ones digit, the tens, the hundredths, etc.

In the binary number system, we have the ones digit, the twos, the fours, the eights, etc.

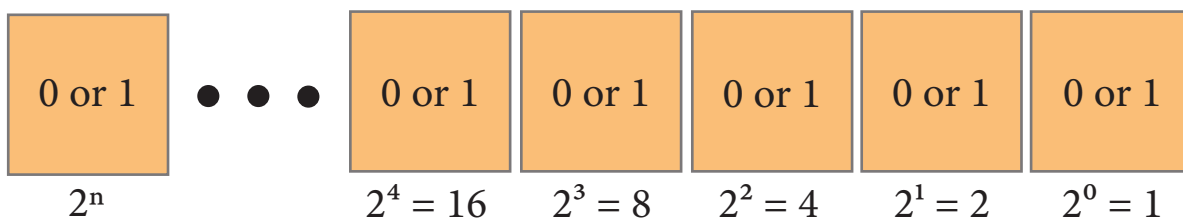
*Notice that each position is a **multiple of two!***

All numbers in the decimal number system can be represented as a binary number.

Example: The number 9 can be written as the binary number, **1001**. To see why:

$$(8 \times 1) + (4 \times 0) + (2 \times 0) + (1 \times 1) = 9$$

Each position holds a power of 2 starting at $2^0 = 1$. A binary number of length n would be:



To find the binary number as a **decimal number**, we simply multiply the 0 or 1 in each position by the power of 2 of that position and we add it all together.

Example: The binary number 101101 is the decimal number **45**. To see why:

$$(32 \times 1) + (16 \times 0) + (8 \times 1) + (4 \times 1) + (2 \times 0) + (1 \times 1) = 45$$

Exercise Set 1

1. Using 5 digits, represent the following decimal numbers as a binary number.

$$0 = \underline{00000}$$

$$1 = \underline{00001}$$

$$2 = \underline{00010}$$

$$3 = \underline{00011}$$

$$4 = \underline{00100}$$

$$5 = \underline{00101}$$

$$6 = \underline{00110}$$

$$7 = \underline{00111}$$

$$8 = \underline{01000}$$

$$9 = \underline{01001}$$

$$10 = \underline{01010}$$

$$11 = \underline{01011}$$

$$12 = \underline{01100}$$

$$13 = \underline{01101}$$

$$14 = \underline{01110}$$

$$15 = \underline{01111}$$

$$16 = \underline{10000}$$

$$17 = \underline{10001}$$

$$18 = \underline{10010}$$

$$19 = \underline{10011}$$

$$20 = \underline{10100}$$

$$21 = \underline{10101}$$

$$22 = \underline{10110}$$

$$23 = \underline{10111}$$

$$24 = \underline{11000}$$

$$25 = \underline{11001}$$

$$26 = \underline{11010}$$

$$27 = \underline{11011}$$

$$28 = \underline{11100}$$

$$29 = \underline{11101}$$

$$30 = \underline{11110}$$

2. Represent the following binary numbers as a decimal number.

$$(a) 1 = \underline{1}$$

$$(e) 100000 = \underline{32}$$

$$(b) 1000000 = \underline{64}$$

$$(f) 00000000 = \underline{0}$$

$$(c) 111111 = \underline{63}$$

$$(g) 0010 = \underline{2}$$

$$(d) 00000001 = \underline{1}$$

$$(h) 11 = \underline{3}$$

Text in Binary

Think of all the letters in the alphabet from *a* to *z*.

Let's assign the following numbers to each letter:

plaintext	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Now we can represent each letter in its binary form by writing the binary number representing the position number.

Example:

D	O	G
4	15	7
00100	01111	00111

Similarly we can find what word a series of binary numbers represents.

Example:

00001	00011	00101
1	3	5
A	C	E

Exercise: Convert the following binary numbers into text. Notice that it's 2 words!

01101 00001 10100 01000 00011 01001 10010 00011 01100 00101 10011

First convert each binary number to its decimal number to get:

13 1 20 8 3 9 18 3 12 5 19

Using the chart above to get the corresponding letters, we get: **“Math Circles”**.

Exercise: Convert your name into a binary code.

Answers may vary.

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.

Why do we even sort things?

Look at the following list and find the smallest number:

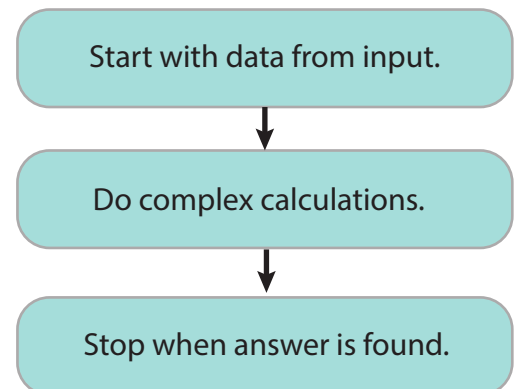
8 1 5 13

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 scientist 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.

Algorithm

An **algorithm** is a set of steps to accomplish a task. 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. We tell computers what to do by creating an algorithm for them to follow. In Computer Science, algorithms such as the one on the right are used often.



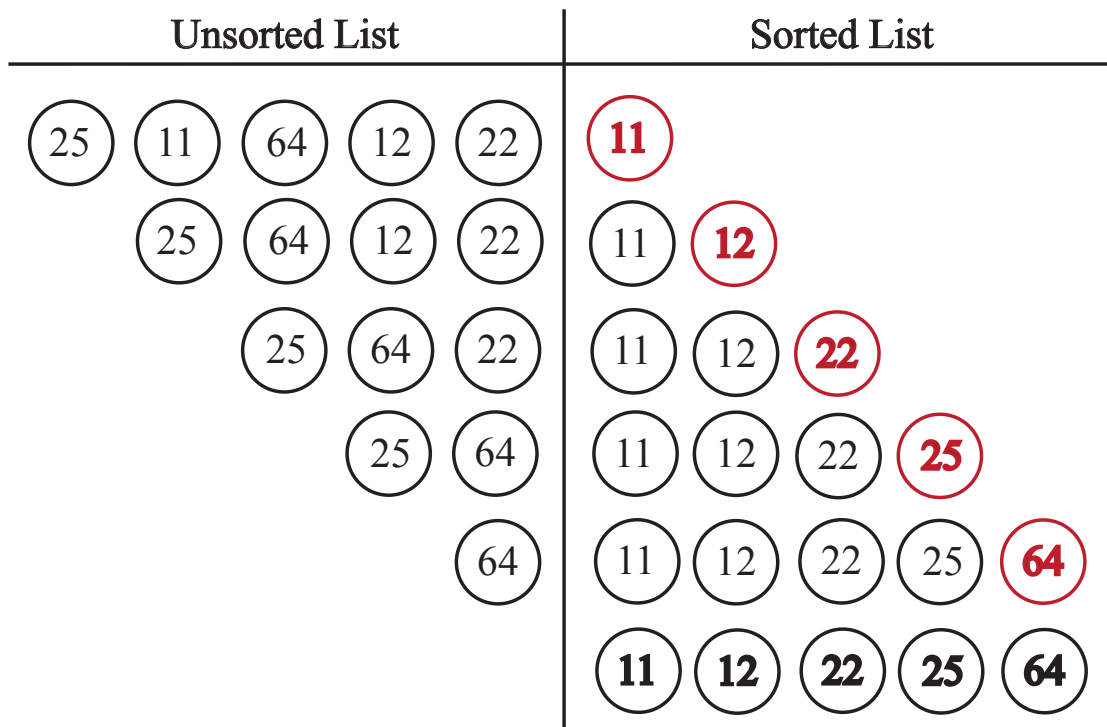
Selection Sort

A **selection sort** uses the following algorithm to sort a list of numbers into **ascending** (increasing) order:

1. Create an empty list for the sorted numbers.
2. Find the smallest number in the unsorted list and add it to the end of the sorted list.
3. Repeat step 2 until the unsorted list is empty.

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

25 11 64 12 22



Exercise: What would we have to change to get the list in *descending* order?

Instead of finding the smallest number each time, we would find the largest number and add it to the end of the sorted list.

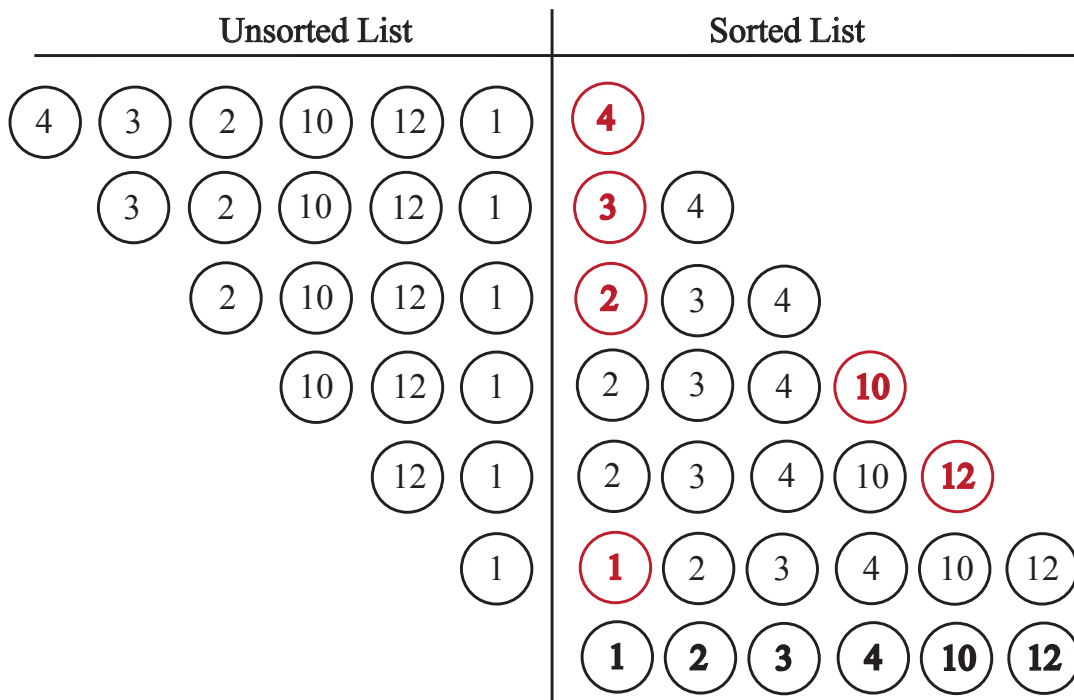
Insertion Sort

An **insertion sort** sorts a list of numbers similar to how we sort playing cards in our hands. This sort uses the following algorithm to sort a list of numbers into **ascending** (increasing) order:

1. Create an empty list for the sorted numbers.
2. Move the first number from the unsorted list to the sorted list.
3. Move the next number from the unsorted to the end of sorted and keep comparing it to the numbers before it until the new number is **inserted** into the right spot.
4. Repeat step 3 until the unsorted list is empty.

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

4 3 2 10 12 1



Bubble Sort

Unlike the sorts mentioned above, **bubble sort** does not move elements one at a time from an unsorted list to a sorted list. The bubble sort repeatedly goes through the list and compares each pair until the largest number is moved to the end of the list. The sort repeats these steps till the list is sorted into **ascending** (increasing) order:

1. Look at the first pair of numbers.
2. If they are not in 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:

29 14 38 2 5

29 14 38 2 5 → 14 29 38 2 5

14 29 38 2 5 → 14 29 38 2 5

14 29 38 2 5 → 14 29 2 38 5

14 29 2 38 5 → 14 29 2 5 38

14 29 2 5 38 → 14 29 2 5 38

14 29 2 5 38 → 14 2 29 5 38

14 2 29 5 38 → 14 2 5 29 38

14 2 5 29 38 → 14 2 5 29 38

Continuing the bubble sort we get:

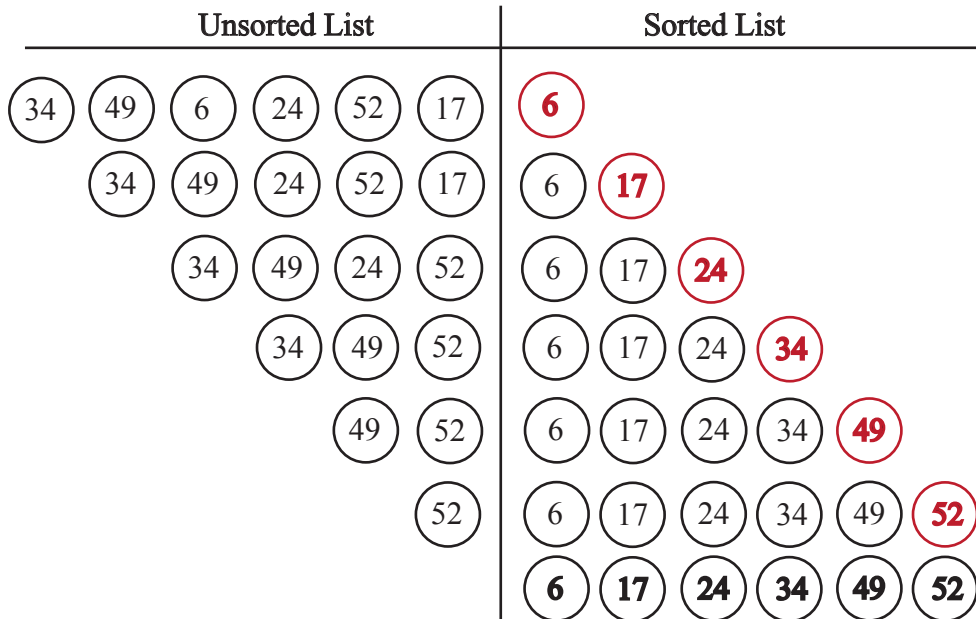
14 2 5 29 38 → 2 14 5 29 38
2 14 5 29 38 → 2 5 14 29 38
2 5 14 29 38 → 2 5 14 29 38
2 5 14 29 38 → 2 5 14 29 38
2 5 14 29 38 → 2 5 14 29 38
2 5 14 29 38 → 2 5 14 29 38
2 5 14 29 38 → 2 5 14 29 38
2 5 14 29 38 → 2 5 14 29 38

Since no swaps were made in the last cycle of the Bubble Sort, then we are done and our list is sorted.

Exercise Set 2

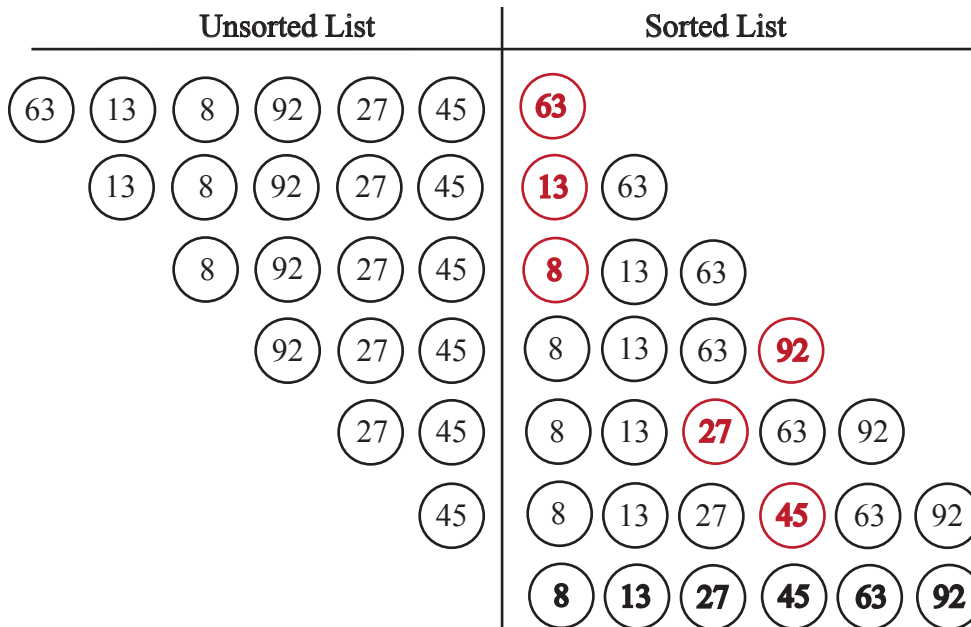
1. Sort the following list of numbers using selection sort.

34 49 6 24 52 17



2. Sort the following list of numbers using insertion sort.

63 13 8 92 27 45



3. Sort the following list of numbers using bubble sort.

11 31 65 20 7 48

11 31 65 20 7 48 → 11 31 65 20 7 48
 11 31 65 20 7 48 → 11 31 65 20 7 48
 11 31 65 20 7 48 → 11 31 20 65 7 48
 11 31 20 65 7 48 → 11 31 20 7 65 48
 11 31 20 7 65 48 → 11 31 20 7 48 65

11 31 20 7 48 65 → 11 31 20 7 48 65
 11 31 20 7 48 65 → 11 20 31 7 48 65
 11 20 31 7 48 65 → 11 20 7 31 48 65
 11 20 7 31 48 65 → 11 20 7 31 48 65
 11 20 7 31 48 65 → 11 20 7 31 48 65

11 20 7 31 48 65 → 11 20 7 31 48 65
 11 20 7 31 48 65 → 11 7 20 31 48 65
 11 7 20 31 48 65 → 11 7 20 31 48 65
 11 7 20 31 48 65 → 11 7 20 31 48 65
 11 7 20 31 48 65 → 11 7 20 31 48 65

11 7 20 31 48 65 → 7 11 20 31 48 65
 7 11 20 31 48 65 → 7 11 20 31 48 65
 7 11 20 31 48 65 → 7 11 20 31 48 65
 7 11 20 31 48 65 → 7 11 20 31 48 65
 7 11 20 31 48 65 → 7 11 20 31 48 65

Go through the list again to see there are no swaps before deciding that it's sorted.

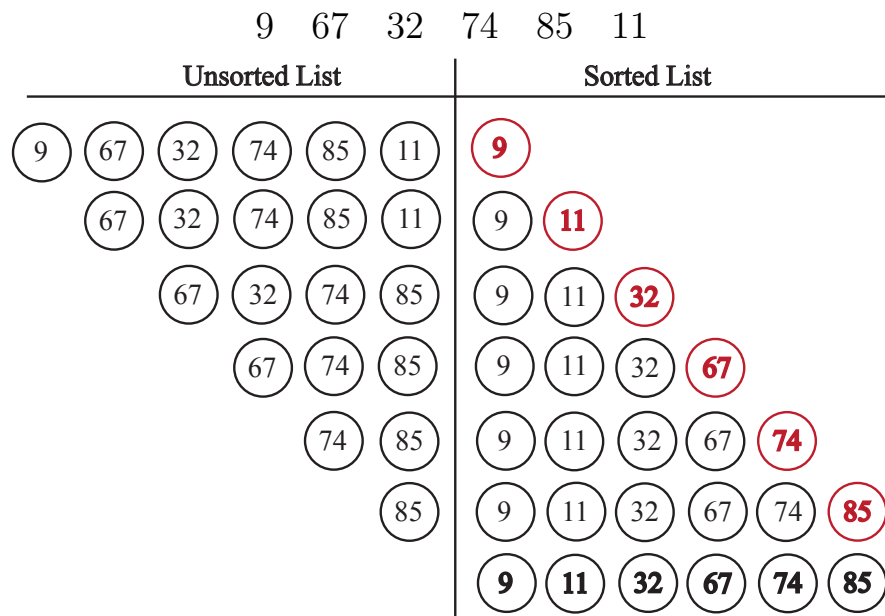
4. What is another algorithm to sort a list of numbers? Answers may vary.

Problem Set

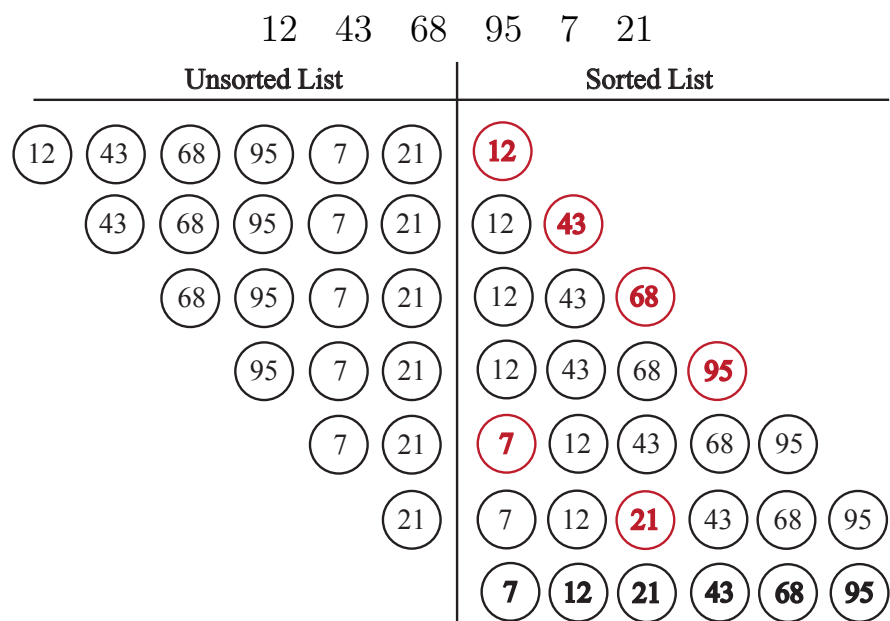
1. Give an example of when you have used an algorithm in your own life.

Answers may vary.

2. Sort the following list of numbers using selection sort.



3. Sort the following list of numbers using insertion sort.



4. Sort the following list of numbers using bubble sort.

41 84 14 79 26 53

41 84 14 79 26 53 → 41 84 14 79 26 53

41 84 14 79 26 53 → 41 14 84 79 26 53

41 14 84 79 26 53 → 41 14 79 84 26 53

41 14 79 84 26 53 → 41 14 79 26 84 53

41 14 79 26 84 53 → 41 14 79 26 53 84

41 14 79 26 53 84 → 14 41 79 26 53 84

14 41 79 26 53 84 → 14 41 79 26 53 84

14 41 79 26 53 84 → 14 41 26 79 53 84

14 41 26 79 53 84 → 14 41 26 53 79 84

14 41 26 53 79 84 → 14 41 26 53 79 84

14 41 26 53 79 84 → 14 41 26 53 79 84

14 41 26 53 79 84 → 14 26 41 53 79 84

14 26 41 53 79 84 → 14 26 41 53 79 84

14 26 41 53 79 84 → 14 26 41 53 79 84

14 26 41 53 79 84 → 14 26 41 53 79 84

14 26 41 53 79 84 → 14 26 41 53 79 84

14 26 41 53 79 84 → 14 26 41 53 79 84

14 26 41 53 79 84 → 14 26 41 53 79 84

14 26 41 53 79 84 → 14 26 41 53 79 84

14 26 41 53 79 84 → 14 26 41 53 79 84

5. Sort the following list using selection sort, insertion sort, and bubble sort.

25 13 6 9 1 38

(a) How many steps did the selection sort take?

(Each time you move a number it is a step)

Each time you find the smallest number in the unsorted list and add it to the sorted list so there are 6 steps.

(b) How many steps did the insertion sort take?

(Each time you move a number and each time you compare it to the numbers in the sorted list till it is in the right spot is a step)

Each time you move the next number in the unsorted list to the end of the sorted list is a step so there are 6 moves over. You first move 25. Once you've moved 25, you move 13 and make a comparison and move 13 so it's before 25 and so you've made another move (3 moves so far). Now you move 6 but 6 needs to come before 25 and before 13 so you move it 2 moves over and you have 6 moves so far. Repeating this you'll have overall 3 moves for moving 9, 5 moves for moving 1 and 1 move for moving 38. In total you have $6 + 3 + 5 + 1 = 15$ steps.

(c) How many steps did the bubble sort take?

(The comparison of each pair is a step whether you swap them or not)

In each cycle through the list, there are 5 comparisons and if you correctly complete the cycles, you should end up with 5 cycles in total (*don't forget the last cycle where you go through to make sure there are no comparisons*). Thus there are 25 steps in total.

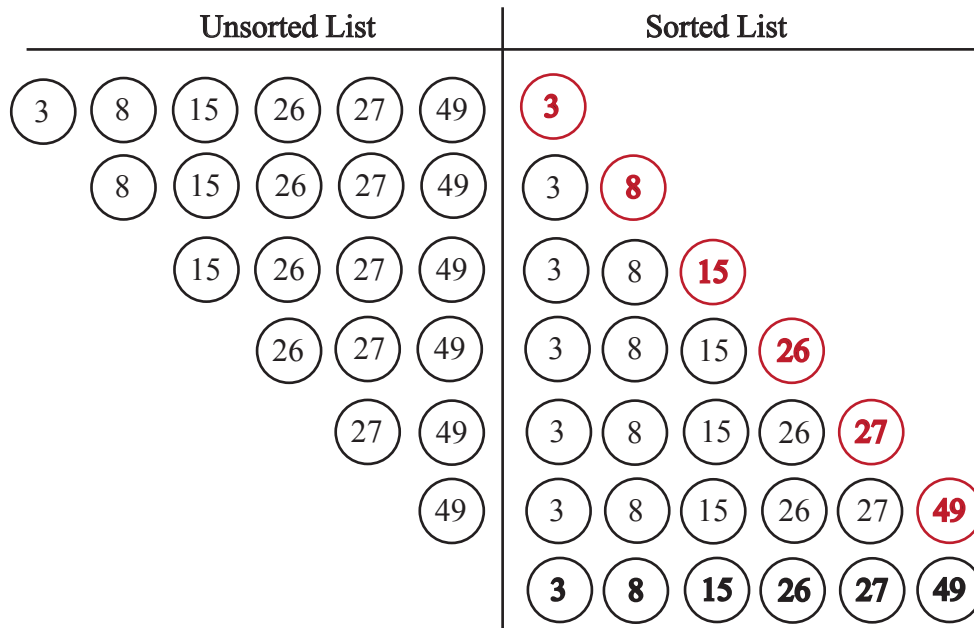
(d) Which of these methods is the most efficient or the fastest?

In order, the fastest sorts are the selection sort, the insertion sort and finally the bubble sort.

6. As humans, we can quickly observe that the following list is sorted and does not need to be sorted any further. However, computers can't see this and given this list, they will still complete the steps to any of the sorting algorithms above. Go through the following list using selection sort, insertion sort, and bubble sort as a computer would.


3 8 15 26 27 49

The selection sort and insertion sort would look the same for this list and would be as follows:

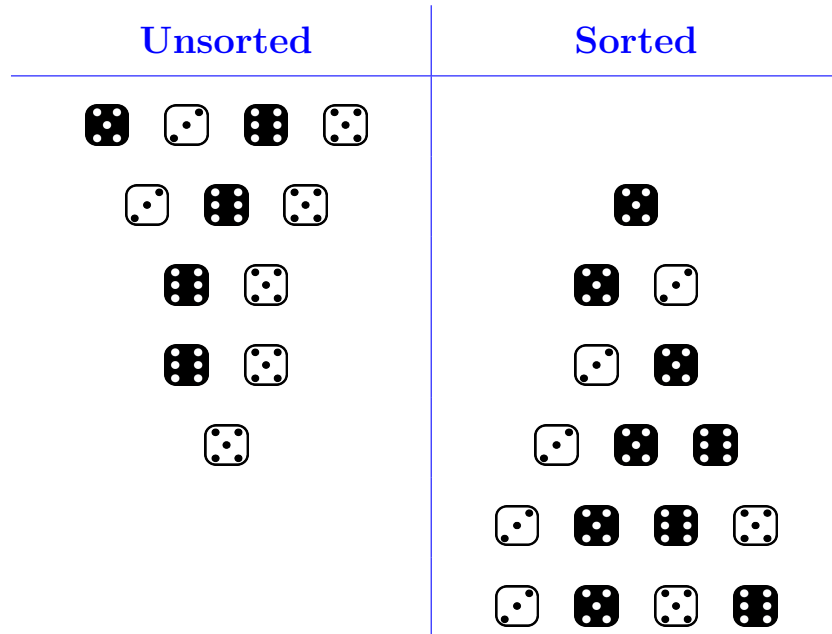


The bubble sort would look as follows:

3 8 15 26 27 49 → 3 8 15 26 27 49
 3 8 15 26 27 49 → 3 8 15 26 27 49
 3 8 15 26 27 49 → 3 8 15 26 27 49
 3 8 15 26 27 49 → 3 8 15 26 27 49
 3 8 15 26 27 49 → 3 8 15 26 27 49

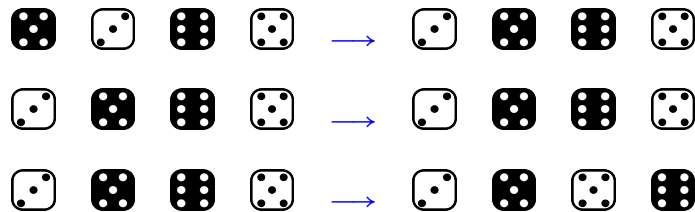
7. Sort the following dice using the algorithm specified below: 
 Make sure the show which colour each die is.

(a) Use insertion sort.

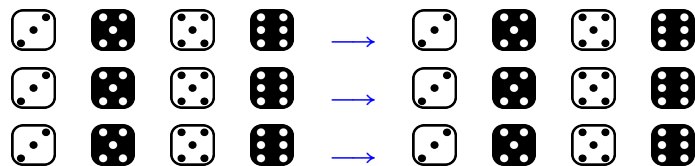


Remember that in the algorithm for insertion sort, we stop swapping elements in the sorted list when the previous element is smaller than or equal to the one being moved along the list.



(b) Use bubble sort.



2 swaps were made so repeat.



No swaps were made so sorting is complete.

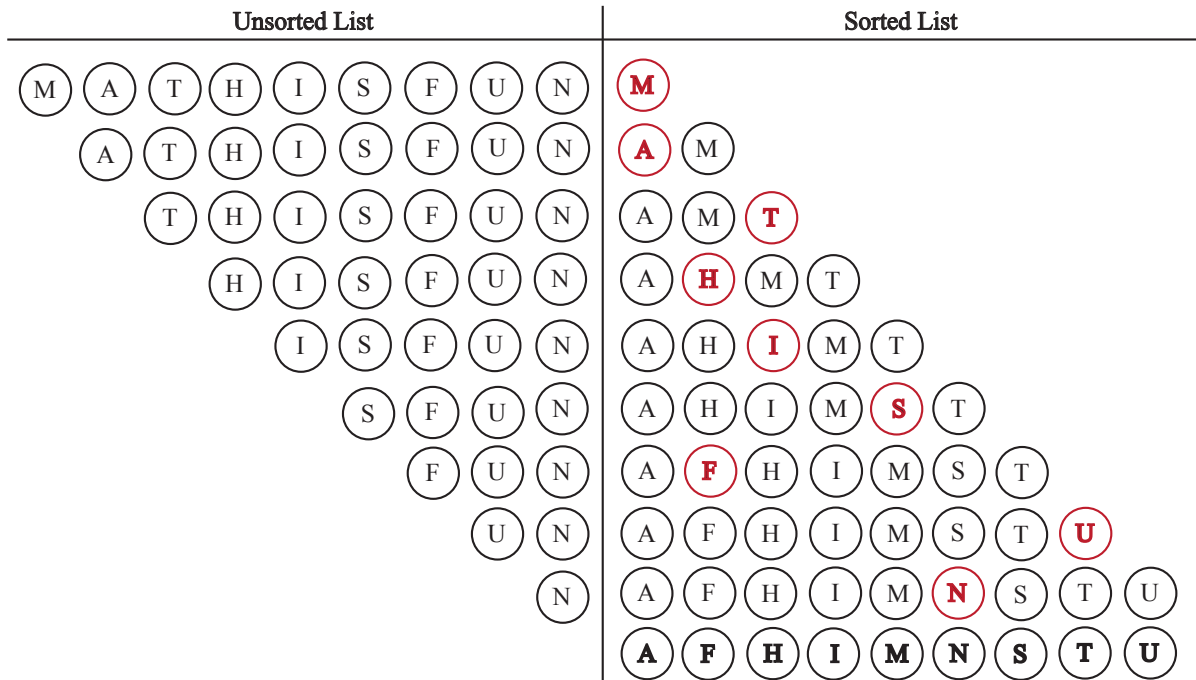
(c) You probably noticed that there were two repeated elements in this list. We say a sorting algorithm is **stable** if repeated elements stay in the same order before and after a list is sorted. Which of the two algorithms we just looked at are stable? (Which algorithms resulted in a sorted list where  is before ?)

In this case, both the insertion sort and the bubble sort are stable.

8. Sort the following letters in alphabetical order using the algorithms specified below.

M A T H I S F U N

(a) Use insertion sort.



(b) Use bubble sort.

You should end up with the same results as above.

9. Based on the following steps showing a list of 3 digit numbers being sorted with an algorithm called **Radix Sort**, write the down what you think the algorithm is.

365 502 560 299 101 462 401 902

365 502 560 299 101 462 401 902

↓

560 101 401 502 462 902 365 299

↓

560 101 401 502 462 902 365 299

↓

101 401 502 902 560 462 365 299

↓

101 401 502 902 560 462 365 299

↓

101 299 365 401 462 502 560 902

- (a) Sort the list the third digit (ones digit).
- (b) Sort the list by the second digit (tens digit).
- (c) Sort the list by the third digit (hundreds digit).

What makes this algorithm efficient is that instead of sorting the numbers themselves, each step places the associated digit into its correct category. You can think of each step as the computer having different buckets for the digits 0 to 9 and, depending on what that digit is in each element, the algorithm puts the element in its appropriate bucket. This way, you are not comparing the numbers to each other but individually putting each in a bucket three times based on its three digits.

10. This question compares insertion sort and radix sort, the algorithm you observed in the previous question.

(a) Sort the following lists of numbers with both insertion and radix:

i. 256 702 524 816

Insertion sort

Unsorted	Sorted
256 702 524 816	
702 524 816	<u>256</u>
524 816	256 <u>702</u>
816	256 702 <u>524</u>
816	256 <u>524</u> 702
	256 524 702 <u>816</u>

Radix sort

```

256 702 524 816
  ↓
702 524 256 816
  ↓
702 816 524 256
  ↓
256 524 702 816

```

ii. 256 702 524 816 130

Insertion sort

Unsorted	Sorted
256 702 524 816 130	
702 524 816 130	<u>256</u>
524 816 130	256 <u>702</u>
816 130	256 702 <u>524</u>
816 130	256 <u>524</u> 702
130	256 524 702 <u>816</u>
	256 524 702 816 <u>130</u>
	256 524 702 <u>130</u> 816
	256 524 <u>130</u> 702 816
	256 <u>130</u> 524 702 816
	<u>130</u> 256 524 702 816

Radix sort

256 702 524 816 130
 ↓
130 702 524 256 816
 ↓
702 816 524 130 256
 ↓
130 256 524 702 816

iii. 256 702 524 816 130 888

Insertion sort

Unsorted	Sorted
256 702 524 816 130 888	
702 524 816 130 888	<u>256</u>
524 816 130 888	256 <u>702</u>
816 130 888	256 702 <u>524</u>
816 130 888	256 <u>524</u> 702
130 888	256 524 702 <u>816</u>
888	256 524 702 816 <u>130</u>
888	256 524 702 <u>130</u> 816
888	256 524 <u>130</u> 702 816
888	256 <u>130</u> 524 702 816
888	<u>130</u> 256 524 702 816
	130 256 524 702 816 <u>888</u>

Radix sort

256 702 524 816 130 888
 ↓
130 702 524 256 816 888
 ↓
702 816 524 130 256 888
 ↓
130 256 524 702 816 888

(b) Which algorithm took less steps to complete for the three lists?

In general, radix sort uses the least number of steps.

(c) What is the relationship between the length of the lists and the number of steps it takes to sort them for both algorithms.

For radix sort, only 3 new “steps” were added for every additional element (putting the new element into the appropriate order for each digit). For insertion sort however, there is generally a significant increase in the number of steps/comparisons needed to sort the list with the added element.

11. * For every algorithm we looked at in Math Circles and the problem set, create a worst case list from the numbers 1, 2, 3, 4, 5, 6, 7, 8. The worst case is the sequence of the eight numbers which will take the most steps to sort.

Selection sort: 8, 7, 6, 5, 4, 3, 2, 1

The computer has to go through each number to find the smallest number so in this case, each time it is looking for the smallest number it has to go through the entire list.

Insertion sort: 8, 7, 6, 5, 4, 3, 2, 1

Each time a number is moved to the unsorted list, it needs to be compared to every number before it before it reaches the right spot.

Bubble sort: 8, 7, 6, 5, 4, 3, 2, 1

The pattern in a bubble sort is to move the largest unsorted number to the end of the list in each cycle so in a reverse ordered list, it would take the longest.

Radix sort: The initial sequence does not matter since these are all 1 digit numbers.

12. * When we use bubble sort, we go through and compare each pair in the list one last time at the end of the algorithm to make sure every element is sorted into its correct place. However, during these last comparisons, the list is already sorted! How can we change the algorithm so it does not need to go through the already sorted list one last time to check that every element is in the correct order.

Go back and look at the previous examples of bubble sort and see if you can notice a pattern when you are carrying out the steps of sorting.

Let's take a look at the Tournament of Algorithms example. Notice that after going through every pair in the list the first time, the largest number has been moved to its correct spot at the end of the list. After going through the entire list a second time, the second largest number has been moved to its correct spot as well. If you use the worst case of Bubble sort from the previous question, you can see that this pattern will continue until the whole list is sorted. You can change the bubble sort algorithm so that every time when it finishes going through the entire list, it cuts off the last element because it is already in the correct place and ignores it for the rest of the sorting. This means that in some cases we can ignore the last step of checking if everything is in the right order because we know that to sort n elements, bubble sort will need to go through the entire list at most $n - 1$ times and that each time, there will be one less comparison you have to make.