AROR UNIVERSITY
OF ART, ARCHITECTURE,
DESIGN & HERITAGE,
SUKKUR, SINDH

CSC-331

Analysis of Algorithms

Fall- 2025

Lecture - 01

Lecture Outline

- Introduction
- Course Outline
- Algorithms
- Analysis

What is an Algorithm?

- **Definition:**
  An algorithm is a finite sequence of well-defined instructions for solving a problem or performing a task.
- **Key Components:**
  - **Input:** Data provided to the algorithm
  - **Process:** Step-by-step operations
  - **Output:** Desired result
- **Example:** Finding the largest number in a list

Why Study Algorithms?

- Algorithms are core to computing:
- Efficiency = Performance
- Correctness = Reliability
- Applications: AI, cybersecurity, search engines, networking

Why Study Algorithms?

- Can you name 3 algorithms you use every day without realizing?

Role of Algorithms in Computing Systems

- **OS Scheduling**: CPU task allocation
- **Networking**: Routing algorithms (Dijkstra, Bellman-Ford)
- **Databases**: Query optimization algorithms
- **AI & Machine Learning**: Training models using gradient descent
- **Example: Google search → PageRank Algorithm**

Characteristics of a Good Algorithm

Correctness – Produces correct output

Finiteness – Terminates after finite steps

Definiteness – Each step is clear and unambiguous

Efficiency – Optimal in time and space

Generality – Works for all valid inputs

Real-Life Examples of Algorithms

- **Google Maps**: Shortest path algorithms (Dijkstra, A*)
- **Amazon**: Recommendation algorithms
- **Cryptography**: RSA algorithm for security
- **Finance**: Algorithmic trading
- **Healthcare**: DNA sequence alignment algorithms

Key Metrics for Algorithm Performance

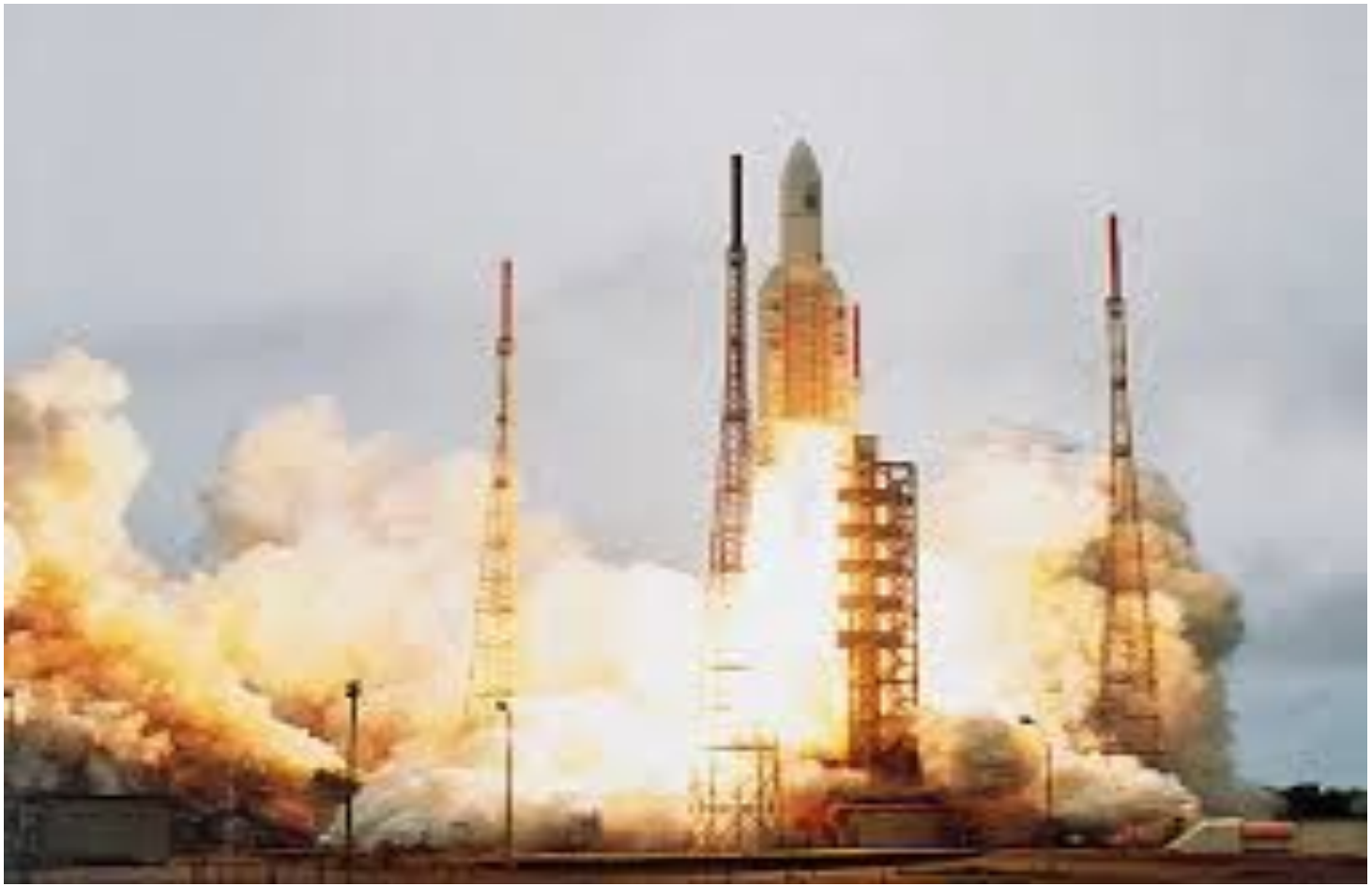
Time Complexity: How execution time grows with input size

Space Complexity: Memory usage by the algorithm

Scalability: Handling large inputs efficiently

Analysis of Algorithm

Analysis of Algorithm is the process of evaluating and measuring the efficiency of an algorithm in terms of the resources it consumes, mainly time (execution speed) and space (memory usage), as the size of input grows.



Why Analysis of Algorithms

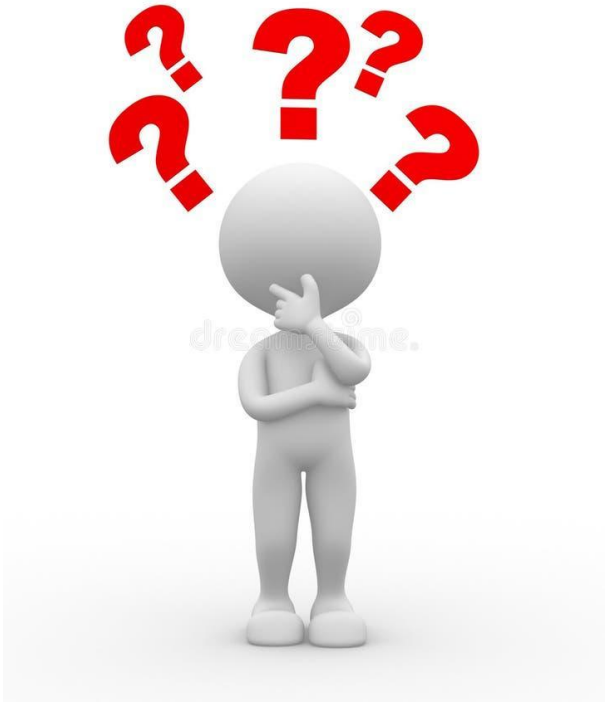
- **Ariane 5 Rocket Failure (1996)**
- **What happened:** During the maiden flight of the Ariane 5 rocket, software that was originally designed for the earlier Ariane 4 reused arithmetic code that failed under the new rocket's flight conditions.
- **Root cause:** An **integer overflow** occurred in the inertial reference software. The exception was mishandled, causing the inertial navigation system to crash and send the rocket off-course. Within 40 seconds post-launch, the rocket self-destructed

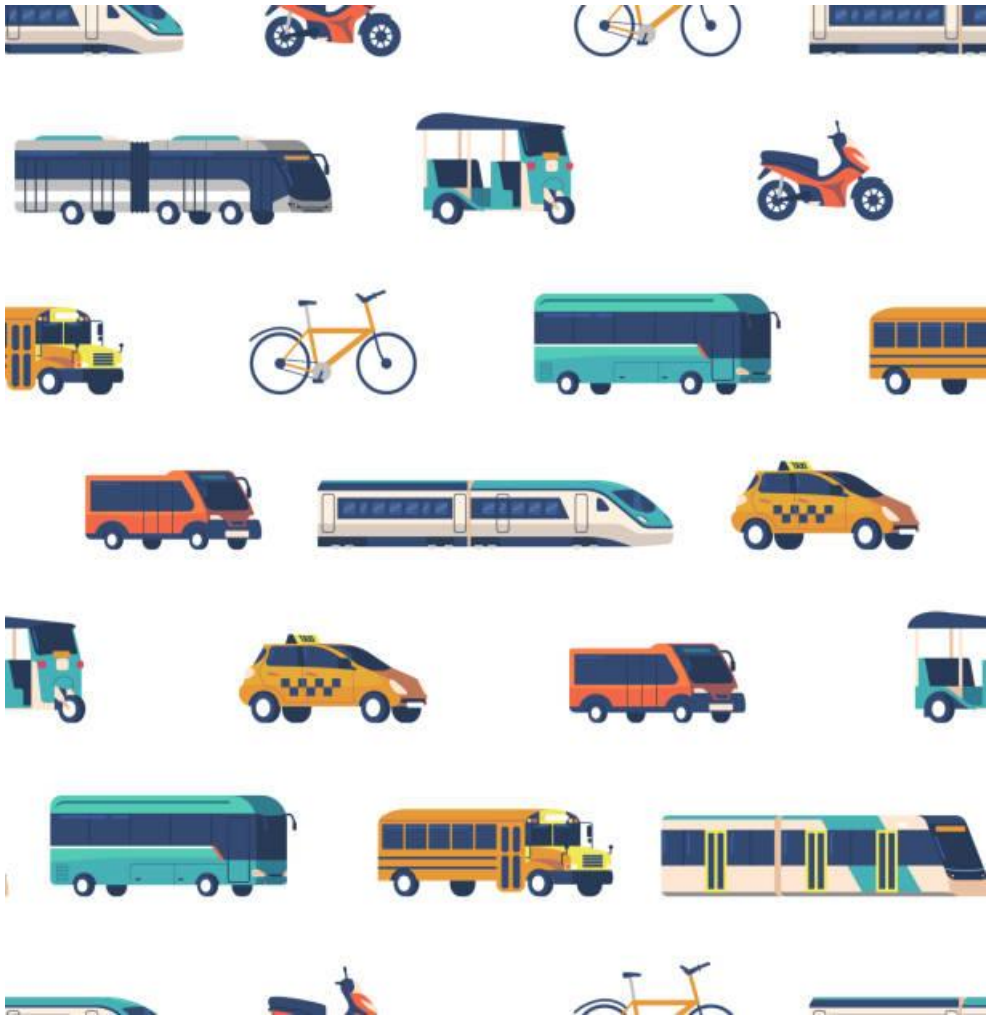
Analysis

- Speed
- Memory
- Input

Discussion

I am planning to travel from Sukkur to Larkana. Suggest and discuss the possible ways.





Where to use……

- Software Development
- To compare algorithms for efficiency before implementation
- Choose optimal data structures & algorithm based on input-size.
- Operating Systems
- CPU and disk scheduling…..

Insertion Sort

```
INSERTION-SORT(A, n)
1  for i = 2 to n
2      key = A[i]
3      // Insert A[i] into the sorted subarray A[1 : i − 1].
4      j = i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] = A[j]
7          j = j − 1
8      A[j + 1] = key
```

What is RAM Model?

- The **RAM model** is an **abstract computational model** used for analyzing algorithms.
- It assumes:
  - The machine has **a single CPU** and **unlimited memory**.
  - Each **basic operation (like +, -, , /, assignment, comparison)* takes **constant time (O(1))**.
  - **Memory access is uniform** (accessing any memory cell takes O(1)).
- It ignores **CPU cache**, **parallelism**, and **disk I/O delays** for simplicity.

Why use RAM model?

- To **count the exact number of primitive operations** in an algorithm.
- To estimate **time complexity** in a theoretical, machine-independent way.

Steps to Perform RAM Model Analysis

1. Write down the algorithm or pseudo-code.
2. Identify all primitive operations:
- Assignments (a = b)
- Arithmetic operations (+, -, *, /)
- Comparisons (<, >, ==)
- Array indexing (A[i])
- Loop control operations

Steps to Perform RAM Model Analysis

3. Assign a constant cost (usually 1 unit) to each primitive operation.

4. Count the total number of operations in the worst case (or best/average if required).

5. Express the count as a function of input size n.

6. Convert it into asymptotic notation (Big-O).

Example: Analyze a Simple Algorithm

- Algorithm: Sum of first n numbers:

```
int sum = 0;                    // Line 1
for (int i = 1; i <= n; i++)  // Line 2
    sum = sum + i;              // Line 3
```

Step 1: Count operations using RAM model

- Line 1: sum = 0 ⟶ 1 operation
- Line 2: i = 1 ⟶ 1 operation
- Line 2: i <= n check ⟶ executed n+1 times
- Line 2: i++ increment ⟶ executed n times
- Line 3: sum = sum + i ⟶ executed n times (1 addition + 1 assignment = 2 operations)

Step 2: Calculate total cost

- Initialization: 1 + 1 = 2
- Comparisons: (n + 1)
- Increments: n
- Sum update: 2n

Total = 2 + (n + 1) + n + 2n = 4n + 3

Step 3: Express as Big-O

- $T(n) = 4n + 3 \longrightarrow O(n)$

Algorithm: Find Maximum

```
int max = A[0];                    // Line 1
for (int i = 1; i < n; i++) {      // Line 2
    if (A[i] > max)                // Line 3
        max = A[i];                // Line 4
}
```

Step 1: Identify Primitive Operations

- Step 1: Identify Primitive Operations
- Line 1: max = A[0] $\longrightarrow$ 1 operation (assignment) + 1 (array access) = 2 operations
- Line 2: i = 1 $\longrightarrow$ 1 operation
- Line 2: i < n $\longrightarrow$ checked n times (from i = 1 to i = n)
- Line 2: i++ $\longrightarrow$ executed n-1 times
- Line 3: A[i] > max $\longrightarrow$ executed n-1 times
- Line 4: max = A[i] $\longrightarrow$ executed at most n-1 times (only when condition is true, but assume worst case)

Step 2: Count Total Cost

- Initialization: max = A[0] ⟶ 2 operations
- i = 1 ⟶ 1 operation
- Total so far = 3 operations
- For loop: Comparison i < n: executed n times ⟶ n
- Increment i++: executed n-1 times ⟶ n-1
- Condition check A[i] > max: executed n-1 times ⟶ n-1
- Assignment max = A[i]: executed n-1 times (worst case) ⟶ n-1
- Total inside loop = (n) + (n-1) + (n-1) + (n-1) = 4n – 3
- Add initialization = (3) + (4n - 3) = 4n

Step 3: Express as Big-O

- $T(n) = 4n \longrightarrow O(n)$


Q & A