Problem 1. * I import hashlib, requests, and datetime.

```python
# Function to load password list from URL
def load_password_list(url):
    response = requests.get(url)
    if response.status_code == 200:
        return response.text.splitlines()
    else:
        print("Failed to download the password list.")
        return None
```

⇒ we use load_password_list function to get the password in password list

```python
# Function to break SHA1 hash using password list
def break_sha1_hash(hash_to_break, password_list):
    start_time = datetime.now()
    attempts = 0
    for password in password_list:
        attempts += 1
        hashed_password = hashlib.sha1(password.encode()).hexdigest()
        if hashed_password == hash_to_break:
            end_time = datetime.now()
            time_taken = end_time - start_time
            time_taken_str = str(time_taken.total_seconds())
            print(f"Hash: {hash_to_break}")
            print(f"Password: {password}")
            print(
                f"Took {attempts} attempts to crack input hash. Time Taken: {time_taken_str}",
            )
            return
    print("Unable to crack the hash with the given password list.")
```

⇒ In this break_sha1_hash function, we use for loop to compare the hash of each password in the password list one by one with hash provided.

(a) result:

```
Easy hash:
Hash: ef0ebbb77298e1fbd81f756a4efc35b977c93dae
Password: orange
Took 124 attempts to crack input hash. Time Taken: 0.001011
```

(b) result:

```
medium hash:
Hash: 0bc2f4f2e1f8944866c2e952a5b59acabd1cebf2
Password: starfish
Took 2681 attempts to crack input hash. Time Taken: 0.003009
```

(c) * I import hashlib, requests, and time.

```
# Function to break SHA1 hash using password list to find salt
def find_salt(hash_to_break, password_list):
    for password in password_list:
        hashed_password = hashlib.sha1(password.encode()).hexdigest()
        if hashed_password == hash_to_break:
            return password
    print("Unable to crack the hash with the given password list.")
```

⇒ First of all, we use find_salt function to find the string of salt term.

```
# Function to break SHA1 hash (with salt) using password list
def break_sha1_hash_with_salt(hash_to_break, password_list, salt):
    start_time = datetime.now()
    attempts = 0
    for password in password_list:
        attempts += 1
        hashed_password = hashlib.sha1((salt + password).encode()).hexdigest()
        if hashed_password == hash_to_break:
            end_time = datetime.now()
            time_taken = end_time - start_time
            time_taken_str = str(time_taken.total_seconds())
            print(f"Hash: {hash_to_break}")
            print(f"Password: {password}")
            print(
                f"Took {attempts} attempts to crack input hash. Time Taken: {time_taken_str}"
            )
            return
    print("Unable to crack the hash with the given password list.")
```

⇒ Then, we use break_sha1_hash_with_salt function to find salt term concatenating to which password.

result:

```
Leet hacker hash:
salt: redbull
Hash: 9d6b628c1f81b4795c0266c0f12123c1e09a7ad3
Password: puppy
Took 2854 attempts to crack input hash. Time Taken: 0.004013
```

Problem 2.

(a)

```python
# Function to download file and return its content
def download_file(url):
    response = requests.get(url)
    if response.status_code == 200:
        return response.content
    else:
        return None
```

⇒ We use download_file function to get information of the video.

```python
def compare_hash_speed(url):
    data = download_file(url)
    if data:
        algorithms = [
            "md5",
            "sha1",
            "sha224",
            "sha256",
            "sha512",
            "sha3_224",
            "sha3_256",
            "sha3_512",
        ]
        results = {}
        for algorithm in algorithms:
            time_taken = measure_time(algorithm, data)
            results[algorithm] = time_taken

        # Sort results based on time taken
        sorted_results = sorted(results.items(), key=lambda x: x[1])

        # Print results
        print("Hash Algorithm\t\tTime Taken (s)")
        for algorithm, time_taken in sorted_results:
            if len(algorithm) < 7:
                print(f"{algorithm}\t\t\t{time_taken:.6f}")
            else:
                print(f"{algorithm}\t\t{time_taken:.6f}")

        # Return the fastest algorithm
        fastest_algorithm = sorted_results[0][0]
        return fastest_algorithm
    else:
        print("Failed to download the file.")
        return None
```

⇒ we use this compare_hash_speed function to measure time of each algorithm. Then, compare their speed to choose the fastest and rank.

(b·c) Speed rank

```
Hash Algorithm          Time Taken (s)
sha1                    0.071115
sha224                  0.079911
sha256                  0.083691
sha512                  0.178190
md5                     0.196235
sha3_224                0.304312
sha3_256                0.319581
sha3_512                0.593552

The fastest hash algorithm is: sha1
```

→ sha1 is the fastest.

Problem 3

There are 98 letters in the cipher text.
Because we know that the message fills the rectangle, this suggests either a 7×14 or 14×7 array.

① 7×14 array

													Number of vowels	Difference
U	I	H	I	S	T	E	X	T	D	E	N	Q	S	5	0.6
O	H	I	E	W	I	F	T	T	Y	O	I	N	G	6	0.4
N	G	G	C	P	E	D	R	A	F	E	O	A	N	5	0.6
C	E	I	S	N	N	O	S	R	S	C	D	E	O	5	0.6
S	P	R	T	I	O	W	R	T	O	O	A	L	P	5	0.6
V	A	L	E	T	I	E	X	L	V	H	A	A	T	6	0.4
A	A	B	C	E	E	C	S	O	N	E	R	F	E	7	1.4

⇒ The average of difference is 0.66

② 14×7 array

							Number of vowels	Difference
U	H	S	E	T	E	Q	3	0.2
O	I	W	F	T	O	N	3	0.2
N	G	P	D	A	E	A	3	0.2
C	I	N	O	R	C	E	3	0.2
S	R	I	W	T	O	L	2	0.8
V	L	T	E	L	H	A	2	0.8
A	B	E	C	O	E	F	4	1.2
I	I	T	X	D	N	S	2	0.8
H	E	I	T	Y	I	G	3	0.2
G	C	E	R	F	O	N	2	0.8
E	S	N	S	S	D	O	2	0.8
P	T	O	R	O	A	P	3	0.2
A	E	I	X	V	A	T	4	1.2
A	C	E	S	N	R	E	3	0.2

⇒ The average of difference is 0.56

$0.56 < 0.66$ It appears that the 14×7 rectangle is more likely.

⇒ we rearrange the 7 columns

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

⇒ 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