# **Leetcode 241 – Different Ways to Add Parentheses**

## Problem Understanding

You are given a string expression of numbers and operators (+, -, \*).  
Return **all possible results** from computing all the different **ways to group numbers and operators** using parentheses.  
You may return the answer in **any order**.

### Example:

Input: "2-1-1"

Output: [0, 2]

Explanation:

(2 - (1 - 1)) = 2

((2 - 1) - 1) = 0

## Optimized Java Solution (Recursion + Memoization)

class Solution {

Map<String, List<Integer>> memo = new HashMap<>();

public List<Integer> diffWaysToCompute(String expression) {

if (memo.containsKey(expression)) return memo.get(expression);

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

for (int i = 0; i < expression.length(); i++) {

char c = expression.charAt(i);

if (c == '+' || c == '-' || c == '\*') {

// Split expression into two parts

List<Integer> left = diffWaysToCompute(expression.substring(0, i));

List<Integer> right = diffWaysToCompute(expression.substring(i + 1));

// Combine all combinations from left and right

for (int l : left) {

for (int r : right) {

if (c == '+') result.add(l + r);

else if (c == '-') result.add(l - r);

else if (c == '\*') result.add(l \* r);

}

}

}

}

// Base case: expression is a number (no operator)

if (result.isEmpty()) {

result.add(Integer.valueOf(expression));

}

memo.put(expression, result);

return result;

}

}

## Dry Run Using Table

### Input: "2\*3-4\*5"

We recursively split at each operator and solve each subexpression.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Expression | Split at | Left | Right | Resulting Expressions | Output |
| 2\*3-4\*5 | \* | 2 | 3-4\*5 | 2 \* result(3-4\*5) | … |
| 3-4\*5 | - | 3 | 4\*5 | 3 - result(4\*5) | … |
| 4\*5 | \* | 4 | 5 | 4 \* 5 = 20 | [20] |
| 3-20 | - |  |  | -17 | [-17] |
| 2 \* -17 | - |  |  | -34 | [-34] |
| … |  |  |  | more groupings… | … |

✅ Final output: [-34, -14, -10, -10, 10]  
(All valid evaluations using different parenthesis groupings)

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | **Exponential**: O(2ⁿ × n) (in worst case) |
| Space | O(2ⁿ × n) (memoization cache + recursion depth) |

* Each operator can create two branches → **binary recursion tree**
* n = number of operators

## Alternate Approaches

### 1. ****Pure Recursion (No Memoization)****

Same logic, but without caching — results in **redundant recomputation**.

### 2. ****DP with Tabulation****

Possible for fixed-length expression, but hard to implement cleanly due to string parsing.

### 3. ****Postfix Evaluation + Parenthesis Generator****

Generate all parenthesis placements, convert to postfix, then evaluate — not as intuitive.