**LAB DAY-5**

1.Strassen algorithm:

PROGRAM:

def add(A,B):

return [[A[i][j]+B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

def sub(A,B):

return [[A[i][j]-B[i][j] for j in range(len(A[0]))] for i in range(len(A))]

def strassen(A,B):

if len(A) == 1:

return [[A[0][0]\*B[0][0]]]

mid = len(A) // 2

A11 = [row[:mid] for row in A[:mid]]

A12 = [row[mid:] for row in A[:mid]]

A21 = [row[:mid] for row in A[mid:]]

A22 = [row[mid:] for row in A[mid:]]

B11 = [row[:mid] for row in B[:mid]]

B12 = [row[mid:] for row in B[:mid]]

B21 = [row[:mid] for row in B[mid:]]

B22 = [row[mid:] for row in B[mid:]]

P1 = strassen(addmatrix(A11,A22),addmatrix(B11,B22))

P2 = strassen(addmatrix(A21,A22),B11)

P3 = strassen(A11,submatrix(B12,B22))

P4 = strassen(A22,submatrix(B21,B11))

P5 = strassen(addmatrix(A11,A12),B22)

P6 = strassen(submatrix(A21,A11),addmatrix(B11,B12))

P7 = strassen(submatrix(A12,A22),addmatrix(B21,B22))

C11 = addmatrix(submatrix(addmatrix(P1, P4), P5), P7)

C12 = addmatrix(P3,P5)

C21 = addmatrix(P2,P4)

C22 = addmatrix(submatrix(addmatrix(P1, P3), P2), P6)

C = []

for i in range(mid):

C.append(C11[i] + C12[i])

for i in range(mid):

C.append(C21[i] + C22[i])

return C

A = [

[1,2,3,4],

[5,6,7,8],

[1,2,3,4],

[5,6,7,8]

]

B = [

[1,2,1,3],

[1,4,1,5],

[1,6,1,7],

[1,8,1,9]

]

C = strassen(A,B)

for row in C:

print(row)

OUTPUT:

[10, 60, 10, 70]

[26, 140, 26, 166]

[10, 60, 10, 70]

[26, 140, 26, 166]

2.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.

Example 1:

Input: list1 = [1,2,4], list2 = [1,3,4]

Output: [1,1,2,3,4,4]

Example 2:

Input: list1 = [], list2 = []

Output: []

Example 3:

Input: list1 = [], list2 = [0]

Output: [0]

Constraints:

● The number of nodes in both lists is in the range [0, 50].

● -100 <= Node.val <= 100

● Both list1 and list2 are sorted in non-decreasing order.

PROGRAM:

class ListNode:

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

self.val = val

self.next = next

def mergeTwoLists(list1, list2):

dummy = ListNode()

current = dummy

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 list1:

current.next = list1

else:

current.next = list2

return dummy.next

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

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

merged\_list = mergeTwoLists(list1, list2)

OUTPUT:

[1,1,2,3,4,4]

2. Merge k Sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it.

Example 1:

Input: lists = [[1,4,5],[1,3,4],[2,6]]

Output: [1,1,2,3,4,4,5,6]

Explanation: The linked-lists are:

[

1->4->5,

1->3->4,

2->6

]

merging them into one sorted list:

1->1->2->3->4->4->5->6

Example 2:

Input: lists = []

Output: []

Example 3:

Input: lists = [[]]

Output: []

Constraints:

● k == lists.length

● 0 <= k <= 104

● 0 <= lists[i].length <= 500

● -104 <= lists[i][j] <= 104

● lists[i] is sorted in ascending order.

● The sum of lists[i].length will not exceed 104.

PROGRAM:

import heapq

class ListNode:

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

self.val = val

self.next = next

def mergeKLists(lists):

min\_heap = []

for idx, l in enumerate(lists):

if l:

heapq.heappush(min\_heap, (l.val, idx, l))

dummy = ListNode()

current = dummy

while min\_heap:

val, idx, node = heapq.heappop(min\_heap)

current.next = node

current = current.next

if node.next:

heapq.heappush(min\_heap, (node.next.val, idx, node.next))

return dummy.next

lists = [ListNode(1, ListNode(4, ListNode(5))), ListNode(1, ListNode(3, ListNode(4))), ListNode(2, ListNode(6))]

merged\_list = mergeKLists(lists)

OUTPUT:

[1,1,2,3,4,4,5,6]

3. Remove Duplicates from Sorted Array

Given an integer array nums sorted in non-decreasing order, remove the duplicates in

place 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.

Custom Judge:

The judge will test your solution with the following code:

int[] nums = [...]; // Input array

int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;

for (int i = 0; i < k; i++) {

assert nums[i] == expectedNums[i];

}

If all assertions pass, then your solution will be accepted.

Example 1:

Input: nums = [1,1,2]

Output: 2, nums = [1,2,\_]

Explanation: Your function should return k = 2, with the first two elements of nums being

1 and 2 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

Example 2:

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

Output: 5, nums = [0,1,2,3,4,,,,,\_]

Explanation: Your function should return k = 5, with the first five elements of nums being

0, 1, 2, 3, and 4 respectively.

It does not matter what you leave beyond the returned k (hence they are underscores).

Constraints:

● 1 <= nums.length <= 3 \* 104

● -100 <= nums[i] <= 100

● nums is sorted in non-decreasing order.

PROGRAM:

def removeDuplicates(nums):

if not nums:

return 0

unique\_ptr = 0

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

if nums[i] != nums[unique\_ptr]:

unique\_ptr += 1

nums[unique\_ptr] = nums[i]

return unique\_ptr + 1

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

k = removeDuplicates(nums)

OUTPUT:

[0,1,2,3,4]

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[n

1], 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.

Example 1:

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

Output: 4

Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3

Output: -1

Example 3:

Input: nums = [1], target = 0

Output: -1

Constraints:

● 1 <= nums.length <= 5000

● -104 <= nums[i] <= 104

● All values of nums are unique.

● nums is an ascending array that is possibly rotated.

● -104 <= target <= 104

PROGRAM:

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]:

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

right = mid - 1

else:

left = mid + 1

else:

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

left = mid + 1

else:

right = mid - 1

return -1

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

target = 0

index = search(nums, target)

OUTPUT:4

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.

Example 1:

Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

Example 2:

Input: nums = [5,7,7,8,8,10], target = 6

Output: [-1,-1]

Example 3:

Input: nums = [], target = 0

Output: [-1,-1]

Constraints:

● 0 <= nums.length <= 105

● -109 <= nums[i] <= 109

● nums is a non-decreasing array.

● -109 <= target <= 109

PROGRAM:

def searchRange(nums, target):

def findLeft(nums, target):

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

while left <= right:

mid = (left + right) // 2

if nums[mid] < target:

left = mid + 1

else:

right = mid - 1

return left

def findRight(nums, target):

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

while left <= right:

mid = (left + right) // 2

if nums[mid] <= target:

left = mid + 1

else:

right = mid - 1

return right

left\_idx = findLeft(nums, target)

right\_idx = findRight(nums, target)

if left\_idx <= right\_idx and left\_idx < len(nums) and nums[left\_idx] == target:

return [left\_idx, right\_idx]

else:

return [-1, -1]

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

target = 8

result = searchRange(nums, target)

OUTPUT:

[3,4]

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.

Example 1:

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

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

Example 2:

Input: nums = [2,0,1]

Output: [0,1,2]

Constraints:

● n == nums.length

● 1 <= n <= 300

● nums[i] is either 0, 1, or 2.

PROGRAM:

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]

low += 1

mid += 1

elif nums[mid] == 1:

mid += 1

else:

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

high -= 1

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

sortColors(nums)

OUTPUT:

[0,0,1,1,2,2]

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.

Example 1:

Input: head = [1,1,2]

Output: [1,2]

Example 2:

Input: head = [1,1,2,3,3]

Output: [1,2,3]

Constraints:

● The number of nodes in the list is in the range [0, 300].

● -100 <= Node.val <= 100

● The list is guaranteed to be sorted in ascending order.

PROGRAM:

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

else:

current = current.next

return head

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

result = deleteDuplicates(head)

OUTPUT:

[1,2,3]

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.

Example 1:

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

Output: [1,2,2,3,5,6]

Explanation: The arrays we are merging are [1,2,3] and [2,5,6].

The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Example 2:

Input: nums1 = [1], m = 1, nums2 = [], n = 0

Output: [1]

Explanation: The arrays we are merging are [1] and [].

The result of the merge is [1].

Example 3:

Input: nums1 = [0], m = 0, nums2 = [1], n = 1

Output: [1]

Explanation: The arrays we are merging are [] and [1].

The result of the merge is [1].

Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the

merge result can fit in nums1.

Constraints:

● nums1.length == m + n

● nums2.length == n

● 0 <= m, n <= 200

● 1 <= m + n <= 200

● -109 <= nums1[i], nums2[j] <= 109

PROGRAM:

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

i, j, k = m - 1, n - 1, m + n - 1

while i >= 0 and j >= 0:

if nums1[i] > nums2[j]:

nums1[k] = nums1[i]

i -= 1

else:

nums1[k] = nums2[j]

j -= 1

k -= 1

while j >= 0:

nums1[k] = nums2[j]

j -= 1

k -= 1

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

nums2 = [2,5,6]

merge(nums1, 3, nums2, 3)

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.

Example 1:

Input: nums = [-10,-3,0,5,9]

Output: [0,-3,9,-10,null,5]

Explanation: [0,-10,5,null,-3,null,9] is also accepted:

Example 2:

Input: nums = [1,3]

Output: [3,1]

Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.

Constraints:

● 1 <= nums.length <= 104

● -104 <= nums[i] <= 104

● nums is sorted in a strictly increasing order.

PROGRAM:

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def sorted (nums):

if not nums:

return None

mid = len(nums) // 2

root = TreeNode(nums[mid])

root.left = sorted (nums[:mid])

root.right = sorted (nums[mid+1:])

return root

nums = [-10, -3, 0, 5, 9]

tree = sorted (nums)

OUTPUT:

[0,-3,9,-10,null,5]

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.

PROGRAM:

class ListNode:

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

self.val = val

self.next = next

def insertion (head):

dummy = ListNode()

current = head

while current:

prev = dummy

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

prev = prev.next

nextnode = current.next

current.next = prev.next

prev.next = current

current = nextnode

return dummy.next

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

result = insertion(head)

OUTPUT:

[1,2,3,4]

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.

Example 1:

Input: s = "tree"

Output: "eert"

Explanation: 'e' appears twice while 'r' and 't' both appear once.

So 'e' must appear before both 'r' and 't'. Therefore "eetr" is also a valid answer.

Example 2:

Input: s = "cccaaa"

Output: "aaaccc"

Explanation: Both 'c' and 'a' appear three times, so both "cccaaa" and "aaaccc" are valid answers.

Note that "cacaca" is incorrect, as the same characters must be together.

Example 3:

Input: s = "Aabb"

Output: "bbAa"

Explanation: "bbaA" is also a valid answer, but "Aabb" is incorrect.

Note that 'A' and 'a' are treated as two different characters.

Constraints:

● 1 <= s.length <= 5 \* 105

● s consists of uppercase and lowercase English letters and digits.

Example 1:

Input: head = [4,2,1,3]

Output: [1,2,3,4]

Example 2:

Input: head = [-1,5,3,4,0]

Output: [-1,0,3,4,5]

Constraints:

● The number of nodes in the list is in the range [1, 5000].

● -5000 <= Node.val <= 5000

PROGRAM:

from collections import Counter

def frequencySort(s):

freq = Counter(s)

sorted\_chars = sorted(freq, key=lambda x: (-freq[x], x))

# Step 3: Construct the output string

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

return result

print(frequencySort("tree"))

OUTPUT:

"eert"

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.

Example 1:

Input: arr = [4,3,2,1,0]

Output: 1

Explanation:

Splitting into two or more chunks will not return the required result.

For example, splitting into [4, 3], [2, 1, 0] will result in [3, 4, 0, 1, 2], which isn't sorted.

Example 2:

Input: arr = [1,0,2,3,4]

Output: 4

Explanation:

We can split into two chunks, such as [1, 0], [2, 3, 4].

However, splitting into [1, 0], [2], [3], [4] is the highest number of chunks possible.

Constraints:

● n == arr.length

● 1 <= n <= 10

● 0 <= arr[i] < n

● All the elements of arr are unique.

PROGRAM:

def Sorted(arr):

seen = 0

chunks = 0

for i, num in enumerate(arr):

seen = max(max\_seen, num)

if seen == i:

chunks += 1

return chunks

print(maxChunksToSorted([4,3,2,1,0]))

OUTPUT:

1

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.

Example 1:

Input: arr1 = [1,2,3,4,5], arr2 = [1,2,5,7,9], arr3 = [1,3,4,5,8]

Output: [1,5]

Explanation: Only 1 and 5 appeared in the three arrays.

Example 2:

Input: arr1 = [197,418,523,876,1356], arr2 = [501,880,1593,1710,1870], arr3 =

[521,682,1337,1395,1764]

Output: []

Constraints:

● 1 <= arr1.length, arr2.length, arr3.length <= 1000

● 1 <= arr1[i], arr2[i], arr3[i] <= 2000

PROGRAM:

def Intersection(arr1, arr2, arr3):

i, j, k = 0, 0, 0

result = []

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

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

result.append(arr1[i])

i += 1

j += 1

k += 1

else:

if arr1[i] < arr2[j]:

i += 1

elif arr2[j] < arr3[k]:

j += 1

else:

k += 1

return result

print(arraysIntersection([1,2,3,4,5], [1,2,5,7,9], [1,3,4,5,8]))

OUTPUT:

[1, 5]

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.

Example 1:

Input: mat = [[3,3,1,1],[2,2,1,2],[1,1,1,2]]

Output: [[1,1,1,1],[1,2,2,2],[1,2,3,3]]

Example 2:

Input: mat =

[[11,25,66,1,69,7],[23,55,17,45,15,52],[75,31,36,44,58,8],[22,27,33,25,68,4],[84,28,14,11,5,50]]

Output:

[[5,17,4,1,52,7],[11,11,25,45,8,69],[14,23,25,44,58,15],[22,27,31,36,50,66],[84,28,75,33,55,68]]

Constraints:

● m == mat.length

● n == mat[i].length

● 1 <= m, n <= 100

● 1 <= mat[i][j] <= 100

PROGRAM:

def diagonalSort(mat):

from collections import defaultdict

import heapq

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

diagonals = defaultdict(list)

for i in range(m):

for j in range(n):

heapq.heappush(diagonals[i - j], mat[i][j])

for i in range(m):

for j in range(n):

mat[i][j] = heapq.heappop(diagonals[i - j])

return mat

mat1 = [[3,3,1,1], [2,2,1,2], [1,1,1,2]]

print(diagonalSort(mat1))

OUTPUT:

[[1,1,1,1], [1,2,2,2], [1,2,3,3]]