Technique: Permutations/Combinations using Auxiliary Buffer

Level: Medium

Coin Change Problem: Given a set of coin denominations, print out the different ways you can make a target amount. You can use as many coins of each denomination as you like.

For example: If coins are [1,2,5] and the target is 5, output will be:

[1,1,1,1,1]
[1,1,1,2]
[1,2,2]
[5]

Questions to Clarify:
Q. Does [1,2] and [2,1] count as one, i.e, do we care about permutations?
A. No, we only care about combinations, so [1,2] and [2,1] will count as the same.

Q. Can we assume that all coins will be integers greater than 0?
A. Yes

Solution:
This problem is similar to generating combinations, except that a number can be repeated several times. So, while picking candidates that go into the buffer, we will also consider the previous candidate that went in.

Note: In this solution, we are using a stack as a buffer. You can also use an array of size target.

Pseudocode:
```
printCoinsHelper(coins, target, stack buffer, startIndex, currentSum)
   if currentSum > target
         return
   if currentSum == target // we have found an answer
         print buffer
         return

   for i: startIndex to coins.length
         place coin onto buffer
         // recurse
         printCoinsHelper(coins, target, buffer, i, currentSum + coins[i])
         remove coin from buffer
```

Test Cases:
Edge Cases: Coins array is null/empty, Target is 0 or -ve.
Base Cases: Coins array is size 1
Regular Cases: Target is equal to a coin, target is greater than all coins

Time Complexity: Factorial Complexity

Note: Typically, interviewers will not ask the time complexity of such complex recursions.

This one is particularly complex. We are generating combinations with repetitions, which is hard to calculate even in combinatorics. It comes down to a formula that involves factorials. If you're interested, you can read more about it here, but we suggest not spending too much time on this unless you're just curious.

Space Complexity: O(Target), because in the worst case, both our buffer and the recursion stack will be the size of the target. (if we use all 1's to make target)

```
public static void printCoins(int[] coins, int target) {
    if (coins == null || coins.length == 0 || target <= 0)
        return;

    printCoinsHelper(coins, target, 0, new Stack<Integer>(), 0);
}

public static void printCoinsHelper(int[] coins, int target, int startIndex,
Stack<Integer> buffer, int sum) {
    // termination cases
    if (sum > target) {
        return;
    }
    if (sum == target) {
        print(buffer);
        return;
    }

    // find candidates that go into buffer
    for(int i = startIndex; i < coins.length; i++) {
        // place candidate into buffer and recurse
        buffer.push(coins[i]);
        printCoinsHelper(coins, target, i, buffer, sum + coins[i]);
        buffer.pop();
    }
}
```