

**Visvesvaraya Technological University  
Belagavi-590 018, Karnataka**



**A MINI PROJECT REPORT ON**

**“File Compression using LZW Algorithm”**

**Mini-Project Report submitted in Partial fulfillment of the requirement for the 6<sup>th</sup>  
Semester File Structures Laboratory with Mini-Project  
[17ISL68]**

**Bachelor of Engineering  
In  
Information Science and Engineering**

**Submitted by**

**DIWAKAR N[1JT17IS009]**

**Under the Guidance of  
Mr.Vadiraja A  
Asst.Professor,Department of ISE**



**Department of Information Science and Engineering  
Jyothy Institute of Technology,  
Tataguni, Bengaluru-560082**

**Jyothy Institute of Technology,**  
**Department of Information Science and Engineering**  
**Tataguni, Bengaluru-560082**



**CERTIFICATE**

Certified that the mini project work entitled “ **File Compression**” carried out by **DIWAKAR N [1JT17IS009]** bonafide student of Jyothy Institute of Technology, in partial fulfillment for the award of **Bachelor of Engineering in Information Science and Engineering** Department of the **Visvesvaraya Technological University, Belagavi** during the year **2020-2021**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of Project work prescribed for the said Degree.

**Mr. Vadiraja A**

Guide, Asst, Professor

Dept. Of ISE

**Dr. Harshvardhan Tiwari**

Associate Professor and HoD

Dept. OF ISE

External Viva Examiner

Signature with Date :

- 
-

## **ACKNOWLEDGMENT**

The satisfaction that accompanies the successful completion of my project Would be incomplete without mentioning the people who made it possible, I am thankful to **Dr. Gopalakrishna K**, Principal, JIT, Bangalore for Being kind enough to provide us an opportunity to work on a project in this Institution and I am thankful to **Dr. Harshvardhan Tiwari**, HoD, Dept of ISE for his co-operation and encouragement at all moments of our approach and Whose constant guidance and encouragement crowns all the efforts with Success. I would greatly mention enthusiastic influence provided by **Mr. Vadiraja A**, Asst. Professor, Dept. of ISE for his ideas and co-operation Showed on us during our venture and making this project a great success. Finally, it's pleasure and happiness to the friendly co-operation showed by all The staff members of Information Science Department, JIT.

**DIWAKAR N**  
**1JT17IS009**

## **ABSTRACT**

File compression is enabled through a file or data compression software that creates a compressed version of each processed file. Typically, file compression works by scanning an entire file, identifying similar or repetitive data and patterns and replacing duplicates with a unique identifier. This identifier is usually much smaller in size than the original word and consumes less space. Thus, the size of the compressed file is considerably smaller.

The File Compression is the data Compression Method in which the logical size of the file is reduced to Save disk Space For easier and Faster Transmission Over a Network. It Enables the Creation of a one or more files the Same data at a Size Substantially Smaller than Orginal file.

File Compression Method Using LZW(Lempel-Ziv-Welch) Algorithm. This Algorithm is Typically used In GIF and Optionally in POF and TIFF. LZW Compression Works by Reading a Sequence of Symbols into Strings and Converting Strings into Codes.Because the Codes take up Less Space than the Strings they Replace We Get Compression.

The File Compression Typically Works by Finding Similar Strings within a File and Replacing Those Strings with a Temporary Binary Representation to make overall Filesize Smaller.

This project has been created using Eclipse, with the platform WINDOWS. The project title-“Text File Compression” talks about how we Compress the Given File and data accepted from the Editor and also use it to retrieve Information.

## TABLE OF CONTENTS

SERIAL NO.	DESCRIPTION	PAGE NO.
	<b>Chapter 1</b>	
1	<b>Introduction</b>	1 – 5
1.1	Introduction to File Structures	1
1.2	Introduction to File System	2
1.3	Data Compression	3-5
	<b>Chapter 2</b>	
2	Requirement analysis and design	6-7
2.1	Domain Understanding	6
2.2	Classification of Requirements	6
2.3	System Analysis	6
	<b>Chapter 3</b>	
3	<b>Implementation</b>	7-11
	<b>Chapter 4</b>	
4	<b>Result and Analysis</b>	12-17

# **CHAPTER 1**

## **INTRODUCTION**

# INTRODUCTION

## 1.1 Introduction to File Structures

In simple terms, a file is a collection of data stored on mass storage (e.g., disk or tape). But there is one important distinction that must be made at the outset when discussing file structures. And that is the difference between the logical and physical organization of the data.

On the whole a file structure will specify the logical structure of the data, that is the relationships that will exist between data items independently of the way in which these relationships may actually be realized within any computer. It is this logical aspect that we will concentrate on. The physical organization is much more concerned with optimizing the use of the storage medium when a particular logical structure is stored on, or in it. Typically for every unit of physical store there will be a number of units of the logical structure (probably records) to be stored in it.

For example, if we were to store a tree structure on a magnetic disk, the physical organization would be concerned with the best way of packing the nodes of the tree on the disk given the access characteristics of the disk.

Like all subjects in computer science the terminology of file structures has evolved higgledy-piggledy without much concern for consistency, ambiguity, or whether it was possible to make the kind of distinctions that were important.

It was only much later that the need for a well-defined, unambiguous language to describe file structures became apparent. In particular, there arose a need to communicate ideas about file structures without getting bogged down by hardware considerations.

## 1.2 Introduction to File System

In computing, a file system or file system controls how data is stored and retrieved. Without a file system, information placed in a storage medium would be one large body of data with noway to tell where one piece of information stops and the next begins. By separating the data into pieces and giving each piece a name, the information is easily isolated and identified.

Taking its name from the way paper-based information systems are named, each groups of data is called a “file”. The structure and logic rules used to manage the groups of information and their names is called a “file system”.

There are many different kinds of file systems. Each one has different structure and logic, properties of speed, flexibility, security, size and more. Some file systems have been designed to be used for specific applications. For example, the ISO 9660 file system is designed specifically for optical discs.

File systems can be used on numerous different types of storage devices that use different kinds of media. The most common storage device in use today is a hard disk drive. Other kinds of media that are used include flash memory, magnetic tapes, and optical discs. In some cases, such as with tmpfs, the computer's main memory (random-access memory, RAM) is used to create a temporary file system for short-term use.

Some file systems are used on local data storage devices; others provide file access via a network protocol(for example, NFS, SMB, or 9P clients). Some file systems are “virtual”. Meaning that the supplied “files” (called virtual files) are computed on request (e.g. procfs) or are merely a mapping into a different file system used as a backing store. The file system manages access to both the content of files and the metadata about those files. It is responsible for arranging storage space; reliability, efficiency, and tuning with regard to the physical storage medium are important design considerations.



## 1.3 Data Compression

Data compression is the process of modifying, encoding or converting the bits structure of data in such a way that it consumes less space on disk. It enables reducing the storage size of one or more data instances or elements. Data compression is also known as source coding or bit-rate reduction.

Data compression enables sending a data object or file quickly over a network or the Internet and in optimizing physical storage resources. Data compression has wide implementation in computing services and solutions, specifically data communications. Data compression works through several compressing techniques and software solutions that utilize data compression algorithms to reduce the data size.

### How compression works

Compression is performed by a program that uses a formula or algorithm to determine how to shrink the size of the data. For instance, an algorithm may represent a string of bits -- or 0s and 1s -- with a smaller string of 0s and 1s by using a dictionary for the conversion between them, or the formula may insert a reference or pointer to a string of 0s and 1s.

file compression can be as simple as removing all unneeded characters, inserting a single repeat character to indicate a string of repeated characters and substituting a smaller bit string for a frequently occurring bit string. Data compression can reduce a text file to 50% or a significantly higher percentage of its original size.

file compression (or data compression) is the act of reducing the size of a file while preserving the original data. Doing so allows the file to take up less space on a storage device, in addition to making it easier to transfer over the internet or otherwise. It's important to note that compression is not infinite.

## **LZW (Lempel–Ziv–Welch) Compression technique**

Lempel–Ziv–Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. It was published by Welch in 1984 as an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978. The algorithm is simple to implement and has the potential for very high throughput in hardware implementations. It is the algorithm of the widely used Unix file compression utility compress and is used in the GIF image format.

- LZW compression works best for files containing lots of repetitive data. This is often the case with text and monochrome images. Files that are compressed but that do not contain any repetitive information at all can even grow bigger!
- LZW compression is fast.
- LZW is a fairly old compression technique. All recent computer systems have the horsepower to use more efficient algorithms.
- Royalties have to be paid to use LZW compression algorithms within applications (see below).

LZW compression works by reading a sequence of symbols, grouping the symbols into strings, and converting the strings into codes. Because the codes take up less space than the strings they replace, we get compression. Characteristic features of LZW includes,

- LZW compression uses a code table, with 4096 as a common choice for the number of table entries. Codes 0-255 in the code table are always assigned to represent single bytes from the input file.
- When encoding begins the code table contains only the first 256 entries, with the remainder of the table being blanks. Compression is achieved by using codes 256 through 4095 to represent sequences of bytes.
- As the encoding continues, LZW identifies repeated sequences in the data, and adds them to the code table.
- Decoding is achieved by taking each code from the compressed file and translating it through the code table to find what character or characters it represents.

## LZW (Lempel–Ziv–Welch) DeCompression technique

The LZW decompressor creates the same string table during decompression. It starts with the first 256 table entries initialized to single characters. The string table is updated for each character in the input stream, except the first one. Decoding is achieved by reading codes and translating them through the code table being built.

The decompressor algorithm only requires the compressed text as an input, since it can build an identical string table from the compressed text as it is recreating the original text. However, an abnormal case shows up whenever the sequence character/string/character/string/character (with the same character for each character and string for each string) is encountered in the input and character/string is already stored in the string table. When the decompressor reads the code for character/string/character in the input, it cannot resolve it because it has not yet stored this code in its table. This special case can be dealt with because the decompressor knows that the extension character is the previously-encountered character.

### Pseudo-Code

```
1  Initialize table with single character strings
2  OLD = first input code
3  output translation of OLD
4  WHILE not end of input stream
5      NEW = next input code
6      IF NEW is not in the string table
7          S = translation of OLD
8          S = S + C
9      ELSE
10         S = translation of NEW
11     output S
12     C = first character of S
13     OLD + C to the string table
14     OLD = NEW
15 END WHILE
```

# **CHAPTER 2**

## **REQUIREMENT ANALYSIS AND DESIGN**

# **REQUIREMENT ANALYSIS AND DESIGN**

## **2.1 Domain Understanding**

The Main Objective of this Project is to Compress The Given PDF File. The outcome is it gives a Compressed PDF File in an Efficient, Friendly and In understandable way.

## **2.2 Classification of Requirements**

### **System Requirements**

- OS: Windows
- Java Installed

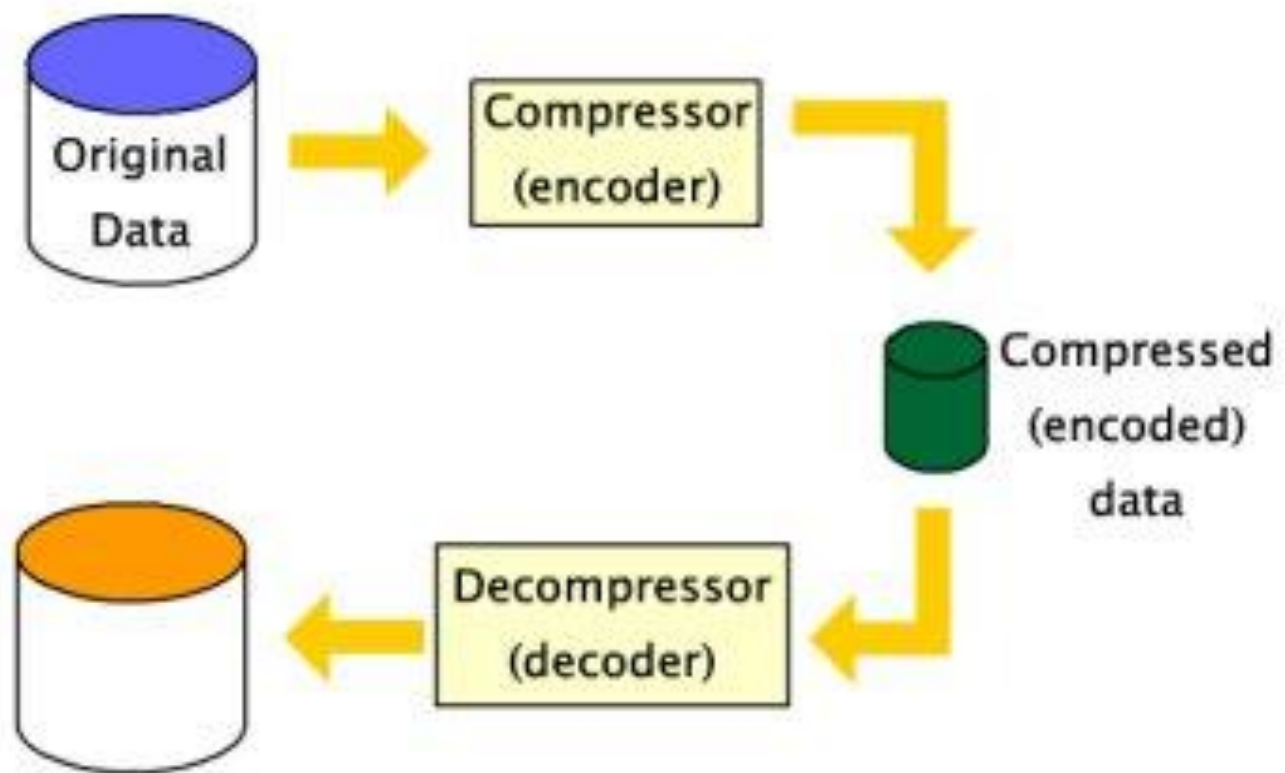
### **Software and Hardware Requirements**

Programming Languages: – JAVA

## **2.3 System Analysis**

When the program is executed, which primarily asks the user to Choose the Compression or Decompression whatever they want to access. If the user chooses the Compression, then it asks to input the PDF File path which they want to compress that. After the LZW File is obtained, the entire File is viewed for verification of Size of the file with name and location.

When user choose Decompression it starts to Decompress the LZW File and it takes only LZW File. After the completion of decompressing, the user is given information about decompression such as time taken, Decompressed pdf file Size, and The New File Path.



**Figure 1:Process Of File Compression**

# **CHAPTER 3**

## **IMPLEMENTATION**

## IMPLEMENTATION

The PDF File Compression Using LZW Algorithm is Implemented Through Two Types of Technique

- LZW COMPRESSION
- LZW DECOMPRESSION

- **LZW COMPRESSION**

Algorithm for LZW Compression:

Step1: Initialize string table with basic characters

Step2: Initialize prefix [ ] = empty

Repeat until no character left

Step3: create variable current Stream by reading the next character C from character stream

Step4: Check if current String in the string table

- If yes then[ ] [ ]C;go to step3
- Else add to[ ]C the string table
- Output code for[ ] to codestream
- [ ] = next character
- Go to step3

The Above Algorithm Shows How The LZW Algorithm Compresses The Given File.



## LZW DECOMPRESSION

### Algorithm for LZW Decompression

Step-1: Initialize table with single character strings

Step-2: OLD = first input code

Step-3: output translation of OLD

Step-4: WHILE not end of input stream

Step-5: NEW = next input code

Step-6: IF NEW is not in the string table

Step-7: S = translation of OLD

Step-8: S = S + C

ELSE

Step-9: S = translation of NEW

Step-10: output S

Step-11: C = first character of S

Step-12: OLD + C to the string table

Step-13: OLD = NEW

END WHILE

The Above Algorithm Shows How The LZW Algorithm DeCompresses The Given File.

# COMPRESSION

```

public void compress(String uncompressed) throws IOException {

    for (int i = 0; i < 256; i++) {
        dictionary.put(Character.toString((char) i), i);
    }

    RandomAccessFile read = new RandomAccessFile(uncompressed, "r");
    RandomAccessFile out = new RandomAccessFile(uncompressed.concat( ".lzw"), "rw");

    try {

        inputByte = read.readByte();
        int i = new Byte(inputByte).intValue();
        if (i < 0) {
            i += 256;
        }
        char ch = (char) i;
        str = "" + ch;

        // Reads Character by Character
        while (true) {
            inputByte = read.readByte();
            i = new Byte(inputByte).intValue();

            if (i < 0) {
                i += 256;
            }
            System.out.print(i + ", ");
            ch = (char) i;

            // If str + ch is in the dictionary..
            // Set str to str + ch
            if (dictionary.containsKey(str + ch)) {
                str = str + ch;
            } else {
                String s12 = to12bit(dictionary.get(str));

                // Store the 12 bits into an array and then write it to the
                // output file
                if (onleft) {
                    buffer[0] = (byte) Integer.parseInt(
                        s12.substring(0, 8), 2);
                    buffer[1] = (byte) Integer.parseInt(
                        s12.substring(8, 12) + "0000", 2);
                } else {
                    buffer[1] += (byte) Integer.parseInt(
                        s12.substring(0, 4), 2);
                    buffer[2] = (byte) Integer.parseInt(
                        s12.substring(4, 12), 2);
                    for (int b = 0; b < buffer.length; b++) {
                        out.writeByte(buffer[b]);
                        buffer[b] = 0;
                    }
                }
            }
        }
    }
}

```

**Figure 2:Compression Function**

## DECOMPRESSION

```

public void LZW-Decompress(String input) throws IOException {
    Array_char = new String[4096];

    for (int i = 0; i < 256; i++) {
        dictionary.put(i, Character.toString((char) i));
        Array_char[i] = Character.toString((char) i);
    }

    RandomAccessFile in = new RandomAccessFile(input, "r");
    RandomAccessFile out = new RandomAccessFile(input.replace( ".lzw", "" ), "rw");

    try {
        buffer[0] = in.readByte();
        buffer[1] = in.readByte();
        priorword = getvalue(buffer[0], buffer[1], onleft);
        onleft = !onleft;
        out.writeBytes(Array_char[priorword]);

        // Reads every 3 bytes and generates corresponding characters
        while (true) {
            if (onleft) {
                buffer[0] = in.readByte();
                buffer[1] = in.readByte();
                currword = getvalue(buffer[0], buffer[1], onleft);
            } else {
                buffer[2] = in.readByte();
                currword = getvalue(buffer[1], buffer[2], onleft);
            }
            onleft = !onleft;

            if (currword >= dictSize) {
                if (dictSize < 4096) {
                    Array_char[dictSize] = Array_char[priorword]
                        + Array_char[priorword].charAt(0);
                }
                dictSize++;
                out.writeBytes(Array_char[priorword]
                    + Array_char[priorword].charAt(0));
            } else {
                if (dictSize < 4096) {
                    Array_char[dictSize] = Array_char[priorword]
                        + Array_char[currword].charAt(0);
                }
                dictSize++;
                out.writeBytes(Array_char[currword]);
            }
            priorword = currword;
        }
    } catch (EOFException e) {
        in.close();
        out.close();
    }
}

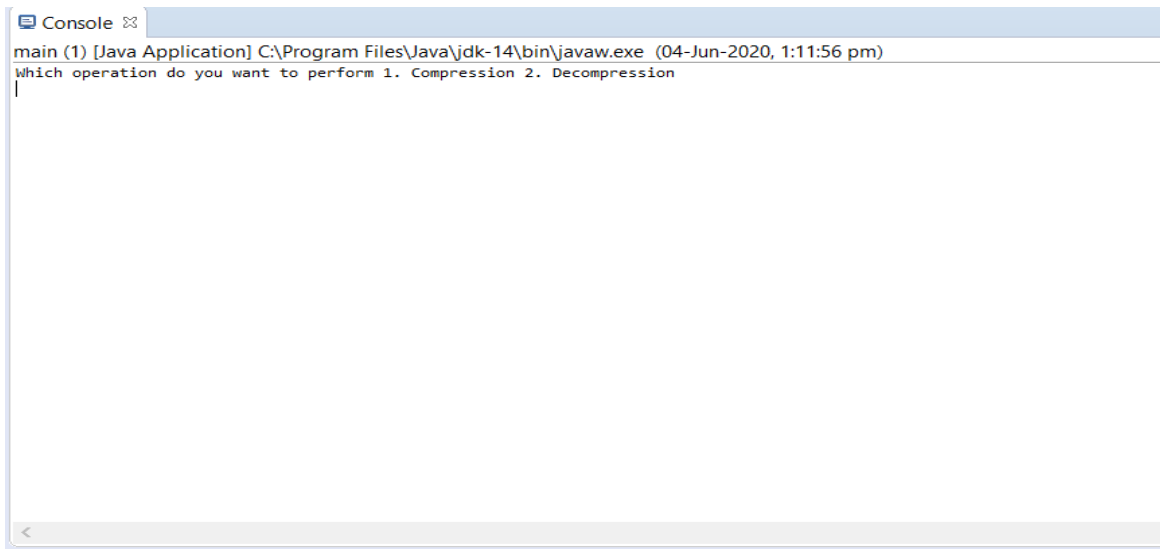
```

**Figure 3: Decompression Function**

# **CHAPTER 4**

## **RESULT AND ANALYSIS**

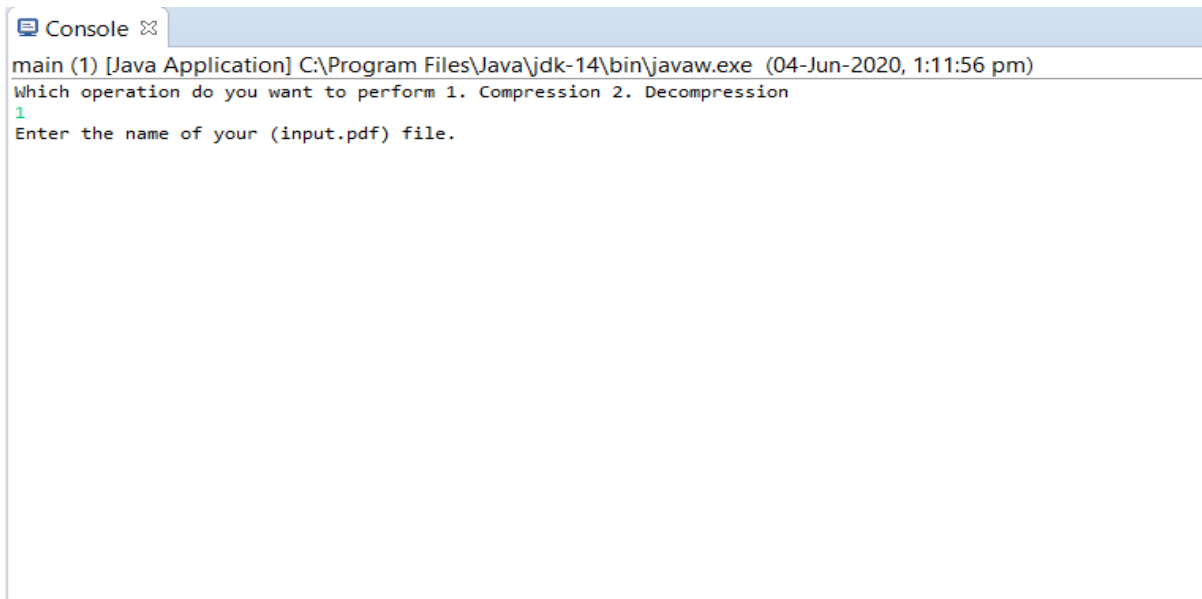
## RESULT AND ANALYSIS



```
Console
main (1) [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (04-Jun-2020, 1:11:56 pm)
Which operation do you want to perform 1. Compression 2. Decompression
|
```

**Figure 4:**

**The Above Figure is the Output Path Screen Which Contains Two operations Compression And Decompressions in which user has to Choose.**



```
Console
main (1) [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (04-Jun-2020, 1:11:56 pm)
Which operation do you want to perform 1. Compression 2. Decompression
1
Enter the name of your (input.pdf) file.
```

**Figure 5:**

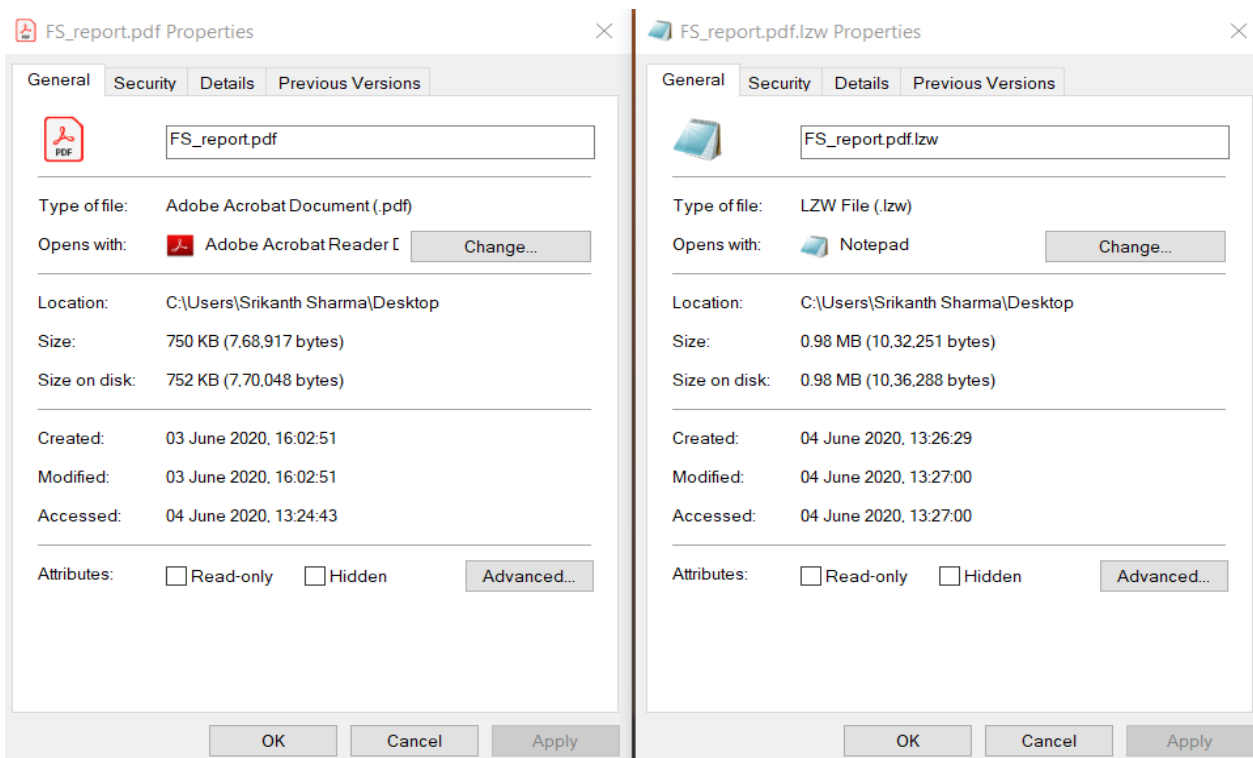
**The Above Figure Displays the Compression Operation in which User has to Give the PDF File Path in which They want to Compress.**

```

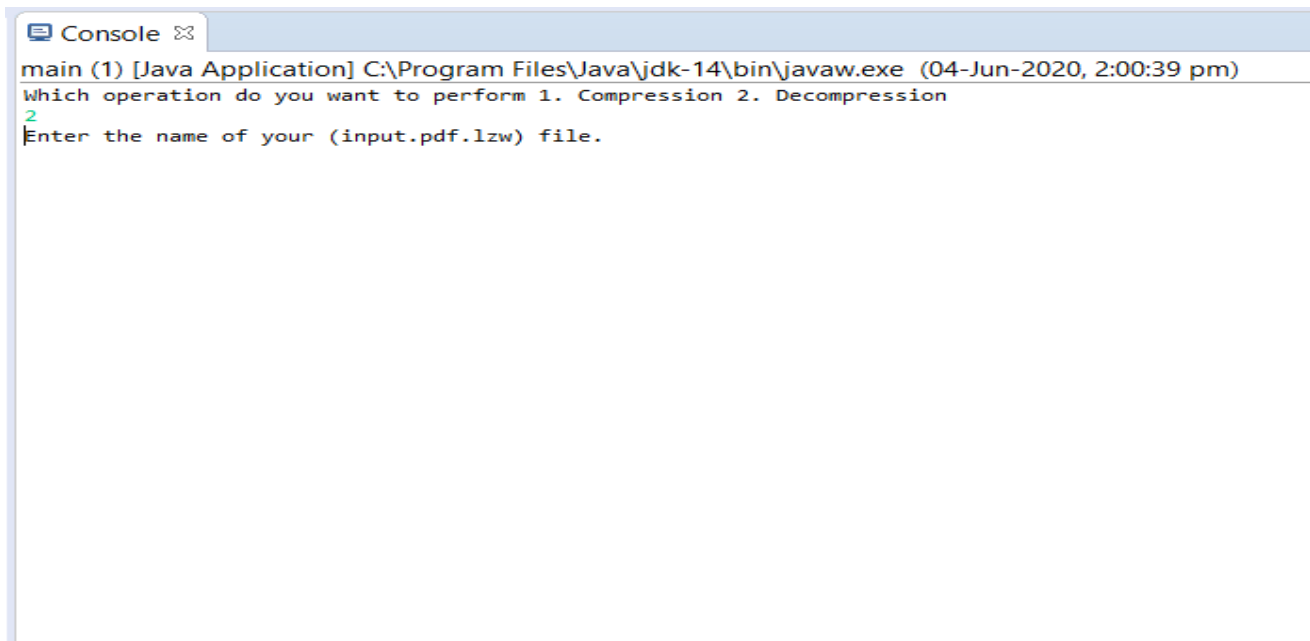
Console
main (1) [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (04-Jun-2020, 1:11:56 pm)
30, 68, 70, 45, 49, 46, 53, 13, 10, 37, 181, 181, 181, 181, 13, 10, 49, 32, 48, 32, 111, 98, 106, 13, 10, 6
Compression of your file is complete!
31
Your new file is named: C:/Users/Srikanth Sharma/Desktop/FS_report.pdf.lzw
Which operation do you want to perform 1. Compression 2. Decompression

```

**Figure 6:**  
The Above Figure shows The Result of The Compression File And It Shows The Compressed Time and Compressed File Path.



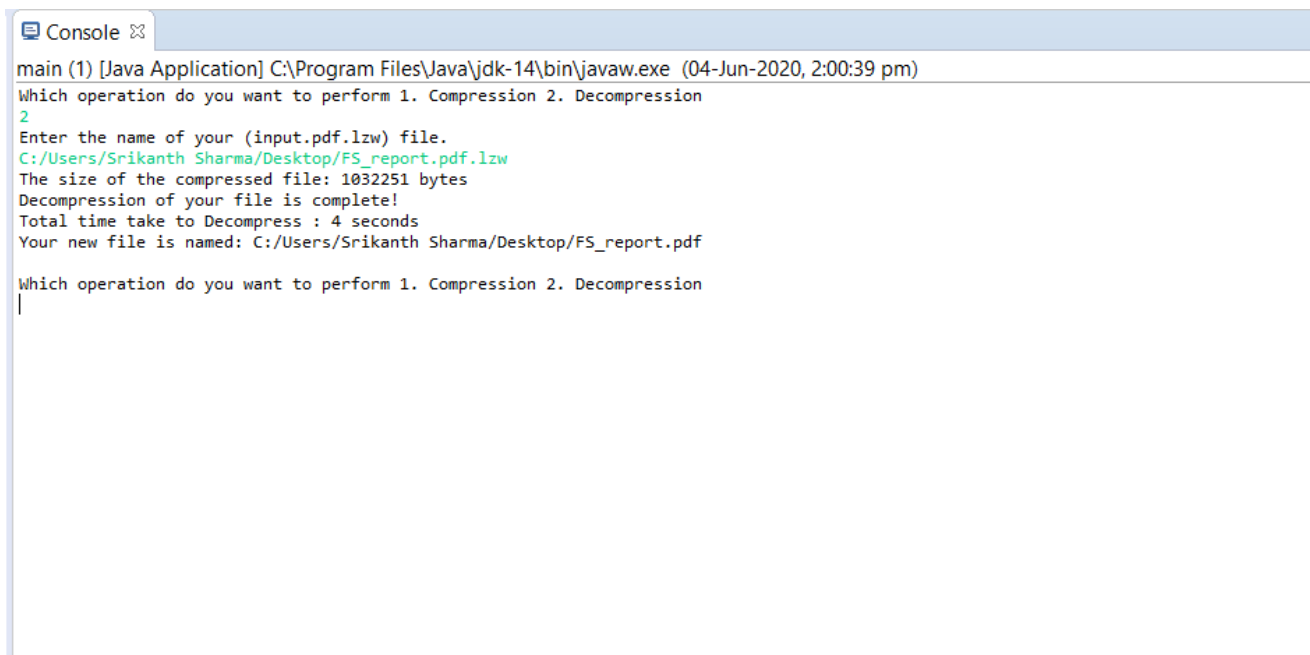
**Figure 7:**  
The Comparision B/w Orginal File and Compressed File



```
main (1) [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (04-Jun-2020, 2:00:39 pm)
Which operation do you want to perform 1. Compression 2. Decompression
2
Enter the name of your (input.pdf.lzw) file.
```

**Figure 8:**

**The Above Figure Displays the DeCompression Operation in which User has to Give the LZW File Path in which They want to DeCompress.**

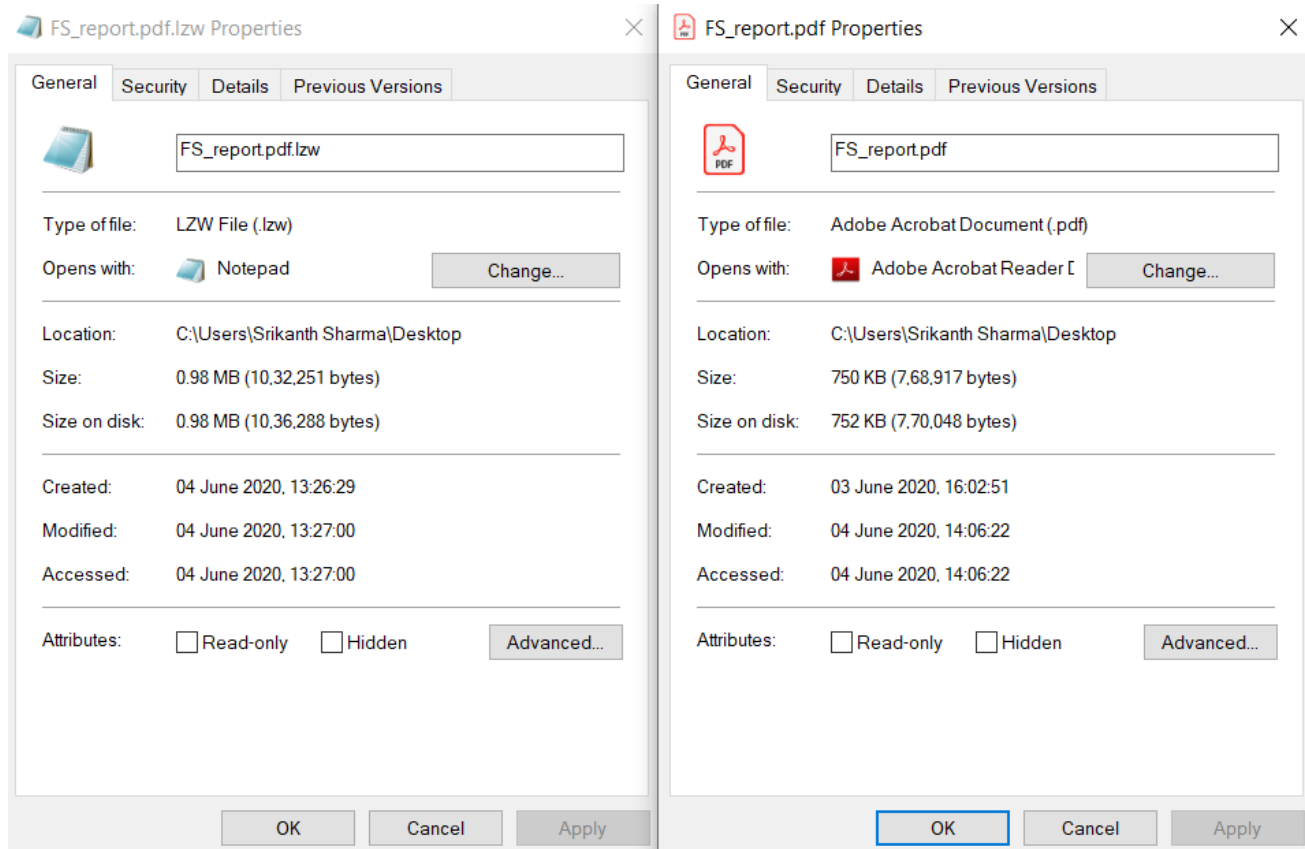


```
main (1) [Java Application] C:\Program Files\Java\jdk-14\bin\javaw.exe (04-Jun-2020, 2:00:39 pm)
Which operation do you want to perform 1. Compression 2. Decompression
2
Enter the name of your (input.pdf.lzw) file.
C:/Users/Srikanth Sharma/Desktop/FS_report.pdf.lzw
The size of the compressed file: 1032251 bytes
Decompression of your file is complete!
Total time take to Decompress : 4 seconds
Your new file is named: C:/Users/Srikanth Sharma/Desktop/FS_report.pdf

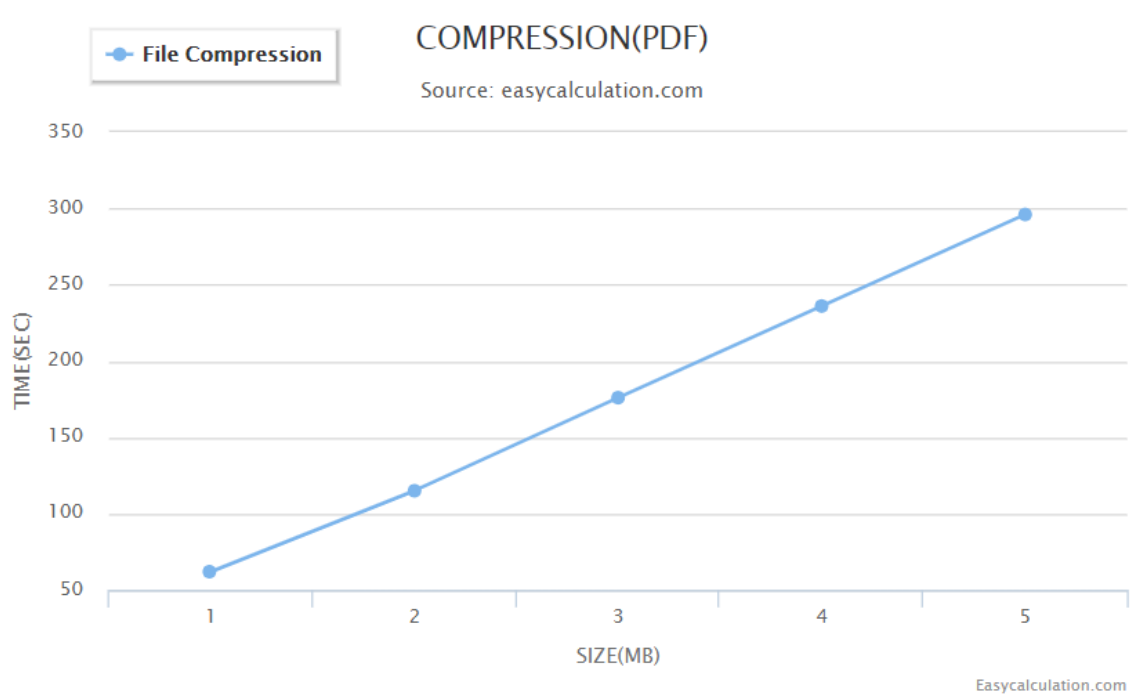
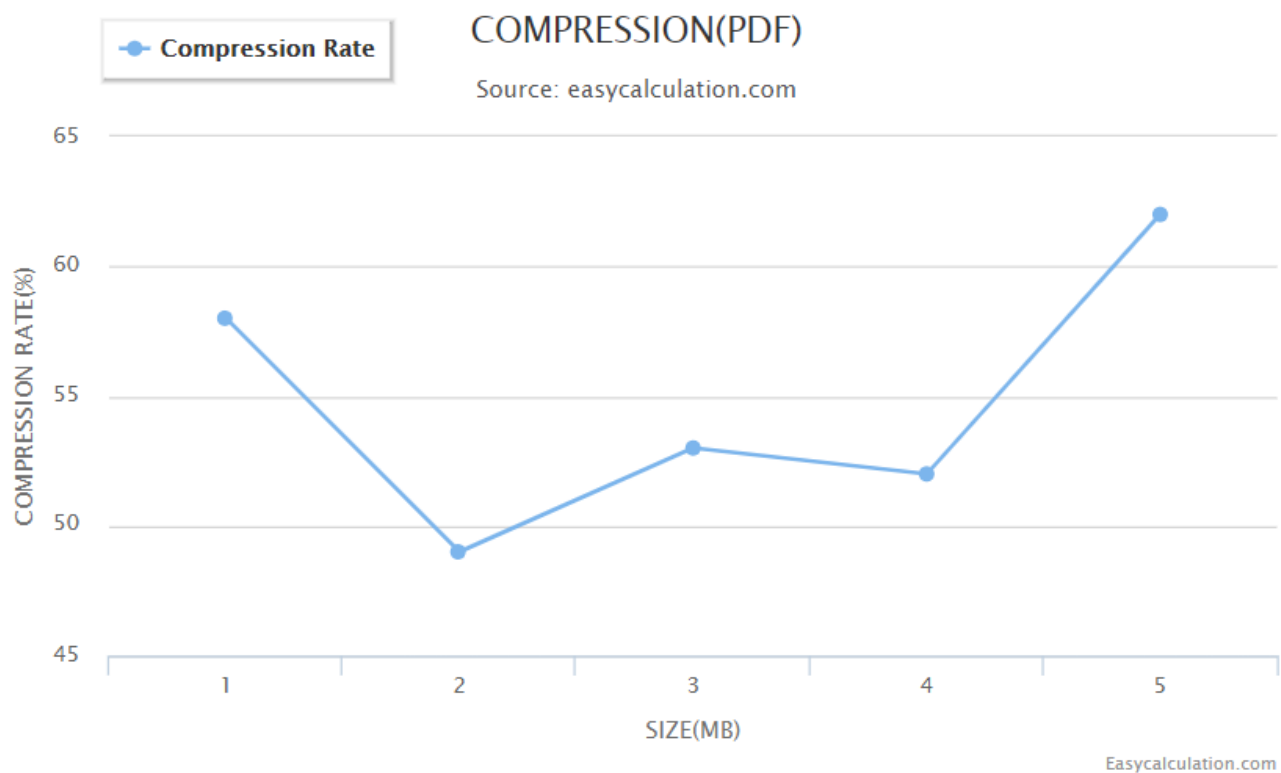
Which operation do you want to perform 1. Compression 2. Decompression
|
```

**Figure 9:**

**The Above Figure shows The Result of The DeCompression File And It Shows The DeCompressed Time and DeCompressed File Path.**

**Figure10:****The Original File and DeCompressed File.**



**Figure11: Compression Graph(Time V/s Size)****Figure 12: Compression graph(Size V/S Compression Rate)**

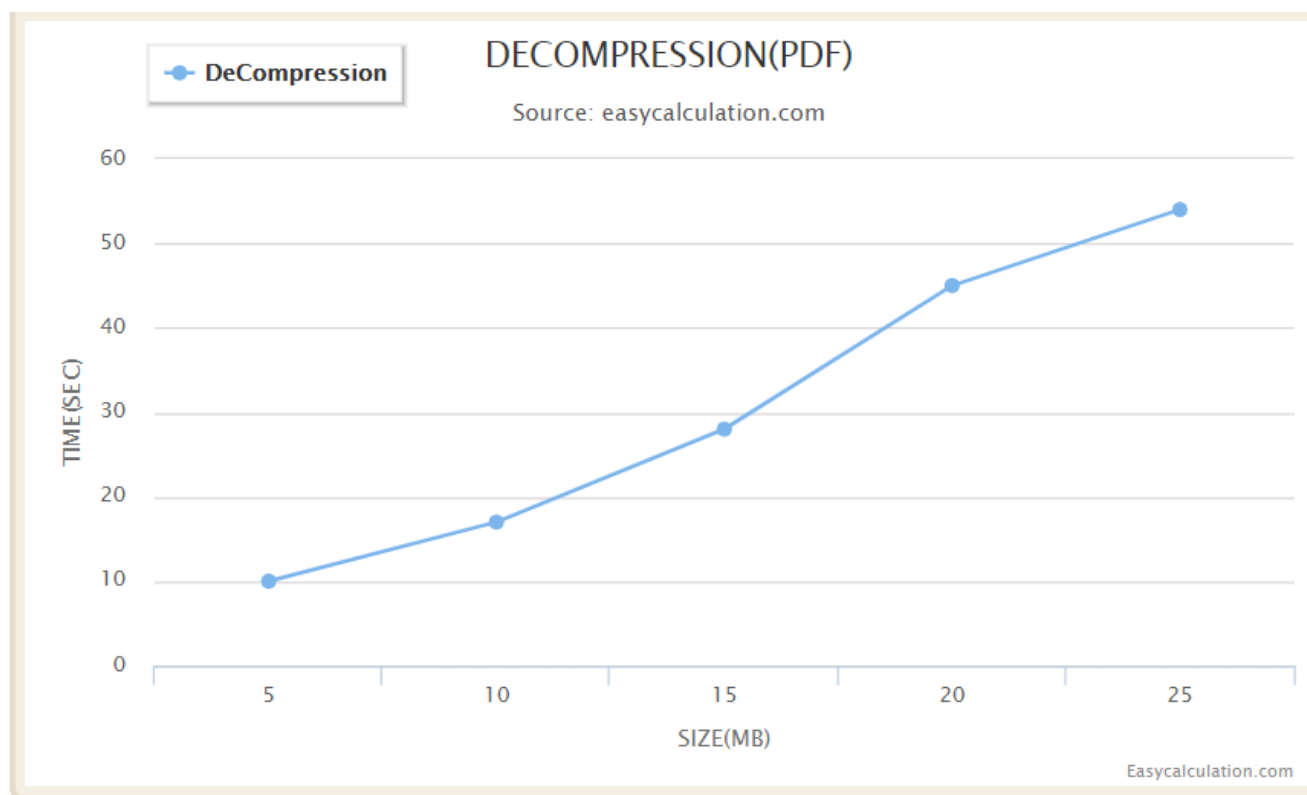


Figure 13: DeCompression graph(Size V/S Time)

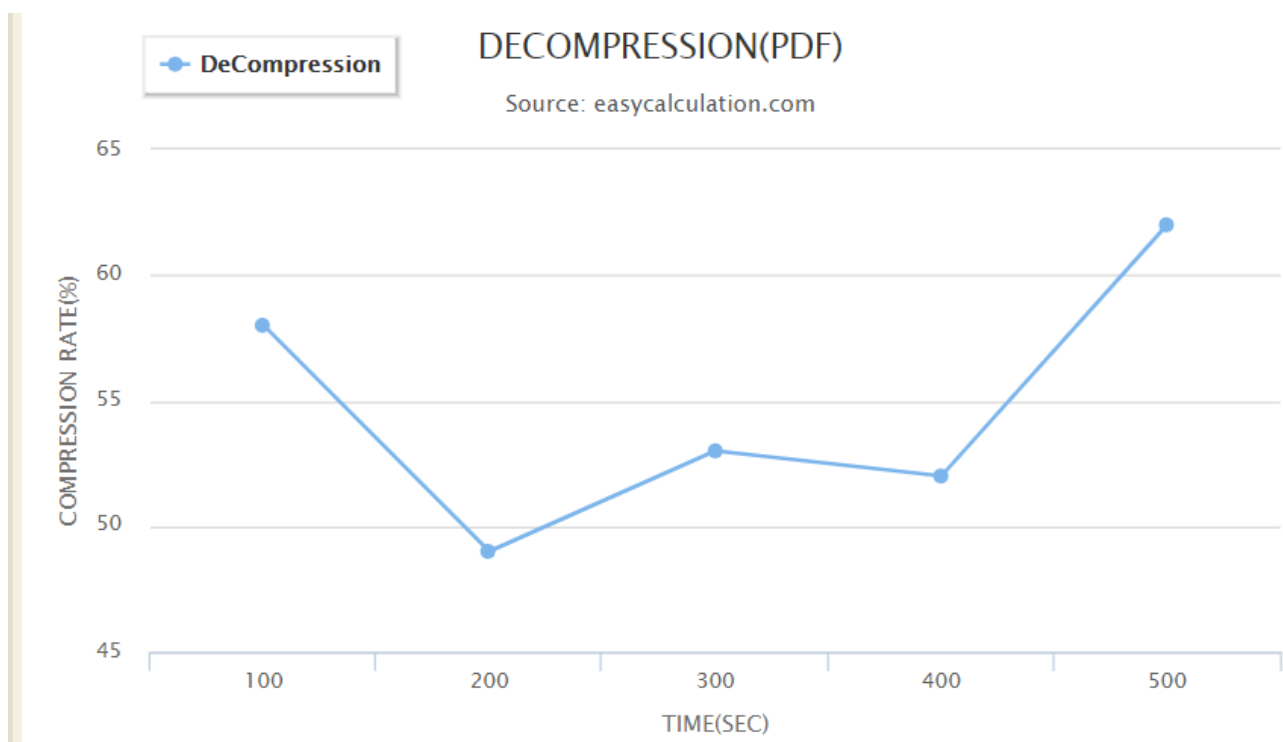


Figure 14: Compression graph(Size V/S Compression Rate)

## REFERENCES:

- [1] <https://www.sciencedirect.com/topics/computer-science/data-compression>
- [2] <https://meta.stackoverflow.com/questions/288599/compress-compression-tags>
- [3] <https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/>
- [4] <https://in.mathworks.com/matlabcentral/fileexchange/4899-lzw-compression-algorithm>
- [5] <https://whatis.techtarget.com/definition/LZW-compression>

## CONCLUSION:

Compression also keeps an instrument range within the range of your recording equipment, enabling you to record a more clear, clean sound. Technically speaking, if instruments become too loud or too soft during a recording, the sound levels can pitch too high or too low within the range your equipment can capture.

In This Mini-Project I Learnt About How Data Compression Works and How Different Forms of Algorithms Works with Compression.





