Backtracking — Illustrated Guide

What is Backtracking?

Backtracking is a systematic trial-and-error technique to build solutions incrementally.
At each step (state), choose an option, recurse, and upon return undo the choice (backtrack).
Prune branches that cannot lead to valid solutions.

Core Template

```
def backtrack(state):
    if is_complete(state):
        record_solution(state)
        return
    for choice in choices(state):
        make(choice)
        backtrack(state)
        undo(choice)
```

Key Uses

- Subsets/Combinations/Permutations
- Constraint satisfaction (N-Queens, Sudoku)
- Path generation and decision trees (Letter Combinations)
- Sum/Partition problems with pruning

Heuristics & Pruning

- Sort choices to prune early (e.g., break when candidate > remaining).
- Track constraints with fast checks (sets/bitmasks for used or threatened positions).
- Use bounds (like remain < 0) to cut branches.

Complexities

Backtracking explores an exponential search tree in the worst case.
Time and space vary by problem; typical patterns:

- Subsets: O(2^n * n)
- Permutations: O(n * n!)
- Combination Sum: exponential, with pruning
- N-Queens: exponential; heavy pruning with diagonals/columns sets

Included Files

- backtracking_all_examples.py (combined runner)
- backtracking_example1_subsets.py
- backtracking_example2_permutations.py

- backtracking_example3_combination_sum.py
- backtracking_example4_nqueens.py
- backtracking_example5_letter_combinations.py
- README.txt