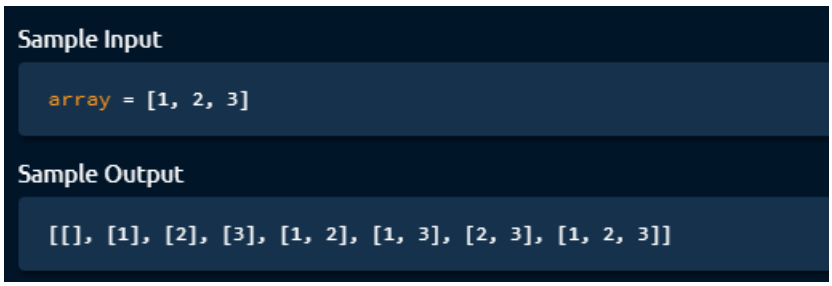
Topic - Recursion

Easy – Difficulty: Powerset

Write a function that takes in an array of unique integers and returns its powerset. The powerset P(X) of a set X is the set of all subsets of X. For example, the powerset of [1,2] is [[], [1], [2], [1,2]].

Note that the sets in the powerset do not need to be in any particular order.



PowerSet Hint:

Try thinking about the base cases. What is the powerset of the empty set? What is the powerset of sets of length 1?

If you were to take the input set X and add an element to it, how would the resulting powerset change?

Can you solve this problem recursively? Can you solve it iteratively? What are the advantages and disadvantages of using either approach?

Time-Space Complexity:
O(n*2^n) time | O(n*2^n) space - where n is the length of the input array

Solution:

```python
# O(n*2^n) time | O(n*2^n) space
def powerset(array, idx=None):
    if idx is None:
        idx = len(array) - 1
    if idx < 0:
        return [[]]
    ele = array[idx]
    subsets = powerset(array, idx - 1)
    for i in range(len(subsets)):
        currentSubset = subsets[i]
        subsets.append(currentSubset + [ele])
    return subsets
```

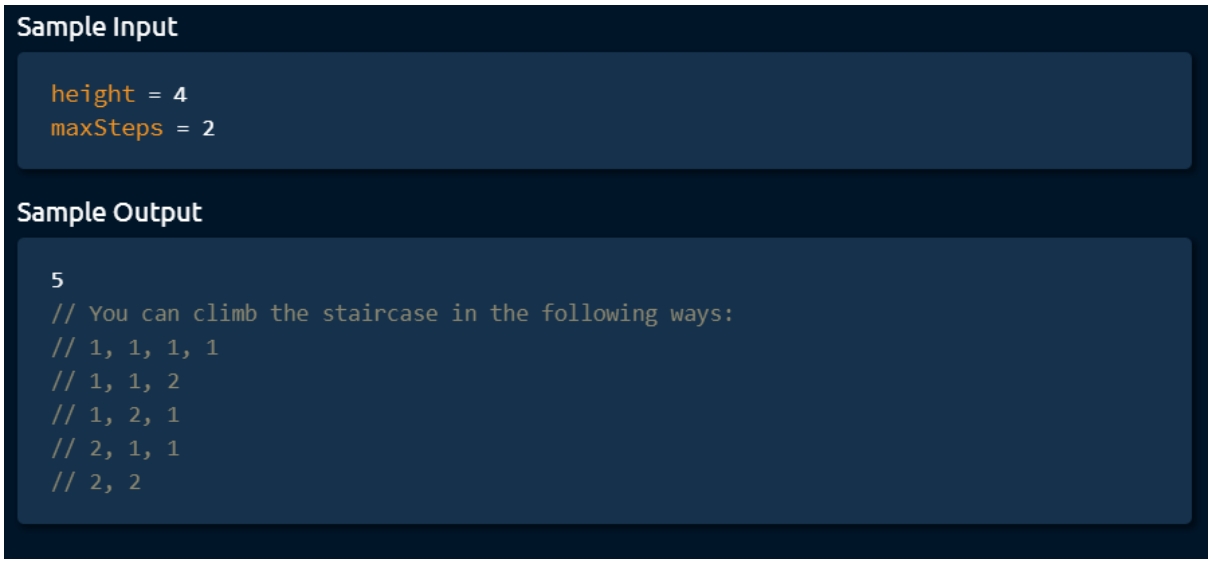
```python
# O(n*2^n) time | O(n*2^n) space
def powerset(array):
    subsets = [[]]
    for ele in array:
        for i in range(len(subsets)):
            currentSubset = subsets[i]
            subsets.append(currentSubset + [ele])
    return subsets
```

Easy – Difficulty: Staircase Traversal

You're given two positive integers representing the height of a staircase and the maximum number of steps that you can advance up the staircase at a time. Write a function that returns the number of ways in which you can climb the staircase.

For example, if you were given a staircase of height = 3 and maxSteps = 2 you could climb the staircase in 3 ways. You could take 1 step, 1 step, then 1 step,

you could also take 1 step, then 2 steps , and you could take 1 step, then 2 steps , and you could take 2 steps, then 1 step



Staircase Traversal Hint:

If you can advance 2 steps at a time, how many ways can you reach a staircase of height 1 and of height 2? Think

recursively.

if you know the number of ways to climb a staircase of height 1 and of height 2, how many ways are there to climb a staircase of height 3 (assuming the same max steps of 2)?

The number of ways to climb a staircase of height k with a max number of steps s is: numWays[k - 1] + numWays[k - 2] + ... + numWays[k - s] . This is because if you can advance between 1 and s steps, then from each step k - 1, k - 2, ..., k - s, you can directly advance to the top of a staircase of height k. By adding the number of ways to reach all steps that you can directly advance to the top step from, you determine how many ways there are to reach the top step.

Time-Space Complexity:

O(n) time | O(n) space - where n is the height of the staircase

Solution:

```python
# O(k^n) time | O(n) space - where n is the height of the staircase and k is the number of allowed steps
def staircaseTraversal(height, maxSteps):
    return numberOfWaysToTop(height, maxSteps)


def numberOfWaysToTop(height, maxSteps):
    if height <= 1:
        return 1

    numberOfWays = 0
    for step in range(1, min(maxSteps, height) + 1):
        numberOfWays += numberOfWaysToTop(height - step, maxSteps)

    return numberOfWays
```

```python
# O(n * k) time | O(n) space - where n is the height of the staircase and k is the number of allowed steps
def staircaseTraversal(height, maxSteps):
    waysToTop = [0 for _ in range(height + 1)]
    waysToTop[0] = 1
    waysToTop[1] = 1

    for currentHeight in range(2, height + 1):
        step = 1
        while step <= maxSteps and step <= currentHeight:
            waysToTop[currentHeight] = waysToTop[currentHeight] + waysToTop[currentHeight - step]
            step += 1

    return waysToTop[height]
```

Medium – Difficulty: Phone Number Mnemonics

If you open the keypad of your mobile phone, it'll likely look like this:

Almost every digit is associated with some letters in the alphabet; this allows certain phone numbers to spell out actual words. For example, the phone number 8464747328 can be written as timisgreat ; similarly, the phone number 2686463 can be written as antoine or as ant6463.

It's important to note that a phone number doesn't represent a single sequence of letters, but rather multiple combinations of letters. For instance, the digit 2 can represent three different letters (a, b, and c).

A mnemonic is defined as a pattern of letters, ideas, or associations that assist in remembering something. Companies oftentimes use a mnemonic for their phone number to make it easier to remember.

Given a stringified phone number of any non-zero length, write a function that returns all mnemonics for this phone number, in any order.

For this problem, a valid mnemonic may only contain letters and the digits 0 and 1. In other words, if a digit is able to be represented by a letter, then it

must be. Digits 0 and 1 are the only two digits that don't have letter representations on the keypad.

Note that you should rely on the keypad illustrated above for digit-letter associations.

Sample Input

```
phoneNumber = "1905"
```

Sample Output

```
[
  "1w0j",
  "1w0k",
  "1w0l",
  "1x0j",
  "1x0k",
  "1x0l",
  "1y0j",
  "1y0k",
  "1y0l",
  "1z0j",
  "1z0k",
  "1z0l",
]
// The mnemonics could be ordered differently.
```

Phone Number Mnemonics Hint:

The first thing you'll need to do is create a mapping from digits to letters. You can do this by creating a hash table mapping all string digits to lists of their respective characters.

This problem can be solved fairly easily using recursion. Try generating all characters for the first digit in the phone number one at a time, and for each character, recursively performing the same action on the next digit, and then on the digit after that, and so on and so forth until you've done so for all digits in the phone number.

You can recursively generate characters one digit at a time and store the intermediate results in an array. Once you've reached the last digit in the phone number, you can add the currently generated mnemonic (stored in the previously mentioned array) to a final array that stores all the results.

Time-Space Complexity:

O(4^n * n) time | O(4^n * n) space - where n is the length of the phone number

Solution:

```
# n is the length of the phone number
def phoneNumberMnemonics(phoneNumber):
    currentMnemonic = ["0"] * len(phoneNumber)
    mnemonicsFound = []

    phoneNumberMnemonicsHelper(0, phoneNumber, currentMnemonic, mnemonicsFound)
    return mnemonicsFound


def phoneNumberMnemonicsHelper(idx, phoneNumber, currentMnemonic, mnemonicsFound):
    if idx == len(phoneNumber):
        mnemonic = "".join(currentMnemonic)
        mnemonicsFound.append(mnemonic)
    else:
        digit = phoneNumber[idx]
        letters = DIGIT_LETTERS[digit]
        for letter in letters:
            currentMnemonic[idx] = letter
            phoneNumberMnemonicsHelper(idx + 1, phoneNumber, currentMnemonic, mnemonicsFound)


DIGIT_LETTERS = {
    "0": ["0"],
    "1": ["1"],
    "2": ["a", "b", "c"],
    "3": ["d", "e", "f"],
    "4": ["g", "h", "i"],
    "5": ["j", "k", "l"],
    "6": ["m", "n", "o"],
    "7": ["p", "q", "r", "s"],
    "8": ["t", "u", "v"],
    "9": ["w", "x", "y", "z"],
}
```