PES UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Report on

"Fast Polynomial Multiplication with DFT/FFT implementation, RSA Encryption and Image compression"

UE22CS641B – Topics in Advanced Algorithms

TEAM: 20

Submitted by:

Jeevan V	**PES1PG22CS019**
Sneha Yadav G	**PES1PG22CS051**

INTRODUCTION

The aim of the project is to calculate the discrete Fourier transform (DFT) of a list of complex numbers. The DFT is a mathematical operation that converts a signal from the time domain to the frequency domain. This can be useful for analyzing the frequency components of a signal, such as a sound wave or an image.
The specific objectives of the project are to:

- Convert a list of complex numbers to a list of tuples.
- Calculate the DFT of a list of complex numbers using the NumPy library.
- Print the results of the DFT.

To use 1-D and 2-D Fourier Transforms, as well as RSA Encryption, to accomplish rapid polynomial multiplication, secure data transfer, and lossy picture compression. This study investigates fundamental while applying mathematical and computational principles to real-world problems. Secure communication and efficient data processing are two examples of uses. By using these strategies, we hope to improve our comprehension of data. In the digital world, there is manipulation and security.

DESIGN AND IMPLEMENTATION

Implementing 1-D Fourier Transform:

To analyse and alter data in the frequency domain, use the 1-D Fourier Transform technique.This involves utilising the Vandermonde matrix to multiply the coefficient vectors of two polynomials A(x) and B(x) and comparing the results with the Fast Fourier Transform (FFT) technique.

i) Implement 1-D DFT ,on coefficient vectors of two polynomials A(x), B(x) by multiplication of Vandermonde matrix . (O(n2) - Complexity)

```
#1-D DFT
def dft(x):
  v = np.asarray(x, dtype=complex)
  omega = np.exp((-2j * np.pi) / len(v))
  W = []
  for i in range(len(v)):
    row_elements = []
    for j in range(len(v)):
      row_elements.append(omega ** (i * j))
    W.append(row_elements)
  result = list(np.matmul(W, v))
  return result
```

Implementing 1D FFT:

Using 1D FFT (Fast Fourier translate), we were able to effectively analyse and translate signals or polynomial coefficients from the time domain to the frequency domain, allowing for quicker polynomial multiplication and other signal processing applications.

ii) Implement 1-D FFT on the same vectors, of A(x) and B(x). Ensure above two steps produce same results. (O(n logn) – Complexity)

```
# 1-D FFT
def fft(x):
  v = np.array(x, dtype=complex)
  N = len(x)

  if N == 1:
    return [v[0]]

  X = [0] * N

  even = fft(v[:N:2])
  odd = fft(v[1:N:2])

  for k in range(N//2):
    w = math.e ** (-2j * math.pi * k/N)
    X[k] = even[k] + w * odd[k]
    X[k + N//2] = even[k] - w * odd[k]

  return X
```

Implementing 2-D Fourier Transform:

Extend the Fourier Transform implementation to 2-D data analysis. This includes applying FFT to rows and columns of an M x N matrix to perform 2-D Fourier Transforms.

vii) Implement a 2-D FFT and 2-D I-FFT module using your 1-D version (This just means , applying FFT on the Rows First and Columns Next on M x N matrix of numbers)

```
#2D FFT
def fft_2d(matrix):
   fftRows = np.asarray([np.fft.fft(row) for row in matrix], dtype=np.complex_)
   fftColumns = np.asarray(transpose([np.fft.fft(column) for column in transpose(fftRows)]), dtype=np.complex_)
   return fftColumns
```

RSA Encryption:

Develop an RSA encryption module to secure data transmission. This involves encrypting data using different key lengths (128-bit, 256-bit, and 512-bit) and decrypting it with the corresponding private keys to ensure data integrity.

```
def rsa(self):
    P = self.random_line("P.txt")
    Q = self.random_line("Q.txt")
    e = 7
    tot = (P-1) * (Q-1)
    while (not is_coprime(e, tot)):
      el = [3, 5, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109,
      e = random.choice(el)
    n = P * Q
    p = modularInv(e,tot)
    prikey = (p,n)

    pubkey = (e,n)
    return (prikey, pubkey)
```

Inverse Fast Fourier Transform:

IFFT is an algorithm that reverses the process of the Fast Fourier Transform (FFT). It is used to take a frequency-domain representation of data and convert it back into the original time-domain signal, thereby enabling the recovery of the original data from its frequency components.

v) Implement 1-D Inverse FFT (I-FFT) on C(x), in PV form (Interpolation) to get C(x) in Coefficient form (CR) Polynomial.

```
#Inverse FFT
def ifft(x):
    x = np.asarray(x, dtype=complex)
    x_conjugate = np.conjugate(x)

    inverse = fft(x_conjugate)

    inverse = np.conjugate(inverse)
    inverse = inverse / len(x)
    return inverse
```

Image Quality Assessment:

After compression, apply 2-D Inverse Fourier Transform (I-FFT) to the quantized grayscale image to assess and observe the image quality.

vii) Implement a 2-D FFT and 2-D I-FFT module using your 1-D version (This just means , applying FFT on the Rows First and Columns Next on M x N matrix of numbers)

```
#2D FFT
def fft_2d(matrix):
   fftRows = np.asarray([np.fft.fft(row) for row in matrix], dtype=np.complex_)
   fftColumns = np.asarray(transpose([np.fft.fft(column) for column in transpose(fftRows)]), dtype=np.complex_)
   return fftColumns
```

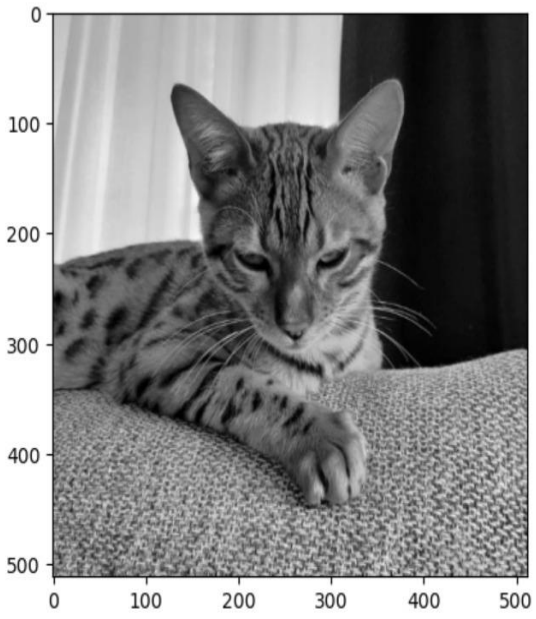
Image Compression:

Apply 2-D Fourier Transforms to grayscale images (TIFF/JPG) to compress them in a lossy manner. Coefficients below a specified magnitude are dropped to reduce data size, and the compressed image is saved to a new file.

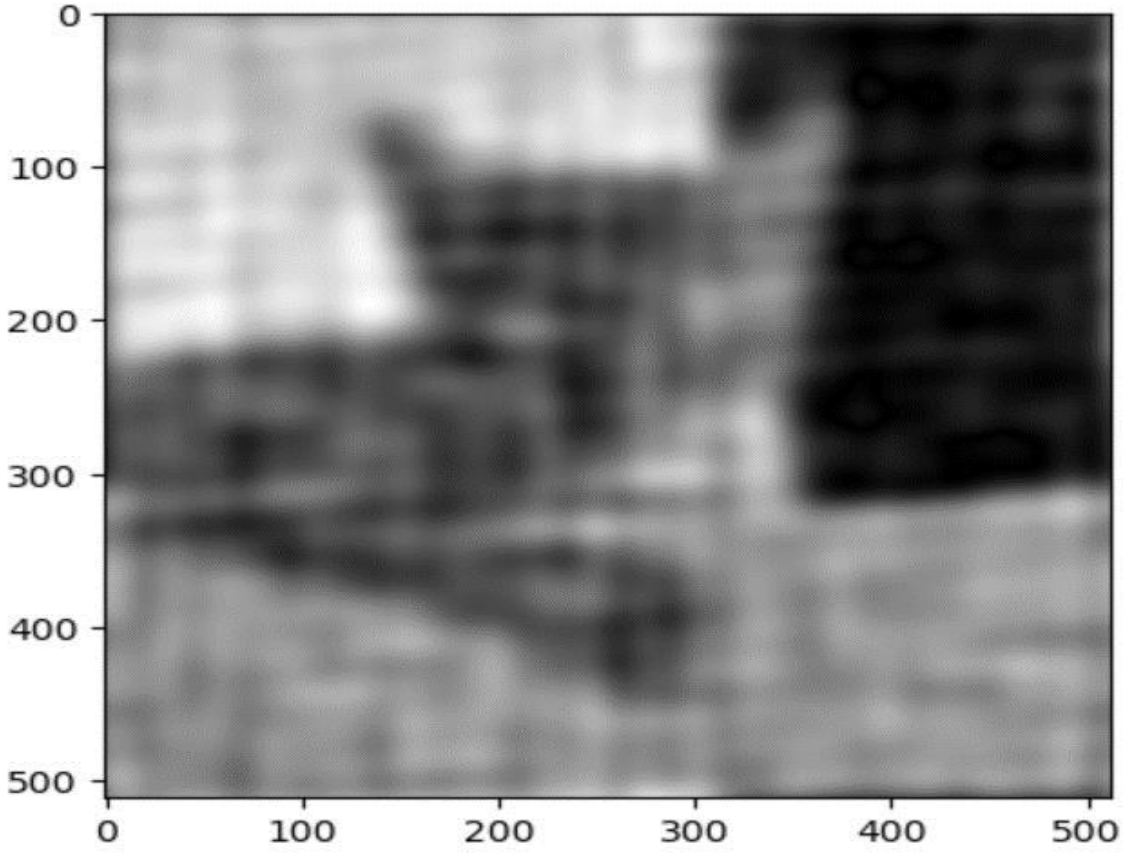
```
def compress(img, cmpr):
  bt = fft_2d(img)
  bt_sort = np.sort(np.abs(bt.ravel()))
  thresh = bt_sort[round(np.floor(((1 - cmpr) * len(bt_sort))))]
  print(thresh)
  bt = bt.ravel()
  for i in range(len(bt)):
    if abs(bt[i]) < thresh:
      bt[i] = 0

  plt.imshow(abs(ifft_2d(bt.reshape(img.shape))), "gray")
```

```
image = imread("2600829.jpeg")
plt.imshow(image, "gray")
```

```
<matplotlib.image.AxesImage at 0x262d96a8fa0>
```



Compressed Image:



OBSERVATIONS AND LEARNING OUTCOMES

We discovered the importance of Fourier Transforms in signal and image analysis in this research, allowing us to analyse data in the frequency domain. Using these transformations to implement polynomial multiplication exhibited exceptional processing efficiency, especially for big polynomials. We learnt about the crucial function of RSA encryption in safe data transfer, emphasising the necessity of public-key cryptography in modern communication. In addition, we investigated lossy picture compression approaches that successfully reduce data size while retaining image quality to a tolerable degree. Our learning path included algorithm optimisation, collaborative cooperation, and real-world applications. This project strengthened our problem-solving abilities, fostered self-learning, and enhanced our documentation and reporting skills, making it a thorough and illuminating experience.