1019. Next Greater Node In Linked List
We will use a stack to store all previous index that hasn't found larger number. Every time we will compare current val and arr[stack[-1]] and assign res associated value if applicable. TC is O(n)

```
class Solution:
    def nextLargerNodes(self, head: ListNode) -> List[int]:
        arr = []
        stack = []
        while head:
            arr.append(head.val)
            head = head.next
        res = [0] * len(arr)
        for i, val in enumerate(arr):
            while stack and val > arr[stack[-1]]:
                res[stack.pop()] = val
            stack.append(i)
        return res
```

901. Online Stock Span
We will use a stack to record previous val and count. Every time we will compare from tail to head and accumulate all count of qualified number. TC is O(n)

```
class StockSpanner:

    def __init__(self):
        self.stack = []
    def next(self, price: int) -> int:
        res = 0
        while self.stack and price >= self.stack[-1][0]:
            res += self.stack[-1][1]
            self.stack.pop()
        self.stack.append([price, res + 1])
        return res + 1
```

208. Implement Trie (Prefix Tree)
We will use dict to implement trie and use ‘isword’ to mark whether it’s the end of a word. TC is O(len(word))

```
from collections import defaultdict
class TrieNode:
    def __init__(self):
        self.children = defaultdict(TrieNode)
        self.isWord = False

class Trie:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.root = TrieNode();
    def insert(self, word: str) -> None:
        """
        Inserts a word into the trie.
        """
        node = self.root
        for i in word:
            node = node.children[i]
        node.isWord = True

    def search(self, word: str) -> bool:
        """
        Returns if the word is in the trie.
        """
        node = self.root
        for i in word:
            if i not in node.children:
                return False
            node = node.children[i]
        return node.isWord

    def startsWith(self, prefix: str) -> bool:
        """
        Returns if there is any word in the trie that starts with
the given prefix.
        """
        node = self.root
        for i in prefix:
            if i not in node.children:
                return False
            node = node.children[i]
        return True
```

17. Letter Combinations of a Phone Number
We will use dfs to solve this question. We will iterate through all digit’s letters and append it to the current one. Until cur’s length is equal to digit’s length. TC is O(len(digits))

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        mapping = {'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
                   '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'}
        res = []
        if not digits:
            return res
        def dfs(i, cur):
            if len(cur) == len(digits):
                res.append(cur)
                return
            for e in mapping[digits[i]]:
                dfs(i + 1, cur + e)
        dfs(0, '')
        return res
```

46. Permutations

We will use bfs to iterate through all possible previous combinations and insert current num to every position of previous num. TC is O(n * n!)

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        res = [[]]
        for num in nums:
            next_ite = []
            length = len(res[0]) + 1
            for e in res:
                for i in range(length):
                    next_ite.append(e[:i] + [num] + e[i:])
            res = next_ite
        return res
```