



# ASSIGNMENT D1

Software Setup for CS4DS

**Submitted By: Ahmed Dider Rahat**

Matriculation Number: 916146

**Assignment Code:** After implementing all the assignment I pushed my code on my [Github](#) link.

### **Answer to the question no. 1**

I have written the implementation of `'def is_sorted(_list: []) -> bool:'` function in the file name Question1.py.

### **Answer to the question no. 2**

I have implement the code using recursion. So, for the recursion process I consider the following assumptions.

1. **Base Conditions:** I assum base conditions are:
  - a. If list is empty then list is sorted.
  - b. If list has one element then it is sorted.
  - c. If list has exactly two elements then:
    - i. If first element is less then two then list is sorted
    - ii. Otherwise list is not sorted.
2. **Recursive call:** Whenever I found the list has elements more than two I check if first two elements are sorted or not. If sorted then call the same method but except the first elements. Otherwise return false to identify the list is not sorted.

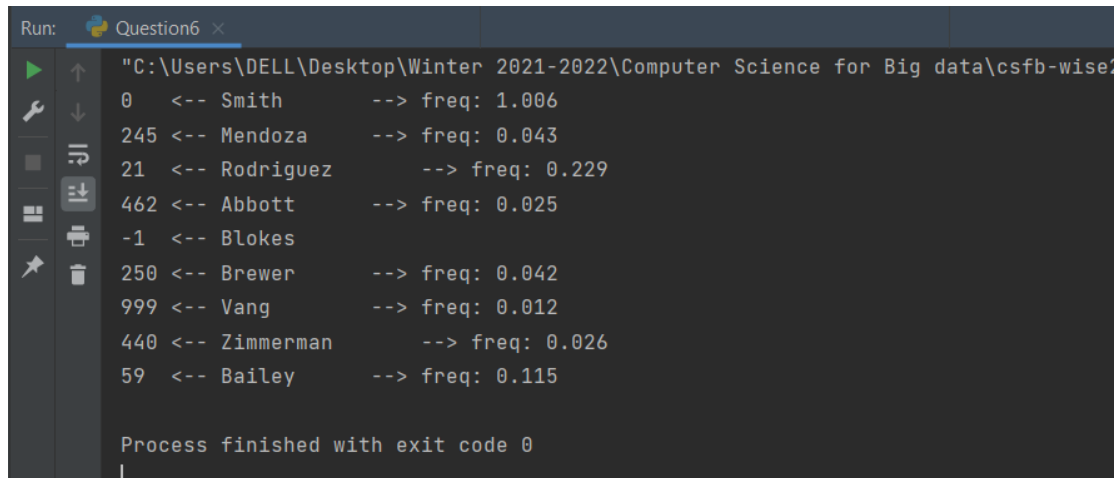
For calculating the time complexity we have to see the recursive call of the function. It is clear that if the list is empty, or has one element, or has only two element we just need to call the function once. And as the function could not contains any loop inside it, the time complexity become  $O(1)$ . Now in case the function contains  $n$  element and all the elements from index 0 to  $n-2$  become sorted but the last element or  $(n-1)$  element is less then the element  $(n-2)$  then the worst case happened and that time we nneed to call the function  $n-1$  times and each time need  $O(1)$  times. So, the final complexity arise to  $O(n)$ .

So, the final complexity is like:

1. Best case:  $O(1)$
2. Worst case:  $O(n)$
3. Avarage case:  $O(n/2)$

### Answer to the question no. 3

**Test the Function:** After testing the function we got the following output:

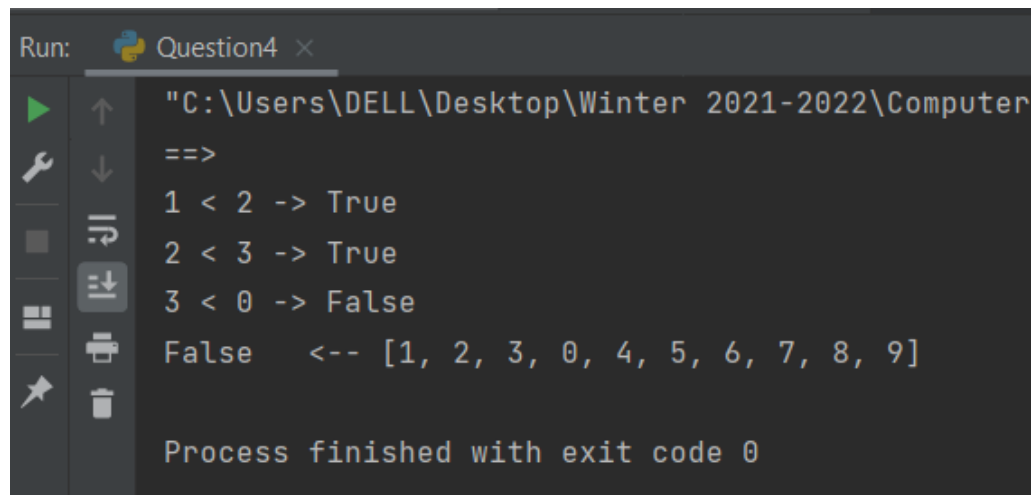


```
Run: Question6 x
"C:\Users\DELL\Desktop\Winter 2021-2022\Computer Science for Big data\csfb-wise2
0 <-- Smith --> freq: 1.006
245 <-- Mendoza --> freq: 0.043
21 <-- Rodriguez --> freq: 0.229
462 <-- Abbott --> freq: 0.025
-1 <-- Blikes
250 <-- Brewer --> freq: 0.042
999 <-- Vang --> freq: 0.012
440 <-- Zimmerman --> freq: 0.026
59 <-- Bailey --> freq: 0.115

Process finished with exit code 0
```

### Answer to the question no. 4

After adding the `_log` feature, I found the recursive call immediately stop when the first unsorted scenario found. The output of the log function given below:



```
Run: Question4 x
"C:\Users\DELL\Desktop\Winter 2021-2022\Computer
==>
1 < 2 -> True
2 < 3 -> True
3 < 0 -> False
False <-- [1, 2, 3, 0, 4, 5, 6, 7, 8, 9]

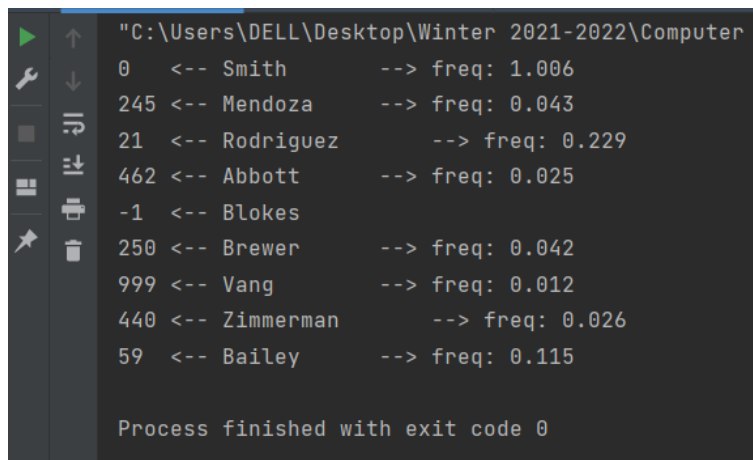
Process finished with exit code 0
```

### **Answer to the question no. 5**

I have imported the given 'name dataset' to my code base and implement my method for searching the name in the name list. The implementation file is Question6.py.

### **Answer to the question no. 6**

The test function output is given bellow:



```
"C:\Users\DELL\Desktop\Winter 2021-2022\Computer S
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

Process finished with exit code 0
```

### **Answer to the question no. 7**

For name searching problem I used python filter function to get the index of a given name. So, the time-complexity of the function is same as the filter function.

Best case:  $O(1)$

Worst case:  $O(n)$

Average case:  $O(n/2)$