Computer Programming – CS 6011
Lecture 6: Interfaces

Fall 2023

Topics

- Generic Programming
- Interfaces
- Synthesizer Project

Generic Programming

- In our game program we wrote two weeks ago… Many people had objects such as:
  - Player, Ball, Enemies, Paddle, ..
  - To make them move, etc, what did we do?
    - update()
    - draw()

Generic Programming

```
vector<Ball> balls;
vector<Enemy> enemies;
for( Ball    ball : balls )  {
            ball.update()
            ball.draw()
}
```

- Now what about enemies?
  - Use the same for loop structure…
  - How could we fix this?

Generic Programming

- Could create a templated function:

```
template <typename T>
void updateAndDraw( vector<T> items ) {
	for( auto & item : items ) {
		item.update();
		item.draw();
	}
}
updateAndDraw( enemies );
updateAndDraw( balls );
```

- What assumptions have we made about items? (Remember, items can be enemies, balls, paddles, etc.)
  - They all have an update() method, and a draw() method.

Generic Programming

- We would really like to have:

```
vector< GameObject > ballsAndEnemies;
```

- Because an enemy, and a ball, etc, are all “Game Objects”.
- What is generic programming?
- What did the program need to know about the game object?
- What if we need to work with other types that we have not worked with yet?
- To define this, we use Java *Interfaces.*
- So all objects that are going to be Game Objects, must have what in common?
  - update() and draw() methods.
- We can enforce this using *Interfaces.*

Interface for Game Object Example

```
public interface GameObject {
      void update();
      void draw();
      int anotherMethod( int param );
}
```

- Note, all methods are by default public.
- Interfaces may NOT have data members?
  - Why?
- Interfaces just **describe** the methods signatures, not their implementations.

Implementing an Interface

- To tell Java that a class we are writing has (and must have) all the methods declared by an interface, we *implement* that interface in our class:

```
public class Enemy implements GameObject {
        @Override
        public void update() { /* Code that does actual work for updating an Enemy. */ }
        ...
}
```

- Note: you must implement ALL methods declared in the Interface.
- Of course Enemy can have other methods, constructors, and member variables that are used to help with implementing the Interface methods.
- Note: You cannot create a GameObject directly:

```
GameObject go = new GameObject(); // WILL NOT COMPILE – DOES NOT MAKE SENSE.
GameObject go = new Enemy(); // This is OK, an Enemy is a game object.
```

Another Interface Example

- As another example, let's think about the Scanner class.
- What common data/functions have we used with it?
- Does other classes use the same methods?
- To ensure that they all implement the same methods, we use the concept of *Interface.*
- An `Interface` is a data-type that does not implement its methods, only specifies their signatures.

Class objects that share an Interface can be added to the same ArrayList

```
Enemy e = new Enemy();
Ball  b = new Ball();

ArrayList< GameObject > gos = new ArrayList<>();
gos.add( e );
gos.add( b );

GameObject b2 = new Ball(); // Also valid
```

When to use Interfaces?

- Classes that are used in the same context but do not share a lot of code.
- Do similar things, but probably in different ways.

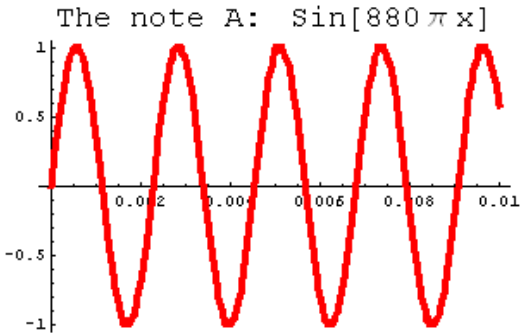
Synthesizer Project

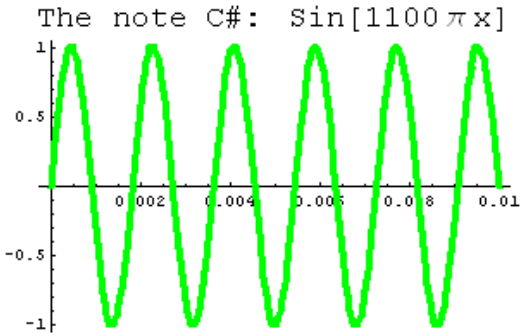
- Components
- Functions you might need

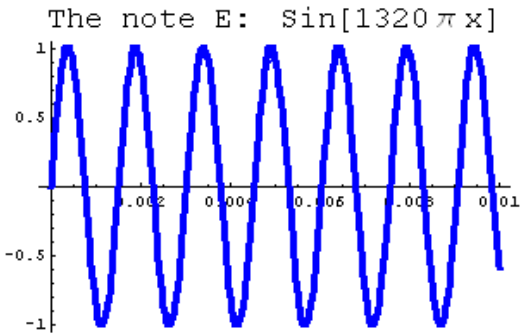
Synthesizer Project

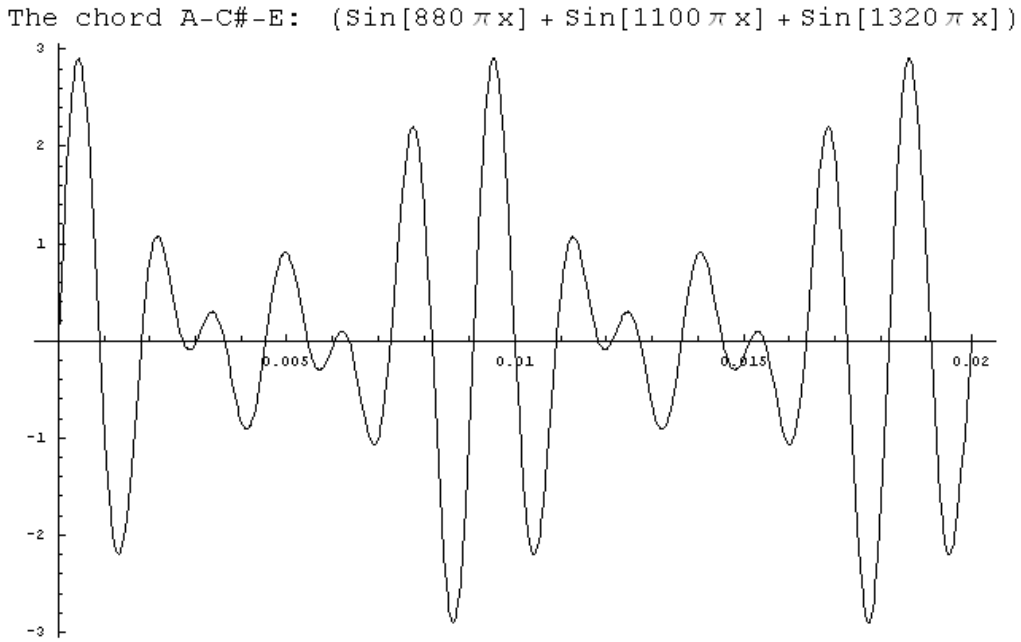
Synthesizer

- What are sounds?
  - Waves.
- What about multiple sounds at the same time?
  - Additive waves.

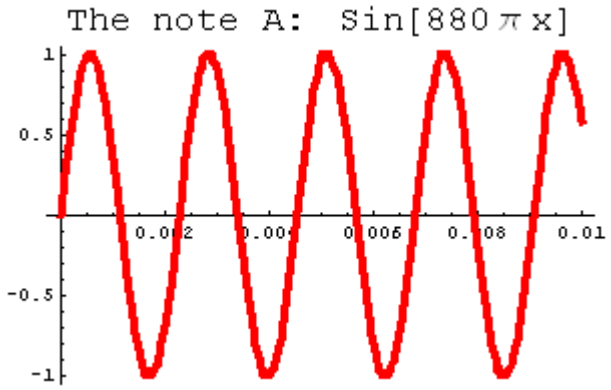








Sound Wave

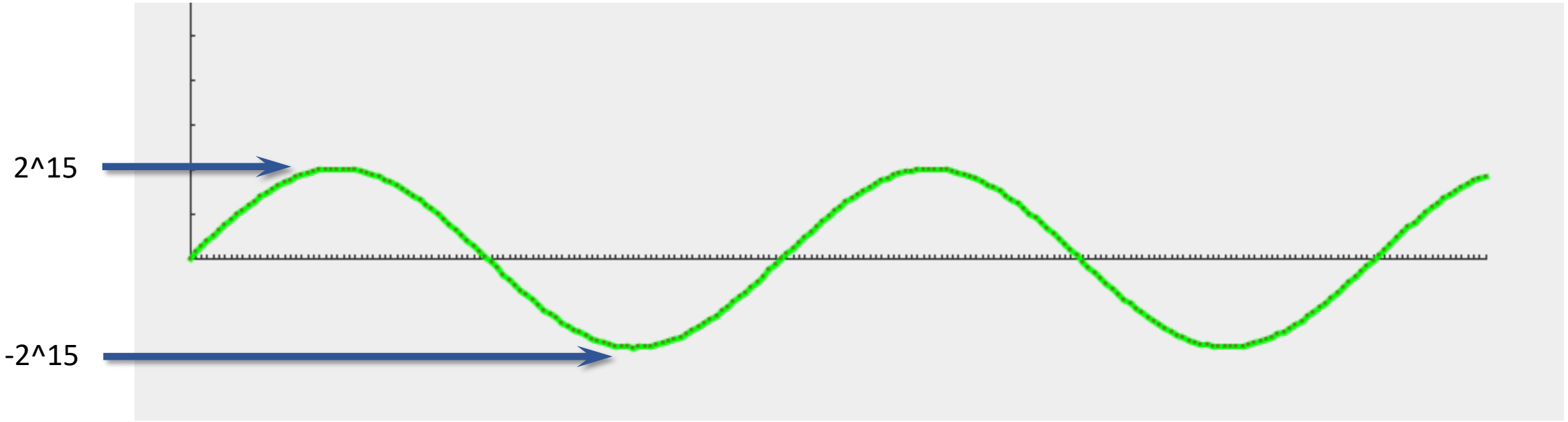


- Sound waves are continuous functions.
  - What does this mean?
    - At X == 0.00000006 there is a value.
    - At X == 0.00000007 there is a value.
    - But there is also a value at:
    - X == 0.000000065
    - And every other position – thus an infinite number of values across all times.
- Can we store this in a computer?
  - No – So how could we store it?
  - Discretize (sample) it at a certain number of X values.
    - x == 0, x == .1, x == .2, x == .3, etc

Synthesizer Project

- Mono, 16 bit per sample, 44.1 KHz audio clips
- 2 seconds in duration

Discrete Sound Wave

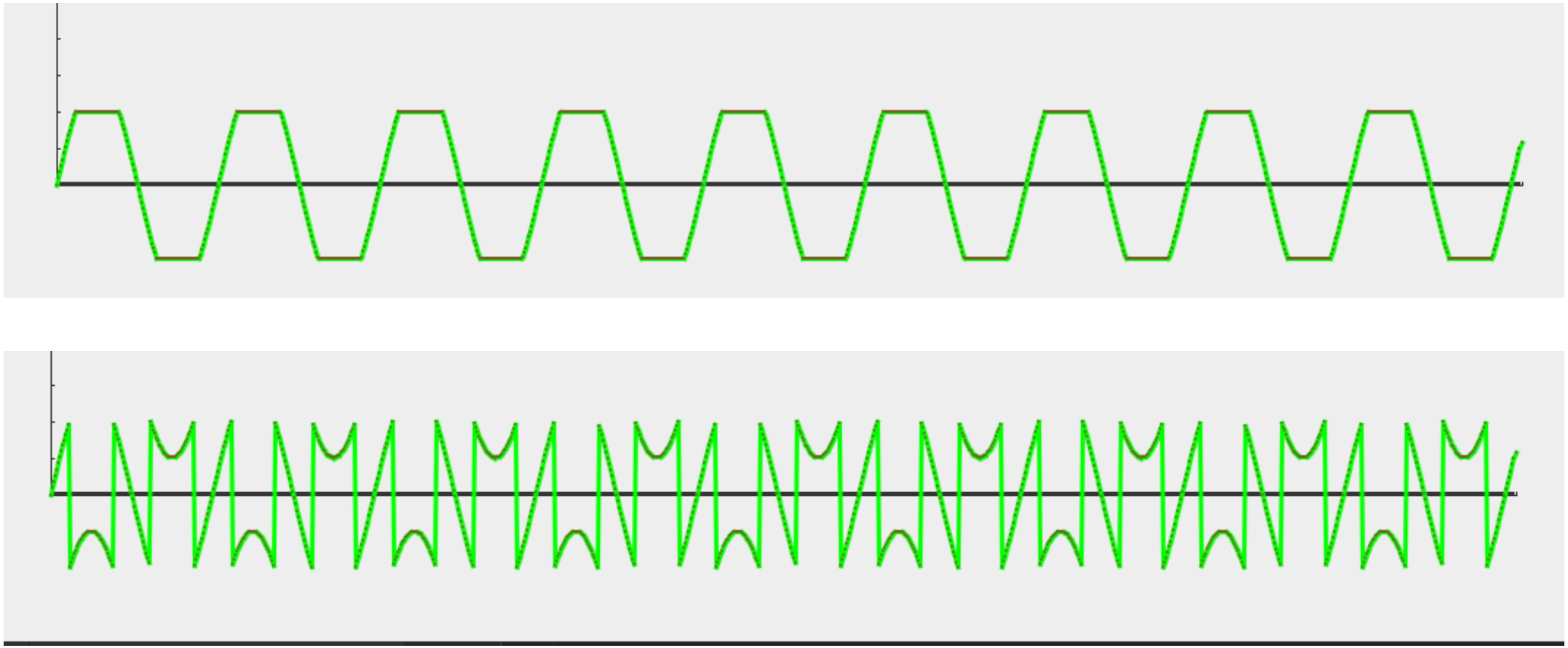


- What data structure can we use to hold this information

```
byte[] data;          // And array of bytes over time.
                      // 44100 samples per second (Data rate for CDs.)
```

- What type can store -2^15 to 2^15?
  - Short

Clamped vs Unclamped Overflow



Storing These Waves

- We need something to store these waves and allow us to manipulate them.
  - Need a class.
  - What should we name it?
    - A sensible name: AudioClip
  - What information does it store?
    - The array of bytes.
  - What functionality does it provide? What do we need to be able to do to that array?
    - Set values in it.
    - Get values from it.
    - Get the entire array.
- Note, the Java audio library has a class named Clip. While we will use this to send audio to the library, do not confuse it with our AudioClip.

Audio Clips

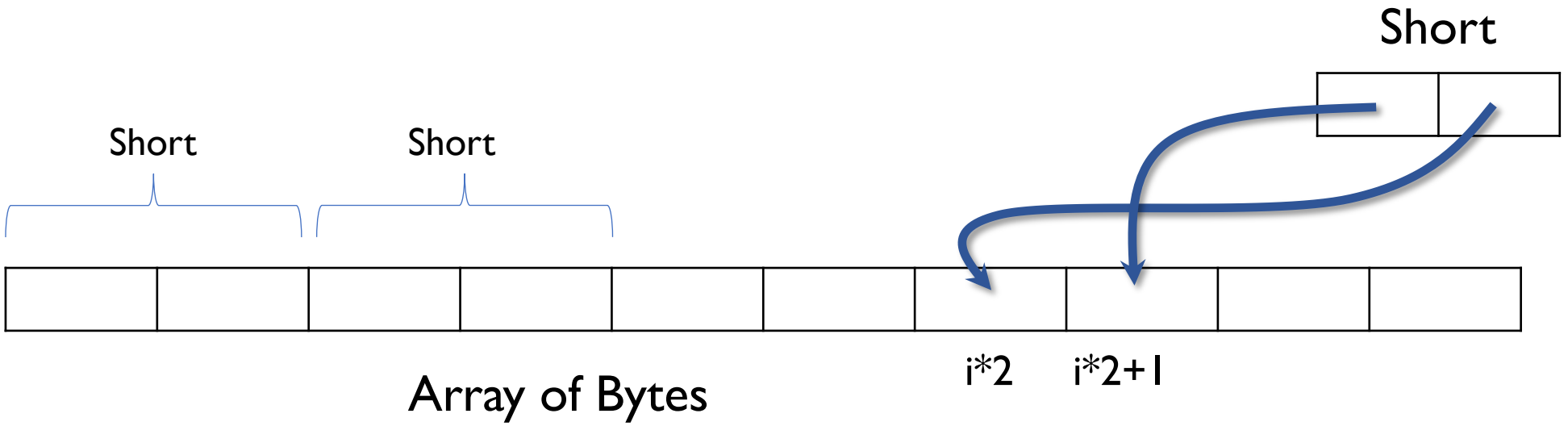
- Each sample is stored as:
  - Short
- Because the audio library requires an array of byes, but we need Short data types to represent the range of values we need, we have to encode the shorts into bytes.
  - all audio data sent to them needs to be converted into a byte array.
- So our AudioClip class will store a byte[], but when we set or get a single sample (a short), we will touch 2 bytes.

Short to Bytes

- Bitwise operations to put a short into a byte
- >>
  - Bit shifting with sign extension
- Endianess
  - Order of bytes. What is the order of these two bytes?
  - Going to use Little Endian
  - Least significant byte in first position in array, Most significant byte in the second position of array.

Short to Byte in Array

- Set the i^{th} position in an array of bytes to the value in a short.
  - Which bytes in array represent the i^{th} short?
    - `i*2, i*2+1`
    - But it what order?



Audio Components

- Play Sound
- Change Volume
- Generate White Noise
- What do these have in common?
  - All create a wave of data to send to the audio library.
  - We will call these AudioComponents for our synthesizer
  - Since multiple of these components will provide the same type of functions, we use:
    - An Interface

AudioComponent Interface

```
interface AudioComponent {
   AudioClip getClip();
   Boolean hasInputs();
   void connectInput( AudioComponent ac );
}

AudioComponent note = new SineWave( 440 );
AudioComponent vol = new VolumeAdjuster();
vol.connectInput( note );
```

VolumeAdjuster AudioComponent

- Must implement all interface methods.
- What other data does a volume adjuster need to store?

```
class VolumeAdjuster implements AudioComponent {
    @Override
    public AudioClip getClip() {
        // How do I get the sound? Where is the sound coming into this
        // VolumeAdjuster? // The input_
        AudioClip out = input_.getClip();
        // for each sample in out
        }
    private AudioComponent  input_;
    private float scale_;
}
```

Monday Assignment(s)

- Assignment – Synthesizer, Part 1
  - Not a group project, but collaborate with your classmates, but everyone will submit their own solution.
  - While you should strive to finish Part 1 today, we will have time tomorrow to finish it up.
- Code Review: Basic HTTP Web Server
- Lab – Sorting Fractions