1. An important feature of Object Oriented Programming is using codes as objects. Describe an example of how objects are used in Python programming. • Please note that we did not ask for a code. We require a description of how objects are used, not the code itself.

An object is a program component that performs actions and interacts with other elements in object-oriented programming (OOP). For example, a class named 'student' can have attributes like name, ID, grade, and age, and functions like student_info and student_grade to perform actions. An object of the 'student' class represents a specific student with its details.

2.1. A program generates all the combinations of elements of a set and writes the output to a file at a rate of 1100 combinations per second. How long will it take to generate all the combinations of a set with 4 distinct elements?

According to the law of set, for each n elements the total subset of that n elements is 2^n

- Distinct elements(n) is 4, number of subsets(2^n) = 2^4 =16

- rate = 1100

Time = Total number of combinations / Rate

= 16 /1100

= 0.0145

So, it will take approximately 0.0145 seconds to generate all the combinations of a set with 4 distinct elements.

2.2. What is the Big-O time complexity of the given arithmetic function? Show working.

```
def Arithmetic(n): a = 0
 for i in range(1, n//2):
 for j in range(1, i//4):
 a += n** i − j
return a
```

This arithmetic function includes O(n) times by an outer loop and inner loop compiles O(i) times for loop iteration between 1 to n/2. The Big-O time complexity is $O(n^2)$ with increasing quadratically with the input size.

3.1 Write pseudocode or code for an algorithm to find the product of a sequence, P, of n integers.

Pseudocode:

```
Function ProductOfSequence(sequence):
    Initialize p to 1
    For each n in sequence:
        p = p * n
        Return p
```

The code initializes the product to 1, loops continue each integer variable in the sequence [1, 2, 3, 4, 5] and multiply then returns the final product of all integers after the loop is completed. The result is 120 that is the product of all the numbers in the sequence.

3.2 What are the time complexity and space complexity of your algorithm? Note that this will depend on your code.

Time Complexity: O(n), the algorithm goes through each number in the list once.

Space Complexity: O(1), the algorithm only uses one variable, no matter how many numbers are in the list.

4.1 Write a recursive function to generate the nth Fibonacci number

A recursive function solves problems by calling itself with a modified argument.

Recursive Function:

```
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
```

```
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

The recursive function fibonacci(n) computes the nth Fibonacci number by breaking down the problem into smaller subproblems, demonstrating the fundamental concept of recursion. If n=10, the sequence 0 to 10 is calculated, and the result is 55.

4.2 What are the time complexity and space complexity of this function?

Time Complexity: $O(2^n)$, the time it takes to run the function increases very quickly as the input number gets larger because it calls itself many times.

Space Complexity: O(n), the amount of space or memory needed depends on how deep the function calls go, which is equal to the input number.

4.3 Is this method more efficient than the iterative method?

Compared to the iterative method, recursion takes more time when dealing with Fibonacci numbers. It calculates each number once, using a fixed amount of memory and a linear time complexity. However, recursive method is inefficient for large numbers due to repeated calculations.

5. What is the complexity of the queue and dequeue operations on a queue? Please create a data structure for a queue with different complexity**.**

O(1) is the complexity of the queue and dequeue operations on a queue. In queue, a dequeue removes elements from front where a enqueue adds element to the end.

Data Structure Code:

```
class StudentQueue:
    def __init__(self):
        self.queue = []

    def enqueue(self, student_name):
        """Add a student to the end of the queue."""
        self.queue.append(student_name)
```

```
    def dequeue(self):
        """Remove and return the student from the front of the queue."""
        if self.is_empty():
            raise IndexError("Dequeue from an empty queue")
        return self.queue.pop(0)

    def is_empty(self):
        """Check if the queue is empty."""
        return len(self.queue) == 0

    def size(self):
        """Return the number of students in the queue."""
        return len(self.queue)
```

The StudentQueue class is a Python class that creates a queue data structure using an empty list. It consists of three components: __init__(self), enqueue(self, student_name), dequeue(self), and is_empty(self), which initialize and remove students.

6.1 Write pseudocode for calculating the Greatest Common Divisor (GCD) of two numbers. There are many ways to do this. Zero marks for code alone.

Pseudocode for Greatest Common Divisor (GCD) of two numbers :

```
Function GCD(c, d):
    While c ≠ d:
        If c > d:
            c = c - d
        Else:
            d = d - c
    Return c
```

Step:

Start: Function start with two variables, c and d.

Subtract: Continue reducing the lower value from the greater.

Repeat: Continue this process until both numbers become the same.

Result: The sum of their combined values is the initial numbers of GCD.

6.2 What is the time complexity of this calculation in Big-O notation? This depends on your implementation**.**

The subtraction method subtracts a smaller number from a larger one, requiring iterations and steps proportional to the difference between the two numbers. The correct time complexity is O(\max(c, d)), as each operation reduces one number by at least the smaller number.

6.3 Show that the output of this algorithm is indeed the Greatest Common Divisor. Do not simply state that it is; demonstrate why it is.

The algorithm of the subtraction consider minimizes the size of the problem while maintaining the common divisor. The GCD of c-d and c-d is the same as the GCD of two integers. When c = d, the process stops, confirming that the result is the greatest common divisor.