

172. Factorial Trailing Zeroes

Easy

Given an integer n , return the number of trailing zeroes in $n!$.

Examples:

Input: 3

Output: 0

Explanation: $3! = 6$, no trailing zero.

Input: 5

Output: 1

Explanation: $5! = 120$, one trailing zero.

Note: Your solution should be in logarithmic time complexity.

```
class Solution(object):
    def trailingZeroes(self, n):
        """
        :type n: int
        :rtype: int
        """
        res=0 #find how many 5
        while n>0:
            res+=n//5
            n//=5
        return res
```

179. Largest Number

Medium

Given a list of non negative integers, arrange them such that they form the largest number.

Examples:

Input: [10,2]

Output: "210"

Input: [3, 30, 34, 5, 9]

Output: "9534330"

Note: The result may be very large, so you need to return a string instead of an integer.

```
import random
class Solution(object):
    def largestNumber(self, nums):
        """
        :type nums: List[int]
        :rtype: str
        """
        def quickSort(start, end):
            if start >= end: return
            q = partition(start, end)
            quickSort(start, q-1)
            quickSort(q+1, end)

        def partition(start, end):
            a, j = random.randrange(start, end+1), start-1
            nums[a], nums[end] = nums[end], nums[a]
            for i in range(start, end):
                if self.compare(nums[i], nums[end]):
                    j += 1
                    nums[j], nums[i] = nums[i], nums[j]
            nums[j], nums[end] = nums[end], nums[j]
            return j

        allZero = 1
        for i in range(len(nums)):
            if nums[i]: allZero = 0
            nums[i] = str(nums[i])
        if allZero: return "0"
        quickSort(0, len(nums)-1)
        return "".join(nums)

    def compare(self, n1, n2):
        return n1+n2 > n2+n1
```

69. Sqrt(x)

Easy

Implement `int sqrt(int x)`.

Compute and return the square root of x , where x is guaranteed to be a non-negative integer. Since the return type is an integer, the decimal digits are truncated and only the integer part of the result is returned.

Example 1:

Input: 4

Output: 2

Example 2:

Input: 8

Output: 2

Explanation: The square root of 8 is 2.82842..., and since the decimal part is truncated, 2 is returned.

```
class Solution(object):
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        l, r = 0, x
        while l <= r:
            mid = (l+r)//2
            if mid * mid <= x < (mid+1)*(mid+1):
                return mid
            elif x < mid * mid:
                r = mid
            else:
                l = mid + 1
```

131. Palindrome Partitioning

Medium

Given a string s , partition s such that every substring of the partition is a palindrome.

Return all possible palindrome partitioning of s .

Example:

Input: "aab"

Output:

```
[  
  ["aa", "b"],  
  ["a", "a", "b"]  
]
```

```
class Solution(object):  
    def partition(self, s):  
        """  
        :type s: str  
        :rtype: List[List[str]]  
        """  
        n=len(s)  
        dp=[[[] for i in range(n+1)]  
        dp[0]=[[[]]  
        for i in range(n):  
            for j in range(i+1):  
                if s[j:i+1]==s[j:i+1][::-1]:  
                    for x in dp[j]:  
                        dp[i+1].append(x+[s[j:i+1]])  
        return dp[-1]
```

127. Word Ladder

Medium

Given two words (*beginWord* and *endWord*), and a dictionary's word list, find the length of shortest transformation sequence from *beginWord* to *endWord*, such that:

1. Only one letter can be changed at a time.
2. Each transformed word must exist in the word list. Note that *beginWord* is *not* a transformed word.

Note:

- Return 0 if there is no such transformation sequence.
- All words have the same length.
- All words contain only lowercase alphabetic characters.
- You may assume no duplicates in the word list.
- You may assume *beginWord* and *endWord* are non-empty and are not the same.

Examples:

Input:

```
beginWord = "hit",  
endWord = "cog",  
wordList = ["hot","dot","dog","lot","log","cog"]
```

Output: 5

Explanation: As one shortest transformation is "hit" -> "hot" -> "dot" -> "dog" -> "cog", return its length 5.

Input:

```
beginWord = "hit"  
endWord = "cog"  
wordList = ["hot","dot","dog","lot","log"]
```

Output: 0

Explanation: The endWord "cog" is not in wordList, therefore no possible transformation.

```

class Solution(object):
    def ladderLength(self, beginWord, endWord, wordList):
        """
        :type beginWord: str
        :type endWord: str
        :type wordList: List[str]
        :rtype: int
        """
        # wordList=set(wordList)
        # queue=collections.deque([[beginWord, 1]])
        # while queue:
        #     word, length=queue.popleft()
        #     if word==endWord:
        #         return length
        #     for i in range(len(word)):
        #         for c in "abcdefghijklmnopqrstuvwxyz":
        #             word1=word[:i]+c+word[i+1:]
        #             if word1 in wordList:
        #                 wordList.remove(word1)
        #                 queue.append([word1, length+1])
        # return 0

        #approach 2: two end method much more faster
        wordList=set(wordList)
        if endWord not in wordList: return 0
        forward, backward, path=set([beginWord]), set([endWord]), 2
        while forward and backward:
            if len(forward)>len(backward):
                forward, backward=backward, forward
            nextStep=set()
            for word in forward:
                for i in range(len(word)):
                    first, second=word[:i], word[i+1:]
                    for c in "abcdefghijklmnopqrstuvwxyz":
                        candidate=first+c+second
                        if candidate in backward: return path
                        if candidate in wordList:
                            wordList.remove(candidate)
                            nextStep.add(candidate)
            forward, path=nextStep, path+1
        return 0

```

130. Surrounded Regions

Medium

Given a 2D board containing 'X' and 'O' (the letter O), capture all regions surrounded by 'X'. A region is captured by flipping all 'O's into 'X's in that surrounded region.

Example:

After running your function, the board should be:

X X X X	X X X X
X O O X	X X X X
X X O X	X X X X
X O X X	X O X X

Explanation: Surrounded regions shouldn't be on the border, which means that any 'O' on the border of the board are not flipped to 'X'. Any 'O' that is not on the border and it is not connected to an 'O' on the border will be flipped to 'X'. Two cells are connected if they are adjacent cells connected horizontally or vertically.

```
class Solution(object):
    def solve(self, board): # in-place modify
        if not board: return
        m, n = len(board), len(board[0])
        def dfs(i, j):
            if board[i][j] != 'O': return
            board[i][j] = 'X'
            if i != 0: dfs(i-1, j)
            if i != m-1: dfs(i+1, j)
            if j != 0: dfs(i, j-1)
            if j != n-1: dfs(i, j+1)
        for i in range(n):
            dfs(0, i)
            dfs(m-1, i)
        for i in range(1, m):
            dfs(i, 0)
            dfs(i, n-1)
        for i in range(m):
            for j in range(n):
                if board[i][j] == '':
                    board[i][j] = 'O'
                elif board[i][j] == 'O':
                    board[i][j] = 'X'
```