OPEN ENDED ASSIGNMENT

PROBLEM STATEMENT:

Determine the compression efficiency and/ or error correction in speech signal.
i. Convert your Speech signal to Binary data.
ii. Simulate: On the generated binary data, apply source coding technique to compress data and find its efficiency.

ALGORITHM:

1. Convert the audio file to text using *recognize_google()* function present in *speech_recognition* library.
2. Convert each character from acquired text into its respective ASCII value using *ascii()*.
3. Convert the ASCII value from decimal to binary using *bin()*.
4. Create a tree that takes the characters as input and produces an appropriate Huffman code. Help functions - *sorted(), append()*
5. Calculate H(X) = Σp*log(1/p), and L = Σp*(length of code)
6. Calculate the efficiency of Huffman code, efficiency = H(X)/L

PART 1 OF SOLUTION:

Code:

```
!pip install SpeechRecognition
import speech_recognition as sr

s = "Itct.wav"
r = sr.Recognizer()
with sr.AudioFile(s) as source: #load audio to memory
  audio_data = r.record(source) #convert from speech to text
  text = r.recognize_google(audio_data)
print(text)

ascii=[]
for i in range(0, len(text)):
  ascii.append(ord(text[i]))
  #add each character's ASCII value into list
print(ascii)

intascii=[]
for i in range(0, len(text)):
  intascii.append(int(ascii[i]))
```

```
print(intascii)

def decimalToBinary(n): #defining a function to allow repetitive usage
  return bin(n).replace("0b","")
for i in range(0, len(ascii)):
  intascii[i]=decimalToBinary(intascii[i]) #convert to binary
  print(intascii[i])
```

Output:

```
she sells seashells on the seashore

[115, 104, 101, 32, 115, 101, 108, 108, 115, 32, 115, 101, 97, 115,
104, 101, 108, 108, 115, 32, 111, 110, 32, 116, 104, 101, 32, 115, 101,
97, 115, 104, 111, 114, 101]

1110011
1101000
1100101
100000
1110011
  ...
1110011
1101000
1101111
1110010
1100101
```

PART 2 OF SOLUTION:

Code:

```
string = text
class NodeTree(object):
def __init__(self, left=None, right=None):
  self.left = left
  self.right = right
def children(self):
  return (self.left, self.right)
def nodes(self):
  return (self.left, self.right)
def __str__(self):
  return '%s_%s' % (self.left, self.right)

#main function implementing Huffman coding
def huffman_code_tree(node, left=True, binString=''):
    if type(node) is str:
        return {node: binString}
    (l, r) = node.children()
    d = dict()
```

```
    d.update(huffman_code_tree(l, True, binString + '0'))
    d.update(huffman_code_tree(r, False, binString + '1'))
    return d

#calculating frequency
freq = {}
for c in string:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1
freq = sorted(freq.items(), key=lambda x: x[1], reverse=True)
nodes = freq
while len(nodes) > 1:
  (key1, c1) = nodes[-1]
  (key2, c2) = nodes[-2]
  nodes = nodes[:-2]
  node = NodeTree(key1, key2)
  nodes.append((node, c1 + c2))
  nodes = sorted(nodes, key=lambda x: x[1], reverse=True)

huffmanCode = huffman_code_tree(nodes[0][0])

print(' Char | Huffman code ')
print('----------------------')
for (char, frequency) in freq:
    print(' %-4r |%12s' % (char, huffmanCode[char]))

#finding efficiency
import math
freq = {} #dictionary
for c in string:
    if c in freq:
        freq[c] += 1
    else:
        freq[c] = 1
hx=0
for i in range(0,len(value)):
  prob=freq[text[i]] / len(text)
  lo=math.log10(1/prob)
  hx+=prob*lo
L=0
for i in range(0,len(text)):
  L+=(len(huffmanCode[text[i]]))/len(text)
print(hx)
print("L=")
print(L)
eff=hx/L
```

```
print("Efficiency=")
print(eff)
```

Output:

```
Char | Huffman code
----------------------
 's'  |          10
 'e'  |          00
 ' '  |         110
 'h'  |         011
 'l'  |         010
 'a'  |       11111
 'o'  |       11110
 'n'  |       11100
 't'  |      111011
 'r'  |      111010

H(X)=
2.396667332704547
L=
3.028571428571427
Efficiency=
0.79135242117603
```

CONCLUSION:

For this assignment, I have used Python as it has in-built functions readily available. I have run my code online using Google Collaboratory.

Huffman code is a variable length coding technique which assigns shorter codes to more probable characters and longer codes to fewer probable characters.

Initially, the first audio file said ‘information theory and coding Techniques’ which was only 35% effiecient.
Later I ran the code again for ‘she sells seashells on the seashore’ which caused a drastic increase in effieciency to 80%.

Thus, we can conclude that in applications where the dictionary is smaller and occurrence of each character is higher, Huffman coding technique seems to be more efficient than in other applications.