1.

class Solution:

def hasCycle(self, head: ListNode) -> bool:

slow, fast = head, head

while fast and fast.next:

slow = slow.next

fast = fast.next.next

if slow == fast:

return True

return False

2.

class Solution:

def minPathSum(self, grid: List[List[int]]) -> int:

m = len(grid)

n = len(grid[0])

memo = [[-1 for \_ in range(n)] for \_ in range(m)]

return self.dp(grid, m - 1, n - 1, memo)

def dp(self, grid: List[List[int]], i: int, j: int, memo: List[List[int]]) -> int:

if i == 0 and j == 0:

return grid[0][0]

if i < 0 or j < 0:

return float("inf")

if memo[i][j] != -1:

return memo[i][j]

memo[i][j] = min(

self.dp(grid, i - 1, j, memo),

self.dp(grid, i, j - 1, memo)

) + grid[i][j]

return memo[i][j]

3.

class Solution:

def mergeKLists(self, lists: List[ListNode]) -> ListNode:

if not lists:

return None

dummy = ListNode(-1)

p = dummy

pq = []

for head in lists:

if head:

heapq.heappush(pq, (head.val, id(head), head))

while pq:

node = heapq.heappop(pq)[2]

p.next = node

if node.next:

heapq.heappush(pq, (node.next.val, id(node.next), node.next))

p = p.next

return dummy.next