| **Aspect** | **Recursion** | **Backtracking** |
| --- | --- | --- |
| **Definition** | A method where a function calls itself to solve smaller subproblems. | A *refined form of recursion* where we explore possible solutions and **undo (backtrack)** decisions when they don’t lead to a valid or optimal solution. |
| **Purpose** | To divide a problem into smaller subproblems (like factorial, Fibonacci). | To explore all possible configurations/paths and backtrack when a path fails (like generating permutations, solving mazes). |
| **Key Idea** | Breaking a problem into smaller ones. | Searching for all solutions or the right one by trying, rejecting, and undoing steps. |
| **State Reversal?** | ❌ No — recursion just goes down and returns. | ✅ Yes — backtracking *removes* or *undoes* choices to explore other options. |
| **Goal** | Compute a value. | Find one or all valid configurations. |
| **Example Problems** | Factorial, Fibonacci, sum of array, binary search. | N-Queens, Sudoku Solver, Subset/Combination generation, Maze pathfinding. |

Summary:

| **Feature** | **Recursion** | **Backtracking** |
| --- | --- | --- |
| Calls itself | ✅ | ✅ |
| Explores all possibilities | ❌ | ✅ |
| Undoes previous choice | ❌ | ✅ |
| Used for | Divide & conquer | Search & decision problems |
| Example | Factorial | Sudoku, N-Queens |

Difference between dp ,backtracking and backtracking

| **Feature / Situation** | **Recursion** | **Backtracking** | **Dynamic Programming (DP)** |
| --- | --- | --- | --- |
| **Core Purpose** | Break a big problem into smaller subproblems. | Explore all possible choices/configurations and undo wrong ones. | Optimize recursive problems with overlapping subproblems by storing results. |
| **When to Use** | When each subproblem is independent and has no “choice” branching. | When you have multiple “choices” or “paths” to explore, and must find all or valid ones. | When the recursive problem repeats the same subproblems (overlapping). |
| **Problem Nature** | Single path computation (e.g., factorial, sum, binary search). | Multiple choices and constraints (e.g., N-Queens, subsets, permutations, Sudoku). | Optimization or counting problems (e.g., Knapsack, Fibonacci, Grid paths). |
| **Output Expected** | A single result/value. | Multiple valid results or one valid configuration. | Best/maximum/minimum result (optimized). |
| **Key Technique** | Function calls itself once per step. | Function calls itself for every possible decision and “undoes” it. | Recursion + Memoization or Tabulation to avoid recomputation. |
| **Example Keywords in Question** | “Compute”, “Find the result of”, “Return the total”. | “Generate all”, “Find all valid”, “Arrange”, “Combinations”, “Paths”. | “Find minimum/maximum”, “Count number of ways”, “Optimal solution”. |
| **State Reversal (Undo step)?** | ❌ No | ✅ Yes | ❌ No (but stores past results) |
| **Memory Usage** | Moderate (stack only). | High (stack + temporary states). | High (stack + DP array). |
| **Typical Example Problems** | Factorial, Fibonacci (naive). | Subsets, Permutations, N-Queens, Sudoku. | Fibonacci (optimized), Knapsack, Longest Common Subsequence, Coin Change. |
| **Base Case Stops When** | Problem becomes smallest unit. | You reach the end of all decisions. | You reach precomputed state or end condition. |
| **Identifying Clue** | “No multiple choices at each step.” | “Need to explore all choices and backtrack wrong ones.” | “Overlapping subproblems or need to store results for efficiency.” |

**Memory Shortcut :**

| **Check this...** | **If YES → Use** | **Why** | **Example Problems** |
| --- | --- | --- | --- |
| Does the problem have smaller subproblems? | Recursion | Divide into smaller tasks | Factorial, Tree Traversal |
| Do you have multiple choices at each step? | Backtracking | Explore all possibilities | Subsets, N-Queens |
| Do you have overlapping subproblems (reusing results)? | Dynamic Programming | Optimize repeated calculations | Fibonacci, Knapsack |
| Do you need to “undo” a previous step to try another? | Backtracking | Trial and error with undo | Sudoku, Word Search |
| Do you only go deeper without reversing state? | Recursion | Single-direction computation | Binary Search, Merge Sort |
| Do you store previous results for reuse? | Dynamic Programming | Memoization/tabulation | Coin Change, LCS |

Flow to choice which algorithm use for the given problem statement:

START

│

├──► Step 1: Does the problem involve "repeating the same task" on smaller inputs?

│ │

│ ├──► NO → Probably iterative (use loops, not recursion)

│ │

│ └──► YES ↓

│

├──► Step 2: At each step, do you have MULTIPLE CHOICES or BRANCHES to explore?

│ │

│ ├──► NO → It's a single flow of smaller subproblems

│ │ ✅ Use \*\*Recursion\*\*

│ │ (e.g., Factorial, Binary Search, Tree Traversal)

│ │

│ └──► YES ↓

│

├──► Step 3: Do you need to explore ALL possible combinations or configurations?

│ │

│ ├──► YES → Is there a need to "UNDO" wrong choices or backtrack?

│ │ │

│ │ ├──► YES → ✅ Use \*\*Backtracking\*\*

│ │ │ (e.g., Subsets, N-Queens, Permutations, Sudoku)

│ │ │

│ │ └──► NO → Could still be recursion generating outputs

│ │

│ └──► NO ↓

│

├──► Step 4: Are there overlapping subproblems (recalculating same things)?

│ │

│ ├──► YES → ✅ Use \*\*Dynamic Programming\*\*

│ │ (e.g., Fibonacci optimized, Knapsack, LCS)

│ │

│ └──► NO → Simple recursion still works

│──► END