DIT376 Python for Data Science
Assignment 4 – Individual Assignment!

There are two problems in this assignment.
To submit:

- A Jupyter notebook (.ipynb) with clearly marked solutions for: Problem 1 and Problem 2. The notebook must show examples that you used to run the codes, and the outputs that you get after running the codes.
- An html file (a result of "Save and Export Notebook As" html of your Jupyter notebook) after you run all the codes or a link to your notebook if you use a collaborative tool like Colab or Deepnote.

Do not forget to add your name at the very top of your notebook.
Note: If you add any details or make any assumptions, please clearly describe these in your submission.

1 Using generator

The task in this problem is to write an *encoder* and *decoder* for strings. The encoder compresses a non-digit string by replacing each consecutive sequence of the same letter by this letter and its frequency, for example, the string:

```
It's __sooooo__ wowww!!!
```

encoded as

```
I1t1'1s1 1_2s1o5_2 1w1o1w3!3
```

The decoder reverses the encoder process by transforming a compressed string, with a fix format containing a repetition of non-digit character followed by a positive integer, to its full representation. Given the following:

```
I1t1’1s1 1_2s1o5_2 1w1o1w3!3
```

The decoder will return:

```
It’s __sooooo__ wowww!!!
```

Solve this problem by using **generators**. Optional: Use regular expression for solving this task.

Example of usage 1:

```
s = "It’s __sooooo__ wowww!!!"
encoded = ’’.join(encode(s))
print(encoded)
decoded = ’’.join(decode(encoded))
print(decoded)
```

Output:

```
I1t1’1s1 1_2s1o5_2 1w1o1w3!3
It’s __sooooo__ wowww!!!
```

Example of usage 2:

```
s = "‘‘‘‘‘´´´´´´*******^^^^^"
encoded = ’’.join(encode(s))
print(encoded)
decoded = ’’.join(decode(encoded))
print(decoded)
```

Output:

```
‘5´6*7^5
‘‘‘‘‘´´´´´´*******^^^^^
```

2 Analysing Algorithms Complexity

You are in a recruitment process for a data science role in a small company, so your role may cover coding, analysis, presentation, etc. The company wants to test your ability to read someone else's codes. You are provided with two sorting algorithms (see code below), with no comments. You are asked to answer the following questions/tasks and explain this code while answering the questions/tasks. You can answer the questions/tasks in the order you want.

- Explain the connection between the two algorithms.
- What is the complexity of 'sort1' in the best case, in the worst case, and on average? Give details on your analysis to get to the answer.
- What is the complexity of 'sort2' in the best case, in the worst case, and on average? Give details on your analysis to get to the answer.
- Compare the two algorithms and summarise the advantages and disadvantages of using each of the algorithms.

```python
def merge(left, right):
    if len(left) == 0:
        return right
    elif len(right) == 0:
        return left
    else:
        result = []
        i = j = 0
        while len(result) < len(left) + len(right):
            if left[i] <= right[j]:
                result.append(left[i])
                i += 1
                if i == len(left):
                    result += right[j:]
                    break
            else:
                result.append(right[j])
                j += 1
                if j == len(right):
                    result += left[i:]
                    break
        return result


```

```python
def sort1(L, start=0, end=None):
    if end is None:
        end = len(L)-1

    for i in range(start+1, end+1):
        key = L[i]
        j = i-1
        while j >= start and L[j] > key:
            L[j+1] = L[j]
            j -= 1
        L[j+1] = key
    return L

def sort2(L):
    n = len(L)
    slice_size = 32

    for i in range(0, n, slice_size):
        sort1(L, i, min((i + slice_size - 1), n-1))

    size = slice_size
    while size < n:
        for start in range(0, n, size*2):
            middle_point = start + size - 1
            end = min((start + size*2 - 1),(n-1))
            merged_list = merge(
                left = L[start:middle_point+1],
                right = L[middle_point+1:end+1])
            L[start:start+len(merged_list)] = merged_list
        size *= 2
    return L
```