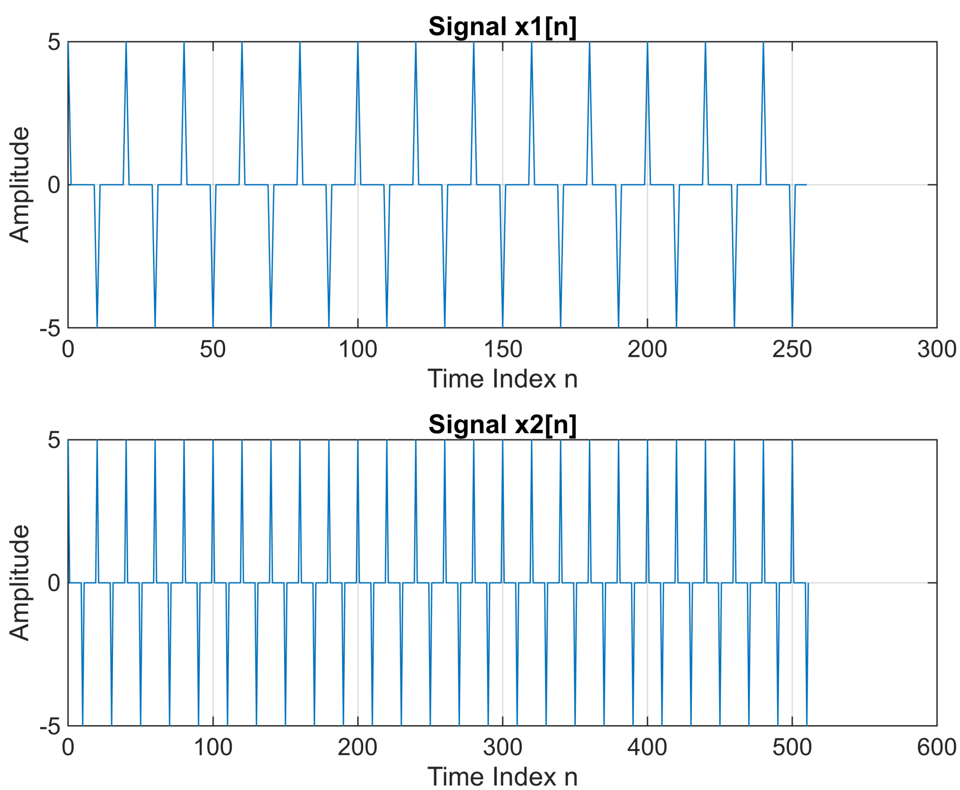
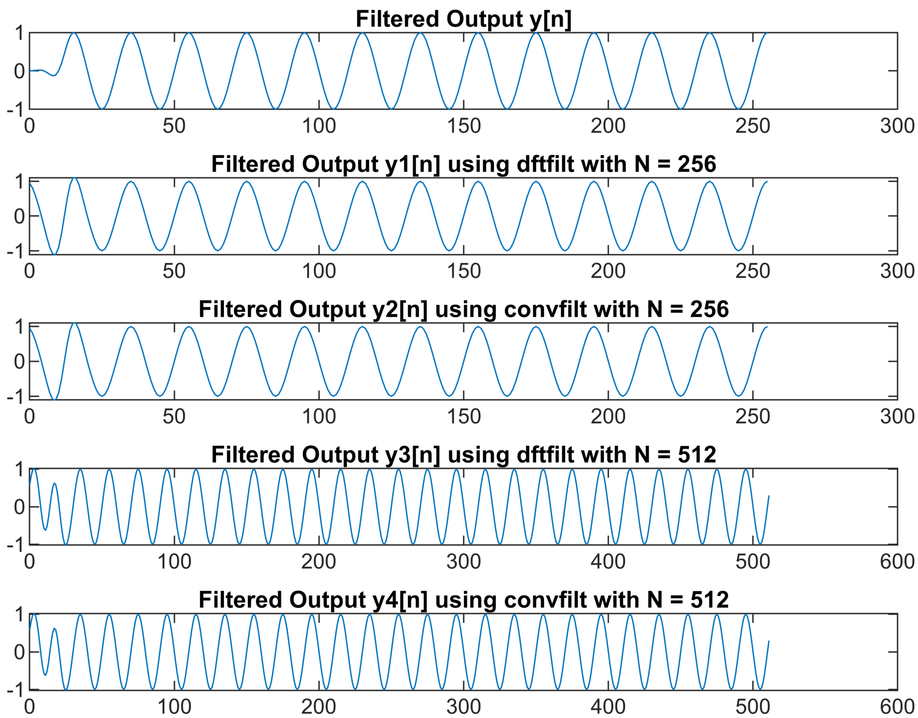
PART C

```
n1 = 0:255;
x1 = cos(2 * pi * n1 * 0.05) + cos(2 * pi * n1 * 0.15) + ...
     cos(2 * pi * n1 * 0.25) + cos(2 * pi * n1 * 0.35) + ...
     cos(2 * pi * n1 * 0.45);

n2 = 0:511;
x2 = cos(2 * pi * n2 * 0.05) + cos(2 * pi * n2 * 0.15) + ...
     cos(2 * pi * n2 * 0.25) + cos(2 * pi * n2 * 0.35) + ...
     cos(2 * pi * n2 * 0.45);

figure;
subplot(2,1,1);
plot(n1, x1);
title('Signal x1[n]');
xlabel('Time Index n');
ylabel('Amplitude');
grid on;

subplot(2,1,2);
plot(n2, x2);
title('Signal x2[n]');
xlabel('Time Index n');
ylabel('Amplitude');
grid on;
```



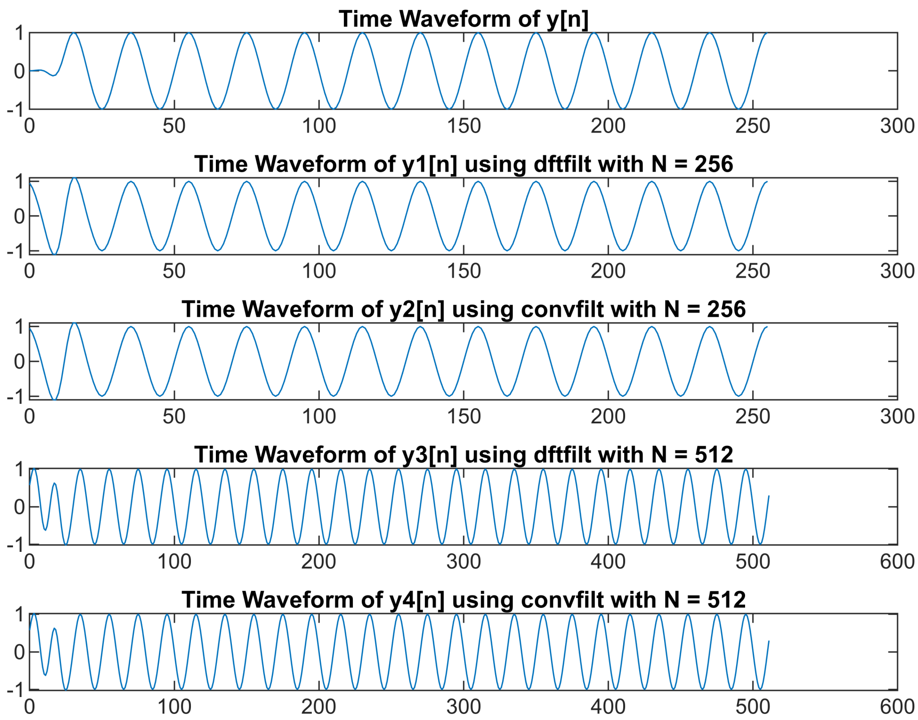
PART D

```
N = 30;
wc = 0.2 * pi;
h1 = fir1(N, wc/(pi), 'low');

n1 = 0:255;
x1 = cos(2 * pi * n1 * 0.05) + cos(2 * pi * n1 * 0.15) + ...
     cos(2 * pi * n1 * 0.25) + cos(2 * pi * n1 * 0.35) + ...
     cos(2 * pi * n1 * 0.45);

n2 = 0:511;
x2 = cos(2 * pi * n2 * 0.05) + cos(2 * pi * n2 * 0.15) + ...
     cos(2 * pi * n2 * 0.25) + cos(2 * pi * n2 * 0.35) + ...
     cos(2 * pi * n2 * 0.45);

y = filter(h1, 1, x1);
y1 = dftfilt(x1, h1, 256);
y2 = convfilt(x1, h1, 256);
y3 = dftfilt(x2, h1, 512);
y4 = convfilt(x2, h1, 512);

figure;
subplot(5,1,1);
plot(n1, y);
title('Filtered Output y[n]');

subplot(5,1,2);
plot(n1, y1);
title('Filtered Output y1[n] using dftfilt with N = 256');

subplot(5,1,3);
plot(n1, y2);
title('Filtered Output y2[n] using convfilt with N = 256');

subplot(5,1,4);
plot(n2, y3);
title('Filtered Output y3[n] using dftfilt with N = 512');

subplot(5,1,5);
plot(n2, y4);
title('Filtered Output y4[n] using convfilt with N = 512');
```

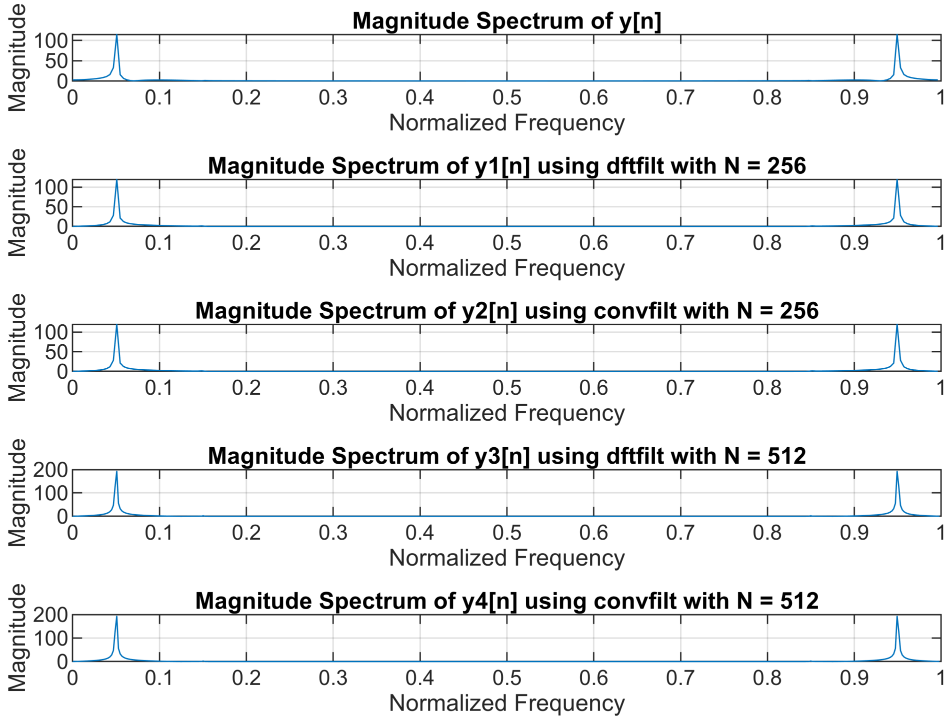


PART E

```
figure;
subplot(5,1,1);
plot(n1, y);
title('Time Waveform of y[n]');

subplot(5,1,2);
plot(n1, y1);
title('Time Waveform of y1[n] using dftfilt with N = 256');

subplot(5,1,3);
plot(n1, y2);
title('Time Waveform of y2[n] using convfilt with N = 256');

subplot(5,1,4);
plot(n2, y3);
title('Time Waveform of y3[n] using dftfilt with N = 512');

subplot(5,1,5);
plot(n2, y4);
title('Time Waveform of y4[n] using convfilt with N = 512');
```




```
figure;
subplot(5,1,1);
plotMagnitudeSpectrum(y, 256);
title('Magnitude Spectrum of y[n]');

subplot(5,1,2);
plotMagnitudeSpectrum(y1, 256);
title('Magnitude Spectrum of y1[n] using dftfilt with N = 256');

subplot(5,1,3);
plotMagnitudeSpectrum(y2, 256);
title('Magnitude Spectrum of y2[n] using convfilt with N = 256');

subplot(5,1,4);
plotMagnitudeSpectrum(y3, 512);
title('Magnitude Spectrum of y3[n] using dftfilt with N = 512');

subplot(5,1,5);
plotMagnitudeSpectrum(y4, 512);
title('Magnitude Spectrum of y4[n] using convfilt with N = 512');
```

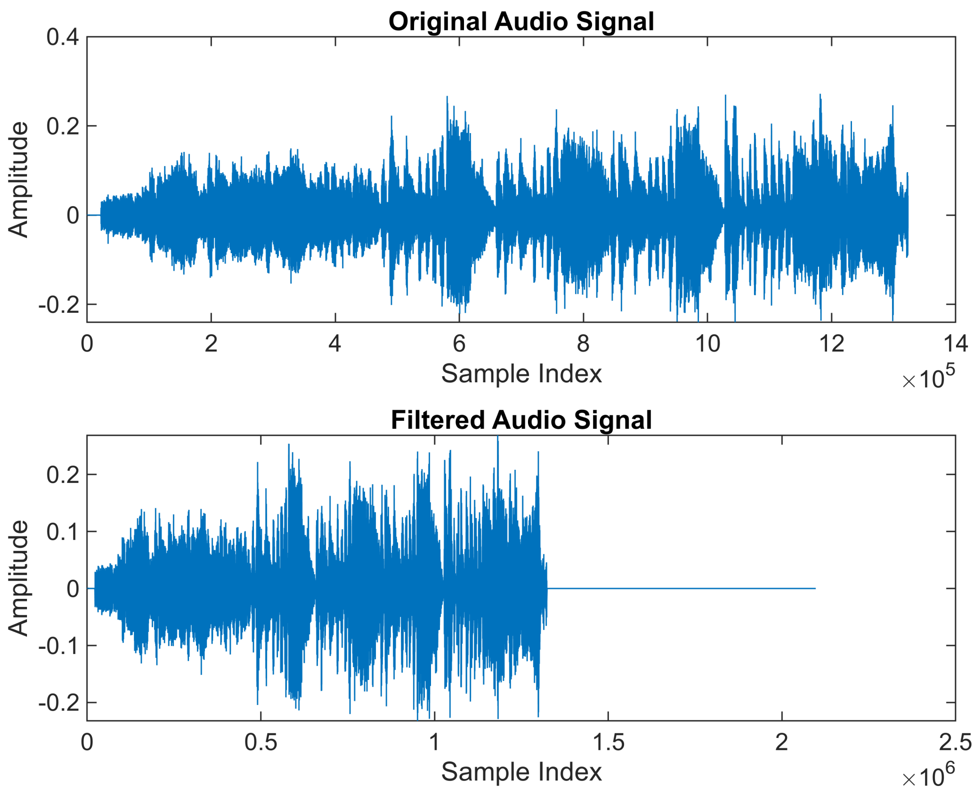



```
% Explanation of results:

% The time waveforms and magnitude spectra of y[n], y1[n], y2[n], y3[n], and y4[n]
% should be compared to determine which outputs are equal and which represent the
% true filtered outputs.

% The 'filter' function performs time-domain convolution which should give the
% true filtered output y[n].

% The 'dftfilt' and 'convfilt' functions perform frequency-domain multiplication
% which is equivalent to circular convolution in time domain.

% For N = 256, y1[n] and y2[n] should be identical to y[n] if the filter length
% is less than or equal to 256-N+1.

% For N = 512, y3[n] and y4[n] should be identical to y[n] extended with zeros,
% assuming the filter length is less than or equal to 512-N+1.

% Any discrepancies can be due to edge effects or the circular nature of the
% DFT-based convolution.
```

PART F

```
[y, fs] = audioread('C:\soundExp6.wav');
N = 2^nextpow2(length(y));
```

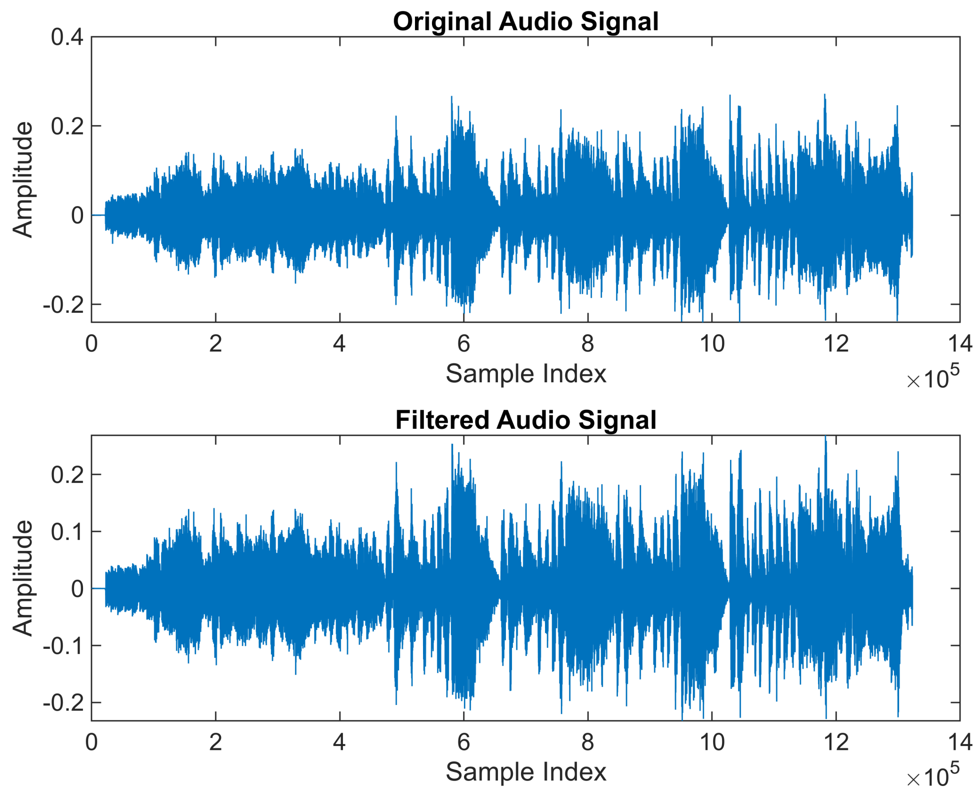
```
filtered_y = dftfilt1(y, h1, N);
player1 = audioplayer(filtered_y, fs);
%play(player1);

figure;
subplot(2,1,1);
plot(y);
title('Original Audio Signal');
xlabel('Sample Index');
ylabel('Amplitude');

subplot(2,1,2);
plot(filtered_y);
title('Filtered Audio Signal');
xlabel('Sample Index');
ylabel('Amplitude');
```



PART G

```
[y, fs] = audioread('C:\soundExp6.wav');
L = 1024;
filtered_y = convsave(y, h1, L);
player2 = audioplayer(filtered_y, fs);
%play(player2);

figure;
```

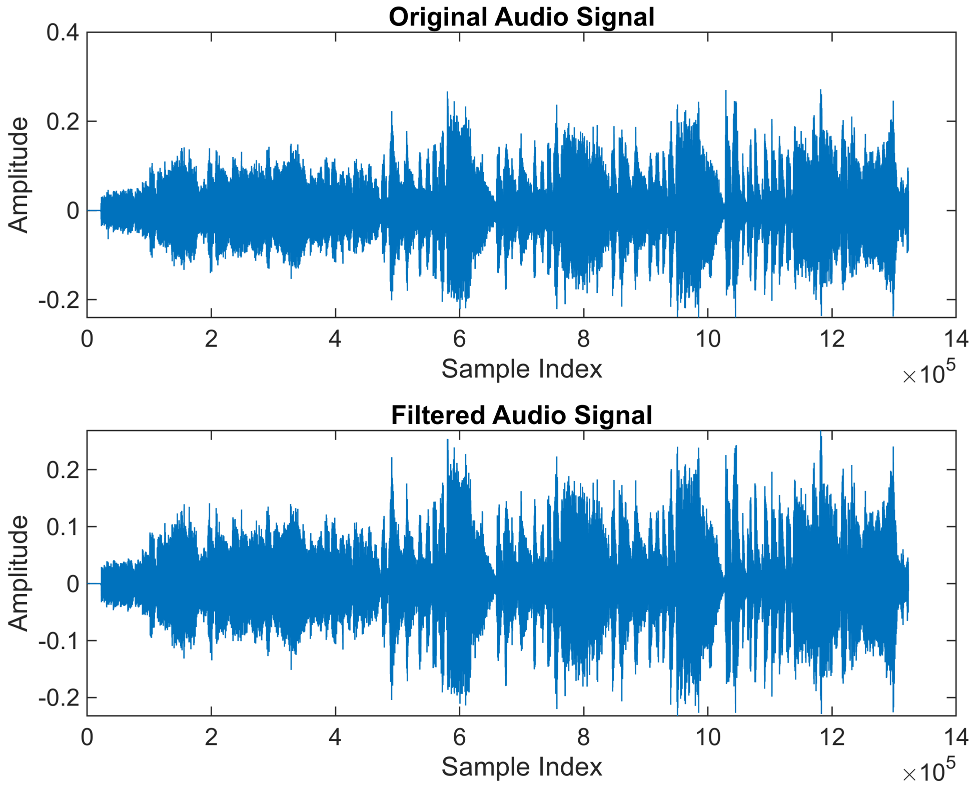
```
subplot(2,1,1);
plot(y);
title('Original Audio Signal');
xlabel('Sample Index');
ylabel('Amplitude');

subplot(2,1,2);
plot(filtered_y);
title('Filtered Audio Signal');
xlabel('Sample Index');
ylabel('Amplitude');
```



PART H

```
[y, fs] = audioread('C:\soundExp6.wav');
L = 1024;
filtered_y = dftsave(y, h1, L);
player3 = audioplayer(filtered_y, fs);
%play(player3);

figure;
subplot(2,1,1);
plot(y);
title('Original Audio Signal');
xlabel('Sample Index');
ylabel('Amplitude');
```

```
subplot(2,1,2);
plot(filtered_y);
title('Filtered Audio Signal');
xlabel('Sample Index');
ylabel('Amplitude');
```



PART I

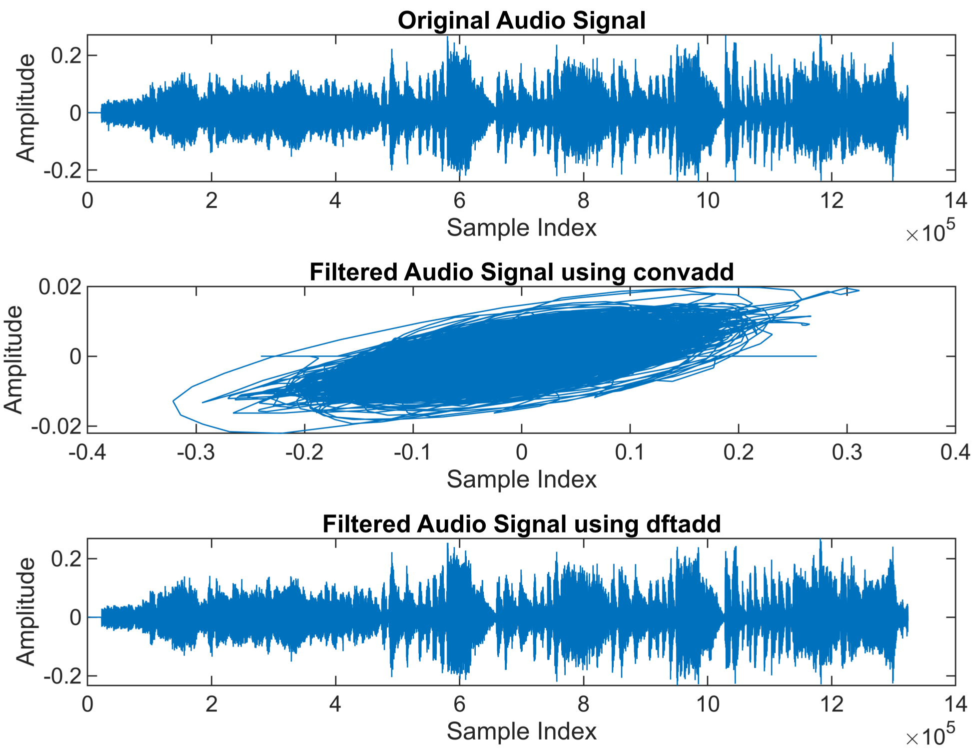
```
% Theoretical complexity analysis for dftfilt1, convsave, and dftsave functions

% For dftfilt1 (Part F):
% The complexity is dominated by the FFT operation, which has a complexity of
% O(N*log(N)) where N is the length of the DFT. There are two FFTs and one IFFT
% per block, plus N complex multiplications.

% For convsave (Part G):
% The complexity is similar to dftfilt1, but the operation is performed in
% blocks. Each block involves an FFT, a point-wise multiplication, and an IFFT.
% The block size L affects the number of operations: smaller blocks mean more
% FFTs but fewer multiplications per block.

% For dftsave (Part H):
% The complexity is also similar to dftfilt1 and convsave, but it uses the
% overlap-save method. This method involves additional operations to handle
% the overlap, but it can be more efficient when filtering long signals because
```

```
% it reduces the number of redundant calculations.

% To compare the operational load, we would need to count the actual number of
% multiplications and additions for each method. This can be done by implementing
% a counter within each function that increments every time a multiplication or
% addition is performed.

% Note: This script provides a theoretical overview and does not perform actual
% calculations. To perform a practical comparison, you would need to modify the
% functions to count and return the number of operations, or use MATLAB's
% built-in profiling tools to measure the execution time.

% Example of counting operations (not implemented in this script):
% multiplication_counter = 0;
% addition_counter = 0;
% for each operation:
%     multiplication_counter = multiplication_counter + number_of_multiplications;
%     addition_counter = addition_counter + number_of_additions;

% After running each function with the counters, compare the values of
% multiplication_counter and addition_counter to determine the operational load
% of each method.
```

PART J

```
[y, fs] = audioread('C:\soundExp6.wav');
L = 1024;
filtered_y_convadd = convadd(y, h1, L);
filtered_y_dftadd = dftadd(y, h1, L);
player_convadd = audioplayer(filtered_y_convadd, fs);
player_dftadd = audioplayer(filtered_y_dftadd, fs);
%play(player_convadd);
%play(player_dftadd);

figure;
subplot(3,1,1);
plot(y);
title('Original Audio Signal');
xlabel('Sample Index');
ylabel('Amplitude');

subplot(3,1,2);
plot(filtered_y_convadd);
title('Filtered Audio Signal using convadd');
xlabel('Sample Index');
ylabel('Amplitude');

subplot(3,1,3);
plot(filtered_y_dftadd);
title('Filtered Audio Signal using dftadd');
```

```
xlabel('Sample Index');
ylabel('Amplitude');
```




```
% Theoretical complexity analysis for convadd and dftadd functions

% For convadd:
% The complexity is dominated by the FFT operation, which has a complexity of
% O(L*log(L)) where L is the length of the block. There are two FFTs and one
% IFFT per block, plus L complex multiplications.

% For dftadd:
% The complexity is similar to convadd, but the operation is performed in blocks.
% Each block involves an FFT, a point-wise multiplication, and an IFFT. The block
% size L affects the number of operations: smaller blocks mean more FFTs but fewer
% multiplications per block.

% To compare the operational load, we would need to count the actual number of
% multiplications and additions for each method. This can be done by implementing
% a counter within each function that increments every time a multiplication or
% addition is performed.

% Note: This script provides a theoretical overview and does not perform actual
% calculations. To perform a practical comparison, you would need to modify the
% functions to count and return the number of operations, or use MATLAB's built-in
% profiling tools to measure the execution time.
```

```
% Example of counting operations (not implemented in this script):
% multiplication_counter = 0;
% addition_counter = 0;
% for each operation:
%     multiplication_counter = multiplication_counter + number_of_multiplications;
%     addition_counter = addition_counter + number_of_additions;

% After running each function with the counters, compare the values of
% multiplication_counter and addition_counter to determine the operational load
% of each method.
```

FUNCTIONS

```
function y = dftfilt(x, h, N)
    x_padded = [x zeros(1, N-length(x))];
    h_padded = [h zeros(1, N-length(h))];
    X = fft(x_padded, N);
    H = fft(h_padded, N);
    Y = X .* H;
    y = ifft(Y, N);

    if isreal(x)
        y = real(y);
    end
end


function y = dftfilt1(x, h, N)
    x = x(:).';
    h = h(:).';
    x_padded = [x, zeros(1, N-length(x))];
    h_padded = [h, zeros(1, N-length(h))];
    X = fft(x_padded, N);
    H = fft(h_padded, N);
    Y = X .* H;
    y = ifft(Y, N);

    if isreal(x)
        y = real(y);
    end
end


function y = convfilt(x, h, N)
    x_padded = [x zeros(1, N-length(x))];
    h_padded = [h zeros(1, N-length(h))];
    X = fft(x_padded, N);
    H = fft(h_padded, N);
    Y = ifft(X .* H);
```

```
    if isreal(x)
        y = real(Y);
    else
        y = Y;
    end
end


function plotMagnitudeSpectrum(x, N)
    X = fft(x, N);
    X_mag = abs(X);
    f = (0:N-1) * (1/N);
    plot(f, X_mag);
    xlabel('Normalized Frequency');
    ylabel('Magnitude');
    grid on;
end


function y = convsave(x, h, L)
    M = length(h);

    if L <= M
        error('Block size L must be greater than the filter length M.');
    end

    h_padded = [h zeros(1, L-M)];
    H = fft(h_padded, L);
    y = zeros(size(x));
    num_blocks = ceil((length(x) + M-1) / (L-M));
    output_start_index = 1;

    for k = 0:num_blocks-1
        start_index = k * (L-M) + 1;
        end_index = start_index + L - 1;
        x_block = x(start_index:min(end_index, length(x)));
        x_block_padded = [zeros(1, M-1), x_block(:).'];
        X_block = fft(x_block_padded, L);
        Y_block = ifft(X_block .* H, L);
        valid_length = min(L-M+1, length(x) - output_start_index + 1);
        y(output_start_index:output_start_index+valid_length-1) =
Y_block(M:M+valid_length-1);
        output_start_index = output_start_index + valid_length;
    end
end


function y = dftsave(x, h, L)
    h = h(:).';
```

```
    M = length(h);

    if L <= M
        error('Block size L must be greater than the filter length M.');
    end

    N = 2^nextpow2(L + M - 1);
    h_padded = [h, zeros(1, N - M)];
    H = fft(h_padded);
    y = zeros(1, length(x) + M - 1);
    num_blocks = ceil(length(x) / L);

    for k = 0:num_blocks-1
        start_index = k * L + 1;
        end_index = start_index + L - 1;
        x_block = x(start_index:min(end_index, length(x)));
        x_block_padded = [zeros(1, M - 1), x_block(:).'];
        x_block_padded = [x_block_padded, zeros(1, N - length(x_block_padded))];
        X_block = fft(x_block_padded);
        Y_block = ifft(X_block .* H);
        y(start_index:start_index + L - 1) = Y_block(M:M + L - 1);
    end
    y = y(1:length(x));
end


function y = convadd(x, h, L)
    M = length(h);

    if L <= M
        error('Block size L must be greater than the filter length M.');
    end

    h_padded = [h zeros(1, L)];
    H = fft(h_padded, L);
    y = zeros(1, length(x) + M - 1);
    num_blocks = ceil(length(x) / L);
    overlap = zeros(1, M - 1);

    for k = 0:num_blocks-1
        start_index = k * L + 1;
        end_index = start_index + L - 1;
        x_block = x(start_index:min(end_index, length(x)));
        x_block_padded = [x_block zeros(1, L - length(x_block))];
        X_block = fft(x_block_padded, L);
        Y_block = ifft(X_block .* H, L);
        y_block = [overlap + Y_block(1:M-1), Y_block(M:end)];
        y(start_index:start_index + length(y_block) - 1) = y_block;
        overlap = Y_block(end - M + 2:end);
    end
```

```
    y = y(1:length(x));
end


function y = dftadd(x, h, L)
    h = h(:).';
    M = length(h);

    if L <= M
        error('Block size L must be greater than the filter length M.');
    end

    N = 2^nextpow2(L + M - 1);
    h_padded = [h, zeros(1, N - M)];
    H = fft(h_padded);
    y = zeros(1, length(x) + M - 1);
    num_blocks = ceil(length(x) / L);

    for k = 0:num_blocks-1
        start_index = k * L + 1;
        end_index = start_index + L - 1;
        x_block = x(start_index:min(end_index, length(x)));
        x_block_padded = [x_block(:).', zeros(1, N - length(x_block))];
        X_block = fft(x_block_padded);
        Y_block = ifft(X_block .* H);
        y(start_index:start_index + L - 1) = y(start_index:start_index + L - 1) +
Y_block(1:L);
        if k < num_blocks - 1
            y(start_index + L:start_index + L + M - 2) = Y_block(L + 1:L + M - 1);
        end
    end
    y = y(1:length(x));
end
```