**📘 Lecture 1 – Introduction & Algorithm Analysis**

* Basics: What is an **algorithm**, what is a **data structure**
* Properties of algorithms (input, output, definiteness, effectiveness, finiteness)
* Algorithm techniques: **Brute Force, Divide & Conquer, Dynamic Programming, Greedy, etc.**
* **Algorithm analysis** framework (RAM model, primitive operations)
* Pseudo code
* Time complexity: Best, Worst, Average case
* Asymptotic notations: **O, Ω, Θ**
* Basic math: **logs, induction, series (AP/GP)**

**📘 Lecture 2 – ADTs, Lists, Stacks, Queues**

* **Abstract Data Type (ADT):** model + operations, independent of implementation
* **Lists:**
  + Array & Linked representation
  + Operations: insert, erase, get, indexOf
  + Issues: resizing, shifting
* **Linked Lists:**
  + Singly, Doubly, Circular
  + Insertion/deletion algorithms
* **Stacks:**
  + LIFO, operations (push, pop, top, empty)
  + Applications: recursion, expression evaluation, Towers of Hanoi
  + Infix, Prefix, Postfix conversions
* **Queues:**
  + FIFO, operations (push, pop, front, back)
  + Array vs Linked implementation
  + Applications (scheduling, graph traversal, simulations)

**📘 Lecture 3 – Algorithm Analysis 2 & Recurrence**

* Analysis of **non-recursive & recursive algorithms**
* Merge sort analysis
* **Recurrence relations**
  + Substitution method
  + Recursion tree method
  + Master’s theorem (simplified version with cases & examples)
* General rules for time complexity (loops, nested loops, if-else, logarithmic complexity)
* Recursive algorithms (definition, requirements, base case, examples)
* Tracing recursion

**📘 Lecture 4 – Trees**

* **Tree basics**: nodes, root, parent, child, siblings, leaves, internal nodes, paths
* Definitions: depth, height, level, subtree, degree, forest
* **Tree ADT** (operations & queries)
* **Binary Trees:**
  + Proper (strict/full), complete, perfect
  + Properties & formulas (height, no. of nodes, leaves, etc.)
  + Array representation
  + Linked representation
* **Tree traversals:** preorder, inorder, postorder

**📘 Lecture 5 – Sorting & Heaps**

* **Sorting Algorithms:**
  + Bubble sort, Insertion sort, Selection sort, Bucket sort
  + Quick sort (partition method, complexity best/worst/average)
  + Complexity comparisons
* **Heaps:**
  + Max/Min heap definitions
  + Heap operations: Insertion, Deletion, Heapify
  + Array representation of heap
  + **Heap Sort** process (Build heap + repeated deletion)

**Lecture 1 – Introduction & Algorithm Analysis**

**Short/Conceptual**

1. Define an **algorithm**. What are the properties of a good algorithm?
2. Differentiate between **data structures** and **algorithms**.
3. What is the **RAM model**? Why is it used in algorithm analysis?
4. Explain **Brute Force, Divide & Conquer, Greedy, Dynamic Programming** with one example each.
5. Differentiate between **best case, worst case, and average case analysis**.

**Numerical/Proof-based**

1. Prove by **induction** that 1+2+…+n=n(n+1)/21+2+…+n = n(n+1)/21+2+…+n=n(n+1)/2.
2. Solve: Count the **primitive operations** in the pseudo-code arrayMax(A, n).
3. Given f(n)=n22−n2f(n) = \frac{n^2}{2} - \frac{n}{2}f(n)=2n2​−2n​, find asymptotic notation (O, Ω, Θ).
4. Compare the growth rates of nnn, nlog⁡nn \log nnlogn, n2n^2n2, 2n2^n2n.
5. Show that binary search runs in O(log⁡n)O(\log n)O(logn) time.

**Lecture 2 – Lists, Stacks, Queues**

**Conceptual**

1. Define **Abstract Data Type (ADT)**. Give two examples.
2. Differentiate between **array-based lists** and **linked lists**.
3. Explain **singly linked list vs doubly linked list vs circular linked list**.
4. Explain the operations of a **stack (push, pop, top)** with an example.
5. What are the applications of stacks in **expression evaluation**?
6. Explain **Queue (FIFO)**. Compare **array vs linked implementation**.
7. Define **infix, prefix, postfix** notation with examples.

**Algorithm/Implementation**

1. Write an algorithm to **insert a node** at a given position in a linked list.
2. Write an algorithm to **delete a node** from a doubly linked list.
3. Write an algorithm to **evaluate a postfix expression** using a stack.
4. Convert the infix expression (A+B) \* C into postfix.
5. Implement **queue using an array** and explain how circular array helps.

**Numerical**

1. Trace the stack contents while evaluating:

* Expression: AB+C\* where A=2, B=3, C=4.

1. Convert the following into postfix:

* A + (B \* C - D) / E

1. A queue is implemented using a circular array of size 6. Show the state after operations:

* Insert(10), Insert(20), Insert(30), Delete(), Insert(40), Insert(50).

**Lecture 3 – Algorithm Analysis 2 & Recurrence**

**Conceptual**

1. Differentiate between **iteration** and **recursion**.
2. Explain the steps in **analyzing a recursive algorithm**.
3. State and explain the **Master’s theorem** with cases.
4. What are the three methods to solve recurrences?
5. Explain **recursion tree method** with example.

**Numerical/Proof**

1. Solve the recurrence: T(n)=2T(n/2)+nT(n) = 2T(n/2) + nT(n)=2T(n/2)+n.
2. Solve the recurrence: T(n)=3T(n/4)+n2T(n) = 3T(n/4) + n^2T(n)=3T(n/4)+n2.
3. Prove by recursion tree that merge sort is O(nlog⁡n)O(n \log n)O(nlogn).
4. Find the time complexity of:

for (i=1; i<=n; i++)

for (j=1; j<=n; j++)

count++;

1. Find the time complexity of:

for (i=1; i<=n; i=i\*2)

for (j=1; j<=n; j++)

count++;

**Lecture 4 – Trees**

**Conceptual**

1. Define: root, child, parent, siblings, leaf, internal node, degree, depth, height.
2. Differentiate between **strict (full), complete, and perfect binary trees**.
3. What is a **forest** in data structures?
4. Give the array representation of a **complete binary tree**.
5. Write recursive algorithms for **preorder, inorder, and postorder traversals**.

**Numerical/Proof**

1. A binary tree has 10 external nodes. Find the number of internal nodes.
2. If a binary tree has 15 nodes, what is the minimum and maximum height possible?
3. Draw a binary tree of height 3 and label preorder, inorder, postorder traversal.
4. Given preorder = ABDECF, inorder = DBEAFC, construct the binary tree.
5. 

**Lecture 5 – Sorting & Heaps**

**Conceptual**

1. Explain the working of **Bubble Sort**. Why is it inefficient?
2. Explain **Insertion Sort** with an example.
3. Explain **Selection Sort** with an example.
4. Explain the steps of **Quick Sort** with partitioning.
5. Compare the best, worst, and average time complexities of bubble, insertion, selection, quick sort.
6. Define **Heap, Max Heap, Min Heap**.
7. Explain **heap insertion** with example.
8. Explain **heap deletion (removing max element)** with steps.
9. What is **heapify**? Explain its role in heap construction.
10. Explain **Heap Sort** algorithm.

**Numerical**

1. Sort the array [5, 3, 8, 4, 2] using **insertion sort** (show passes).
2. Sort the array [29, 10, 14, 37, 13] using **selection sort** (show steps).
3. Show quick sort on [9, 7, 5, 11, 12, 2, 14, 3, 10, 6] with pivot = last element.
4. Construct a max-heap for the array [10, 20, 5, 6, 1, 8, 9].
5. Perform heap sort on [12, 11, 13, 5, 6, 7]. Show intermediate heaps.