Design and Analysis of Algorithms – Principles Summary

1. Decomposition

Breaking a big problem into smaller parts makes it easier to solve. Example: Splitting 'build a website' into tasks like frontend, backend, and database setup.

2. Pattern Recognition

Finding similarities or repeated structures in problems to reuse solutions. Example: Recognizing that sorting logic in bubble sort and selection sort follows a comparison pattern.

3. Abstraction

Focusing only on important details and ignoring the rest. Example: A map shows cities and roads, not every building or tree.

4. Brave and Cautious Travel

Brave means going deep first (DFS); cautious means exploring level by level (BFS). Example: DFS explores one full path in a maze before trying others; BFS explores all paths layer by layer.

5. Pruning

Cutting off unnecessary branches of a search to save time. Example: In the N-Queens problem, stop exploring when two queens attack each other.

6. Lazy Propagation / Evaluation

Delay updates in data structures until they are needed to improve efficiency. Example: In a segment tree, update only when a node is accessed instead of updating all nodes immediately.

7. Sliding Window

Use a moving 'window' to process subsets of data efficiently. Example: Finding the maximum sum of 3 consecutive elements in an array.

8. Level Order Traversal

Visit nodes level by level in a tree (like BFS). Example: Printing a binary tree level by level from top to bottom.

9. Hierarchical Data

Organizing data in parent–child form like a tree. Example: File folders in a computer where each folder contains subfolders and files.

10. Edge Relaxation

Updating the shortest distance in a graph if a better path is found. Example: In Dijkstra's algorithm, if A $\rightarrow$ C is shorter via B, update the distance to C.

11. Balancing and Rotations

Keeping a tree balanced by rotating nodes to maintain efficiency. Example: In an AVL tree, perform a right rotation when the left subtree is taller.

12. Kleene Closure

Repetition of patterns or connections; ensures all reachable paths are found. Example: In regular expressions, a* means zero or more occurrences of 'a'.

13. Pre-Computing

Calculate and store results beforehand to save time later. Example: Precomputing factorials for fast calculation of combinations.

14. Parental Dominance

Parent nodes have higher/lower values than children. Example: In a max heap, the parent node always has the largest value.

15. Prefix and Suffix

Substrings from the start or end of a string. Example: For 'apple', prefix = 'app', suffix = 'ple'.

16. Partitioning

Divide a problem into smaller parts for easier solving. Example: In quicksort, elements are partitioned around a pivot.

17. Bit Manipulation

Using bitwise operations for speed and memory efficiency. Example: Checking if a number is even using (n & 1) == 0.

18. Memoization

Store results of previous computations to avoid repetition. Example: In Fibonacci, store fib(5) so you don't recalculate it later.

19. Invariants

A condition that remains true throughout algorithm execution. Example: In bubble sort, after each pass, the largest element is at the end — this remains invariant.

20. Shortest Path Tree

Shows shortest paths from one source to all others in a graph. Example: Using Dijkstra’s algorithm to find shortest delivery routes from one city to many.