Recursion
Exercise 2

1. **Triple Step**: A child is running up a staircase with n steps and can hop either 1 step, 2 steps, or 3 steps at a time. Implement a method to count how many possible ways the child can run up the stairs.

2. **Robot in a Grid**: Imagine a robot sitting on the upper left corner of grid with r rows and c columns. The robot can only move in two directions, right and down, but certain cells are "off limits" such that the robot cannot step on them. Design an algorithm to find a path for the robot from the top left to the bottom right.

3. **Magic Index**: A magic index in an array A[0...n-1] is defined to be an index such that A[i] = i. Given a sorted array of distinct integers, write a method to find a magic index, if one exists, in array A.
   FOLLOW UP: What if the values are not distinct?

4. **Power Set**: Write a method to return all subsets of a set.

5. **Recursive Multiply**: Write a recursive function to multiply two positive integers without using the * operator. You can use addition, subtraction, and bit shifting, but you should minimize the number of those operations.

6. **Towers of Hanoi**: In the classic problem of the Towers of Hanoi, you have 3 towers and N disks of different sizes which can slide onto any tower. The puzzle starts with disks sorted in ascending order of size from top to bottom (i.e., each disk sits on top of an even larger one). You have the following constraints:
   (1) Only one disk can be moved at a time.
   (2) A disk is slid off the top of one tower onto another tower.
   (3) A disk cannot be placed on top of a smaller disk.
   Write a program to move the disks from the first tower to the last using stacks.

7. **Permutations without Dups**: Write a method to compute all permutations of a string of unique characters.

8. **Permutations with Dups**: Write a method to compute all permutations of a string whose characters are not necessarily unique. The list of permutations should not have duplicates.

9. **Parens**: Implement an algorithm to print all valid (e.g., properly opened and closed) combinations of n pairs of parentheses.
   EXAMPLE
   Input: 3
   Output: ((())), (()()) (())(), ()(()), ()()()

10. **Paint Fill**: Implement the "paint fill" function that one might see on many image editing programs. That is, given a screen represented by a two-dimensional array of colours), a point, and a new colour, fill in the surrounding area until the colour changes from the original colour.

11. **Coins**: Given an infinite number of quarters (25 cents), dimes (10 cents), nickels (5 cents), and pennies (1 cent), write code to calculate the number of ways of representing n cents.

12. **Eight Queens**: Write an algorithm to print all ways of arranging eight queens on an 8x8 chess board so that none of them share the same row, column, or diagonal. In this case, "diagonal" means all diagonals, not just the two that bisect the board.

13. **Stack of Boxes**: You have a stack of n boxes, with widths w_i, heights h_i, and depths d_i. The boxes cannot be rotated and can only be stacked on top of one another if each box in the stack is strictly larger than the box above it in width, height, and depth. Implement a method to compute the height of the tallest possible stack. The height of a stack is the sum of the heights of each box.

14. **Boolean Evaluation**: Given a Boolean expression consisting of the symbols 0 (false), 1 (true), & (AND), | (OR), and ^ (XOR), and a desired Boolean result value result. Implement a function to count the number of ways of parenthesizing the expression such that it evaluates to result.
    EXAMPLE
    countEval("1^0|0|1", false) -> 2
    countEval("0&0&0&1^1|0", true) -> 10