LeetCode 77. Combinations

1. Problem Title & Link

- **77. Combinations**
- 🔗 https://leetcode.com/problems/combinations/

2. Problem Statement (Short Summary)

Given two integers n and k, return **all possible combinations** of k numbers chosen from the range [1, n].

Return the answer in **any order**.

3. Examples (Input → Output)

Input: n = 4, k = 2

Output:

[

[1,2],

[1,3],

[1,4],

[2,3],

[2,4],

[3,4]

]

Input: n = 1, k = 1

Output: [[1]]

4. Constraints

- 1 <= n <= 20
- 1 <= k <= n

5. Thought Process (Step by Step)

This is a **backtracking problem** ❤️

We need to **build all possible subsets** of size k from numbers [1...n] in increasing order.

🧠 Step 1: Recursive Choice

For each number i in [start...n]:

1. Include i in the current combination.
2. Recurse with i + 1 as the new starting number.
3. Backtrack (remove i) to explore other options.

When the current combination size = k → add it to result.

🧩 Step 2: Avoid Duplicates

Always move forward (start = i + 1) so we never reuse previous numbers.
This ensures combinations (not permutations).

6. Pseudocode

```
result = []

def backtrack(start, path):
    if len(path) == k:
        result.append(copy(path))
        return

    for i in range(start, n + 1):
        path.append(i)
        backtrack(i + 1, path)
        path.pop()

backtrack(1, [])
return result
```

7. Code Implementation

✅ **Python**

```
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        res = []

        def backtrack(start, path):
            if len(path) == k:
                res.append(path[:])
                return
            for i in range(start, n + 1):
                path.append(i)
                backtrack(i + 1, path)
                path.pop()

        backtrack(1, [])
        return res
```

✅ **Java**

```
class Solution {
    public List<List<Integer>> combine(int n, int k) {
        List<List<Integer>> res = new ArrayList<>();
        backtrack(1, n, k, new ArrayList<>(), res);
        return res;
    }

    private void backtrack(int start, int n, int k, List<Integer> path,
List<List<Integer>> res) {
        if (path.size() == k) {
            res.add(new ArrayList<>(path));
            return;
        }
        for (int i = start; i <= n; i++) {
            path.add(i);
            backtrack(i + 1, n, k, path, res);
            path.remove(path.size() - 1);
        }
    }
}
```

8. Time & Space Complexity

- **Time:** O(C(n, k) * k) → total combinations × cost to copy each
- **Space:** O(k) recursion stack (depth of each combination)

9. Dry Run (Step-by-Step Execution)

👉 Input: n = 4, k = 2

Step	path	start	Action	Result
1	[]	1	Add 1 → recurse	—
2	[1]	2	Add 2 → recurse	—
3	[1,2]	3	✅ Add to result	[[1,2]]
4	[1]	3	Add 3 → recurse	—
5	[1,3]	4	✅ Add	[[1,2],[1,3]]
6	[1]	4	Add 4 → ✅ Add	[[1,2],[1,3],[1,4]]
7	[]	2	Add 2 → recurse	—
8	[2,3]	4	✅ Add	[[1,2],[1,3],[1,4],[2,3]]

9	[2,4]	5	✅ Add	[[...],[2,4]]
10	[3,4]	✅ Add	Final Output	

✅ Output: [[1,2],[1,3],[1,4],[2,3],[2,4],[3,4]]

10. Concept Insight Table

Core Concept	Common Use Cases	Common Traps	Builds / Next Steps
Backtracking (DFS Tree Exploration) — generate all valid combinations by exploring choices recursively and undoing them.	- Combinations / subsets - Permutations - Parentheses / Sudoku / N-Queens	- Forgetting to pop after recursion - Using same start index repeatedly - Shallow copying list (mutates results)	🔷 Builds to LC 78 (Subsets), LC 46 (Permutations) 🔷 Connects to DFS + State Space Tree 🔷 Foundation for constraint-satisfaction problems

11. Common Mistakes / Edge Cases

- Forgetting to path.pop() after recursion → duplicates in result.
- Modifying path without copying (path[:] in Python).
- Off-by-one errors in loop (i <= n must include n).

12. Variations / Follow-Ups

- **LC 78:** Generate all subsets (all possible sizes).
- **LC 46:** Generate all permutations (change order).
- **LC 39 / 40:** Combination Sum (adds target sum constraint).
- **LC 216:** Combination Sum III (limit + target).