**Pattern 1:** If the given input is a sorted (array, list, or matrix), then we will be using a variation of Binary Search or a Two Pointers strategy

**Pattern 2:** If we are dealing with top/maximum/minimum/closest 'k' elements among 'n' elements, we will be using a Heap

**Pattern 3:** If we need to try all combination (or permutations) of the input, we can either use recursive Backtracking or iterative Breadth-First Search.

**Pattern 4:** Most of the questions related toTrees or Graphs can be solved either through Breadth-First Search or Depth-First Search.

**Pattern 5:** Every recursive solution can be converted to an iterative solution using a stack.

**Pattern 6:** If for a problem, there exists a brute-force solution in $O(n^2)$ time and O(1) space, there must exist two other solutions: 1) Using a `Map` or a `Set` for O(n) time and O(n) space, 2) Using sorting for O(n log n) time and O(1) space.

**Pattern 7:** If the problem is asking for optimization (e.g., maximization or minimization), we will need to use **Dynamic Programming** to solve it.

**Pattern 8:** If we need to find some common substring among a set of strings, we will be using a `HashMap` or a `Trie`.

**Pattern 9:** If we need to search among a bunch of strings, `Trie` will be the best datastructure.

**Pattern 10:** If the problem involves a `LinkedList` and we can't use extra space, then use Fast & Slow Pointer approach.