Unit -2

1. Introduction to Recursion and Divide & Conquer

- A **recursive algorithm** calls itself to solve smaller versions of the same problem.
- If the problem can be broken into **smaller sub-problems**, solved individually, and combined back, this is called **Divide and Conquer (D&C)**.
- Steps in D&C:
    1. **Divide** the problem into sub-problems.
    2. **Conquer** by solving them (recursively).
    3. **Combine** their results to get the final solution.

Examples: Merge Sort, Quick Sort, Binary Search, Strassen's Matrix Multiplication.

---

2. Recurrence Relations

- When analyzing recursive algorithms, we often get equations like:
  T(n) = aT(n/b) + f(n)
    - a = number of sub-problems
    - n/b = size of each sub-problem
    - f(n) = extra work (divide + combine)

This is called a **recurrence relation**. We solve it to find time complexity.

---

3. Methods to Solve Recurrence

(a) Substitution Method

- Guess the solution (like O(n log n)) and prove it using induction.
- Example: Binary Search → T(n) = T(n/2) + O(1) → solution = O(log n).

(b) Recursion Tree Method

- Draw a tree where each level shows the work done in recursive calls.
- Add up the cost at each level.
- Example: Merge Sort → tree has log n levels, each level costs O(n). Total = O(n log n).

(c) Master Theorem

For T(n) = aT(n/b) + f(n):

- Compare f(n) with n^(log_b a).

1. If f(n) grows slower → O(n^(log_b a))

2. If $f(n)$ grows same → $O(n^{\log_b a} * \log n)$
3. If $f(n)$ grows faster → $O(f(n))$ (if regularity condition holds).

Example:

- Merge Sort: $T(n) = 2T(n/2) + O(n)$ → Case 2 → $O(n \log n)$.
- Binary Search: $T(n) = T(n/2) + O(1)$ → Case 1 → $O(\log n)$.

---

4. Binary Search

- Works on a **sorted array**.
- Steps:
    1. Find middle element.
    2. If target = mid → found.
    3. If target < mid → search left half.
    4. Else → search right half.
- Time Complexity = **$O(\log n)$**, Space = $O(1)$ (iterative) or $O(\log n)$ (recursive).

---

5. Multiplying Large Integers

- Normal multiplication takes $O(n^2)$.
- Using Divide & Conquer (Karatsuba's algorithm), we reduce work.
- Idea: Split numbers into halves, do fewer multiplications + additions.
- Time Complexity = $O(n^{1.59})$ (faster than $O(n^2)$).

---

6. Matrix Multiplication

- Normal multiplication: $O(n^3)$.
- **Strassen's Algorithm**: reduces number of multiplications from 8 to 7 (by using clever formulas).
- Time Complexity = $O(n^{2.81})$.

---

7. Merge Sort

- Divide array into halves until single elements remain.
- Merge sorted halves step by step.
- Example: [5,2,8,4] → split into [5,2] and [8,4] → sort individually → merge into [2,5,4,8] → final [2,4,5,8].

- Time Complexity: O(n log n)
- Space Complexity: O(n)

---

8. Quick Sort

- Pick a **pivot** element.
- Partition array into two parts:
  - Left → elements < pivot
  - Right → elements > pivot
- Recursively sort partitions.
- Example: [5,3,8,1] → pivot=5 → partition [3,1] and [8] → sort → [1,3,5,8].
- Time Complexity:
  - Best / Avg: O(n log n)
  - Worst: $O(n^2)$ (when pivot is always smallest/largest).
- Space: O(log n) (stack).

✅ **Quick Revision for Viva:**

- Recursion = function calls itself.
- Recurrence Relation = formula for time complexity.
- Methods: Substitution, Recursion Tree, Master Theorem.
- Binary Search = O(log n).
- Merge Sort = O(n log n).
- Quick Sort = O(n log n) avg, $O(n^2)$ worst.
- Strassen's Matrix = O(n^2.81).
- Karatsuba Multiplication = O(n^1.59).