1. Copy List with Random Pointer
   We will go through the whole linkedlist twice, the first time, we will use a memo to new nodes associated with the original nodes and finish all next field. In the second round, we will finish random with the help of memo. TC is O(n)

```
class Solution:
    def copyRandomList(self, head: 'Node') -> 'Node':
        memo = {}
        dummy = Node(0, None, None)
        dummy_mem = dummy
        head_mem = head
        while head:
          dummy.next = Node(head.val, None, None)
          memo[head] = dummy.next
          dummy = dummy.next
          head = head.next
        head = head_mem
        dummy = dummy_mem.next
        while head:
          if head.random:
            dummy.random = memo[head.random]
          head = head.next
          dummy = dummy.next
        return dummy_mem.next
```

2. Word Ladder
   We will use bfs to use all letters to replace every bit of word and check whether there is any word in endWordSet, every time we will compare startWordSet and endWordSet, and iterate through the shorter one. TC is O(n)

```
 from collections import defaultdict
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
        words = 'abcdefghijklmnopqrstuvwxyz'
        length = len(beginWord)
        visited = set([beginWord, endWord])
        beginWordSet = set([beginWord])
        endWordSet = set([endWord])
        wordListSet = set(wordList)
        count = 1
        if endWord not in wordListSet:
          return 0
        while beginWordSet:
          next_ite = set()
          count += 1
          for word in beginWordSet:
            for i in range(length):
              for l in words:
                newWord = word[:i] + l + word[i + 1:]
                if newWord in wordListSet:
                  if newWord in endWordSet:
                    return count
                  if newWord not in visited:
                    visited.add(newWord)
                    next_ite.add(newWord)
          beginWordSet = next_ite
          if len(beginWordSet) > len(endWordSet):
            beginWordSet, endWordSet = endWordSet, beginWordSet
        return 0
```

3. Letter Combinations of a Phone Number

We will use back trace to append each word to each word from the previous iteration. TC is O(3**n)

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

4. Generate Parentheses

We will create parentheses by using recursion. Every time if l > r, we could append ‘)’ or ‘(’, if l == r, we could only append ‘(’, when l == r == n, we will append our valid parentheses to result. O(C) is O(2^n)

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        result = []
        self.helper('', 0, 0, n, result)
        return result
    def helper(self, cur, l, r, n, result):
      if l == r and l == n:
        result.append(cur)
        return
      elif l > n or r > n:
```

```
      return
    if l < n:
      self.helper(cur + '(', l + 1, r, n, result)
    if l > r:
      self.helper(cur + ')', l, r + 1, n, result)
```

5. Combination Sum

We will use dfs to try all different number combination in candidates. If the sum is equal to target, we will append it to our result, if it’s less than our sum, we will keep finding from the current index to prevent duplication.

```
class Solution:
  def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
    result = []
    self.dfs([], result, 0, candidates, target)
    return result

  def dfs(self, cur, result, lastIndex, candidates, target):
    cur_sum = sum(cur)
    for idx in range(lastIndex, len(candidates)):
      if cur_sum + candidates[idx] == target:
        result.append(cur + [candidates[idx]])
      elif cur_sum + candidates[idx] < target:
        self.dfs(cur + [candidates[idx]], result, idx, candidates, target)
```