1. Divide and conquer is a top-down approach.

- **Answer:** True
- **Explanation:** It starts with the main problem and recursively breaks it down into smaller sub-problems.

2. Dynamic programming is a top-down approach.

- **Answer:** False
- **Explanation:** Classic Dynamic Programming is typically a **bottom-up** approach (tabulation), solving smaller sub-problems first to build the final solution. (Note: Memoization is top-down, but DP is generally contrasted with Top-Down D&C).

3. In divide and conquer, sub problems are independent.

- **Answer:** True
- **Explanation:** The sub-problems are disjoint (do not overlap) and can be solved separately.

4. Subproblems of dynamic programming are dependent and overlapping.

- **Answer:** True
- **Explanation:** DP is specifically designed for cases where sub-problems recur and share common solutions.

5. Divide and conquer is more efficient than dynamic programming because it always finds the best way to go.

- **Answer:** False
- **Explanation:** If sub-problems overlap, Divide and Conquer re-calculates the same things repeatedly, making it extremely inefficient compared to DP.

6. Dynamic programming is more efficient due to the saving of intermediate solutions.

- **Answer:** True

- **Explanation:** It stores results in a table (memoization/tabulation) to avoid redundant computations.

7. On backtracking (trial and error) the partial solutions are saved in a table.

- **Answer:** False
- **Explanation:** Backtracking does not store solutions; it explores paths recursively and undoes steps (backtracks) when needed. DP stores solutions.

8. Merge Sort can be solved using dynamic programming.

- **Answer:** False
- **Explanation:** Merge Sort has **independent** sub-problems, so it uses Divide and Conquer, not Dynamic Programming.

9. Matrix multiplication can be solved using dynamic programming.

- **Answer:** True
- **Explanation:** Matrix Chain Multiplication is a classic DP problem because it has overlapping sub-problems and optimal substructure.

10. Dynamic programming is based on divide-and-conquer and it can be applied only when the problem has an optimal substructure.

- **Answer:** True (with context)
- **Explanation:** It shares the concept of combining sub-problem solutions (optimal substructure), but specifically targets **overlapping** sub-problems.

11. When a backtracking algorithm, when a bad choice has been made, the last choice is undone and it tries another option.

- **Answer:** True
- **Explanation:** This is the definition of "backtracking"—reversing the last step to explore a different path.

12. Backtracking guarantees the complete optimal solution.

- **Answer:** True
- **Explanation:** Since it acts as an optimized brute-force search exploring all valid possibilities, it will find the optimal solution if one exists.

13. Divide and conquer does not use recursion as part of its strategy.

- **Answer:** False
- **Explanation:** Divide and Conquer relies heavily on recursion to break problems down and combine results.

14. A recursive algorithm is an algorithm that calls itself directly or indirectly.

- **Answer:** True
- **Explanation:** That is the standard definition of recursion.

15. Some recursive algorithms do not need a base case, since they stop naturally.

- **Answer:** False
- **Explanation:** Every recursive algorithm requires a base case to terminate; otherwise, it causes infinite recursion (Stack Overflow).

16. A recursive algorithm must change its state toward the base case.

- **Answer:** True
- **Explanation:** The input must change (e.g., get smaller) with each call to eventually reach the base case and stop.

17. Backtracking can solve a wide range of problems due to its brute-force nature.

- **Answer:** True
- **Explanation:** It systematically tries all potential configurations, making it applicable to many constraint satisfaction problems.

18. Backtracking algorithms can learn and reuse solutions of previous executions since it uses memoization.

- **Answer:** False
- **Explanation:** Standard backtracking does not use memoization (caching); if it did, it would be considered Dynamic Programming or Memoized Recursion.