**DAY 5**1) **strassen matrix multiplication** :

**Code**:  
import numpy as np

def add\_matrix(a, b):

return np.add(a, b)

def sub\_matrix(a, b):

return np.subtract(a, b)

def strassen(a, b):

if len(a) == 1:

return a \* b

mid = len(a) // 2r

a11, a12, a21, a22 = a[:mid, :mid], a[:mid, mid:], a[mid:, :mid], a[mid:, mid:]

b11, b12, b21, b22 = b[:mid, :mid], b[:mid, mid:], b[mid:, :mid], b[mid:, mid:]

m1 = strassen(add\_matrix(a11, a22), add\_matrix(b11, b22))

m2 = strassen(add\_matrix(a21, a22), b11)

m3 = strassen(a11, sub\_matrix(b12, b22))

m4 = strassen(a22, sub\_matrix(b21, b11))

m5 = strassen(add\_matrix(a11, a12), b22)

m6 = strassen(sub\_matrix(a21, a11), add\_matrix(b11, b12))

m7 = strassen(sub\_matrix(a12, a22), add\_matrix(b21, b22))

c11 = add\_matrix(sub\_matrix(add\_matrix(m1, m4), m5), m7)

c12 = add\_matrix(m3, m5)

c21 = add\_matrix(m2, m4)

c22 = add\_matrix(sub\_matrix(add\_matrix(m1, m3), m2), m6)

top = np.hstack((c11, c12))

bottom = np.hstack((c21, c22))

c = np.vstack((top, bottom))

return c

# Example matrices

a = np.array([[1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12],

[13, 14, 15, 16]])

b = np.array([[17, 18, 19, 20],

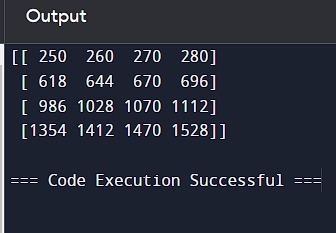
[21, 22, 23, 24],

[25, 26, 27, 28],

[29, 30, 31, 32]])

result = strassen(a, b)

print(result)



1. Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists in a one sorted list. The list should be made by splicing together the

nodes of the first two lists.

Return the head of the merged linked list.

class ListNode:

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

self.val = val

self.next = next

def mergeTwoLists(l1, l2):

dummy = ListNode(0)

current = dummy

while l1 and l2:

if l1.val < l2.val:

current.next = l1

l1 = l1.next

else:

current.next = l2

l2 = l2.next

current = current.next

current.next = l1 or l2

return dummy.next

# Helper function to create a linked list from a list of values

def createLinkedList(values):

if not values:

return None

head = ListNode(values[0])

current = head

for value in values[1:]:

current.next = ListNode(value)

current = current.next

return head

# Helper function to print the linked list

def printLinkedList(head):

values = []

while head:

values.append(head.val)

head = head.next

print(" -> ".join(map(str, values)))

# Example usage

list1 = createLinkedList([1, 2, 4])

list2 = createLinkedList([1, 3, 4])

# Merge the two sorted linked lists

mergedList = mergeTwoLists(list1, list2)

# Print the merged linked list

printLinkedList(mergedList)  
  
  
**3. Remove Duplicates from Sorted Array Given an integer array nums sorted in non-decreasing order, remove the duplicates inplace such that each unique element appears only once. The relative order of the elements should be kept the same. Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the first part of the array nums. More formally, if there are k elements after removing the duplicates, then the first k elements of nums should hold the final result. It does not matter what you leave beyond the first k elements. Return k after placing the final result in the first k slots of nums. Do not allocate extra space for another array. You must do this by modifying the input array in-place with O(1) extra memory.**  
  
**CODE**   
def removeDuplicates(nums):

if not nums:

return 0

# Initialize the index for the next unique element

k = 1

# Start from the second element and iterate through the array

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

# If the current element is different from the previous one, it's unique

if nums[i] != nums[i - 1]:

# Place the unique element at the k-th position

nums[k] = nums[i]

# Move the index for the next unique element

k += 1

# The length of the array with unique elements

return k

# Example usage

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

length = removeDuplicates(nums)

print("Length of array with unique elements:", length)

print("Array after removing duplicates:", nums[:length])  
  
  
**4. Search in Rotated Sorted Array There is an integer array nums sorted in ascending order (with distinct values). Prior to being passed to your function, nums is possibly rotated at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n1], nums[0], nums[1], ..., nums[k-1]] (0-indexed). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2]. Given the array nums after the possible rotation and an integer target, return the index of target if it is in nums, or -1 if it is not in nums. You must write an algorithm with O(log n) runtime complexity**

def search(nums, target):

left, right = 0, nums.length - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

return mid

# Check if the left part is sorted

if nums[left] <= nums[mid]:

# If the target is within the sorted left part

if nums[left] <= target < nums[mid]:

right = mid - 1

else:

left = mid + 1

# Otherwise, the right part must be sorted

else:

# If the target is within the sorted right part

if nums[mid] < target <= nums[right]:

left = mid + 1

else:

right = mid - 1

return -1  
 **5. Find First and Last Position of Element in Sorted Array Given an array of integers nums sorted in non-decreasing order, find the starting and ending position of a given target value. If target is not found in the array, return [-1, -1]. You must write an algorithm with O(log n) runtime complexity.**

def searchRange(nums, target):

def findFirst(nums, target):

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

first = -1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

first = mid

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

elif nums[mid] < target:

left = mid + 1

else:

right = mid - 1

return first

def findLast(nums, target):

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

last = -1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

last = mid

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

elif nums[mid] < target:

left = mid + 1

else:

right = mid - 1

return last

first = findFirst(nums, target)

last = findLast(nums, target)

return [first, last]

**6. Sort Colors Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue. We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively. You must solve this problem without using the library's sort function.**

def sort\_colors(nums):

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

while mid <= high:

if nums[mid] == 0:

nums[low], nums[mid] = nums[mid], nums[low]

low += 1

mid += 1

elif nums[mid] == 1:

mid += 1

else: # nums[mid] == 2

nums[high], nums[mid] = nums[mid], nums[high]

high -= 1 **7. Remove Duplicates from Sorted List Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well**.

class ListNode:

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

self.val = val

self.next = next

def delete\_duplicates(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 create\_linked\_list(elements):

head = ListNode(elements[0])

current = head

for element in elements[1:]:

current.next = ListNode(element)

current = current.next

return head

def print\_linked\_list(head):

current = head

while current:

print(current.val, end=' -> ' if current.next else '')

current = current.next

print()  
  
**8.   
Merge Sorted Array You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively. Merge nums1 and nums2 into a single array sorted in non-decreasing order. The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n**

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

# Pointers for nums1, nums2, and the last position in merged array

p1, p2, p = m - 1, n - 1, m + n - 1

# While there are still elements to compare

while p1 >= 0 and p2 >= 0:

# Place the larger element at the end of nums1

if nums1[p1] > nums2[p2]:

nums1[p] = nums1[p1]

p1 -= 1

else:

nums1[p] = nums2[p2]

p2 -= 1

p -= 1

# If there are still elements in nums2, place them in nums1

while p2 >= 0:

nums1[p] = nums2[p2]

p2 -= 1

p -= 1

merge(nums1, m, nums2, n)

print(nums1) # Output: [1, 2, 2, 3, 5, 6]

**9. Convert Sorted Array to Binary Search Tree Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.  
class TreeNode:**

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

self.val = val

self.left = left

self.right = right

def sorted\_array\_to\_bst(nums):

if not nums:

return None

def helper(left, right):

if left > right:

return None

mid = (left + right) // 2

node = TreeNode(nums[mid])

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

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

return node

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

def inorder\_traversal(root):

if root is not None:

inorder\_traversal(root.left)

print(root.val, end=' ')

inorder\_traversal(root.right)

**10. Insertion Sort List Given the head of a singly linked list, sort the list using insertion sort, and return the sorted list's head. The steps of the insertion sort algorithm: 1. Insertion sort iterates, consuming one input element each repetition and growing a sorted output list. 2. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list and inserts it there. 3. It repeats until no input elements remain. The following is a graphical example of the insertion sort algorithm. The partially sorted list (black) initially contains only the first element in the list. One element (red) is removed from the input data and inserted in-place into the sorted list with each iteration.**

**11. Sort Characters By Frequency Given a string s, sort it in decreasing order based on the frequency of the characters. The frequency of a character is the number of times it appears in the string. Return the sorted string. If there are multiple answers, return any of them**

**from collections import Counter**

def frequencySort(s):

freq = Counter(s)

sorted\_chars = sorted(freq.items(), key=lambda item: item[1], reverse=True)

result = ''.join([char \* count for char, count in sorted\_chars])

return result

**12. Max Chunks To Make Sorted You are given an integer array arr of length n that represents a permutation of the integers in the range [0, n - 1]. We split arr into some number of chunks (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array. Return the largest number of chunks we can make to sort the array**

def maxChunksToSorted(arr):

max\_so\_far = 0

chunks = 0

for i in range(len(arr)):

max\_so\_far = max(max\_so\_far, arr[i])

# If max\_so\_far is equal to the current index, we can form a chunk

if max\_so\_far == i:

chunks += 1

return chunks

**13. Intersection of Three Sorted Arrays Given three integer arrays arr1, arr2 and arr3 sorted in strictly increasing order, return a sorted array of only the integers that appeared in all three arrays.**

def arraysIntersection(arr1, arr2, arr3):

# Initialize pointers for all three arrays

i, j, k = 0, 0, 0

result = []

while i < len(arr1) and j < len(arr2) and k < len(arr3):

# If all elements are the same, add it to the result

if arr1[i] == arr2[j] == arr3[k]:

result.append(arr1[i])

i += 1

j += 1

k += 1

elif arr1[i] < arr2[j]:

i += 1

elif arr2[j] < arr3[k]:

j += 1

else:

k += 1

return result

# Example usage

arr1 = [1, 2, 3, 4, 5]

arr2 = [1, 2, 5, 7, 9]

arr3 = [1, 3, 4, 5, 8]

result = arraysIntersection(arr1, arr2, arr3)

print("Intersection of three arrays:", result)

**14. Sort the Matrix Diagonally A matrix diagonal is a diagonal line of cells starting from some cell in either the topmost row or leftmost column and going in the bottom-right direction until reaching the matrix's end. For example, the matrix diagonal starting from mat[2][0], where mat is a 6 x 3 matrix, includes cells mat[2][0], mat[3][1], and mat[4][2]. Given an m x n matrix mat of integers, sort each matrix diagonal in ascending order and return the resulting matrix.**

def diagonalSort(mat):

from collections import defaultdict

# Create a dictionary to hold the diagonals

diagonals = defaultdict(list)

# Collect all elements in the same diagonal

m, n = len(mat), len(mat[0])

for i in range(m):

for j in range(n):

diagonals[i - j].append(mat[i][j])

# Sort each diagonal

for key in diagonals.keys():

diagonals[key].sort()

# Put sorted elements back into the matrix

for i in range(m):

for j in range(n):

mat[i][j] = diagonals[i - j].pop(0)

return mat