Subject: Professional Development Skills	Lecture - 1	Date: 23.12.2024 / Monday	No. of Programs : 02

Addition of two numbers:

C

```c
#include <stdio.h>
// Include standard input-output library
int main() {
 // Main function where program execution
begins
int num1, num2, sum;
// Declare three integer variables
   // Prompt the user for two numbers
   printf("Enter first number: ");
   scanf("%d", &num1);
   printf("Enter second number: ");
   scanf("%d", &num2);
   // Calculate the sum
   sum = num1 + num2;
   // Print the result
   printf("Sum: %d\n", sum);
   return 0;
 // Indicate successful execution
}
```

JAVA

```java
import java.util.Scanner;
// Import Scanner class for user input
public class Main {
    public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);
// Create a scanner object for input
        // Declare variables
        int num1, num2, sum;
        // Prompt the user for two numbers
 System.out.print("Enter first number: ");
        num1 = sc.nextInt();
System.out.print("Enter second number: ");
       num2 = sc.nextInt();
        // Calculate the sum
        sum = num1 + num2;
        // Print the result
        System.out.println("Sum: " + sum);
    }
}
```

PYTHON

```python
# Take input from the user
num1 = int(input("Enter first number: "))  # Integer
input
num2 = int(input("Enter second number: "))
# Integer input

# Calculate the sum
sum = num1 + num2

# Print the result
print("Sum:", sum)
```

Time Complexity:

In all three programs, the operations are performed in constant time:

Reading two integers from the user.

Performing the addition.

Printing the result.

Each of these operations takes constant time, so the overall **time complexity** for all three programs is **O(1).**

Space Complexity:

Each program uses a fixed amount of memory for storing the two integers (`num1`, `num2`) and the result (`sum`).

There are no dynamic memory allocations or large data structures, so the overall **space complexity** is **O(1) for** all three programs.

Program to accept an integer, a floating-point number, and a character

C

```c
#include <stdio.h>
// Include standard input-output library
int main() {
 // Main function where program execution begins
    int int_num;
// Declare an integer variable
    float float_num;
// Declare a floating-point variable
    char char_val;
// Declare a character variable

    // Take integer input
    printf("Enter an integer: ");
    scanf("%d", &int_num);

    // Take floating-point input
  printf("Enter a floating-point number: ");
    scanf("%f", &float_num);

    // Take character input
    printf("Enter a character: ");
    scanf(" %c", &char_val);
// The space before %c is to consume any leftover newline character

    // Display the inputs
    printf("You entered: \n");
    printf("Integer: %d\n", int_num);
    printf("Floating point number: %.2f\n", float_num);
    printf("Character: %c\n", char_val);

    return 0;
// Indicate successful execution
}
```

JAVA

```java
import java.util.Scanner;
// Import Scanner class for user input

public class Main {
    public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
// Create a scanner object for input

        // Declare variables
        int intNum;
        float floatNum;
        char charVal;

        // Take integer input
        System.out.print("Enter an integer: ");
        intNum = sc.nextInt();

        // Take floating-point input
        System.out.print("Enter a floating-point number: ");
        floatNum = sc.nextFloat();

        // Take character input
      System.out.print("Enter a character: ");
        charVal = sc.next().charAt(0);

        // Display the inputs
        System.out.println("You entered: ");
   System.out.println("Integer: " + intNum);
 System.out.println("Floating point number: " + floatNum);
        System.out.println("Character: " + charVal);
    }
}
```

PYTHON

```python
# Take user inputs
int_num = int(input("Enter an integer: "))
# Integer input
float_num = float(input("Enter a floating-point number: "))  # Floating-point input
char_val = input("Enter a character: ")[0]
# Character input

# Display the inputs
print("You entered: ")
print("Integer:", int_num)
print("Floating point number:", float_num)
print("Character:", char_val)
```

Explanation:

1. **Time Complexity:**
   - In all three programs, we perform constant-time operations:
     - Accepting integer, float, and character inputs, which take constant time.
     - Printing the values, this also takes constant time.
   - Hence, the overall **time complexity** for each program is **O(1)**.
2. **Space Complexity**:
   - We only use a fixed amount of memory for storing three variables (one for each type of input: integer, float, and character).
   - No dynamic memory allocation or large data structures are used.
   - Therefore, the **space complexity** for all three programs is **O(1)**.

Type	Time Complexity (Big-O)	Space Complexity (Big-O)	Description
Constant	$O(1)$	$O(1)$	The algorithm/operation takes the same amount of time/space, regardless of input size.
Logarithmic	$O(\log n)$	$O(\log n)$	The time/space grows logarithmically with the input size (e.g., binary search).
Linear	$O(n)$	$O(n)$	The time/space grows directly proportional to the input size (e.g., iterating over an array).
Linearithmic	$O(n \log n)$	$O(n \log n)$	The time/space grows as the product of input size and its logarithm (e.g., merge sort).
Quadratic	$O(n^2)$	$O(n^2)$	The time/space grows proportional to the square of the input size (e.g., nested loops).
Cubic	$O(n^3)$	$O(n^3)$	The time/space grows proportional to the cube of the input size (e.g., matrix multiplication).
Exponential	$O(2^n)$	$O(2^n)$	The time/space doubles with every additional unit of input (e.g., recursive problems without optimization).
Factorial	$O(n!)$	$O(n!)$	The time/space grows factorially with the input size (e.g., permutations generation).
Polynomial	$O(n^k)$ $(k > 3)$	$O(n^k)$	The time/space grows as a polynomial function of input size (e.g., certain brute-force algorithms).
Log-Linear	$O((\log n)^n)$	$O((\log n)^n)$	A less common complexity, appearing in specialized problems.