Project Title: Data Compression Comparison

1. Project Overview

- **Programming Language**: Python
- **Proficiency in Other Languages**: R (R studio), Java, C#
- **Study Program**: Bachelor of Science in Computer Science (BSc)
- **Documentation Language**: English

2. Problem Definition

This project aims to compare different compression algorithms in order to test efficiency and effectiveness in compressing files without loss in information. The two different algorithms that will be investigated are Huffman Coding and Lempel-Ziv-Welch (LZW).

Objective: The primary aim is to achieve a compressed size of 40-60% of the original for text files. For images and audio, which will use lossy compression methods, the goal is to reduce size while retaining core visual or auditory quality, accepting some degree of data loss. This allows for an exploration of both lossless and lossy data compression approaches.

Lossy and lossless data compression will be investigated, the former being more used in media such as audio and image compression. This is to test the level of compression attainable whilst still retaining core data (audio still recognizable, image still recognizable). The main goal being able to achieve a 40-60% reduction in file size whilst still allowing data to be usable.

3. Algorithms and Data Structures

- **Algorithms**:
  - **Huffman Coding**: A variable-length encoding algorithm that uses a binary tree structure to assign shorter codes to more frequent symbols, achieving lossless compression for text files.
  - **Lempel-Ziv-Welch (LZW)**: A dictionary-based algorithm that identifies and stores patterns in the input as unique codes, which then represent sequences in the compressed file. This approach is also lossless and effective for text files with repeating patterns.
- **Data Structures**:
  - **Binary Tree**: Used in Huffman Coding to organize and store the encoding scheme, assigning the shortest codes to the most frequent characters.
  - **Dictionary**: Python's built-in dictionary data structure is utilized in LZW to store recurring patterns, allowing efficient access and updates during compression.

4. Inputs and Usage

- **Inputs**:
  - **Text Files** (.txt): Used for lossless compression with Huffman and LZW where the compressed files can be decompressed to their original form.
  - **Media Files** (.jpg, .wav): Lossy compression will be applied to these formats to achieve significant size reductions, though decompression will not reconstruct the original file.

- **Usage**:
  - The program will accept a file as input, apply the chosen compression algorithm, and output a standalone compressed file. Each compressed file will be self-contained and independent of the program once created.

5. Complexity Analysis

The expected computational complexities for the algorithms are as follows:

- **Huffman Coding:**
  - **Time Complexity**: O(n*logn) for building the Hoffman tree and encoding text, where n is the number of unique symbols.
  - **Space complexity**: Proportional to the number of unique symbols, as each requires a position in the binary tree.
  -
- **LZW**:
  - **Time Complexity:** O(n) for compressing data, assuming minimal dictionary resizing.
  - **Space Complexity:** Depends on the dictionary size and input length, as the dictionary grows with unique patterns encountered in the text.

6. References

- **Primary Resources**:
  - Wikipedia (*Huffman Coding* and *Lempel-Ziv-Welch (LZW))*.
  - Stack Overflow