Kowshik Ahmed Tiash

s357478

Assessment 1

HIT220 ALGORITHMS AND COMPLEXITY

**Question 1. Ans:**

Let's consider an example where encapsulation is used in a banking program:  
Encapsulation would be employed in the creation of a "Customer" class in a banking application. The class would include getter methods to access and setter methods to modify the customer's private attributes, which would include things like name, address, and contact information. As a result, the sensitive information of the customer is safeguarded and can only be accessed or changed through defined and regulated interfaces offered by the class.

**Question 2.(1) Ans:** Calculate the time it takes for the computer to generate all permutations of a set with 6  
distinct elements at a rate of 1500 permutations per second:  
Time (seconds) = Total permutations / Permutations per second  
Time (seconds) = 6! / 1500 ≈ 0.48 seconds  
It will take approximately 0.48 seconds.

**Question 2.(2) Ans:**def Arithmetic(n):  
a=0  
for i in range(1, n):  
for j in range(1, (i + 1) // 3):  
a += n \* i \* j  
return a

**Time Complexity**: O(n) - The function has a linear time complexity since the outer loop runs n-1  
times and the inner loop runs at most n // 3 times.**Space Complexity**: O(1) - The space complexity is constant as the function uses only a fixed  
amount of memory regardless of the input size n.

**Question 3.(1) Ans:** Here's a pseudo-code algorithm for finding the average value in a sequence P of n  
integers:  
Algorithm: FindAverageValue  
Input: Sequence P of n integers  
1. Initialize a variable sum to 0.  
2. For each element num in the sequence P:  
2.1. Add num to the sum.  
3. Compute the average by dividing the sum by n and store it in a variable avg.  
4. Return avg as the average value of the sequence P.

**Question 3.(2) Ans:**Time Complexity: O(n)  
Space Complexity: O (1)

**Question 4.(1) Ans:**def generate\_binary\_numbers(n):  
if n == 0:  
return ['']  
if n % 2 != 0:  
raise ValueError("n must be a power of 2.")

prev\_numbers = generate\_binary\_numbers(n // 2)  
results = [num + bit for num in prev\_numbers for bit in ['0', '1']]  
return results  
# Example usage:  
n=4  
binary\_numbers = generate\_binary\_numbers(n)  
print(binary\_numbers)

**Question 4.(2) Ans:** Time and space complexity:  
**Time Complexity**: O(n) - The function has a linear time complexity as it generates binary numbers of length n by combining the results of n/2 binary numbers.  
**Space Complexity**: O(2^n) - The function generates and returns 2^n binary numbers of length n.  
Therefore, the space complexity is O(2^n).

**Question 5. Ans:** Push and pop operations on the Stack, as well as enqueue and dequeue operations on the Queue, must be O(1) in any data structure implementation to ensure efficient constant-time access to the top (Stack) or front (Queue) elements. This allows maintaining their core behaviors, such as LIFO for Stack and FIFO for Queue, regardless of the data size.

**Question 6. (1) Ans:** Here's a simple pseudo code for calculating the Greatest Common Divisor (GCD) of two numbers using the Euclidean algorithm:  
function gcd(a, b):

while b != 0: remainder = a % b a=b  
b = remainder

return a

**Question 6. (2) Ans:** The time complexity of the GCD calculation using the Euclidean algorithm is O(log(min(a, b))). The algorithm is effective and has a logarithmic time complexity. The size of one of the two input values (a or b) is reduced by roughly half during each iteration of the while loop, which is the cause of the time complexity. Since the algorithm divides the larger number by the smaller number continuously while updating the data, the reduction occurs quickly and the number of iterations is logarithmically enormous.

**Question 6. (3) Ans:** Due to the Euclidean method, the result of the given code is the two numbers 'a' and 'b''s greatest common divisor (GCD). The algorithm calculates the remaining of a%b and updates 'a' to 'b' and 'b' to the remainder for each iteration. This procedure is carried out until 'b' equals zero. As the highest number that divides both 'a' and 'b' without leaving any residue, 'a' now holds the GCD. The final output of the function is the GCD of variables 'a' and 'b', which is precisely determined by the code.