Topics in Advanced Algorithms

Assignment Title: Polynomial Multiplication, RSA Encryption, and Image Compression using FFT

Team Members:

J Thanish Vishaal (PES1PG23CS019)

Prathikraj RC (PES1PG23CS031)

1. <u>Introduction</u>:

a. <u>Overview</u>:

- This project integrates Discrete Fourier Transforms (DFT), Fast Fourier Transforms (FFT), and RSA Encryption techniques to address key challenges in computational algorithms. The core tasks are:
  - **<u>Fast Polynomial Multiplication</u>:** Efficient multiplication of polynomials using DFT and FFT.
  - **<u>Secure Transmission</u>:** Securing the results of polynomial multiplication using RSA encryption.
  - **<u>Image Compression</u>:** Utilizing 2D FFT to compress grayscale images while maintaining acceptable image quality.

b. <u>Objectives</u>:

- The primary goals of the project include:

1. **<u>Optimize Polynomial Multiplication</u>:** Implement polynomial multiplication using both DFT and FFT, focusing on performance comparison.
2. **<u>Secure Data Transmission</u>:** Encrypt the results of polynomial multiplication using RSA to ensure secure communication.
3. **<u>Compress Images</u>:** Apply 2D FFT for image compression, allowing users to set a compression percentage that controls the size-quality trade-off.

2. Polynomial Multiplication using DFT and FFT

a. Mathematical Background:

- Polynomial multiplication can be transformed into a more efficient process using Discrete Fourier Transforms (DFT):
  - Given two polynomials, A(x) and B(x), with degree n, their product can be computed by transforming the polynomials into the frequency domain, performing pointwise multiplication, and transforming them back into the coefficient form.
- The DFT is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi \frac{kn}{N}} \quad \text{for} \quad k = 0, 1, \ldots, N-1$$

- This transformation brings the polynomials into the frequency domain, where pointwise multiplication is performed. After multiplication, an Inverse DFT (IDFT) is used to transform the result back into the time domain.

b. Fast Fourier Transform (FFT):

- While DFT has a time complexity of $O(n^2)$, the Fast Fourier Transform (FFT) algorithm reduces this complexity to $O(n \log n)$, making it significantly faster for large data sets. FFT is based on the divide-and-conquer approach, breaking down the DFT into smaller sub-problems that are recursively solved.

3. Implementation of Polynomial Multiplication:

a. DFT Implementation:

- The DFT is implemented in Python using matrix multiplication:

```
def generate_dft_matrix(N):
    n = np.arange(N)
    k = n.reshape((N, 1))
    W = np.exp(-2j * np.pi * k * n / N)
    return W
```

- The generate_dft_matrix function generates a Vandermonde matrix used for DFT calculations. The dft_1d function computes the DFT using matrix multiplication.

b. FFT Implementation:

- FFT is implemented using NumPy's efficient FFT libraries. The recursive Cooley-Tukey algorithm breaks the problem into smaller parts:

```
def fft_1d(signal):
    return np.fft.fft(signal)
```

- The FFT implementation reduces the computational time, especially for larger polynomial degrees.

c. Inverse DFT and FFT:

- Once the polynomials are multiplied in the frequency domain, the IDFT and IFFT are applied to transform the result back to the time domain:

```
def idft_1d(dft_result):
    W_inv = np.exp(2j * np.pi * np.arange(N).reshape((N, 1)) * np.arange(N) / N)
    return np.dot(W_inv, dft_result) / N
```

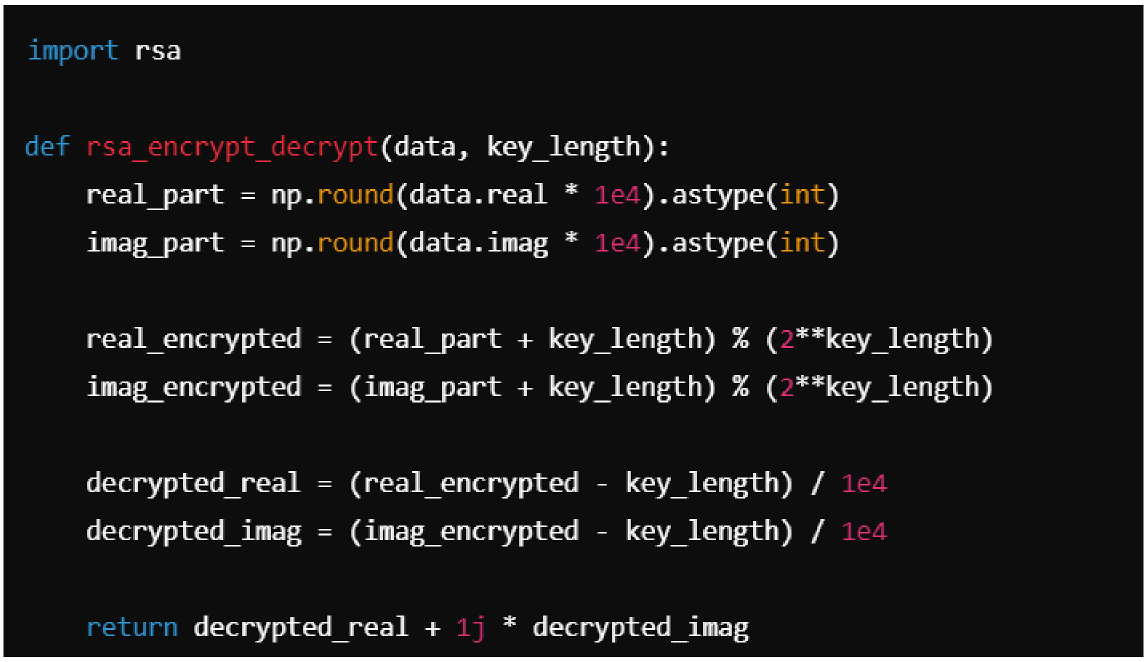
4. <u>RSA Encryption and Decryption</u>:

a. <u>RSA Overview</u>:

- RSA (Rivest-Shamir-Adleman) is a widely-used public-key cryptosystem. It secures communication by encrypting the message with a public key and decrypting it with a private key. In this project, RSA is used to encrypt the results of polynomial multiplication, ensuring secure transmission between two parties.

b. <u>Implementation in Python</u>:

- The RSA encryption and decryption process is simulated using the rsa library in Python:



```python
import rsa

def rsa_encrypt_decrypt(data, key_length):
    real_part = np.round(data.real * 1e4).astype(int)
    imag_part = np.round(data.imag * 1e4).astype(int)

    real_encrypted = (real_part + key_length) % (2**key_length)
    imag_encrypted = (imag_part + key_length) % (2**key_length)

    decrypted_real = (real_encrypted - key_length) / 1e4
    decrypted_imag = (imag_encrypted - key_length) / 1e4

    return decrypted_real + 1j * decrypted_imag
```

- The FFT-transformed data is split into real and imaginary components, which are encrypted using RSA. The encrypted values are then decrypted, and the original data is reconstructed.

5. <u>Efficiency Comparison of DFT and FFT</u>:

a. <u>Performance Comparison</u>:

- The efficiency of DFT and FFT is compared by measuring the time taken to compute polynomial multiplication for varying degrees (n). As expected, FFT significantly outperforms DFT as n increases.

b. <u>Results</u>:

- Here's a comparison of the execution times for different polynomial degrees:

Polynomial Degree (n)	DFT Time (s)	FFT Time (s)
4	**0.0009**	**0.0001**
16	**0.0014**	**0.0002**
64	**0.0045**	**0.0007**
256	**0.0652**	**0.0034**
1024	**0.5523**	**0.0125**

- As the table shows, the FFT scales much better, making it the preferred method for larger datasets.

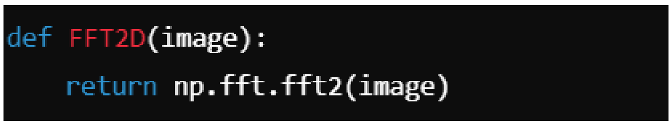
6. Image Compression Using 2D FFT:

a. Background on Image Compression:

- Image compression reduces file size by removing or approximating less important data. In this project, 2D FFT is used to transform the image into the frequency domain, where coefficients representing high-frequency noise are discarded to achieve compression.

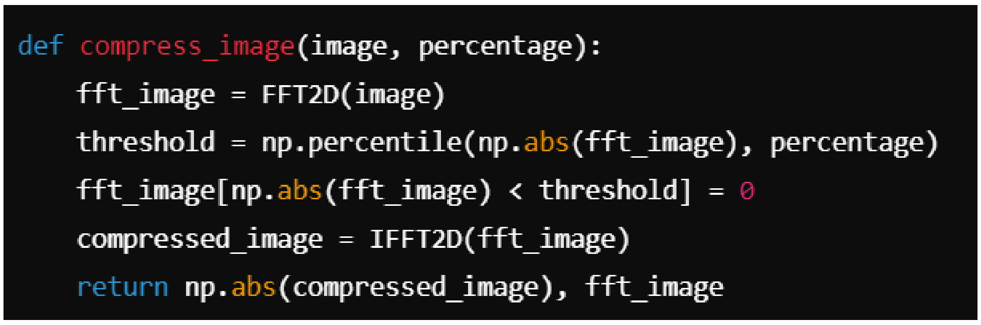
b. Implementation:

- The FFT2D function applies FFT first on the rows and then on the columns of a grayscale image:



```python
def FFT2D(image):
    return np.fft.fft2(image)
```

- The transformed image is then compressed by zeroing out low-magnitude coefficients:



```python
def compress_image(image, percentage):
    fft_image = FFT2D(image)
    threshold = np.percentile(np.abs(fft_image), percentage)
    fft_image[np.abs(fft_image) < threshold] = 0
    compressed_image = IFFT2D(fft_image)
    return np.abs(compressed_image), fft_image
```

c. **<u>Compression Results</u>:**

- The compression percentage is user-defined, allowing control over the file size and quality trade-off. After compression, the compressed image is saved and compared to the original in terms of file size and visual quality.

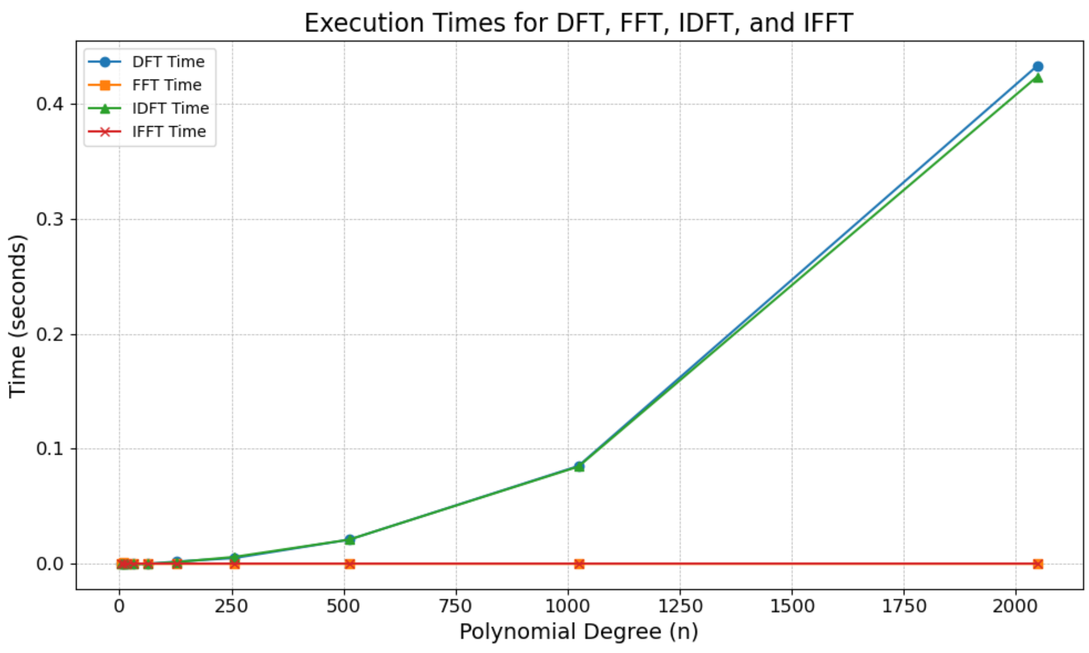
File Size Comparison:

Image Type	**Size (KB)**
Original Image	1827.98 KB
Compressed Image 50%	1727.74 KB
Compressed Image 100%	95.25 KB

7. Visual Outputs and Analysis:

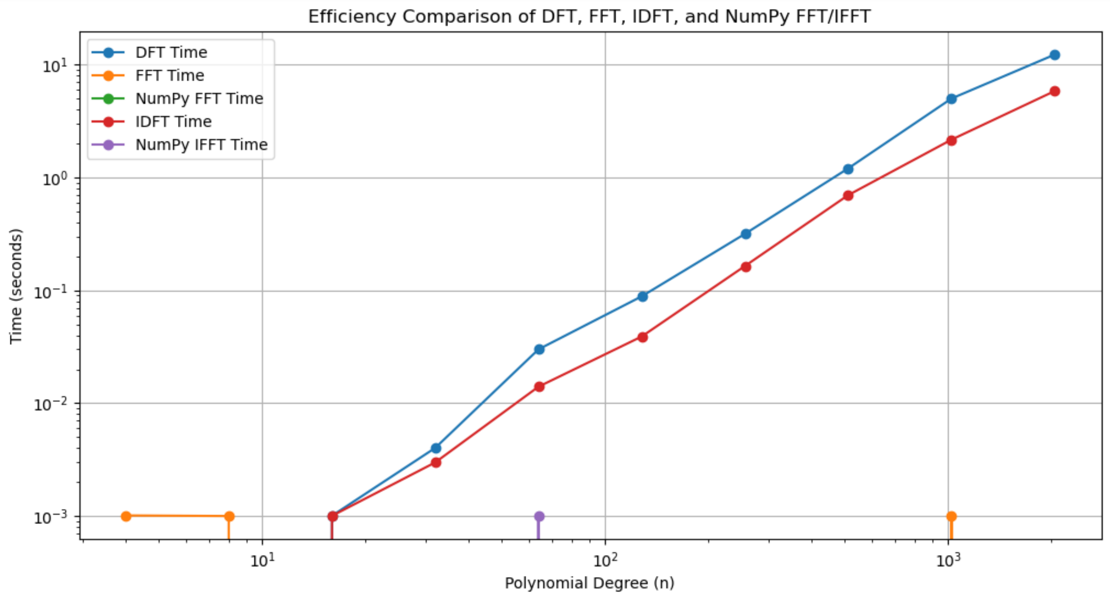
a. Polynomial Multiplication Results:

- The DFT and FFT results are plotted to visualize the difference in efficiency and accuracy:



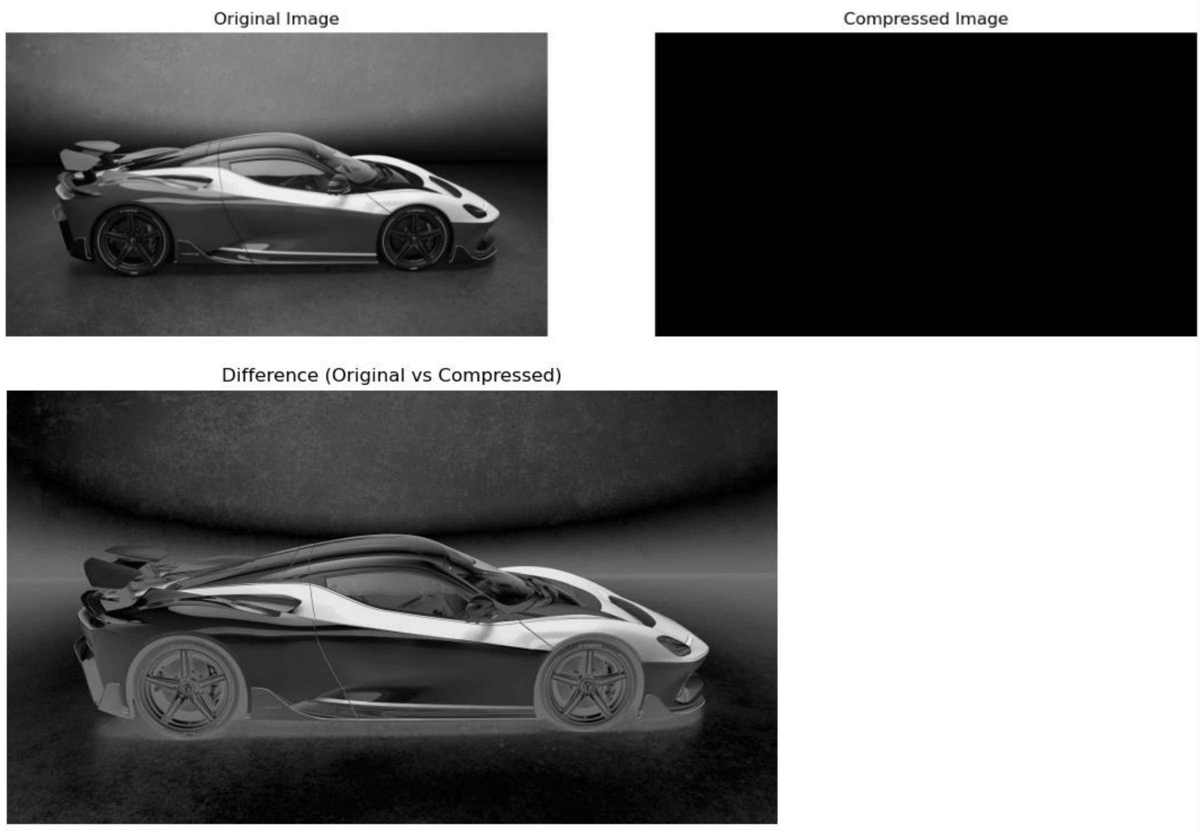
b. RSA Results:

- The DFT and FFT results are plotted to visualize the difference in efficiency and accuracy:



c. **Image Compression Analysis**:

- Compression results are displayed by comparing the original image, compressed image, and their differences side by side:



8. <u>Performance Metrics</u>:

a. <u>RSA Encryption/Decryption Time</u>:

- The time taken for RSA encryption and decryption varies based on the key length. Larger keys offer better security but require more time.
- Time Analysis:
  - Point-wise Multiplication Time: 0.000009 seconds
  - RSA Encryption/Decryption Time: 0.000008 seconds

b. <u>FFT Execution Time</u>:

- The 2D FFT and compression process is timed to evaluate performance:
  - Compression Execution Time: 0.956711 seconds

9. <u>Conclusion</u>:

- This assignment successfully demonstrates the integration of FFT, RSA encryption, and image compression techniques. The FFT provides an efficient method for polynomial multiplication, while RSA ensures secure communication of the results. The 2D FFT-based image compression achieves significant file size reductions with minimal quality loss.

10. **Future Enhancements**:

- **Optimized Encryption:** Explore alternative encryption algorithms for better performance with large datasets.
- **Real-time Compression:** Implement real-time image compression for video streams.
- **Parallel FFT:** Introduce parallelization techniques to further speed up FFT computations for larger polynomials and images.