**Patterns:**

**// Optimizing backtracking**

    If it is a selecting problem, if you use a hashmap (for bitmasking essentially) to check if that combination has been tried earlier and use previous result, you get drastically optimized results.

**// Knapsack variations**

**0/1 fractional Knapsack – Greedy**

**0/1 Knapsack – DP Subset-sum / Tabular**

**0/1 Unbounded Knapsack – DP Tabular**

**// If k or kth smallest or largest elements needed, not in order**

    QuickSelect / std::nthelement C++

Note: Generally seen in Heap questions

O(n) Avg O(n^2) Worst

**// If we want to sort based on freq counts and the max freq is known, we can use bucketSort for sorting in O(n).**

**// In meeting room questions if we only want to find maximum number of meetings that can be held in one room, sort by end times, if we want to find min number of meeting rooms to accommodate ALL meetings, sort by start time.**

**// To serialize a tree such that we can uniquely identify it**

    We need a char for null values eg. ‘!’

After each value add a char eg. ‘#’

Only preorder and postorder will work

**// For interval-related questions, sorting is required.**

**Bit Manipulation:**

**// XORing removes duplicate pairs**

**//n&(n-1) flips n’s least significant 1 bit. Can be used to find no. of 1 bits in a number.**

**//Strings (especially character occurrence counts) can be converted into an integer instead of a vector or hashmap. Direct int-to-int comparisons can be done to reduce the O(n) time to O(1).**

**General:**

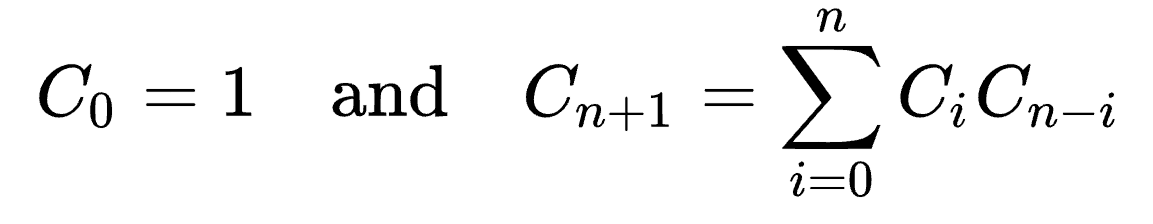
**// For a N-ary tree having height h**

    Number of nodes = Nh+1

**// Number of possible BST combinations from n unique numbers is**

2nCn/(n+1) which is called a Catalan number.

Catalan can be calculated efficiently as:

\_

to prevent overflows.