ASSIGNMENT 1

Q2.3.1, Q2.3.2, Q2.3.3, Q2.4.1, Q2.4.2, Q2.4.3

Section 2: Time and Space Complexity Analysis

2.3 Time Complexity Analysis Questions

Problem 1

Question:

For the given code snippet, find the time complexity in terms of Big O (O), Theta (Θ), and Omega (Ω). Provide a step-by-step explanation of how you arrived at the time complexity.

Answer:

- Analyze the nested loops. The outer loop runs `n` times, and for each iteration of the outer loop, the inner loop also runs `n` times. This results in `n * n = n^2` operations for adjusting `a`.

- The second loop (for `k`) is independent and runs `n` times for adjusting `b`.

Time Complexity:

-**Big O (O)**: The worst-case scenario is O(n^2) due to the nested loops.

-**Theta (Θ)**: Since the dominant term in the total operations is n^2 and there are no scenarios where the algorithm would perform better or worse than this rate of growth, the average-case complexity is also Θ(n^2).

-**Omega (Ω)**: The best-case scenario, which is also Ω(n^2), aligns with the worst-case since the loops will always perform n^2 operations for adjusting "a plus n" operations for adjusting 'b', but n^2 is the dominant term.

Problem 2

Question:

Determine the time complexity for the following function:

Answer:

- Analysis:

  - The first loop runs approximately n/2 times.

  - The second loop is a geometric progression starting from 1 and multiplying by 2 each time until `n`. This runs in O(log n) time.

  - The third loop, identical to the second loop, also runs in O(log n) time.

- **Overall Time Complexity:**

  -**Big O (O):** Considering the combination of loops, the worst-case time complexity is O(n log^2 n) because the outer loop runs n/2 times, and each of the two inner loops runs log n times.

  -**Theta (Θ):** The average-case complexity, considering the nature of loops and their execution path, remains Θ(n log^2 n).

  - **Omega (Ω)**: The best-case scenario, given the lower bounds of the loops, is Ω(n log^2 n).

Problem 3

Question:

Evaluate the time complexity for this recursive function:

Answer:

- Analysis:

  - This function makes two recursive calls for each value of 'a' greater than 0, effectively doubling the number of calls with each decrement of 'a'.

-**Overall Time Complexity:**

-**Big O (O):**The time complexity is O(2^n) because each function call spawns two additional calls.

  -**Theta (Θ)**: The average-case complexity mirrors the worst-case, so it is also Θ(2^n).

 - **Omega (Ω):** The best-case scenario, where `a <= 0`, executes in constant time, Ω(1), but considering the growth rate as `a` increases, the lower bound in terms of growth behavior is Ω(2^n) for `a > 0`.

2.4 Space Complexity Analysis Questions

Problem 1

Question:

For the given code snippet, analyze the space complexity in terms of Big O (O), Theta (Θ), and Omega (Ω). Present a step-by-step explanation of your analysis.

- **Analysis:**

  - This function performs an in-place reversal of an array. The variables `start`, `end`, and `temp` are used for index tracking and swapping elements.

- **Space Complexity:**

  -**Big O (O):** The space complexity is `O(1)` as the memory usage does not scale with the size of the input array. Only a constant amount of extra space (`start`, `end`, `temp`) is used regardless of the input size.

  -**Theta (Θ):**Since the algorithm consistently uses a fixed amount of space, the average-case space complexity is also `Θ(1)`.

- **Omega (Ω):** The best-case scenario also has a space complexity of `Ω(1)`, reflecting that even in the most efficient case, the algorithm requires a constant amount of space.

Problem 2

Question:

Analyze the space complexity of the following function:

- **Analysis:**

  - This function calculates the nth Fibonacci number using dynamic programming. The `dp` array of size `n + 1` stores intermediate Fibonacci values.

-**Space Complexity:**

  - **Big O (O):**The space complexity is `O(n)` as the algorithm allocates an array of size `n + 1` to store the Fibonacci sequence up to `n`.

  -**Theta (Θ):** Given the allocation of memory scales linearly with `n`, the average-case space complexity is `Θ(n)`.

  - **Omega (Ω):**The best-case space complexity is also `Ω(n)`, indicating that the minimum space required by the algorithm grows linearly with the input size.

Problem 3

Question:

Evaluate the space complexity for this function:

-Analysis:

- This function computes the length of the longest common subsequence between two strings `s1` and `s2` using dynamic programming. The `dp` 2D array of size `(m + 1) x (n + 1)` is used to store the lengths of the longest common subsequences for all subproblems.

- **Space Complexity:**

-**Big O (O):** The space complexity is `O(mn)` due to the allocation of a 2D array whose size is directly proportional to the lengths of the input strings `s1` and `s2`.

-**Theta (Θ)**: The average-case space complexity is `Θ(mn)` as the space used by the algorithm scales with the product of the lengths of the two input strings.

-**Omega (Ω)**: The best-case scenario still requires a space complexity of `Ω(mn)` for the dynamic programming table.