# **Leetcode 22 – Generate Parentheses**

## Problem Understanding

Given n pairs of parentheses, generate **all combinations** of well-formed parentheses.

**Constraints:**

* Each open ( must be closed with a ).
* At any point, the number of closing ) must **not exceed** the number of opening (.
* You must use **exactly n open and n close parentheses**.

## Optimized Java Solution (Backtracking)

class Solution {

public List<String> generateParenthesis(int n) {

List<String> result = new ArrayList<>();

backtrack(result, "", 0, 0, n);

return result;

}

private void backtrack(List<String> result, String current, int open, int close, int max) {

if (current.length() == 2 \* max) {

result.add(current);

return;

}

if (open < max) {

backtrack(result, current + "(", open + 1, close, max);

}

if (close < open) {

backtrack(result, current + ")", open, close + 1, max);

}

}

}

## Dry Run (n = 3)

We'll trace the recursive tree path.

### Tree View:

Level 0: ""

/ \

Level 1: "(" [X]

/ \

Level 2: "((" "()"

/ \ \

Level 3: "(((" "(()" "(())"

| | |

"((()))" "(()())" "()(())"

### Combinations for n = 3:

|  |  |  |
| --- | --- | --- |
| Path (Open, Close) | Current String | Valid? |
| (1,0) | ( | ✅ |
| (2,0) | (( | ✅ |
| (3,0) | ((( | ✅ |
| (3,1) | ((() | ✅ |
| (3,2) | ((()) | ✅ |
| (3,3) | ((())) | ✅ Added |
| ... | ... | ... |
| Final Output | ["((()))", "(()())", "(())()", "()(())", "()()()"] |  |

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time Complexity | O(2ⁿ), but generates **Catalan(n)** sequences |
| Space | O(n) recursion depth |
| Output Size | Catalan(n) ≈ (1 / n+1) C(2n, n) |

For n = 3, output size is 5 (Catalan(3) = 5).

## Alternate Approaches

|  |  |  |
| --- | --- | --- |
| Approach | Description | Notes |
| Brute force | Generate all strings of 2n length and check validity | Inefficient for n > 3 |
| Backtracking | ✅ Prunes invalid paths early | Optimal |
| DP (memoized tree building) | Advanced but unnecessary here | Overkill for n ≤ 10 |