Maxwell Richard Tamer-Mahoney ID #: 201804029

CMPS 200 – Assignment 8

1. Express in O notation the worst case asymptotic complexity of the solution you wrote and a short one or two-sentence justification of your answer.
   1. Palindrome, 4.1

O(n), because in the worst case it will check over all of the characters, which means there are n iterations of the while loop. My solution isn’t the best because you could half the number of required iterations.

* 1. longest substring, 4.2

Assuming that the python max() function has a better algorithm than looping over all of the elements, O(n) because the string only must be looped over once constructing all the alphabetical substrings (in my algorithm). Otherwise, O(2n), which is still O(n) if we disregard the coefficient because it is relatively insignificant.

* 1. guess the number, 4.3

Binary search (which is used in this program) is logarithmic in complexity, as each time the search range is halved until it reaches 0, so O(log n).

* 1. smallest positive, 4.5

Assuming that the python min() function has a better algorithm than looping over all of the elements, O(n) because the list comprehension I used loops over the elements once. Otherwise, O(2n), which is still O(n) if we disregard the coefficient because it is relatively insignificant.

* 1. Shuffle, 4.8

O(n) because the elements are looped over once to switch the position of each to a random integer position.

* 1. Pythagorean triplets, 5.4

O(n^3) because of the 3 nested for loops used to generate possible triplets below a given number.

* 1. longest palindrome, 5.5

O(n^2) because of the nested looping done in the list comprehension I used.

* 1. palindrome recursive version, 5.7

O(n) because the cutting of the first and last characters of the string each iteration result in n / 2 iterations.

* 1. Snowflake, 5.8

O(4^n) because each fractal call spawns 4 calls which spawn another 4, and so on.

* 1. towers of hanoi, 6.2

O(2^n) because moving a tower of height n requires 2^n-1 moves. The non-base case only has 2 calls that can spawn child calls, thus it is 2^n not 3^n.

* 1. tree, 6.3

O(3^n) because each tree call spawns 3 more that spawn another 3, etc.

* 1. Pascal’s triangle, 6.5

O(n^2) because there is a for loop nested within another to generate the triangle.

1. The asymptotic complexity of the first calculation method is O(n^2). The asymptotic complexity of Horner’s evaluation method, however, is O(n).
2. The complexity of my algorithm is O(2^n).
3. The worst case complexity of insertion sort is O(n^2). The best case complexity is O(n).
4. Results for different n (sorting unique lists):

2,500: 0.00712 sec

5,000: 0.01675 sec

10,000: 0.0340 sec

Results for different n (list already sorted):

2,500: 0.00246 sec

5,000: 0.00617 sec

10,000: 0.01635 sec

The time for sorting increases about 2x each time. In the worst case it would increase by 4x from 2,500 to 5,000 and 5,000 to 10,000, but this isn’t the worst case. The already sorted list is much faster, as expected, because this is the best case, and is only O(n) complexity instead of O(n^2).

1. When n is 1,000,000, the linear search took on average 0.0624 sec to find the integer. Using binary search, as expected, the time is much shorter, only 0.000011 sec. However, sorting the list took 0.2864 sec. When n is 10,000,000, the linear running time should increase 10x, and the binary running time should increase log(10,000,000) / log(1,000,000), which is 1.16x.

When n is 10,000,000, the linear search average running time was 0.6597 sec and the binary search average running time was 0.000017 sec.

1. The recurrence equation describing the asymptotic complexity of the merge sort algorithm is

whose solution is O(n log n).