Question 2

Code:

```
import sys
sys.setrecursionlimit(20000)
def func1(arr, low, high):
    if low < high:
        pi = func2(arr, low, high)
        func1(arr, low, pi-1)
        func1(arr, pi + 1, high)
def func2(array, start, end):
    p = array[start]
    low = start + 1
    high = end
    while True:
        while low <= high and array[high] >= p:
            high = high - 1
        while low <= high and array[low] <= p:
            low = low + 1
        if low <= high:
            array[low], array[high] = array[high], array[low]
        else:
            break
    array[start], array[high] = array[high], array[start]
    return high
```

Questions:

1. Explain what the code does and perform an average-case complexity analysis. Describe the process, not just the result. (2 pts)
2. Test the code on all the inputs at: https://raw.githubusercontent.com/ldklab/ensf338w23/main/assignme nts/assignment2/ex2.json Plot timing results. Provide your timing/plotting code as ex2.2.py. (2 pts)
3. Compare the timing results with the result of the complexity analysis. Is the result consistent? Why? (2 pts)
4. Change the code – if possible – to improve its performance on the input given in point 2. If possible, provide your code as ex2.4.py and plot the improved results. If not possible, explain why. (2 pts)
5. Alter the inputs given in point 2 – if possible - to improve the performance of the code given in the text of the question. The new inputs should contain all the elements of the old inputs, and nothing more. Plot the results and provide the new inputs as ex2.5.json). If not possible, explain why. (2 pts)

Answers:

1.

This code uses the quick sort method to sort arrays. Quick sort works by recursively subdividing the array and ordering each subdivision. The arrays are sorted around the pivot point, which in this case is the first element of the array. The subdivision will continue until the length of all sub-arrays have a single element.

Average Case Complexity Analysis:

Step 1 It is necessary to establish a hypothetical average partition, that is one that is not always balanced. In this case, assume that the partitioning will never be worse than a 3-to-1 split, so for every partition, one subdivision will receive n/4 elements and the other will get 3n/4.

Step 2 Now, the total partioning time for each subdivision will never exceed cn. (For example, one side will compare n/4 elements, the other will compare 3n/4 elements, resulting in n compared elements.)

Step 3 The node of the partition which receives n/4 elements each time will have log_4(n) layers. This is the shortest path to a subdivision of size one, and so every layer beyond this layer will have a partitioning time < cn.

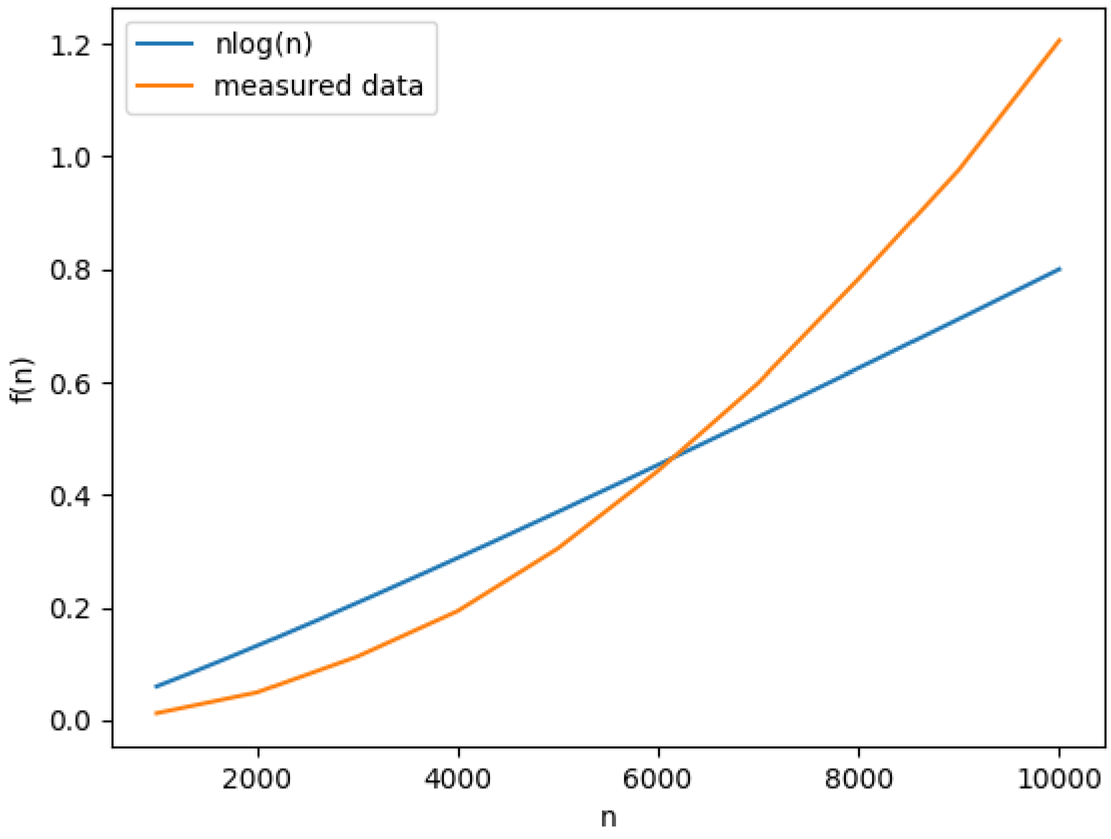
Step 4 It is now necessary to calculate the layers of the longest path to a subdivision size of one. This will be the path that receives 3n/4 elements each partition. In this case, we will reach log_4/3(n) levels.

Step 5 Now, combining the partitioning time of each level with the total number of levels, we have O(nlog_3/4(n)). We can use the fact that log_4/3(n) = log_10(n) / log_10(4/3). Furthermore, since log_10(4/3) is a constant, we can disregard it, so we are left with O(nlog(n)) as an average case complexity.

2.

ex2.2.py is located within the github repo

Plot:



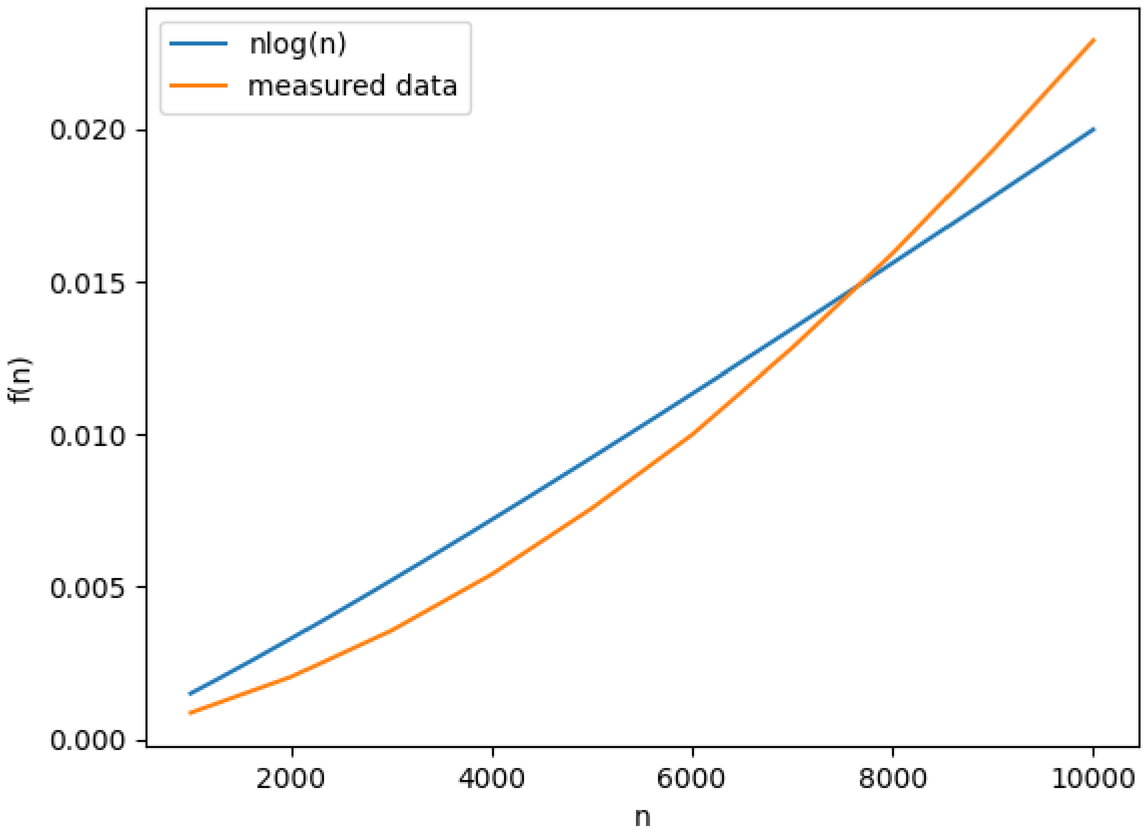
3.

From the plot, it is evident that the results are worse than the average-case complexity for large n. This is likely due to an ineffecient pivot point based on how the input data is formatted. The input data is already almost sorted, and therefore having a pivot point of the first element in the array will always result in a lopsided partition.

4.

ex2.4.py is located within the github repo The code was changed to set the pivot point to the halfway point of the array, as the given input data already seemed roughly sorted, and the quicksort algorithm is most efficient when there is an even partition between the subarrays, therefore a halfway pivot point would produce a closer-to-even partition as opposed to a start pivot point.

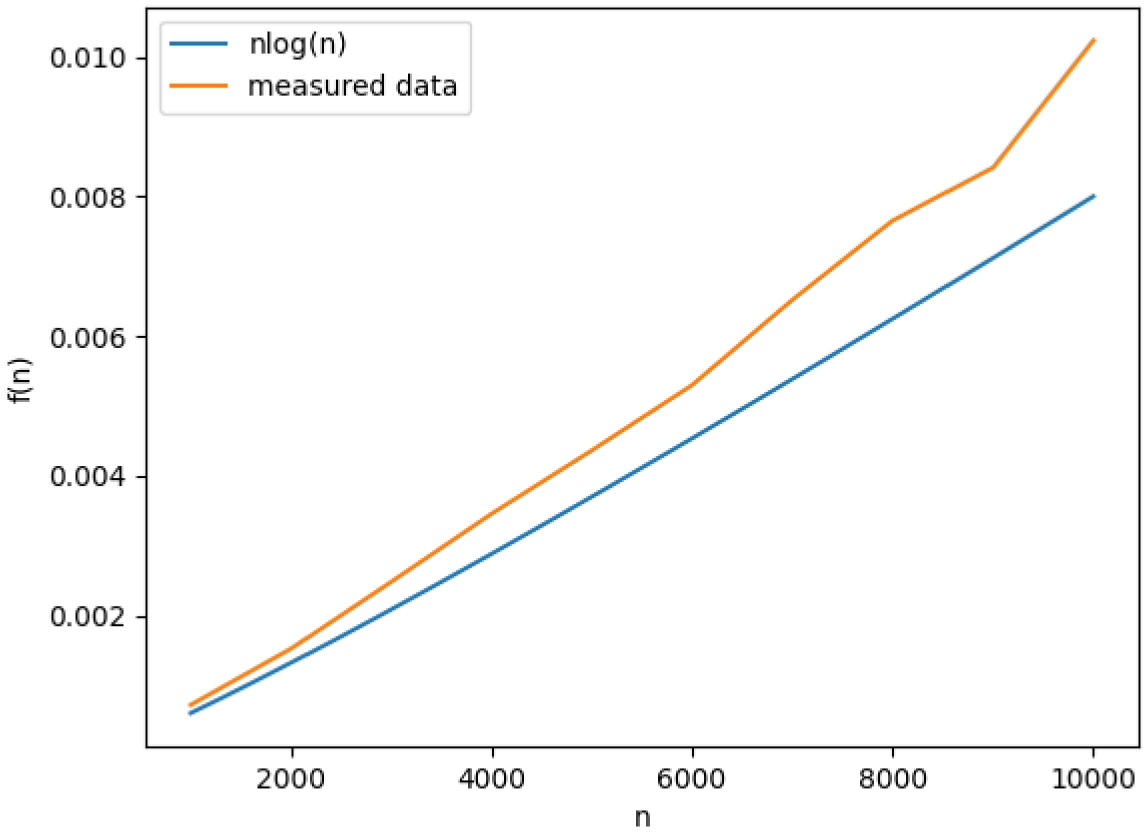
Plot:



5.

ex2.5.json is located within the repo. The data was altered by shuffling all of the elements in the input data. This results in a completely unordered set of data, as opposed to an almost ordered set. This unordered data will work more efficiently with the chosen pivot point.

Plot:


nlog(n)
measured data
0.010
0.008
0.006
0.004
0.002
f(n)
2000
4000
6000
8000
10000
n