1. Sherlock and Anagrams

   We will iterate all substrings and sort them, using a hashmap to record all presenting number. TC is O(length ** 2)

```
def sherlockAndAnagrams(s):
 memo = defaultdict(int)
 length = len(s)
 result = 0
 for i in range(length):
   for j in range(i + 1, length + 1):
     memo[''.join(sorted(s[i:j]))] += 1
 for val in memo.values():
   result += val * (val - 1) // 2
 return result
```

2. Recursive Digit Sum

   We will add all digits iteratively until the result is one digit. We will get digits first and then multiply k to reduce time we used.

```
num_table = {'0':0, '1':1, '2':2, '3':3, '4':4, '5':5, '6':6, '7':7, '8':8, '9':9}
def superDigit(n, k):
 num = n
 while len(num) > 1:
  result = 0
  for i in num:
   result += num_table[i]
  result *= k
  k = 1
  num = str(result)
 return int(num)
```

3. 3D Surface Area

   We will add up all length difference between two adjacent pillars and in the end, adding all outside appearance.TC is O(m*n)

```
def surfaceArea(A):
 rows, cols = len(A), len(A[0])
 result = rows * cols * 2
 max_col_row = [0] * (rows + cols)
 for j in range(cols):
  for i in range(rows):
   if 0 < i:
     result += abs(A[i - 1][j] - A[i][j])
   if 0 < j:
     result += abs(A[i][j - 1] - A[i][j])
   if i == 0:
     result += A[i][j]
   if i == rows - 1:
     result += A[i][j]
   if j == 0 :
     result += A[i][j]
   if j == cols - 1:
     result += A[i][j]
 return result
```

4. Matrix Layer Rotation
   We will rotate it layer by layer and compute its forward step and move the whole layer forward x steps. TC is O(MN)

```
def matrixRotation(matrix, r):
  rows, cols = len(matrix), len(matrix[0])
  step_r = r
  l, b, r, t = 0, rows - 1, cols - 1, 0
  mark = 0
  direction = [[1, 0], [0, 1], [-1, 0], [0, -1]]
  while rows >= 2 and cols >= 2:
    time = rows * 2 + (cols - 2) * 2
    step = step_r % time
    result = []
    i, j = t, l
    for k in range(time):
      if i == t:
        if j == l:
          mark = 1
        else:
          mark = 4
      elif i == b:
        if j == r:
          mark = 3
        else:
          mark = 2
      if j == l:
        if i == b:
          mark = 2
        else:
          mark = 1
      elif j == r:
        if i == t:
          mark = 4
        else:
          mark = 3
      result.append(matrix[i][j])
      d_i, d_j = direction[mark - 1]
      i, j = i + d_i, j + d_j
    if step > rows - 1:
      step = step - rows + 1
      if step > cols - 1:
        step = step - cols + 1
        if step > rows - 1:
          step = step - rows + 1
          i, j = t, r - step
          mark = 4
        else:
          i, j = b - step, r
          mark = 3
      else:
        i, j = b, l + step
        mark = 2
    else:
      i, j = t + step, l
      mark = 1
    for k in range(time):
      if i == t:
        if j == l:
          mark = 1
        else:
          mark = 4
      elif i == b:
        if j == r:
          mark = 3
        else:
          mark = 2
      if j == l:
        if i == b:
          mark = 2
        else:
          mark = 1
      elif j == r:
        if i == t:
            mark = 4
          else:
            mark = 3
        matrix[i][j] = result[k]
        d_i, d_j = direction[mark - 1]
        i, j = i + d_i, j + d_j
      rows -= 2
      cols -= 2
      l, b, r, t = l + 1, b - 1, r - 1, t + 1
    for e in matrix:
      print(' '.join(map(str, e)))
```

5. Missing Number

   We know the boundary of the array, so we only need to calculate the difference of two sums, we will get the missing number. TC is O(n), SC is O(1)

   ```
   class Solution:
       def missingNumber(self, nums: List[int]) -> int:
           return sum(range(len(nums) + 1)) - sum(nums)
   ```