Title: Data Structures and Algorithms - CST3108
Lab 2: Asymptotic Complexity of an Algorithm
Name: Chris Mugabo
Date: 22/01/2025

Lab 2: Asymptotic Complexity of an Algorithm

Task 1:

Answer the following questions regarding the time complexity of various expressions and visualize their growth.

a. Is (n+5)^2 = O(n log n)?

No, (n+5)^2 expands to n^2 + 10n + 25, which is primarily dominated by n^2. Quadratic growth (n^2) is much faster than n log n, making (n+5)^2 not fit within O(n log n).

b. Is n^(14/16) = O(n log n)?

No, n^(14/16) simplifies to n^0.875. Since any polynomial growth n^k where k > 0 grows faster than n log n, n^0.875, being a fractional polynomial, does not fit within O(n log n).

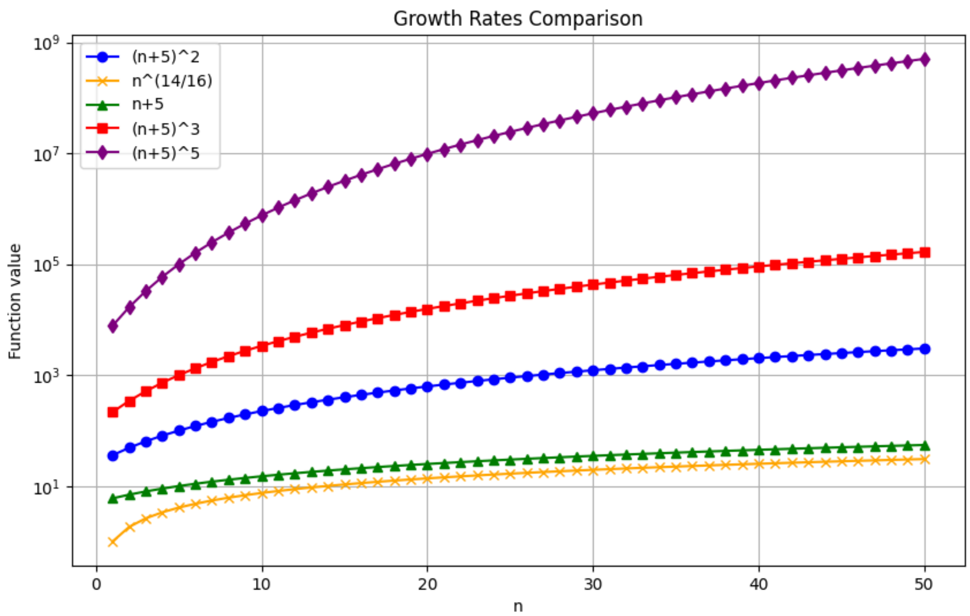
c. What is the time complexity of n+5, (n+5)^3, and (n+5)^5?

The time complexity of n+5 is O(n), reflecting linear growth. For (n+5)^3 and (n+5)^5, the complexities are O(n^3) and O(n^5), respectively, indicating polynomial growth rates.

d. Code for Visualizing Growth Rates

Python code is provided to plot the growth rates of (n+5)^2, n^(14/16), n+5, (n+5)^3, and (n+5)^5 on the same graph, demonstrating their behavior as n increases from 1 to 50.

```
import matplotlib.pyplot as plt
import numpy as np
n = np.arange(1, 51)
plt.figure(figsize=(10, 6))
plt.plot(n, (n+5)**2, label='(n+5)^2')
plt.plot(n, n**(14/16), label='n^(14/16)')
plt.plot(n, n+5, label='n+5')
plt.plot(n, (n+5)**3, label='(n+5)^3')
plt.plot(n, (n+5)^5, label='(n+5)^5')
plt.yscale('log')
plt.xlabel('n')
plt.ylabel('Function value')
plt.title('Growth Rates Comparison')
plt.legend()
plt.grid(True)
plt.show()
```



Task 2:

Analyze the given pseudocode to determine the number of times 'Ping' is executed and compute the time complexities.

Pseudocode 1:

```
for j in 0 . . . n do
   for k in 0 . . . j + 1 do
      Ping
```

Analysis: The outer loop iterates from 0 to n, inclusive, resulting in n+1 iterations. For each iteration of j, the inner loop runs from 0 to j+1, inclusive, resulting in j+2 iterations of 'Ping' for each j.

Calculation of Total 'Ping' Executions:

Total Executions = 2 (when j=0) + 3 (when j=1) + 4 (when j=2) + ... + (n+2) (when j=n).

This is an arithmetic series with the first term 2 and the last term n+2, over n+1 terms.

Sum of the series = (n+1)(2 + (n+2))/2 = (n+1)(n+4)/2.

Time Complexity: O(n^2).

Pseudocode 2:

```
i ← 1
while i ≤ n do
  Ping
  j ← n
  while j > i do
    Ping
    j––
  i++
```

Analysis: The outer loop runs n times, with one execution of 'Ping' per iteration. For each i, the inner loop runs n-i times, decrementing j from n to i+1.

Total Executions = n (from outer loop) + (n-1) + (n-2) + ... + 1 (from inner loop).

Sum of the series = n + n(n-1)/2 = n(n+1)/2.

Time Complexity: O(n^2).

Pseudocode 3:

```
Ping
i ← 1
while i ≤ n do
  Ping
  j ← 1
  while j ≤ i × i do
    Ping
    j++
  i++
```

Analysis: Starts with a single 'Ping'. The outer loop runs n times, executing 'Ping' once per iteration. For each i, the inner loop runs i^2 times.

Total Executions = 1 (initial) + n (outer loop) + 1^2 + 2^2 + ... + n^2 (inner loop).

Sum of squares = 1 + n + n(n+1)(2n+1)/6.

Time Complexity: O(n^3).

Comparison of Pseudocodes:

Pseudocode 1 and 2 both exhibit quadratic time complexities (O(n^2)), suitable for medium-sized datasets. Pseudocode 3, with its cubic time complexity (O(n^3)), is less efficient for larger datasets due to the exponential increase in executions. This makes Pseudocode 1 and 2 preferable for larger inputs where performance is a critical factor.