1180. Count Substrings with Only One Distinct Letter
We will use a dict to record all number of keys. Then we will add all possible combinations to result. TC is O(n)

```
from collections import defaultdict
class Solution:
    def countLetters(self, S: str) -> int:
        memo = defaultdict(int)
        count = 1
        result = 0
        for idx in range(1, len(S)):
          if S[idx - 1] == S[idx]:
            count += 1
          else:
            memo[count] += 1
            count = 1
        memo[count] += 1
        for key, val in memo.items():
          result += (key + 1) * key // 2 * val
        return result
```

1134. Armstrong Number
We will add all digits of numbers and compare with the original number. TC is O(len(n))

```
class Solution:
    def isArmstrong(self, N: int) -> bool:
        num_str = str(N)
        length = len(num_str)
        result = 0
        for i in num_str:
          result += (ord(i) - ord('0')) ** length
        return result == N
```

938. Range Sum of BST
We will iterate all nodes and add all values that are in [L,R], and then keep recuring branches within this interval. TC is O(n)

```
class Solution:
    def rangeSumBST(self, root: TreeNode, L: int, R: int) -> int:
      self.result = 0
      self.helper(root, L, R)
      return self.result

    def helper(self, root, L, R):
        if root:
          if L <= root.val <= R:
```

```
            self.result += root.val
          if root.val <= L:
            self.helper(root.right, L, R)
          elif root.val >= R:
            self.helper(root.left, L, R)
          else:
            self.helper(root.right, L, R)
            self.helper(root.left, L, R)
```

709. To Lower Case

We will transform uppercase to lower if this letter is in upper case. TC is O(n)

```
class Solution:
    def toLowerCase(self, str: str) -> str:
      result = ''
      for c in str:
        if 65 <= ord(c) <= 90:
          result += chr(ord(c) + 32)
        else:
          result += c
      return result
```

804. Unique Morse Code Words

We will use a set to add all words' code and return len(set). TC is O(n)

```
class Solution:
    def uniqueMorseRepresentations(self, words: List[str]) -> int:
      arr =
[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]
      memo = set()
      for word in words:
        key = ''
        for e in word:
          key += arr[ord(e) - ord('a')]
        memo.add(key)
      return len(memo)
```