Introduction to Complexity Analysis of Non-Recursive Algorithms

Session No.: 4
Course Name: Design and analysis of algorithm
Course Code: R1UC407B
Instructor Name: Mili Dhar
Duration: 50 Min.
Date of Conduction of Class:

GALGOTIAS UNIVERSITY

Review of the key concepts

1. Complexity Analysis
2. Types of Complexity
3. Analyzing Non-Recursive Algorithms

Q1: "Can you think of a real-life example where choosing the wrong approach to solving a problem wasted a lot of time?"

Learning Outcome

Analyze time and space complexity for non-recursive algorithms.

Identify best, worst, and average case scenarios.

Session Outline

1 Definition of Complexity Analysis

2 Importance of Analyzing Algorithms

3 Types of Complexity (Time & Space)

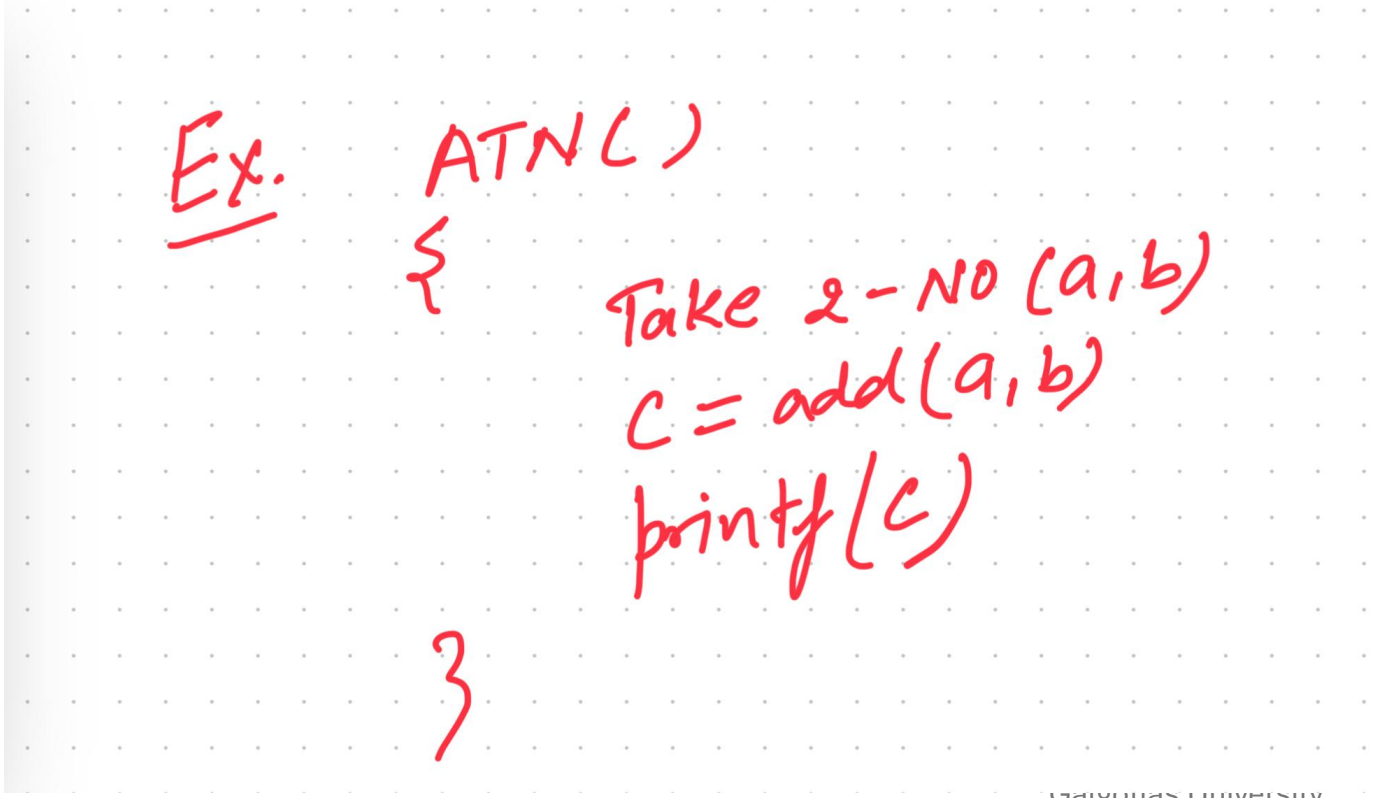
4 Complexity Analysis of Non-Recursive Algorithms

5 Reflection learning activity

6 Conclusion

Definition

- An algorithm is any well-defined computational procedure that take some values as input and produces some values as output.



Types of Complexity

1. Time Complexity:

- Measures the execution time of an algorithm.
  - **Best Case (Ω)**: Minimum time required.
  - **Worst Case (O)**: Maximum time required.
  - **Average Case (Θ)**: Expected time for random inputs.

2. Space Complexity:

- Measures memory required by an algorithm.
- Includes input storage, auxiliary space, and recursion stack.

Complexity Analysis of Non-Recursive Algorithms

Activity (Pen-Paper based)

Example 1: Summing N Numbers

```
int sum(int n)
{
int total = 0;
for (int i = 1; i <= n; i++)
  {
  total += i;
  }
 return total;
}
```

Complexity Analysis of Non-Recursive Algorithms

Activity (Pen-Paper based) Contd…

Solution:

Loop runs n times → Time Complexity: O(n)

Uses constant extra space → **Space Complexity: O(1)**

Analysis

Ex.

```
main()
{
    x = y + z;
    for(i=1; i<=n; i++)
    {
        x = y + z;
    }
    for(i=1; i<=n; i++)
        for(j=1; j<=n; j++)
        {
            x = y + z;
        }
}
```

Analysis

Ex.

```
main()
{
    x = y + z;
    for(i=1; i<=n; i++)
    {
        x = y + z;
    }
    for(i=1; i<=n; i++)
        for(j=1; j<=n/2; j++)
        {
            x = y + z;
        }
}
```

Analysis

Ex.

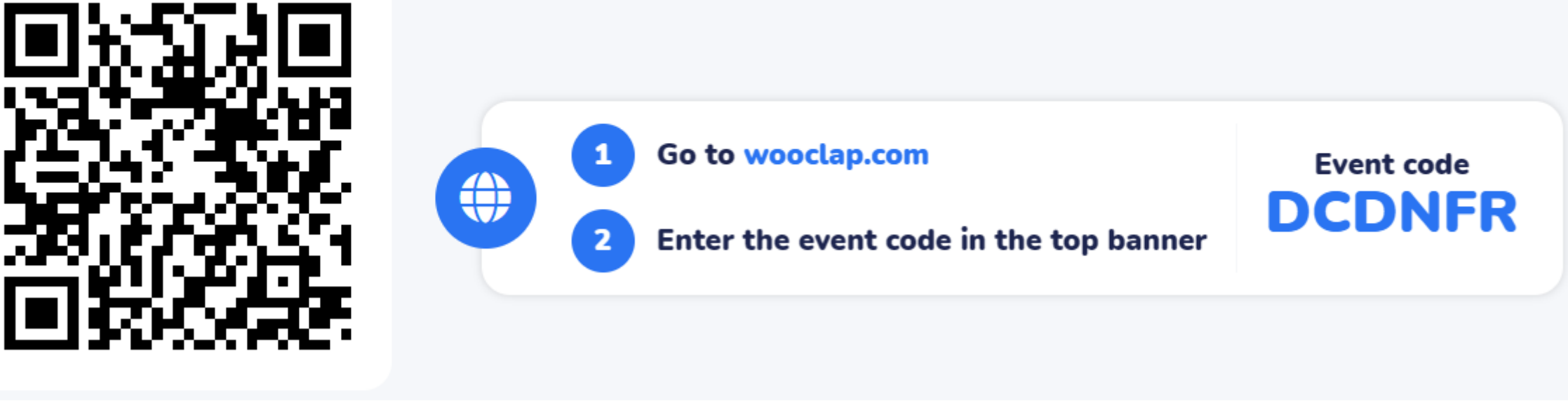
```
main()
{
    x = y + z;
    for(i=1; i<=n; i++)
    {
        x = y + z;
    }
    for(i=1; i<=n/2; i++)
        for(j=1; j<=n/2; j++)
        {
            x = y + z;
        }
}
```

Analysis

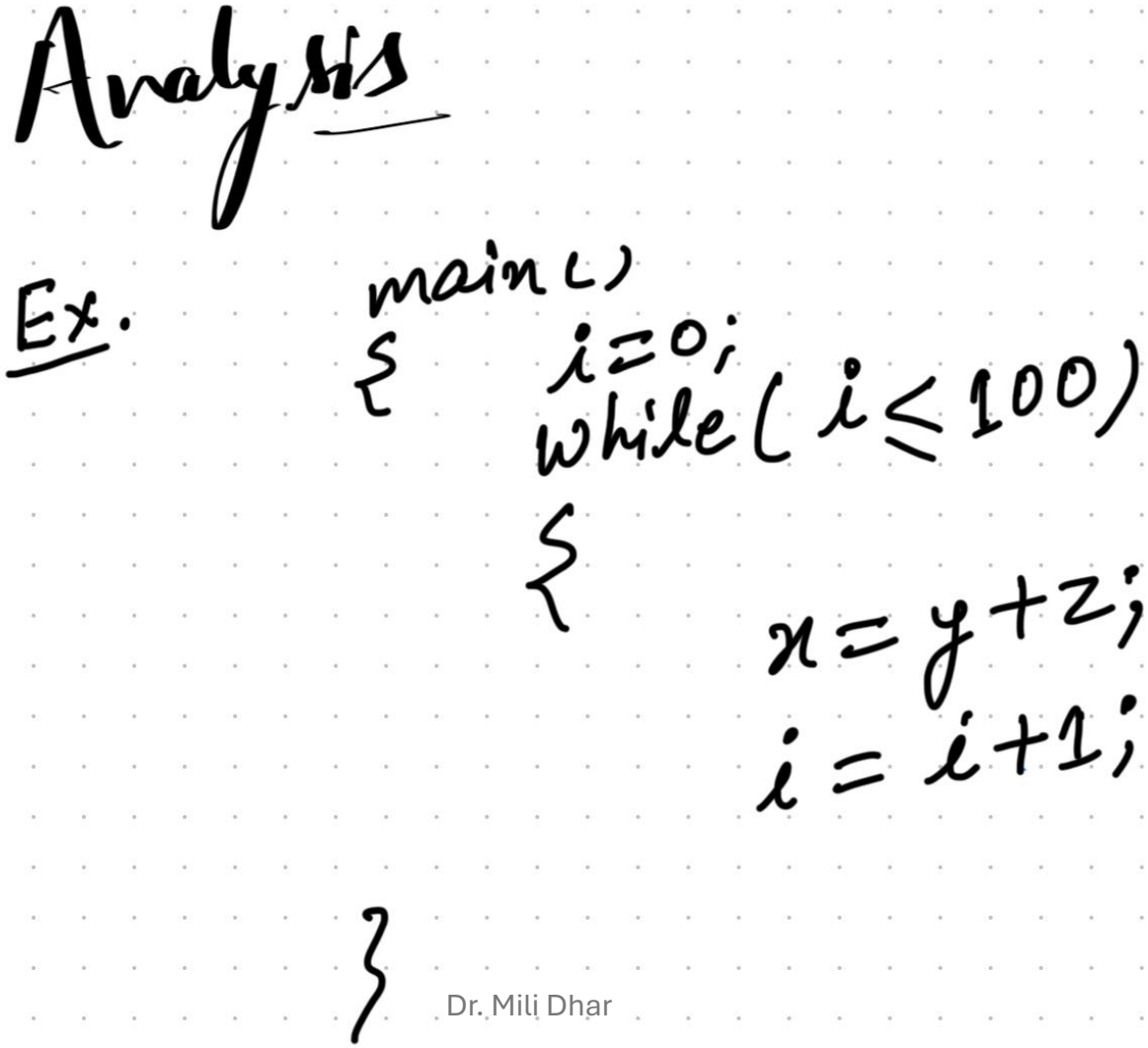
Ex.

```
main()
{
    x = y + z;
    for(i = 1; i <= n; i++)
    {
        x = y + z;
        x = y + z;
    }
}
```

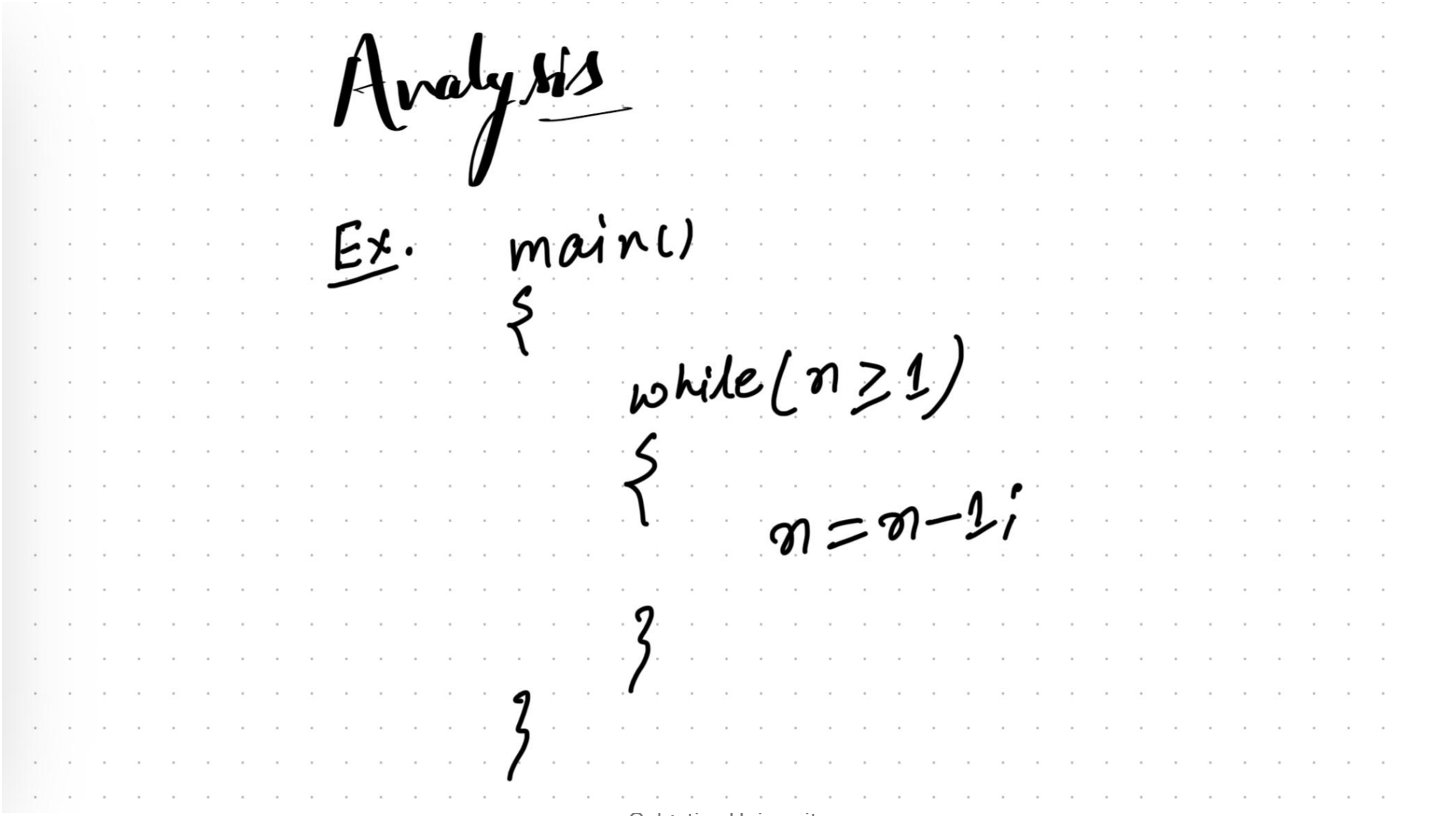
Activity 1 (Wooclap)



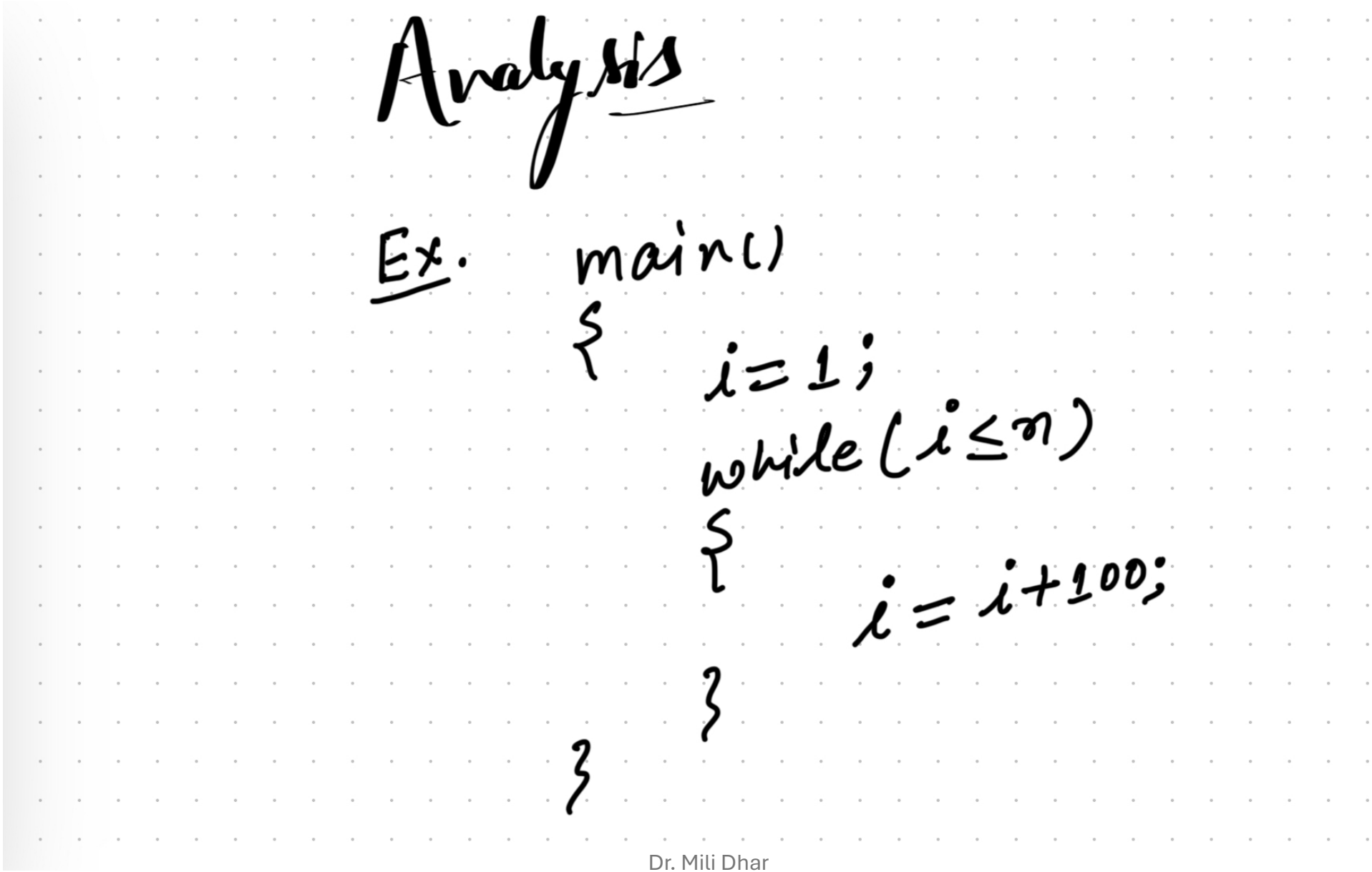
Activity 1 (Wooclap)



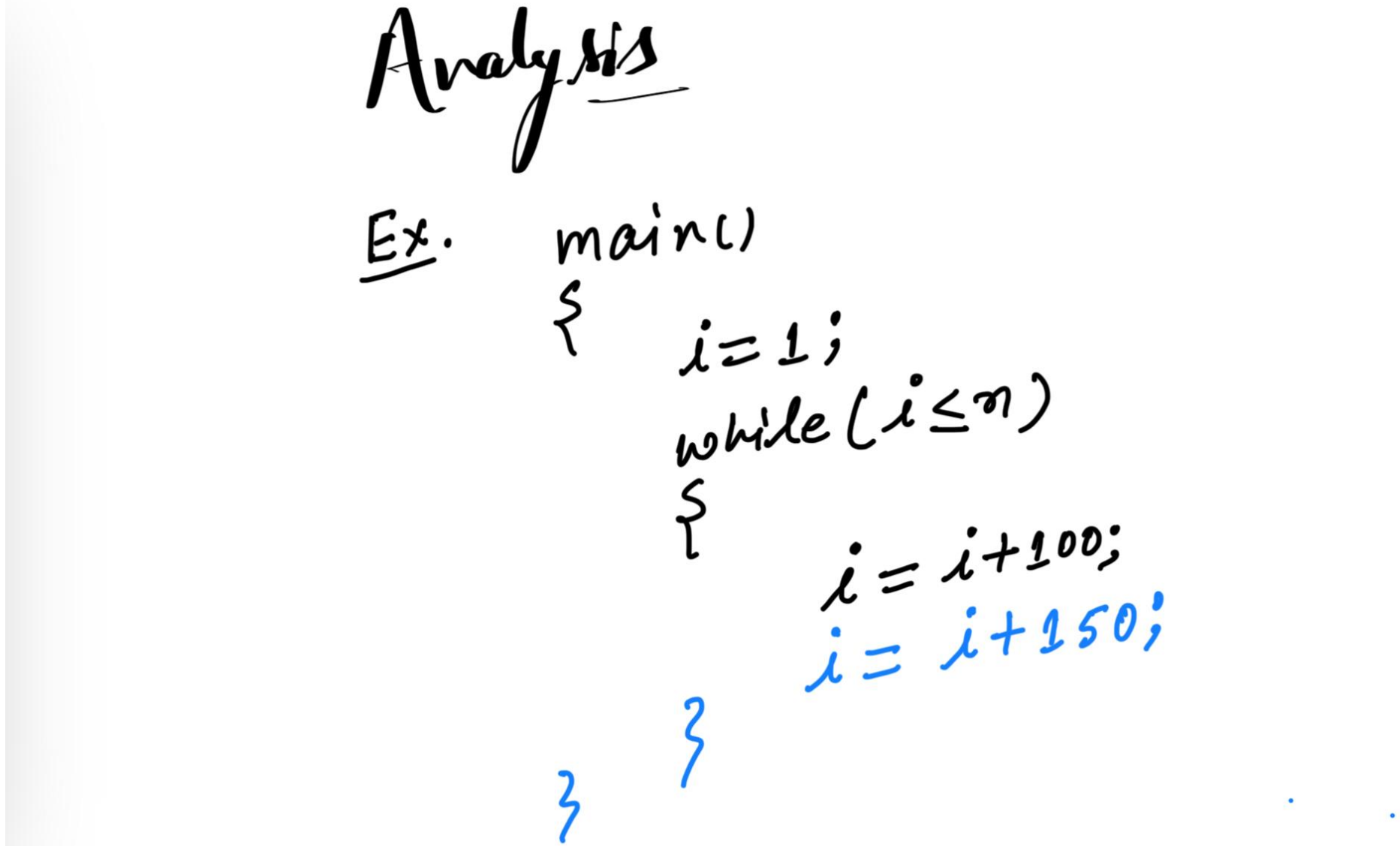
Activity 2



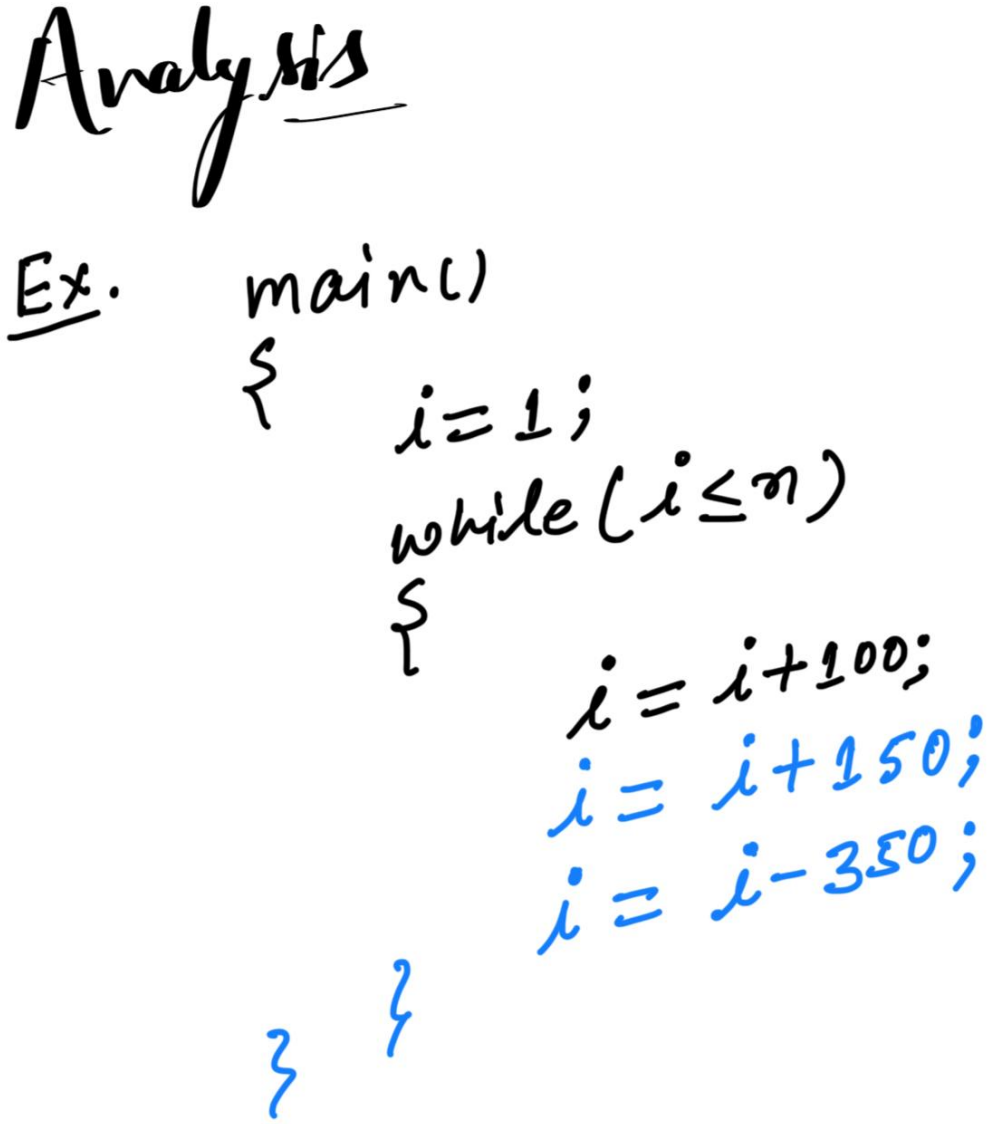
Activity 2



Activity 2



Activity 2 (WoocLap)



Reflection-

How did analyzing this scenario help you understand the importance of algorithm efficiency?

Summery

- Complexity analysis helps determine efficiency. Time complexity measures execution time, while space complexity measures memory usage.

  Non-recursive algorithms are easier to analyze as they do not involve recursion stacks.

Post session activities

Write an algorithm to find the second largest element in an array and analyze its time complexity.

In the next session, Advanced Complexity Analysis of Iterative Algorithms will be discussed in detail.

Review and Reflection from students

What was the most interesting concept you learned today?

Any doubts or areas requiring further clarification?