1. **Merge sorted array :**

def merge(nums1, m, nums2, n):

# Initialize pointers

p1 = m - 1

p2 = n - 1

p = m + n - 1

while p1 >= 0 and p2 >= 0:

if nums1[p1] > nums2[p2]:

nums1[p] = nums1[p1]

p1 -= 1

else:

nums1[p] = nums2[p2]

p2 -= 1

p -= 1

while p2 >= 0:

nums1[p] = nums2[p2]

p2 -= 1

p -= 1

nums1 = [1, 2, 3, 0, 0, 0]

m = 3

nums2 = [2, 5, 6]

n = 3

merge(nums1, m, nums2, n)

print(nums1) # Output: [1, 2, 2, 3, 5]

1. **Convert sorted array to binary search tree**

class TreeNode:

def \_\_init\_\_(self, val=0, left=None, right=None):

self.val = val

self.left = left

self.right = right

def sortedArrayToBST(nums):

if not nums:

return None

def convertListToBST(left, right):

if left > right:

return None

mid = (left + right) // 2

node = TreeNode(nums[mid])

node.left = convertListToBST(left, mid - 1)

node.right = convertListToBST(mid + 1, right)

return node

return convertListToBST(0, len(nums) - 1)

def inorderTraversal(root):

if root:

inorderTraversal(root.left)

print(root.val, end=' ')

inorderTraversal(root.right)

nums = [1, 2, 3, 4, 5, 6, 7]

root = sortedArrayToBST(nums)

inorderTraversal(root)

Output: 1 2 3 4 5 6 7

1. **First and last position of an element in sorted array**

def findFirstPosition(nums, target):

left, right = 0, len(nums) - 1

first\_pos = -1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

first\_pos = mid

right = mid - 1 # Continue searching in the left half

elif nums[mid] < target:

left = mid + 1

else:

right = mid - 1

return first\_pos

def findLastPosition(nums, target):

left, right = 0, len(nums) - 1

last\_pos = -1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

last\_pos = mid

left = mid + 1 # Continue searching in the right half

elif nums[mid] < target:

left = mid + 1

else:

right = mid - 1

return last\_pos

def searchRange(nums, target):

first\_pos = findFirstPosition(nums, target)

if first\_pos == -1:

return [-1, -1] # Target not found

last\_pos = findLastPosition(nums, target)

return [first\_pos, last\_pos]

# Example usage

nums = [5, 7, 7, 8, 8, 10]

target = 8

print(searchRange(nums, target)) # Output: [3, 4]

target = 6

print(searchRange(nums, target))

Output: [-1, -1]

1. **Insertion sort list :**

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def insertionSortList(head):

# Create a dummy node to act as the sorted portion of the list

dummy = ListNode(0)

current = head

while current:

prev = dummy

next\_node = current.next

while prev.next and prev.next.val < current.val:

prev = prev.next

current.next = prev.next

prev.next = current

current = next\_node

return dummy.next

def printList(node):

while node:

print(node.val, end=" -> ")

node = node.next

print("None")

head = ListNode(4, ListNode(2, ListNode(1, ListNode(3))))

print("Original list:")

printList(head)

sorted\_head = insertionSortList(head)

print("Sorted list:")

printList(sorted\_head)

1. **Remove duplicates from sorted array**

def removeDuplicates(nums):

if not nums:

return 0

j = 1 # Pointer for the position of the next unique element

for i in range(1, len(nums)):

if nums[i] != nums[i - 1]: # If the current element is unique

nums[j] = nums[i] # Move it to the next position for unique elements

j += 1 # Increment the position pointer

return j # Number of unique elements

nums = [0,0,1,1,1,2,2,3,3,4]

k = removeDuplicates(nums)

print("Number of unique elements:", k)

print("Array after removing duplicates:", nums[:k])

1. **Remove duplicates from sorted list :**

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def deleteDuplicates(head):

current = head

while current and current.next:

if current.val == current.next.val:

current.next = current.next.next # Skip the duplicate node

else:

current = current.next # Move to the next node

return head

def printList(node):

while node:

print(node.val, end=" -> ")

node = node.next

print("None")

head = ListNode(1, ListNode(1, ListNode(2, ListNode(3, ListNode(3)))))

print("Original list:")

printList(head)

head = deleteDuplicates(head)

print("List after removing duplicates:")

printList(head)

1. **Search in rotated sorted array :**

def search(nums, target):

left, right = 0, len(nums) - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

return mid

if nums[left] <= nums[mid]: # Left part is sorted

if nums[left] <= target < nums[mid]: # Target is in the left part

right = mid - 1

else: # Target is in the right part

left = mid + 1

else: # Right part is sorted

if nums[mid] < target <= nums[right]: # Target is in the right part

left = mid + 1

else: # Target is in the left part

right = mid - 1

return -1 # Target not found

nums = [4,5,6,7,0,1,2]

target = 0

print(search(nums, target))

Output: 4

target = 3

print(search(nums, target))

Output: -1

1. **Sort colors :**

def sortColors(nums):

low, mid, high = 0, 0, len(nums) - 1

while mid <= high:

if nums[mid] == 0:

nums[low], nums[mid] = nums[mid], nums[low] # Swap the 0 to the front

low += 1

mid += 1

elif nums[mid] == 1:

mid += 1 # Move past the 1

else:

nums[high], nums[mid] = nums[mid], nums[high] # Swap the 2 to the end

high -= 1

# Example usage

nums = [2, 0, 2, 1, 1, 0]

sortColors(nums)

print(nums)

Output: [0, 0, 1, 1, 2, 2]

1. **Merge K sorted list :**

from heapq import heappush, heappop

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def mergeKLists(lists):

min\_heap = []

for index, node in enumerate(lists):

if node:

heappush(min\_heap, (node.val, index, node))

dummy = ListNode() # Dummy node to start the merged list

current = dummy # Pointer to build the new list

while min\_heap:

val, index, node = heappop(min\_heap)

current.next = ListNode(val)

current = current.next

if node.next:

heappush(min\_heap, (node.next.val, index, node.next))

return dummy.next

# Helper function to print the list (for testing purposes)

def printList(node):

while node:

print(node.val, end=" -> ")

node = node.next

print("None")

list1 = ListNode(1, ListNode(4, ListNode(5)))

list2 = ListNode(1, ListNode(3, ListNode(4)))

list3 = ListNode(2, ListNode(6))

lists = [list1, list2, list3]

merged\_head = mergeKLists(lists)

printList(merged\_head)

1. **Merge two sorted list** **:**

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def mergeTwoLists(list1, list2):

dummy = ListNode() # Dummy node to start the merged list

current = dummy # Pointer to build the new list

# Traverse both lists and append the smaller value to the merged list

while list1 and list2:

if list1.val < list2.val:

current.next = list1

list1 = list1.next

else:

current.next = list2

list2 = list2.next

current = current.next

# If one of the lists has remaining nodes, append them

if list1:

current.next = list1

else:

current.next = list2

return dummy.next # The merged list starts from the next node of dummy

# Helper function to print the list (for testing purposes)

def printList(node):

while node:

print(node.val, end=" -> ")

node = node.next

print("None")

# Example usage

list1 = ListNode(1, ListNode(2, ListNode(4)))

list2 = ListNode(1, ListNode(3, ListNode(4)))

merged\_head = mergeTwoLists(list1, list2)

printList(merged\_head)