P.HARINI
SEC 7
2300033404

NPS LAB EXPERIMENT 12

```
def rail_fence_encrypt(plaintext, rails):
    fence = [''] * rails
    rail = 0
    direction = 1  # 1 for down, -1 for up

    for char in plaintext:
        fence[rail] += char
        rail += direction
        # Change direction when reaching top or bottom rail
        if rail == rails - 1 or rail == 0:
            direction *= -1
    return ''.join(fence)


def rail_fence_decrypt(ciphertext, rails):
    rail_lengths = [0] * rails
    rail = 0
    direction = 1

        for char in ciphertext:
        rail_lengths[rail] += 1
        rail += direction
        if rail == rails - 1 or rail == 0:
            direction *= -1

        fence = [''] * rails
    index = 0
    for i in range(rails):
        fence[i] = ciphertext[index:index + rail_lengths[i]]
        index += rail_lengths[i]
```

P.HARINI
SEC 7
2300033404

```
    result = []
    rail = 0
    direction = 1
    for _ in range(len(ciphertext)):
        result.append(fence[rail][0])
        fence[rail] = fence[rail][1:]
        rail += direction
        if rail == rails - 1 or rail == 0:
            direction *= -1

    return ''.join(result)



def columnar_encrypt(plaintext, keyword):
    num_cols = len(keyword)
    num_rows = len(plaintext) // num_cols + (1 if len(plaintext) % num_cols else 0)
    grid = [''] * num_cols



    for i, char in enumerate(plaintext):
        grid[i % num_cols] += char



    sorted_indices = sorted(range(len(keyword)), key=lambda i: keyword[i])


        ciphertext = ''.join(grid[i] for i in sorted_indices)
    return ciphertext



def columnar_decrypt(ciphertext, keyword):
    num_cols = len(keyword)
```

```
    num_rows = len(ciphertext) // num_cols


        column_lengths = [num_rows] * num_cols
    for i in range(len(ciphertext) % num_cols):
        column_lengths[i] += 1


        grid = [''] * num_cols
    index = 0
    sorted_indices = sorted(range(len(keyword)), key=lambda i: keyword[i])
    for i in sorted_indices:
        grid[i] = ciphertext[index:index + column_lengths[i]]
        index += column_lengths[i]


    result = []
    for i in range(num_rows):
        for j in range(num_cols):
            if i < len(grid[j]):
                result.append(grid[j][i])


    return ''.join(result)



if __name__ == "__main__":
    # Rail Fence Cipher
    plaintext_rf = "HELLO WORLD"
    rails = 3
    encrypted_rf = rail_fence_encrypt(plaintext_rf, rails)
    decrypted_rf = rail_fence_decrypt(encrypted_rf, rails)
    print(f"Rail Fence Encryption: {encrypted_rf}")
    print(f"Rail Fence Decryption: {decrypted_rf}")


    plaintext_ct = "HELLO WORLD"
```

```
keyword = "KEY"
encrypted_ct = columnar_encrypt(plaintext_ct, keyword)
decrypted_ct = columnar_decrypt(encrypted_ct, keyword)
print(f"Columnar Transposition Encryption: {encrypted_ct}")
print(f"Columnar Transposition Decryption: {decrypted_ct}")
```

- **Rail Fence Cipher**: Characters are arranged in a zigzag pattern across multiple "rails" and then read row-wise for encryption. For decryption, we recreate the zigzag structure to retrieve the original message.
- **Columnar Transposition Cipher**: Characters are filled into columns based on the keyword's length, and then columns are reordered based on the keyword for encryption. For decryption, we reconstruct columns and read row-wise.