Problem 1

Breaks SHA1 hashes in a brute force manner.

Here is the python code.

- Import:
  Hashlib and time

```
1   import hashlib
2   import time
```

- The function to break the hash.
  Using the sha1() function to break the given string.

```
10  def sha1_hash(string):
11      return hashlib.sha1(string.encode('utf-8')).hexdigest()
12
13  def bf_sha1(hash_to_break, password_file):
14      count = 0
15      start_time = time.time()
16      with open(password_file, 'r') as file:
17          for password in file:
18              password = password.strip()  # Remove potential new line characters
19              count += 1
20              # For salted passwords, concatenate the salt and the password
21              password = 'redbull'+ password
22              if sha1_hash(password) == hash_to_break:
23                  end_time = time.time()
24                  return password, count, end_time - start_time
25      end_time = time.time()  # If the password was not found
26      return None, count, end_time - start_time
```

- The output function to print the hash, password and taken time.

```
28  # For leet hacker hash
29  hash_to_break = '9d6b628c1f81b4795c0266c0f12123c1e09a7ad3'  # hash
30  password_file = 'password.txt'  # the path of 100000 passwords list
31  result, tries, time_taken = bf_sha1(hash_to_break, password_file)
32
33
34  if result:
35      print(f"9d6b628c1f81b4795c0266c0f12123c1e09a7ad3") # change the hash if the question need
36      print(f"Password: {result}")
37      print(f"Took {tries} attempts to crack input hash. Time taken: {time_taken} seconds")
38  else:
39      print(f"Password not found in the passwords.txt\n Tries: {tries}, Time: {time_taken} seconds")
```

(a) Easy hash：ef0ebbb77298e1fbd81f756a4efc35b977c93dae

```
ef0ebbb77298e1fbd81f756a4efc35b977c93dae
Password: orange
Took 124 attempts to crack input hash. Time taken: 0.0 seconds
```

(b) Medium hash：0bc2f4f2e1f8944866c2e952a5b59acabd1cebf2

```
0bc2f4f2e1f8944866c2e952a5b59acabd1cebf2
Password: starfish
Took 2681 attempts to crack input hash. Time taken: 0.014507293
701171875 seconds
```

(c) Leet hacker hash：9d6b628c1f81b4795c0266c0f12123c1e09a7ad3

#Hint：The salt term here is：dfc3e4f0b9b5fb047e9be9fb89016f290d2abb06

Due to the answer of the salt (dfc3e4f0b9b5fb047e9be9fb89016f290d2abb06) is redbull.

```
dfc3e4f0b9b5fb047e9be9fb89016f290d2abb06
Password: redbull
Took 2785 attempts to crack input hash. Time taken: 0.0 seconds
```

Hence, I add the salt answer “redbull” before the password in the txt file.

```
9d6b628c1f81b4795c0266c0f12123c1e09a7ad3
Password: redbullpuppy
Took 2854 attempts to crack input hash. Time taken: 0.008421897
888183594 seconds
```

Above is the result.

Problem 2

Calculate the checksums of the downloaded video file by using various hash functions.

(a) Write a Python 3 program to compare the speed of the hash algorithms.

- Imports:

  hashlib: A module containing different hash functions.

  time: A module for measuring time.

- hash functions:
  The code tests multiple hash functions including MD5, SHA-1, SHA-224, SHA-256, SHA-512, SHA3-224, SHA3-256, and SHA3-512.
- Function to Calculate and Time Hash:
  This function takes a hash function and the file path as arguments. It measures the time taken to read the file and compute its hash, and it returns the time taken and the hexadecimal representation of the hash.
- Calculating Hash and Time for Each Hash Function:
  For each hash function, it calculates the execution time and the hash value of the specified video file using the calculate_hash function.

```python
import hashlib
import time

video_path = 'video.mp4'

# Hash functions to test
hash_functions = {
    'MD5': hashlib.md5(),
    'SHA1': hashlib.sha1(),
    'SHA-224': hashlib.sha224(),
    'SHA-256': hashlib.sha256(),
    'SHA-512': hashlib.sha512(),
    'SHA3-224': hashlib.sha3_224(),
    'SHA3-256': hashlib.sha3_256(),
    'SHA3-512': hashlib.sha3_512()
}
```

```python
# Function to calculate and time hash
def calculate_hash(hash_func, file_path):
    start_time = time.time()  # Start timer
    with open(file_path, 'rb') as file:  # Open file in binary mode
        while chunk := file.read(8192):  # Read file in chunks of 8 KB
            hash_func.update(chunk)  # Update hash with each chunk
    end_time = time.time()  # End timer
    return end_time - start_time, hash_func.hexdigest()  # Return time taken and hash value

# Calculate hash and time for each hash function
execution_times = {}
for name, func in hash_functions.items():
    execution_times[name], _ = calculate_hash(func, video_path)
```

The code to calculate the time of hashing and find the password.

```python
# Print out the execution times
print("(a)\nExecution times for each hash function:")
for name, time_taken in execution_times.items():
    print(f"{name}: {time_taken} seconds")

# Find and print the fastest hash function
fastest_hash = min(execution_times, key=execution_times.get)
print("\n(b)")
print(f"The fastest hash function is: {fastest_hash}")

# Print the ranking of hash functions by speed
print("\n(c)\nRanking of hash functions by speed:")
sorted_times = sorted(execution_times.items(), key=lambda x: x[1])
for rank, (name, time) in enumerate(sorted_times, start=1):
    print(f"{rank}. {name}: {time} seconds")
```

Here is the output function.

```
(a)
Execution times for each hash function:
MD5: 0.5509078502655029 seconds
SHA1: 0.12029862403869629 seconds
SHA-224: 0.1297929286956787 seconds
SHA-256: 0.13234639167785645 seconds
SHA-512: 0.2336575984954834 seconds
SHA3-224: 0.30628061294555664 seconds
SHA3-256: 0.3118858337402344 seconds
SHA3-512: 0.5396742820739746 seconds
```

(b) The fastest is SHA1 and this is the output.

```
(b)
The fastest hash function is: SHA1
```

(c) Here is the ranking：

```
(c)
Ranking of hash functions by speed:
1. SHA1: 0.12029862403869629 seconds
2. SHA-224: 0.1297929286956787 seconds
3. SHA-256: 0.13234639167785645 seconds
4. SHA-512: 0.2336575984954834 seconds
5. SHA3-224: 0.30628061294555664 seconds
6. SHA3-256: 0.3118858337402344 seconds
7. SHA3-512: 0.5396742820739746 seconds
8. MD5: 0.5509078502655029 seconds
```

Problem 3

Given the transposition cipher:

UONCS VAIHG EPAAH IGIRL BIECS TECSW PNITE TIENO IEEFD OWECX
TRSRX STTAR TLODY FSOVN EOECO HENIO DAARQ NAELA FSGNO PTE

1. Determine the dimensions of rectangle

The ciphertext is total 98 letters, and the approximately 40% of plaintext of vowels.

Using the function to count the difference, and count the average number.

```python
L = len(text)
for i in range(1, L+1):
    if(L%int(i) == 0):
        print("For", i, "x", int(L/i) ,"rectangle, the avg of the difference is ", end='')
        vowel = L/i*0.4
        freq = [0] * i
        diff = 0.0
        for j in range(L):
            if(text[j] in vowels):
                freq[j%i] += 1
        for j in range(i):
            diff += abs(freq[j]-vowel)
        diff = diff/i # count the avg of diff
        print(diff)
```

```
For 1 x 98 rectangle, the avg of the difference is 0.20000000000000284
For 2 x 49 rectangle, the avg of the difference is 1.5
For 7 x 14 rectangle, the avg of the difference is 0.6571428571428573
For 14 x 7 rectangle, the avg of the difference is 0.557142857142857
For 49 x 2 rectangle, the avg of the difference is 0.5510204081632651
For 98 x 1 rectangle, the avg of the difference is 0.47959183673469374
```

Above is the result.

2. Decrypt the ciphertext

Since the 1 x 98, 2 x 49, 49 x 2 and 98 x 1 cannot have the answer. We take 7 x 14 and 14 x 7 to discuss.

```
rec_7x14=[]
idx=0
for i in range(14):
    rec_7x14.append([])
    for j in range(7):
        #if the word is not the uppercase letter, then skip
        # if(ord(text[idx]) < 65  or ord(text[idx]) > 90):
        #     idx += 1
        #if the  word is from A to Z
        rec_7x14[i].append(text[idx])
        idx += 1

print("for rectangle 7 X 14:")
diff_7x14=0
for i in range(7):
    vowelNum=0
    for j in range(14):
        #count the number of vowels in every rows
        if(rec_7x14[j][i] in vowels):
            vowelNum += 1
        print(rec_7x14[j][i], end=' ')
    diff=round(abs(vowelNum - 0.4 * 14), 2)
    diff_7x14 += diff
    print("\tdifference: " + str(diff))
print("average difference: " + str(diff_7x14/7))
```

The code to count the rectangle 7 x 14.

```
for rectangle 7 X 14:
U I H I S T E X T D E N Q S     difference: 0.6
O H I E W I F T T Y O I N G     difference: 0.4
N G G C P E D R A F E O A N     difference: 0.6
C E I S N N O S R S C D E O     difference: 0.6
S P R T I O W R T O O A L P     difference: 0.6
V A L E T I E X L V H A A T     difference: 0.4
A A B C E E C S O N E R F E     difference: 1.4
average difference: 0.6571428571428571
```

Here is the result of 7x14.

```python
# rectangle 14*7
rec_14x7=[]
idx=0
for i in range(7):
    rec_14x7.append([])
    for j in range(14):
        # if(ord(text[idx]) < 65  or ord(text[idx]) > 90):
        #     idx += 1
        rec_14x7[i].append(text[idx])
        idx += 1

print("\nfor rectangle 14 x 7:")
diff147=0
for i in range(14):
    vowelNum=0
    for j in range(7):
        if(rec_14x7[j][i] in vowels):
            vowelNum += 1
        print(rec_14x7[j][i], end=' ')
    diff=round(abs(vowelNum - 0.4 * 7), 2)
    diff147 += diff
    print("\tdifference: " + str(diff))
print("average difference: " + str(diff147/14))
```

The code to count the rectangle 14 x 7.

```
for rectangle 14 x 7:
U H S E T E Q   difference: 0.2
O I W F T O N   difference: 0.2
N G P D A E A   difference: 0.2
C I N O R C E   difference: 0.2
S R I W T O L   difference: 0.8
V L T E L H A   difference: 0.8
A B E C O E F   difference: 1.2
I I T X D N S   difference: 0.8
H E I T Y I G   difference: 0.2
G C E R F O N   difference: 0.8
E S N S S D O   difference: 0.8
P T O R O A P   difference: 0.2
A E I X V A T   difference: 1.2
A C E S N R E   difference: 0.2
average difference: 0.5571428571428572
```

Here is the result of 14x7.

The average difference of 7 x 14 is larger than 14 x 7, so 14 x 7 rectangle is more likely.

```
56  print("\nThe plain text:")
57  for i in range(14):
58      print(rec_14x7[4][i], end=' ')
59      print(rec_14x7[1][i], end=' ')
60      print(rec_14x7[5][i], end=' ')
61      print(rec_14x7[6][i], end=' ')
62      print(rec_14x7[0][i], end=' ')
63      print(rec_14x7[3][i], end=' ')
64      print(rec_14x7[2][i])
```

```
The plain text:
T H E Q U E S
T I O N O F W
A G E A N D P
R I C E C O N
T R O L S W I
L L H A V E T
O B E F A C E
D I N S I X T
Y E I G H T I
F C O N G R E
S S D O E S N
O T A P P R O
V E A T A X I
N C R E A S E
```

Here is the plain text：

THE QUESTION OF WAGE AND PRICE CONTROLS WILL HAVE TO BE FACED IN SIXTY EIGHT IF CONGRESS DOES NOT APPROVE A TAX INCREASE