2.1 Task 1: Pseudocode Development

Write a detailed pseudocode for a simple program that takes a number as input, calculates the square if it's even or the cube if it's odd, and then outputs the result. Incorporate conditional and looping constructs.

Pseudocode:-

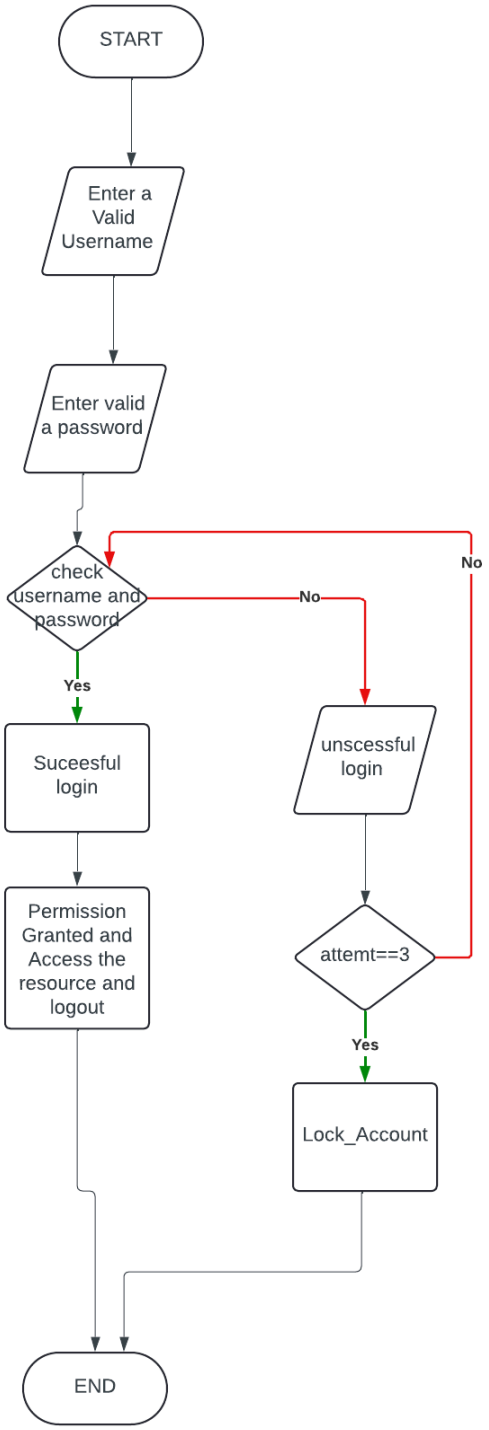
```
START
        PRINT “Enter a valid Number:”
              Var num :Integer
         Var squ :Integer
         Var cub :Integer
         IF num % 2 == 0 THEN
            squ =num*num
            PRINT “Square of The Number:[squ]”
         ELSE
               cub = num*num*num
               PRINT “Cubic of The Number:[cub]”
         ENDIF
  STOP
```

2 Task 2: Flowchart Creation

Design a flowchart that outlines the logic for a user login process. It should include conditional paths for successful and unsuccessful login attempts, and a loop that allows a user three attempts before locking the account.

Explanation:

1. Start: The process begins here.
2. Enter Username: The user inputs their username.
3. Enter Password: The user inputs their password.
4. Validate Credentials: The system checks if the provided username and password are correct.
5. Successful Login: If the credentials are valid, the user is successfully logged in.
6. Unsuccessful Login: If the credentials are invalid, the user is informed of the unsuccessful attempt.
7. Lock Account: If the user exceeds three unsuccessful login attempts, the account is locked.
8. End: The process ends here.


START
Enter a Valid Username
Enter valid a password
check username and password
No
Yes
Suceesful login
Permission Granted and Access the resource and logout
unscessful login
attemt==3
No
Yes
Lock_Account
END

Task 3: Function Design and Modularization
Create a document that describes the design of two modular functions: one that returns the factorial of a number, and another that calculates the nth Fibonacci number. Include pseudocode and a brief explanation of how modularity in programming helps with code reuse and organization.

Function Degin And Modularization
Recurssion Approach

```
Start
 Factorial(n)
    IF n == 0
        return 1
    else
        return n * Factorial(n - 1)
End
```

Iteration Approach:-

```
Start
      Factorial(n)
           Var fact = 1
            For i =1 TO n
                    Fact = fact *i
             Return fact
```

```
End
 Start
Main()
 Print “Enter a Number to calculate
Factorial:”
Var n :Integer
Read n
Var ans :Integer
Factorial(n);
End
Fibonacci Series Recusion Apporach :-
Start
Fibonacci(n)
    if(n==0 || n==1)
          Return 1
   Return Fibonacci(n-1)+Fibonacci(n-2)
End
Start
    Fibonacci(n)
      Var n1:Integer
       Var n2:Integer
         n1=0
         n2 =1
          Print  n1
          Print n2
          Var tt:Integer
```

```
For i=0 to n THEN
    tt=n1+n2
    Print tt
    n1=n2
     n2=tt
  Endfor
End
```

Function Design Document

Function 1: Factorial Calculation

Purpose: The purpose of this function is to calculate the factorial of a given number.

Input: An integer representing the number whose factorial needs to be calculated.

Output: An integer representing the factorial of the input number.

Function Design Document

Function 1: Factorial Calculation

Purpose: The purpose of this function is to calculate the factorial of a given number.

Input: An integer representing the number whose factorial needs to be calculated.

Output: An integer representing the factorial of the input number.

Pseudocode:

typescript

Copy code

```
function factorial number
 if number    0
 return 1
 else
 return number   factorial number   1
```

Explanation:

This function calculates the factorial of a number recursively. If the input number is 0, it returns 1 as the factorial of 0 is defined as 1. Otherwise, it multiplies the input number by the factorial of the number decremented by 1.

Function 2: Fibonacci Number Calculation

Purpose: The purpose of this function is to calculate the nth Fibonacci number.

Input: An integer representing the position of the Fibonacci number to be calculated (n).

Output: An integer representing the nth Fibonacci number.

Explanation:

This function calculates the nth Fibonacci number recursively. If the input position is 0, it returns 0. If the input position is 1, it returns 1. Otherwise, it returns the sum of the (n-1)th and (n-2)th Fibonacci numbers.

Modularity in Programming:

Modularity in programming refers to the practice of breaking down a program into smaller, self-contained modules or functions.

Each module or function performs a specific task, making the code more organized, easier to understand, and easier to maintain.

Advantages of Modularity:

1. **Code Reusability: Modular functions can be reused in different parts of the program or in different programs altogether, reducing redundancy and promoting efficient code reuse.**
2. **Easy Maintenance: Since each module or function is self-contained and focuses on a specific task, it is easier to debug, update, and maintain the codebase. Changes made to one module do not affect other modules, reducing the risk of unintended consequences.**
3. **Improved Readability: Breaking down a program into modular functions improves code readability and comprehension. Each function serves as a building block with a clear purpose, making it easier for developers to understand the overall logic of the program.**

4. **Enhanced Collaboration: Modular programming facilitates collaboration among team members by allowing them to work on different modules independently. This promotes parallel development and enables teams to deliver projects more efficiently.**

In summary, modularity in programming enhances code reusability, maintainability, readability, and collaboration, leading to more robust and scalable software solutions.

In Both Case the time Complexity of factorial and Fibonacci is O(n).