Assignment for Class Computer Science for Big Data D1

Basic Search, Points: 7

1.) Write an elegant, recursive Python function that tests whether a list is sorted (numbers in <= seq.):

```
def is_sorted(_list: []) -> bool:
```

2.) What is the "time-complexity" of your function (how many comparisons are needed for *n*-elements in the input list):

a. for the best case (minimal number of comparisons needed),

b. the worst case (maximal number of comparisons) and

c. on average? [1 Pts]

3.) Test your function: [2 Pts]

```
numbers = [
    [1, 2, 3, 0, 4, 5, 6, 7, 8, 9],
    [4, 8, 12, 12, 24, 36],
    [], [2], [1, 2], [2, 1]
]

for n in numbers:
    print(str(is_sorted(n)) + '\t<-- ' + str(n))

==>
False   <-- [1, 2, 3, 0, 4, 5, 6, 7, 8, 9]
True    <-- [4, 8, 12, 12, 24, 36]
True    <-- []
True    <-- [2]
True    <-- [1, 2]
False   <-- [2, 1]
```

4.) Does your implementation continue, or does it stop when the first pair of out-of-order numbers is found? Add a log-feature to your function that is controlled by an optional argument `_log` that prints each comparison operation when enabled: [1 Pts]

```
def is_sorted(_list: [], _log: bool = False) -> bool:
   ...

n = numbers[0]       # [1, 2, 3, 0, 4, 5, 6, 7, 8, 9]

print(str(is_sorted(n, True)) + '\t<-- ' + str(n))

==>
1 < 2 -> True
2 < 3 -> True
3 < 0 -> False       # stop
False   <-- [1, 2, 3, 0, 4, 5, 6, 7, 8, 9]
```

5.) Site https://namecensus.com/most_common_surnames.htm contains data with name statistics from the U.S. You can download the list of the 1,000 most common surnames from:

https://github.com/sgra64/cs4bigdata/blob/main/C1_us_name_stats.py .

The file contains variables with 1,000 names, the number of occurrences of names and name frequency, all ordered by a rank as shown on the namecensus page. Import data from C1_us_name_stats .py :

```
import C1_us_name_stats as ns
```

6.) Write an (own) Python function that finds a name in `us_names_top1000[]` and returns the index of the name in the list or -1 if the name was not found. The index is used to access data in the other lists.

```
def find_name(_name: str, _list: []) -> int:
```

Test the function: [2 Pts]

```
name_list = ns.us_names_top1000
name_freq = ns.us_name_freq_top1000
names = ['Smith', 'Mendoza', 'Rodriguez', 'Abbott', 'Blokes', 'Brewer',
'Vang', 'Zimmerman', 'Bailey']

for n in names:
    i = find_name(n, name_list)
    print(str(i) + '\t<-- ' + str(n), end='')
    print('\t\t--> freq: ' + str(name_freq[i]) if i >= 0 else "")

==>
0       <-- Smith               --> freq: 1.006
245     <-- Mendoza             --> freq: 0.043
21      <-- Rodriguez           --> freq: 0.229
462     <-- Abbott              --> freq: 0.025
-1      <-- Blokes
250     <-- Brewer              --> freq: 0.042
999     <-- Vang                --> freq: 0.012
440     <-- Zimmerman           --> freq: 0.026
59      <-- Bailey              --> freq: 0.115
```

7.) What is the time-complexity of your function (how many comparisons are needed for *n*-elements in the input list of names):

a. for the best case (minimal number of comparisons needed for a list with *n*-elements),

b. the worst case (maximal number of comparisons) and

c. on average? [1 Pts]

The word "index" is normally used to access data in a list or in an array as their offset from 0. The word index is also used to refer to an intermediary data structure through which underlying data can be accessed faster than through linear search as in our previous cases.

Rather than searching through an array of strings, string search can be significantly accelerated by an index in which strings are ordered.

Cost of an index is to build the extra data structure (cost in terms of run time and space needed). If data is immutable (as in our case of names), the index can be built once and then speed up each lookup. If underlying data changes (add, update, delete), the index must be adjusted accordingly.