Recursion Assignment Questions

Problem 1:

Print all possible strings of length `k` that can be formed from a set of `n` characters.

Example 1:

```
Input:
set[] = {'a', 'b'}, k = 3
Output:
aaa
aab
aba
abb
baa
bab
bba
bbb
```

Example 2:

```
Input:
set[] = {'a', 'b', 'c', 'd'}, k = 1
Output:
a
b
c
d
```

Constraints:

```
0 < n <= 10
0 < k <= 5
```

Problem 2:

Find the number of unique paths in a `n X m` grid, starting from `[0,0]` to `[n-1,m-1]`, where movement is only allowed either 1 cell down or 1 cell right at a time.

Example 1:

```
Input: m = 3, n = 2
Output: 3
Explanation: From the top-left corner, there are a total of 3 ways to reach the bottom
1. Right -> Down -> Down
2. Down -> Down -> Right
3. Down -> Right -> Down
```

Example 2:

```
Input: m = 3, n = 1
Output: 1
Explanation: From the top-left corner, there is exactly 1 way to reach the bottom-righ
1. Right -> Right -> Right
```

Constraints:

```
1 <= m, n <= 100
```

Problem 3:

The head of a singly linked-list is given. The list can be represented as:
`L0 → L1 → … → Ln - 1 → Ln`
Reorder the list to be on the following form:
`L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …`
The values in the list's nodes may not be modified, only nodes themselves may be changed.

Example 1:

```
Input: head = [1,2,3,4]
Output: [1,4,2,3]
```

Example 2:

```
Input: head = [1,2,3,4,5]
Output: [1,5,2,4,3]
```

Constraints:

```
The number of nodes in the list is in the range [1, 5 * 104].
1 <= Node.val <= 1000
```

Problem 4:

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 and print the moves.

Example 1:

```
Input: N = 3
Output:
Move disk from rod 1 to rod 3
Move disk from rod 1 to rod 2
Move disk from rod 3 to rod 2
Move disk from rod 1 to rod 3
Move disk from rod 2 to rod 1
Move disk from rod 2 to rod 3
Move disk from rod 1 to rod 3
```

Constraints:

```
1 <= n <= 10
```

Problem 5:

Given an integer `n`, return a list of all possible full binary trees with `n` nodes. Each node of each tree in the answer must have `Node.val == 0`.
Each element of the answer is the root node of one possible tree. You may return the final list of trees in any order.
A full binary tree is a binary tree where each node has exactly 0 or 2 children.

Example 1:

```
Input: n = 3
Output: [[0,0,0]]
```

Example 2:

```
Input: n = 7
Output: [[0,0,0,null,null,0,0,null,null,0,0],[0,0,0,null,null,0,0,0,0],[0,0,0,0,0,0,0],
```

Constraints:

```
1 <= n <= 20
```

Problem 6:

Print sums of all subsets of a given set of size `n`.

Example 1:

```
Input: arr[] = {2, 3}
```

```
Output: 0 2 3 5
```

Example 2:

```
Input: arr[] = {2, 4, 5}
Output: 0 2 4 5 6 7 9 11
```

Constraints:

```
1 <= n <= 15
```

Problem 7:

0/1 Knapsack Problem: We are given `N` items where each item has some weight and profit associated with it. We are also given a bag with capacity `W`, [i.e., the bag can hold at most `W` weight in it]. The target is to put the items into the bag such that the sum of profits associated with them is the maximum possible.
The constraint here is we can either put an item completely into the bag or cannot put it at all [It is not possible to put a part of an item into the bag].

Example 1:

```
Input: N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}
Output: 3
Explanation: There are two items which have weight less than or equal to 4. If we sele
```

Example 2:

```
Input: N = 3, W = 50, profit[] = {60, 100, 120}, weight[] = {10, 20, 30}
Output: 220
```

Constraints:

```
1 <= N <= 10
1 <= W <= 50
```

Problem 8:

Given a string `s`, partition `s` such that every string of the partition is a palindrome. Return all possible palindrome partitioning of `s`.

Example 1:

```
Input: s = "bcc"
Output: [["b", "c", "c"], ["b", "cc"]]
```

Example 2:

```
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
```

Constraints:

```
1 <= s.length <= 16
s contains only lowercase English letters.
```