**Ruiyan Guo 400256752**
**Xinyu Chen 400221680**

## Fourier Transform Take-home Exercise

According to the output of the main function, it is obvious that the running time increased to 4 times of that before when the number of samples became 2 times, which is consistent with the algorithmic complexity of O(n^2).

## Filter Design Take-home Exercise

In this section, my own function `mySquare(std::vector<float> &t, std::vector<float> &x, float Fs, float interval, float frequency, float amplitude)` is written, in order to implement the square wave. For the input, 't' and 'x' are vectors that store the time and the generated square wave, the values for 'frequency' and 'amplitude' are generated by the rand() function and modulus is used to achieve the desired value range. In the 'main' function, both the time domain and frequency domain are plotted.

## Block Processing Take-home Exercise

To achieve the functionality of a block filter, my own function `mylfilter_w_block(std::vector<float> &coeff, std::vector<float> &data, int size, std::vector<float> &buffer)`, with this function, block processing can be achieved. For the inputs of this function, 'coeff' is the filter coefficient, 'data' is the piece of raw data that is required to be processed, 'size' is the block size, 'buffer' is the vector storing the last few elements from each block. Inside this function, it convolves the filter coefficients with the input raw data, it is achieved by a nested for loop, where the outer loop loops through the raw data, the inner one loops through the coefficients. This function is vector type, it returns the vector 'filtered_data' where one block of  processed data is stored.

To implement the `mylfilter_w_block` stated above, my own function `myBlockfilter_implementation` is writtened. It contains a while loop that breaks the raw audio data into blocks and assigns them to the function `mylfilter_w_block` to process. This function is vector type and returns 'filtered_data', which contains the whole-piece processed data. Notice that, in order to slice the audio data into blocks, my own function `slicing(std::vector<float>& arr, int a, int b)` is introduced, 'arr' is the vector contains the one-piece data, 'a' and 'b represent the start position and the end position of the block, it is a vector type function, it returns a vector 'result' that contains the sliced block data.

Inside the 'main' function, the code for the block processing is shown, the code for the singlepass is commented. If the in-lab exercise needs to be checked, uncomment the singlepass and comment the corresponding block processing part.