* Overall notes:
  + Use subsonic sine wave
    - 18KHz
  + App that he’s testing us with is AudioTestApp on his github
  + Same app for module a and b
    - Use the in class assignment as a starting point
* Module A
  + **Reads from the microphone**
    - Larson’s code
  + **Takes an FFT of the incoming audio stream**
    - Larson’s code
  + **Displays the frequency of the two loudest tones within (+-3Hz) accuracy**
    - Display FFT to start
    - Possible way of doing this:
      * A screenshot of a social media post

        Description automatically generated
      * Use vDSP\_maxviD
        + Gets the max value and index of input vector
        + Remove that element from the swift list with code similar to:

var animals = ["cats", "dogs", "chimps", "moose"]

animals.remove(at: 2) //["cats", "dogs", "moose"]

* + - * + Then run the code again
        + The indexes will tell us the frequency of the loudest tones with the following amount of hertz between our elements in our fft:
        + A close up of text on a white background

          Description automatically generated

Solve Fs/N = 3 where Fs is 44.1KHz

N\_FFT = 14700 at minimum!

* + - **Have a way to "lock in" the last frequencies detected on the display**
      * Not exactly sure what he wants here if we are already displaying the 2 frequencies
  + **Is able to distinguish tones at least 50Hz apart, lasting for 200ms or more**
    - Time vs. N\_FFT tradeoff.
    - Peaks are at least 50hz apart so that helps with peak finding so we don’t just get the value to the left/right of the peak
    - Time permitting:
      * Peak interpolation
        + Just use the highest and one to left and right
        + Quadratic approx.

Good for module a, not for module b (since you know the exact frequency since you’re playing it)

* + - * Higher res FFT
        + Use 0 padding to time signal get finer resolution FFT
        + Keep FFT at same amount of points, but zero pad half the time domain signal
        + Noise in FFT:

FFT of overlapping time points

Then average out FFTs

* + An idea for Exceptional Credit: recognize two tones played on a piano (down to one half step apart) and report them by letter (i.e., A4, A#4). Must work at note A2 and above. Note: this is harder than just identifying two perfect sine waves!!
    - Is this a substitute?
  + **Exceptional Credit Idea (required for 7000 level students): make the FFT analysis follow the model-view-controller framework more closely. That is, make the model an analyzer that is not implemented in the View Controller (i.e., an "analyzer model"). All audio saving and analysis should happen in the model only, not the view controller. The audio analysis should be performed using blocks on a serial queue. Once analysis is complete, a view controller can ask the model for FFT frames, and the view controller can display those frames however it wants. You should design functions for accessing the result of the analyzer such that memory and computation time are reasonable.** 
    - Recognizing 2 tones count?
  + Verify the functionality of the application by taking of video of the app working. The sound source must be external to the phone (i.e., laptop, instrument, another phone, etc.).
* Module B
  + **Reads from the microphone**
    - Larson’s code
  + **Plays a settable (via a slider or setter control) inaudible tone to the speakers (15-20kHz)**
    - Larson’s code
  + **Displays the magnitude of the FFT of the microphone data in decibels**
    - Easy, just scale our FFT values with the following:
      * 20\*log10(|FFT|)
  + **Is able to distinguish when the user is {not gesturing, gestures toward, or gesturing away} from the microphone using Doppler shifts in the frequency**
    - Higher frequency when moving towards
    - Lower frequency when moving away
    - Monitor peaks to left and right or frequency that sine wave is playing
    - Just use thresholding for simplicity
      * Just needs to work most of the time