Introduction to CS for Data Science

HW2 – Recursion & Complexity

Recursion and Time Complexity

In this exercise, you will not be provided with premade scripts. For the time complexity analysis, write your answers in a .pdf file zipped with your python files.

In conducting running time analysis, you are not expected to produce a rigorous proof. You are expected to provide a reasonable explanation like the recitations (i.e., explain how many iterations and why, how many operations per iteration and why, and express using big-O notation).

For the following coding problems, use recursion. You are free to define your recursive functions as you see fit (you may define helper functions). Make sure you test your code on interesting cases and make sure it works well. You are encouraged to write tests.

1. Consider the following function that converts and int into a string and analyze its time complexity. The function assumes *i* is a nonnegative int and returns the string representation of *i*.

```
def int_to_str(i):
    digits = '0123456789'
    if i == 0:
        return '0'
    result = ''
    while i > 0:
        result = digits[i % 10] + result
        i = i // 10
    return result
```

2. The power set of any set S is the set of all subsets of S, including the empty set and S itself. Write a recursive function that takes as input a list (you can assume the list does not have repeats of any element), and returns its power set, and analyze your algorithm's time complexity. Put your answer inside a file named Q2.py.

For example, power_set([1,2,3]) will return [[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]].

3. Write a recursive function that takes as input a non-negative integer n and returns the nth row of Pascal's triangle (assuming that the counting of rows starts at zero). For example, pascal(0) will return [1], pascal(3) will return [1,3,3,1], pascal(4) will return [1,4,6,4,1]. Put your answer inside a file named Q3.py.

4. For the following pairs of functions, state the big O relationship between them and prove it.

   Note that it is possible to have $f = O(g)$ and $g = O(F)$ at the same time.

   a. $5n^2 + 1, 10n^2 + 30$
   b. $\log_3 n, \log^3 n$
   c. $8^{\log n}, \ 3n^3 + 2$
   d. $n^n, 2^n$

5. In this Question we will implement and analyze the insertion sort algorithm.

   a. Implement the Insertion sort algorithm that you have seen in class with the following two steps:
      i. Implement a function to insert an item to a specific positions in a list pushing all other elements forward.
      ii. Use this function to implement insertion sort algorithm.

      Put your implementation inside a file named Q5.py.

   b. Analyze Insertion sort's time complexity.
   c. Design an experiment to test your complexity analysis.
      Generate random inputs of a various sizes and measure the running time of your algorithm (you can use "timeit" or the "time" module for this). calculate average time for every input size. print input size vs. running time. Use Excel or any other tool to plot the result. Include your experiment code in the Q5.py file.

6. **Read the following submission guidelines carefully:**

   a. This exercise is to be submitted individually.

b. All submissions must be zipped and submitted as an archive file with your ID as a filename. For example, compress all relevant files using zip and name it ‘123456789.zip’.

Good luck.