**CODE BOUNTY**

**Q1- Generate Parentheses.**

CORRECT CODE-

class Solution(object):

def generateParenthesis(self,n,currentString = '',close = 0):

if n == 1 and close == 0:

return [currentString+"()"]

elif n == 0 and close == 1:

return [currentString+")"]

allPossibleVariations = []

if n >= 1:

getSubVariations = self.generateParenthesis(n-1,currentString+"(",close+1)

if type(getSubVariations) != str:

for item in getSubVariations:

allPossibleVariations.append(item)

else: allPossibleVariations.append(getSubVariations)

if close >= 1:

getSubVariations = self.generateParenthesis(n,currentString+")",close-1)

if type(getSubVariations) != str:

for item in getSubVariations:

allPossibleVariations.append(item)

else: allPossibleVariations.append(getSubVariations)

return allPossibleVariations

# Instantiate the Solution class

solution = Solution()

# Test the generateParenthesis method

n = 3 # Change the value of n as needed

result = solution.generateParenthesis(n)

# Print the result

for combination in result:

print(combination)

CODE WITH BUGS-

class Solution(object):

def generateParenthesis(self,n,currentString = '',close = 0):

if n == 1 and close == 0:

return [currentString + "()"]

elif n == 0 and close == 1:

return [currentString + ")"]

elif n == 0 and close == 0:

return [currentString]

allPossibleVariations = []

if n >= 1:

getSubVariations = self.generateParenthesis(n-1,currentString+"(",close+1)

if type(getSubVariations) != str:

for item in getSubVariations:

allPossibleVariations.append(item)

else:

allPossibleVariations.append(getSubVariations)

if close >= 1:

getSubVariations = self.generateParenthesis(n,currentString+")",close-1)

if type(getSubVariations) != str:

for item in getSubVariations:

allPossibleVariations.append(item)

else: allPossibleVariations.append(getSubVariations)

return allPossibleVariations

# Instantiate the Solution class

solution = Solution()

# Test the generateParenthesis method

n = 3 # Change the value of n as needed

result = solution.generateParenthesis(n+1)

# Print the result

for combination in result:

print(combination)

**Q2- SORT LIST**

**Correct code-**

class Solution:

def sortList(self, head, buffer\_size=8):

dummy = ListNode(0)

dummy.next = head

steps = 1

while True:

prev = dummy

remaining = prev.next

num\_loops = 0

while remaining:

num\_loops += 1

sublists = [None] \* buffer\_size

sublists\_tail = [None] \* buffer\_size

for i in range(buffer\_size):

sublists[i] = remaining

substeps = steps

while substeps and remaining:

substeps -= 1

sublists\_tail[i] = remaining

remaining = remaining.next

if sublists\_tail[i]:

sublists\_tail[i].next = None

num\_sublists = buffer\_size

while num\_sublists > 1:

subdummy = ListNode()

for i in range(0, num\_sublists, 2):

subprev = subdummy

subprev.next = None

while sublists[i] and sublists[i + 1]:

if sublists[i].val <= sublists[i + 1].val:

subprev.next = sublists[i]

sublists[i] = sublists[i].next

else:

subprev.next = sublists[i + 1]

sublists[i + 1] = sublists[i + 1].next

subprev = subprev.next

if sublists[i]:

subprev.next = sublists[i]

sublists\_tail[i // 2] = sublists\_tail[i]

else:

subprev.next = sublists[i + 1]

sublists\_tail[i // 2] = sublists\_tail[i + 1]

sublists[i // 2] = subdummy.next

num\_sublists //= 2

prev.next = sublists[0]

prev = sublists\_tail[0]

steps \*= buffer\_size

if num\_loops <= 1:

return dummy.next

# Driver code

# You can create a sample input list and call the sortList method on it.

# Replace the sample input list with your own data as needed.

sample\_input\_list = ListNode(4, ListNode(2, ListNode(1, ListNode(3)))

solution = Solution()

sorted\_list = solution.sortList(sample\_input\_list)

# Print the sorted list

while sorted\_list:

print(sorted\_list.val, end=" -> ")

sorted\_list = sorted\_list.next

print("None")

**CODE WITH BUGS**

class Solution:

def sortList(self, head, buffer\_size=10):

dummy = ListNode(0)

dummy.next = head

steps = 1

while True: # Bug: An infinite loop

prev = dummy

remaining = prev.next

num\_loops = 2 # Bug: Incorrectly incrementing num\_loops

while remaining:

num\_loops += 1

sublists = [None] \* buffer\_size

sublists\_tail = [None] \* buffer\_size

for i in range(buffer\_size):

sublists[i] = remaining

substeps = steps

while substeps and remaining:

substeps -= 1

sublists\_tail[i] = remaining

remaining = remaining.next

if sublists\_tail[i]:

sublists\_tail[i].next = None

num\_sublists = buffer\_size

while num\_sublists > 1:

subdummy = ListNode()

for i in range(1, num\_sublists): # Bug: Starting from 1 instead of 0

subprev = subdummy

subprev.next = None

while sublists[i] and sublists[i - 1]: # Bug: Merging sublists in the wrong order

if sublists[i].val > sublists[i - 1].val:

subprev.next = sublists[i] # Bug: Swapped order

sublists[i] = sublists[i].next

else:

subprev.next = sublists[i - 1]

sublists[i - 1] = sublists[i - 1].next

subprev = subprev.next

if sublists[i]:

subprev.next = sublists[i]

sublists\_tail[i // 2] = sublists\_tail[i]

else:

subprev.next = sublists[i - 1]

sublists\_tail[i // 2] = sublists\_tail[i - 1]

sublists[i // 2] = subdummy.next

num\_sublists //= 2

prev.next = sublists[0]

prev = sublists\_tail[0]

steps \*= buffer\_size

if num\_loops >= 4: # Bug: Changing the condition to a larger number

return dummy.next

# Driver code

# You can create a sample input list and call the sortList method on it.

# Replace the sample input list with your own data as needed.

sample\_input\_list = ListNode(4, ListNode(2, ListNode(1, ListNode(3)))

solution = Solution()

sorted\_list = solution.sortList(sample\_input\_list)

# Print the sorted list

while sorted\_list:

print(sorted\_list.val, end=" -> ")

sorted\_list = sorted\_list.next

print("None")

**Q3.Median of Two Sorted Arrays.**

**Correct code-**

class Solution(object):

def findMedianSortedArrays(self, nums1, nums2):

merged=nums1+nums2

merged.sort()

total=len(merged)

if total % 2== 1:

return float(merged[total//2])

else:

middle1 = merged[total//2-1]

middle2= merged[total//2]

return(float(middle1)+float(middle2))/2.0

# Instantiate the Solution class

solution = Solution()

# Test the findMedianSortedArrays method with sample input

nums1 = [1, 3]

nums2 = [2]

result = solution.findMedianSortedArrays(nums1, nums2)

# Print the result

print("Median:", result)

**Code with bugs-**

class Solution(object):

def findMedianSortedArrays(self, nums1, nums2):

merged\_list = nums1 + nums2

merged\_list.sort(reverse=True) # Introducing an error: Sort in descending order

total\_elements = len(merged)

if total % 2 == 1:

return float(merged[total // 2])

else:

middle1 = merged[total // 2 - 1]

middle2 = merged[total // 2 - 1]

return middle1 # Introducing an error: Always return middle1 as an integer

# Instantiate the Solution class

solution = Solution()

# Test the findMedianSortedArrays method with sample input

nums1 = [1, 3, 5]

nums2 = [2, 4, 6]

result = solution.findMedianSortedArrays(nums1, nums2)

# Print the result

print("Median:", result)