<<< Only Problem 1 and 2 will be graded >>>

Problem 1 (sound)

Denoising time with FFT (DFT)

In [1]:
```
# !pip install praat-parselmouth
```

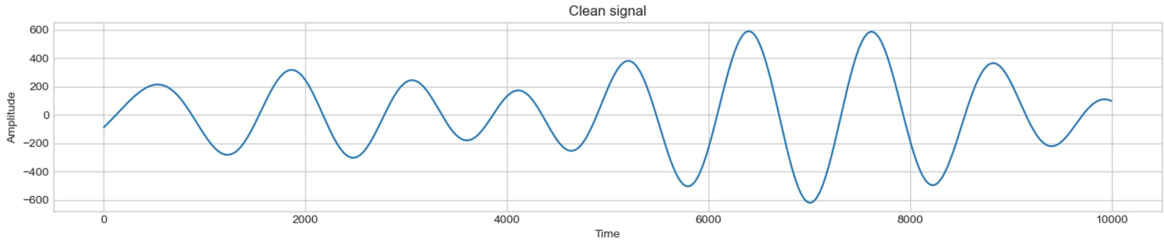
In [2]:
```
import numpy as np
import pandas as pd
from scipy import signal,fftpack
import cv2
from skimage.io import imread
import matplotlib.pyplot as plt
import IPython.display as ipd
import os
# import librosa
# import parselmouth
```

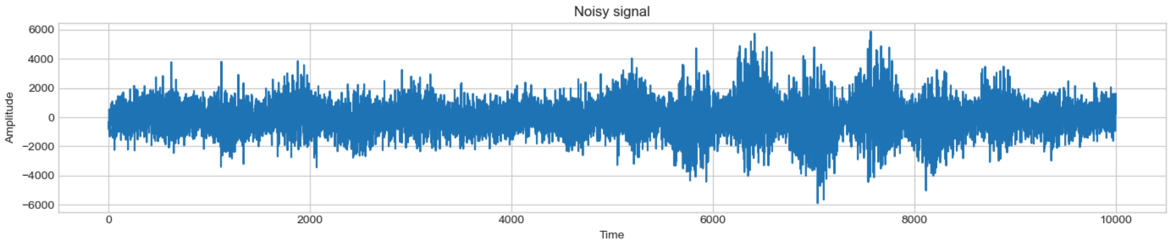
In [3]:
```
# Configuration
np.random.seed(0) # set seed
plt.style.use('seaborn-v0_8-whitegrid')
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
sampling_rate = 32000
N=10001
Nf = 3 # Nf--> num freq
t= np.arange(N,dtype=float)
# pick rand period betwwen 10-2010 and convert to freq

# random period
Ts = np.random.rand(Nf)*2000+10
fs=1/Ts

# fs in sampling rate = 32000
fs_real = fs*sampling_rate

# pick rand Amp and phase
amp = np.random.rand(Nf)*200+ 100
phi = np.random.rand(Nf)*2*np.pi

# create clean signal
h = np.zeros(N)
for i in range(len(fs)):
    h += amp[i]*np.sin(2*np.pi*fs[i]*t + phi[i])

# signal with noise
h_w_noise = h + np.random.randn(N)*3*h + np.random.randn(N)*700
```

In [5]:
```
# TODO 1.1 : plot (1) clean signal and (2) noisy signal with label
```

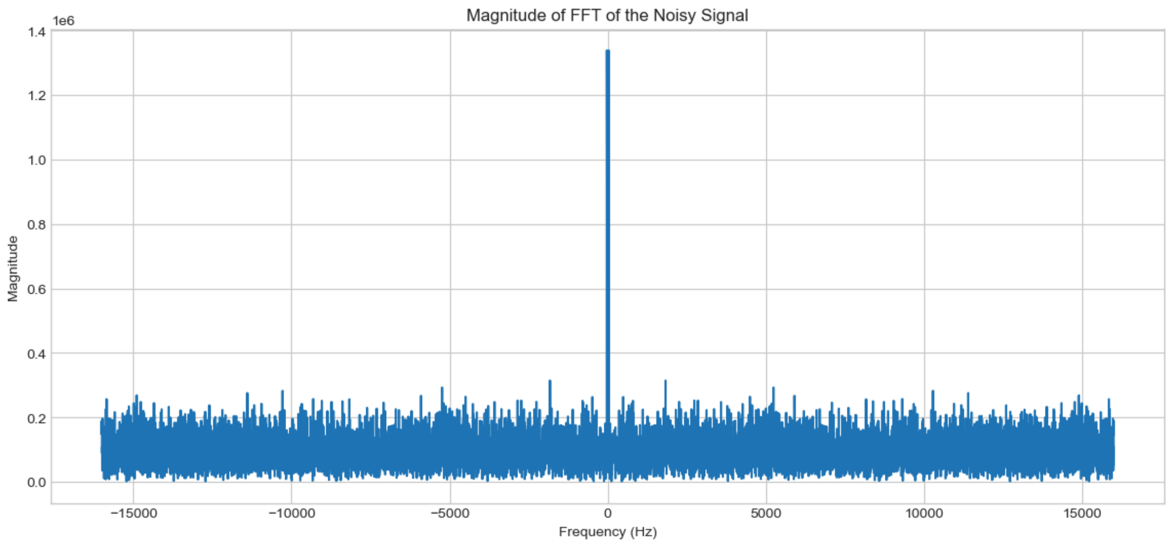
```
plt.figure(figsize=(14,6))

# Clean signal
plt.subplot(2, 1, 1)
plt.plot(t, h)
plt.title('Clean signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid(True)

# Noisy signal
plt.subplot(2, 1, 2)
plt.plot(t, h_w_noise)
plt.title('Noisy signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid(True)

plt.tight_layout()
plt.show()
```



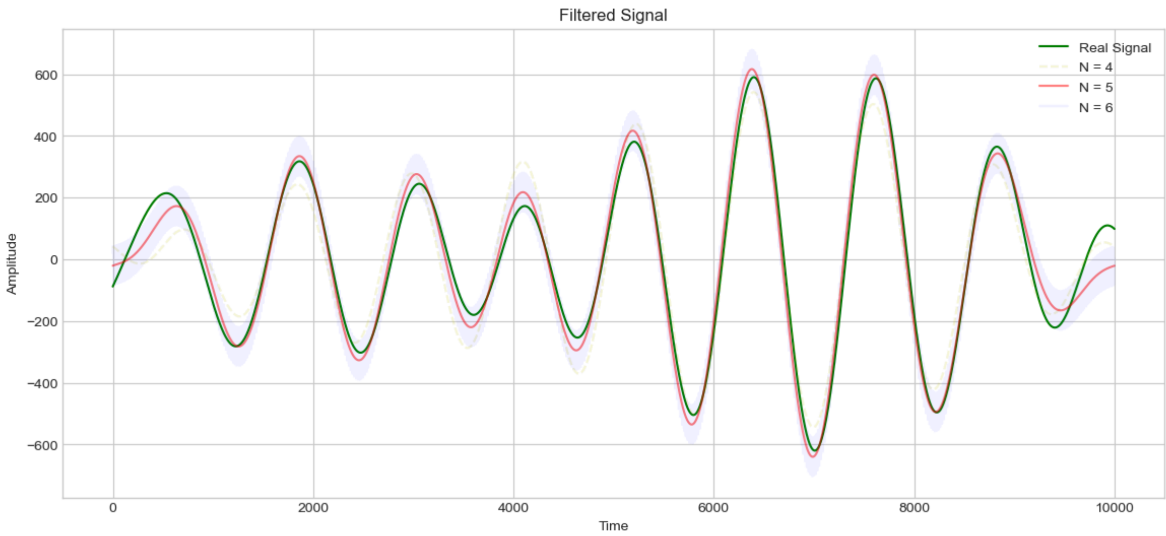


In [6]:

```
# TODO 1.2: plot magnitude of FFT of the noisy signal (freq sort form min

# Compute the FFT of the noisy signal
H_w_noise = np.fft.fft(h_w_noise)
magnitude = np.abs(H_w_noise)

frequencies = np.fft.fftfreq(N, d=1/sampling_rate)
sorted_indices = np.argsort(frequencies)

sorted_frequencies = frequencies[sorted_indices]
sorted_magnitude = magnitude[sorted_indices]

# Plot
plt.figure(figsize=(14, 6))
plt.plot(sorted_frequencies, sorted_magnitude)
plt.title('Magnitude of FFT of the Noisy Signal')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude')
plt.grid(True)
plt.show()
```

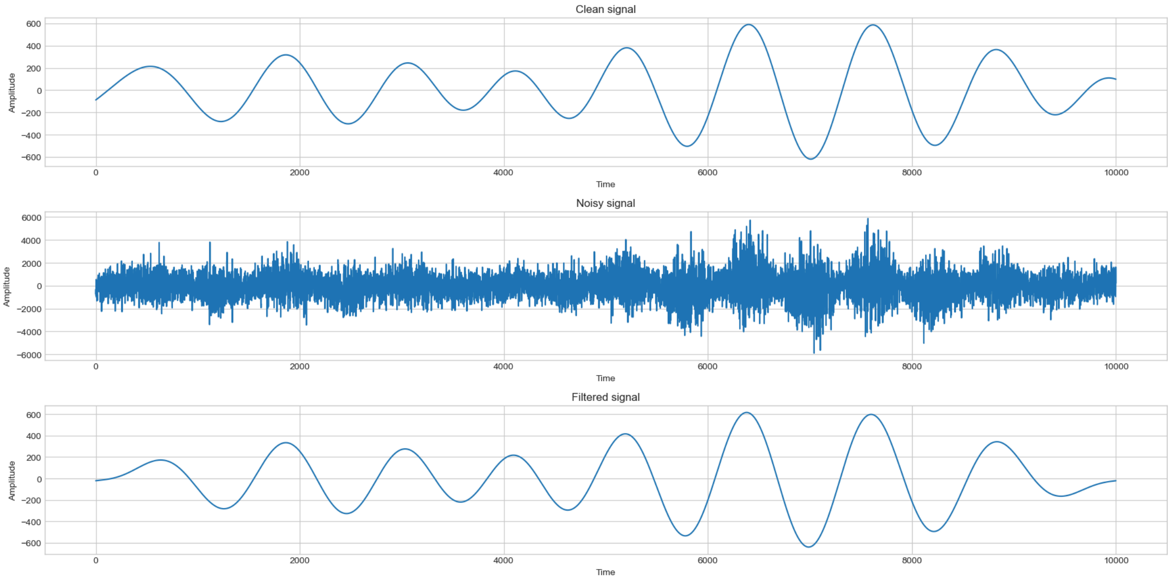


In [7]:
```
# TODO 1.3 : cleaning the noisy signal using magnitude of FFT

def filter_signal(h, top_N):
    # ``` Top N dominant frequencies filtering
    # (Top N including the DC component, but not complex conjugates symme
    # ```
    H = np.fft.fft(h)
    magnitude = np.abs(H)
    threshold = np.sort(np.unique(magnitude))[-top_N]

    H_filtered = np.where(magnitude >= threshold, H, 0)

    # Inverse FFT (IFFT)
    h_filtered = np.fft.ifft(H_filtered)

    return h_filtered

# Plot
plt.figure(figsize=(14, 6))
plt.plot(t, h, label='Real Signal', color='g')
plt.plot(t, filter_signal(h_w_noise, 4).real, label='N = 4', alpha=0.15,
plt.plot(t, filter_signal(h_w_noise, 5).real, label='N = 5', alpha=0.5, c
plt.plot(t, filter_signal(h_w_noise, 6).real, label='N = 6', alpha=0.05,
plt.title('Filtered Signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.legend()
plt.grid(True)
plt.show()
```



In [8]:
```
# Choose N = 5 for filtering
# Top N = 5 dominant frequencies filtering
# because, from graph above, N = 5 is the best choice for filtering

h_filtered = filter_signal(h_w_noise, 5)
```

In [9]:
```
# TODO 1.4 : plot clean signal, noise signal and filtered signal (from yo

plt.figure(figsize=(18, 9))

# Clean signal
plt.subplot(3, 1, 1)
plt.plot(t, h)
plt.title('Clean signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid(True)

# Noisy signal
plt.subplot(3, 1, 2)
plt.plot(t, h_w_noise)
plt.title('Noisy signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid(True)

# Filtered signal
plt.subplot(3, 1, 3)
plt.plot(t, h_filtered)
plt.title('Filtered signal')
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.grid(True)

plt.tight_layout()
plt.show()
```



In [10]:

```
# TODO 1.5 : export with IPython.display, listen to (1) original signal (

print("Clean Signal:")
ipd.display(ipd.Audio(data=h, rate=sampling_rate))

print("Noisy Signal:")
ipd.display(ipd.Audio(data=h_w_noise, rate=sampling_rate))

print("Filtered Signal:")
ipd.display(ipd.Audio(data=h_filtered, rate=sampling_rate))
```

```
Clean Signal:
```

0:00 / 0:00

```
Noisy Signal:
```

0:00 / 0:00

```
Filtered Signal:
```

0:00 / 0:00

In [11]:

```
# TODO 1.6 : Write to explain and analyze the results
```

Answer:

จากการฟังเสียง แลการพิจารณา graph แทบจะแยกความแตกต่างระหว่าง Clean และ Filtered Signal ไม่ออก หากฟังด้วยหูเพียงอย่างเดียวอาจจะแยกไม่ออก ต้องอาศัยการพิจารณากราฟร่วมด้วยจึงจะสังเหตเห็นความแตกต่าง โดยสังเกตตรงบริเวณเริ่มต้นและปลายจะพบว่า filtered signal Amplitude มีการลู่เข้าสู่ 0 ทำให้เสียงในตอนแรกเสียงจะเบากว่า และรู้สึก smooth กว่า ในช่วงกลางนั้นแยกความแตกต่างด้วยหูไม่ออก

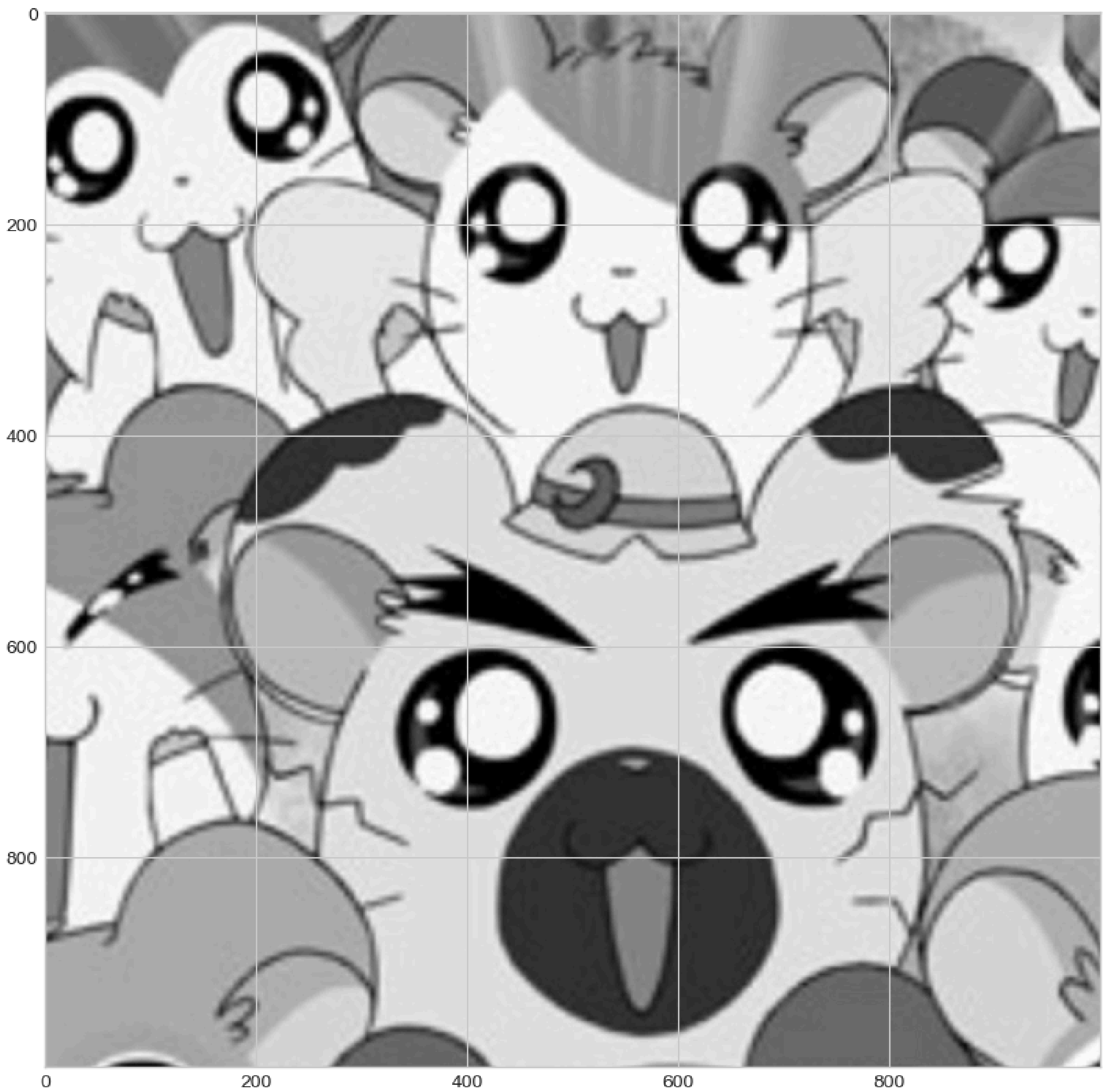
ในส่วนของ Noisy signal กับ Filtered/Clean นั้นจะแตกต่างกันอย่างเห็นได้ชัด โดย noisy signal จะเป็นเสียงซ่าๆ และที่สำคัญจะดังกว่าทั้งสองสัญญาณมาก โดยประมาณจากกราฟ(amplitude) 10 เท่า

Problem 2 (image FFT)

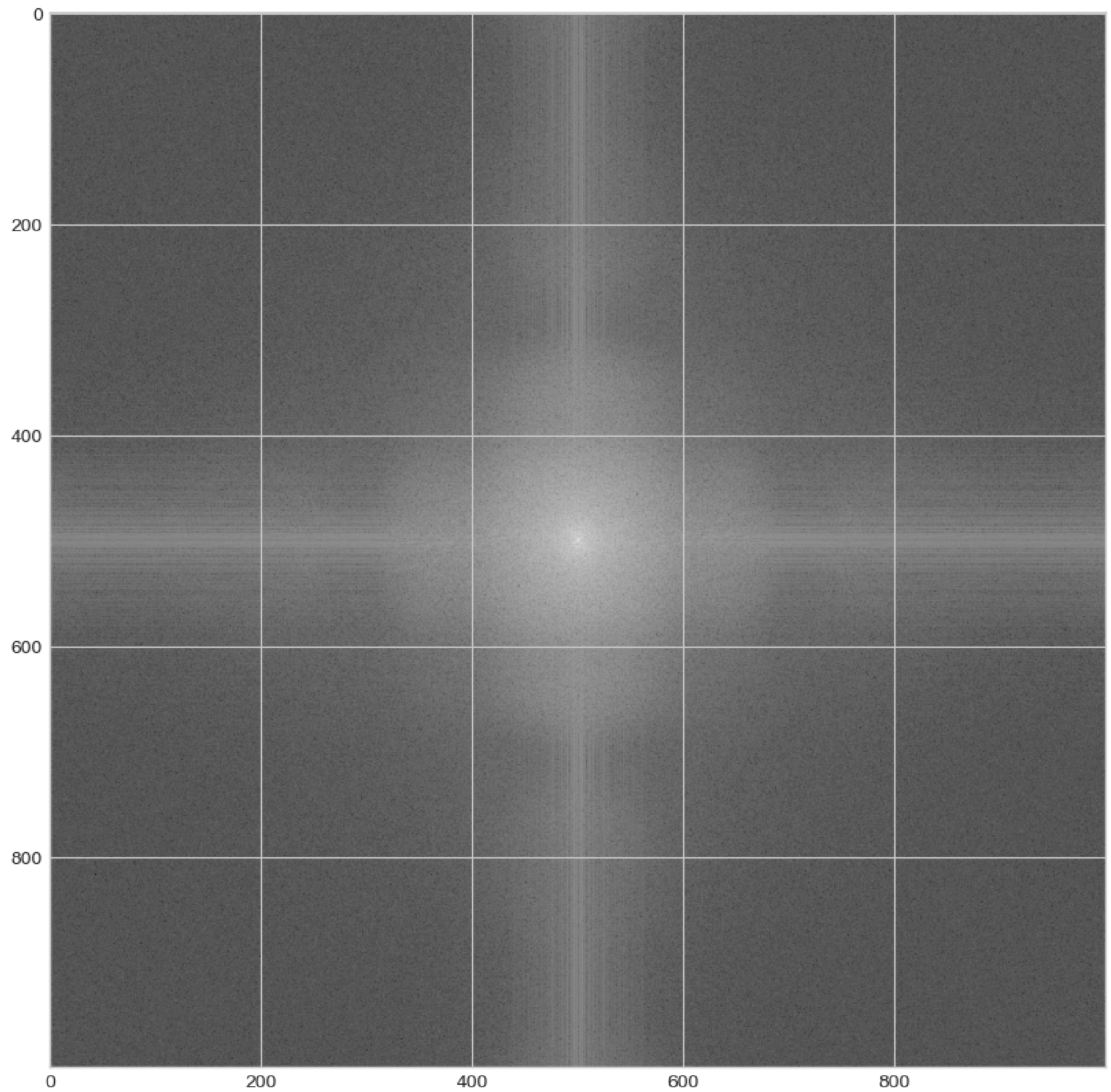
Download a 1000 x 1000 image ("hamtaro.png") below

In [12]:

```
screen_shot = cv2.imread('hamtaro.png',0)

plt.figure(figsize=(10,10))
plt.imshow(screen_shot, cmap='gray')
plt.show()
```



In [13]:
```
# Apply FFT to the given image
F1 = fftpack.fft2((screen_shot).astype(float))
F2 = fftpack.fftshift(F1) # FFT center zeros freq
plt.figure(figsize=(10,10))
plt.imshow( (20*np.log10( 0.1 + np.abs(F2))).astype(int), cmap=plt.cm.gra
plt.show()
```

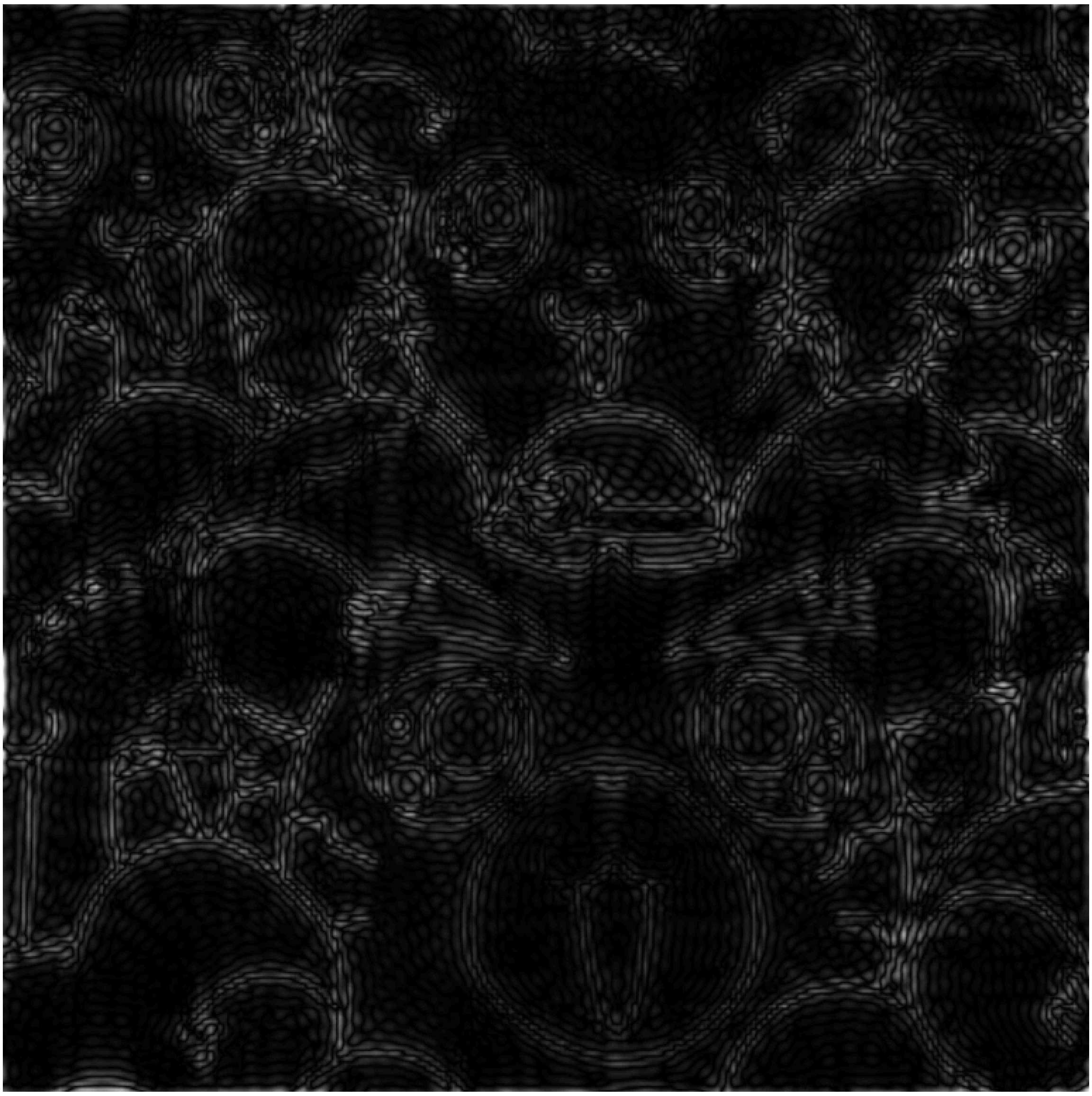


In [14]:

```
# TODO 2.1 : Implement an ideal high-pass filter with a box size of 100x1

rows, cols = screen_shot.shape
# Create a mask
high_pass_mask = np.ones((rows, cols), dtype=np.float32)

# Create a central square of zeros (low-pass removal region)
n = 100  # 100x100 box
center_row, center_col = rows // 2, cols // 2
high_pass_mask[center_row - n//2:center_row + n//2, center_col - n//2:cen

# Apply the filter
F2_filtered = F2 * high_pass_mask

# Perform IFFT Shift and IFFT
F1_filtered = fftpack.ifftshift(F2_filtered)
image_filtered = fftpack.ifft2(F1_filtered)
image_filtered = np.abs(image_filtered)

# Plot
plt.figure(figsize=(10, 10))
plt.imshow(image_filtered, cmap='gray')
plt.title('High-Pass Filtered Image (100x100 Box)')
```

```
plt.axis('off')
plt.show()
```

High-Pass Filtered Image (100x100 Box)



In [15]:

```
# TODO 2.2 : Implement an ideal low-pass filter with a box size of 100x10

rows, cols = screen_shot.shape
# Create a mask
low_pass_mask = np.zeros((rows, cols), dtype=np.float32)

# Create a central square of zeros (low-pass removal region)
n = 100  # 100x100 box
center_row, center_col = rows // 2, cols // 2
low_pass_mask[center_row - n//2:center_row + n//2, center_col - n//2:cent

# Apply the filter
F2_filtered = F2 * low_pass_mask

# Perform IFFT Shift and IFFT
F1_filtered = fftpack.ifftshift(F2_filtered)
image_filtered = fftpack.ifft2(F1_filtered)
image_filtered = np.abs(image_filtered)

# Plot
plt.figure(figsize=(10, 10))
```

```python
plt.imshow(image_filtered, cmap='gray')
plt.title('High-Pass Filtered Image (100x100 Box)')
plt.axis('off')
plt.show()
```

High-Pass Filtered Image (100x100 Box)

Problem 3

A digital signal can be generated from sampling of an analog signal using a periodic impulse-train. Explain how you can reconstruct an analog signal from a digital signal and aliasing problem does not occur when $f_s \leq 2f_{max}$ using frequency analysis.

where f_s is the sampling frequency and f_{max} is the maximum frequency of the analog signal

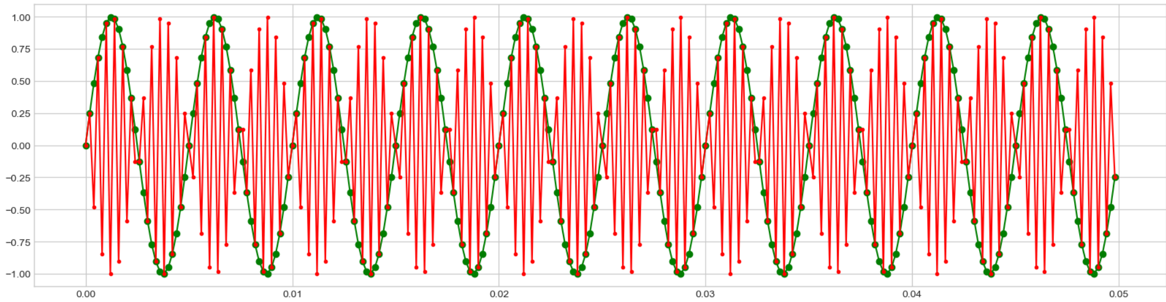
HINT : $\mathscr{F}\left\{\sum_{n=-\infty}^{\infty}\delta(t-nT_s)\right\} = \sum_{n=-\infty}^{\infty}\delta(\omega-n\omega_s)$ if $\omega_s = \frac{2\pi}{T_s} = 2\pi f_s$

Problem 4 : Aliasing

Problem 4.1

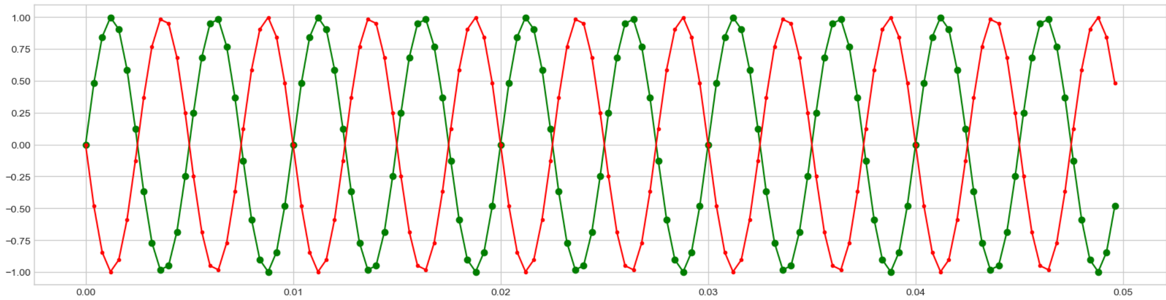
The following code generates two sine waves (x01_ts01 and x02_ts01) which are sampled in a range of t = 0,0.05 with sampling rate = 5000 Hz (f_samp_01). Study and write a report to analyze the results.

In [16]:
```
t_st = 0
t_end = 0.05
f_01 = 200
f_02 = 2300

f_samp_01 = 5000

ts01 = np.linspace(t_st, t_end , int((t_end-t_st)*f_samp_01), endpoint=Fa
x01_ts01 = np.sin(2*np.pi*f_01*ts01)
x02_ts01 = np.sin(2*np.pi*f_02*ts01)

plt.figure(figsize=(20, 5))
plt.plot(ts01, x01_ts01, 'go-', ts01, x02_ts01, 'r.-')
plt.show()
```



The sampling rate is reduced to 2500 Hz (f_samp_02). Study and write a report to compare the results.

In [17]:
```
f_samp_02 = 2500
ts02 = np.linspace(t_st, t_end , int((t_end-t_st)*f_samp_02), endpoint=Fa
x01_ts02 = np.sin(2*np.pi*f_01*ts02)
x02_ts02 = np.sin(2*np.pi*f_02*ts02)

plt.figure(figsize=(20, 5))
plt.plot(ts02, x01_ts02, 'go-', ts02, x02_ts02, 'r.-')
plt.show()
```



Ans.

Problem 4.2

The following code generate audio signals at different frequencies. Play the sound and write a report the analyse the results.

In [18]:
```
t_st = 0
t_end = 5
f_01 = 50
f_02 = 22050 - f_01
f_03 = 22050 + f_01
f_samp_02 = 22050

ts02 = np.linspace(t_st, t_end , int((t_end-t_st)*f_samp_02), endpoint=Fa

# CREATE SIGNAL WITH DIFFERENT FREQ

x01_ts02 = np.sin(2*np.pi*f_01*ts02)
x02_ts02 = np.sin(2*np.pi*f_02*ts02)
x03_ts02 = np.sin(2*np.pi*f_03*ts02)
```

In [19]:
```
x02_ts02
```

Out[19]:
```
array([ 0.        , -0.0142471 , -0.02849132, ...,  0.04272974,
        0.02849132,  0.0142471 ])
```

In [20]:
```
ipd.Audio(x01_ts02, rate=f_samp_02)
```

Out[20]: 0:00 / 0:05

In [21]:
```
ipd.Audio(x02_ts02, rate=f_samp_02)
```

Out[21]: 0:00 / 0:05

In [22]:
```
ipd.Audio(x03_ts02, rate=f_samp_02)
```

Out[22]: 0:00 / 0:05

from Imgflip Meme Generator

TODO : write report

Ans:

Problem 4.3

why many of audio file use sampling rate 44.1 kHz

Ans:

Problem 5

Download the 3 audio files and analyze all 3 signals with preliminary analysis. (HINT : Use a log scale for both frequency and magnitude.)"

1. bass-guitar-single-note --> mixkit-bass-guitar-single-note-2331.wav

explain pattern of signal

In [23]:
```
# !wget https://raw.githubusercontent.com/Pataweepr/ComEngMath2_2023_reso
# !wget https://raw.githubusercontent.com/Pataweepr/ComEngMath2_2023_reso
# !wget https://raw.githubusercontent.com/Pataweepr/ComEngMath2_2023_reso
```