Unit - 1

📌 1. Algorithm – Definition & Characteristics

- **Algorithm** = Step-by-step procedure to solve a problem.
- **Input:** Zero or more inputs.
- **Output:** At least one output.
- **Finiteness:** Must finish in limited steps.
- **Definiteness:** Every step must be clear and unambiguous.
- **Effectiveness:** Steps must be simple and basic enough to execute.
- **Correctness:** Works for all valid inputs.

📌 2. Types of Algorithms

- **Recursive** – Solves problem by calling itself (factorial, Fibonacci).
- **Divide & Conquer** – Break into subproblems (merge sort, quick sort).
- **Dynamic Programming** – Break + store results to avoid recomputation (Floyd–Warshall, knapsack).
- **Greedy** – Take best choice at each step (Prim's, Kruskal's).
- **Backtracking** – Try possible options, backtrack on failure (N-Queens).
- **Branch & Bound** – Search with pruning.
- **Brute Force** – Try all possibilities.
- **Randomized** – Use randomness (Randomized Quick sort).

📌 3. Problem & Instance

- **Problem:** General task to be solved (e.g., sorting numbers).
- **Instance:** Specific input (e.g., [5, 2, 9, 1]).
- **Instance size:** Measures input size (for sorting → number of elements).

📌 4. Analysis of Algorithm

- Goal: Estimate **time** and **space** requirements.

- **Why?** To compare algorithms and pick the most efficient.
- Two resources:
  - **Time complexity** → execution time.
  - **Space complexity** → memory used.

Approaches

1. **Empirical (Posteriori):** Run program and measure time.
   (Problem: Depends on machine, compiler, inputs).
2. **Theoretical (Priori):** Analyze mathematically, independent of implementation.

📌 5. Efficiency of Algorithm

- Measured by growth of time w.r.t. input size n.
- Larger n → more operations → more time.

📌 6. Time Complexity

- **Best Case:** Minimum steps (e.g., already sorted).
- **Average Case:** Expected steps for random inputs.
- **Worst Case:** Maximum steps (e.g., reverse sorted).

Example:

- Linear Search → Best O(1), Worst O(n).
- Sorting → Best, average, worst differ.

📌 7. Asymptotic Notations

- Used to express **time complexity mathematically**.

1. **Big O (O):** Upper bound (worst case).
   Example: $O(n^2)$.
2. **Omega (Ω):** Lower bound (best case).
   Example: $\Omega(n)$.
3. **Theta (Θ):** Tight bound (average case).
   Example: $\Theta(n \log n)$.

Common complexities:

- Constant: $O(1)$
- Logarithmic: $O(\log n)$
- Linear: $O(n)$
- Quadratic: $O(n^2)$
- Cubic: $O(n^3)$
- Exponential: $O(2^n)$

📌 8. Analyzing Control Statements

- **Sequential:** Runs one after another ($O(1)$ each).
- **Loops:** Repeated execution.
  - Single loop → $O(n)$.
  - Nested loop → $O(n^2)$, etc.
- **Conditionals (if/else):** Only one branch executes.

📌 9. Sorting Algorithms

Sorting = Arranging data in order (ascending/descending).
Applications: Phone bills, bank statements, dropdown menus, e-commerce filtering.

🔹 **Bubble Sort**

- Compare adjacent elements and swap if needed.
- Repeated passes.
- **Best $O(n)$, Worst $O(n^2)$.**

🔹 **Selection Sort**

- Repeatedly find minimum and place at beginning.
- **Always $O(n^2)$.**

🔹 **Insertion Sort**

- Insert element into correct position in sorted part.
- **Best $O(n)$, Worst $O(n^2)$.**

🔹 **Heap Sort**

- Build max-heap → repeatedly remove root.

- **$O(n \log n)$.**

🔹 **Shell Sort**

- Improvement of insertion sort with gaps.
- **$O(n \log n)$ to $O(n^2)$.**

🔹 **Radix Sort**

- Sort numbers digit by digit.
- **$O(nk)$, k = digits.**

🔹 **Bucket Sort**

- Distribute into buckets → sort each → merge.
- **$O(n+k)$.**

🔹 **Counting Sort**

- Count frequency of elements.
- Good for small range integers.
- **$O(n+k)$.**

📌 10. Amortized Analysis

- Some operations expensive, but **average cost per operation is small**.
- Used in dynamic arrays, splay trees, hash tables.
- **Methods:**
    1. **Aggregate:** Total cost ÷ number of operations.
    2. **Accounting:** Assign extra cost (credits) to cheap operations to pay for expensive ones.
    3. **Potential:** Similar to accounting but uses "potential energy".