DAY-3 PROGRAM

1. Write a program to perform the following o An empty list o A list with one element o A list with all identical elements o A list with negative numbers

Test Cases: 1. Input: [] o Expected Output: []

Input: [1] o Expected Output: [1]

. Input: [7, 7, 7, 7] o Expected Output: [7, 7, 7, 7]

Input: [-5, -1, -3, -2, -4] o Expected Output: [-5, -4, -3, -2, -1]

Code:

test\_case1 = []

test\_case2 = [1]

test\_case3 = [7, 7, 7, 7]

test\_case4 = [-5, -1, -3, -2, -4]

sorted\_case1 = sorted(test\_case1)

sorted\_case2 = sorted(test\_case2)

sorted\_case3 = sorted(test\_case3)

sorted\_case4 = sorted(test\_case4)

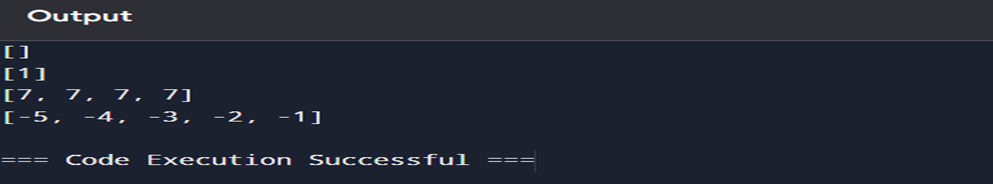
print(sorted\_case1)

print(sorted\_case2)

print(sorted\_case3)

print(sorted\_case4)

output:



2. Describe the Selection Sort algorithm's process of sorting an array. Selection Sort works by dividing the array into a sorted and an unsorted region. Initially, the sorted region is empty, and the unsorted region contains all elements. The algorithm repeatedly selects the smallest element from the unsorted region and swaps it with the leftmost unsorted element, then moves the boundary of the sorted region one element to the right. Explain why Selection Sort is simple to understand and implement but is inefficient for large datasets. Provide examples to illustrate step-by-step how Selection Sort rearranges the elements into ascending order, ensuring clarity in your explanation of the algorithm's mechanics and effectiveness. Sorting a Random Array:

Input: [5, 2, 9, 1, 5, 6]

Output: [1, 2, 5, 5, 6, 9]

Code:

arr = [5, 2, 9, 1, 5, 6]

n = len(arr)

for i in range(n):

min\_index = i

for j in range(i + 1, n):

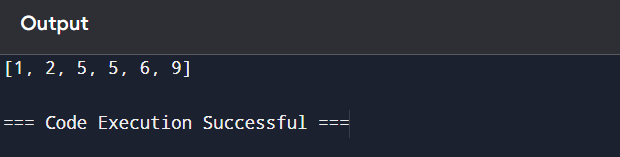
if arr[j] < arr[min\_index]:

min\_index = j

arr[i], arr[min\_index] = arr[min\_index], arr[i]

print(arr)

output:



3. Write code to modify bubble\_sort function to stop early if the list becomes sorted before all passes are completed.

Code:

arr = [64, 34, 25, 12, 22, 11, 90]

n = len(arr)

for i in range(n):

swapped = False

for j in range(0, n-i-1):

if arr[j] > arr[j + 1]:

arr[j], arr[j + 1] = arr[j + 1], arr[j]

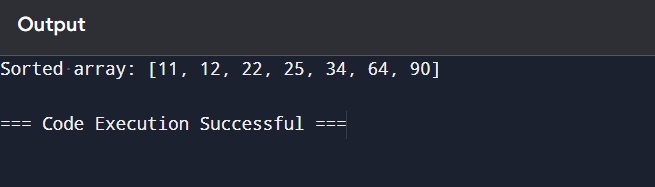
swapped = True

if not swapped:

break

print("Sorted array:",arr)

output:



4. Test Cases: • Test your optimized function with the following lists:

1. Input: [64, 25, 12, 22, 11]

Expected Output: [11, 12, 22, 25, 64]

1. Input: [29, 10, 14, 37, 13]

Expected Output: [10, 13, 14, 29, 37]

3. Input: [3, 5, 2, 1, 4]

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

4. Input: [1, 2, 3, 4, 5] (Already sorted list)

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

Code:

arr\_list = [

[64, 25, 12, 22, 11],

[29, 10, 14, 37, 13],

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

[1, 2, 3, 4, 5]

]

for arr in arr\_list:

n = len(arr)

for i in range(n):

min\_index = i

for j in range(i + 1, n):

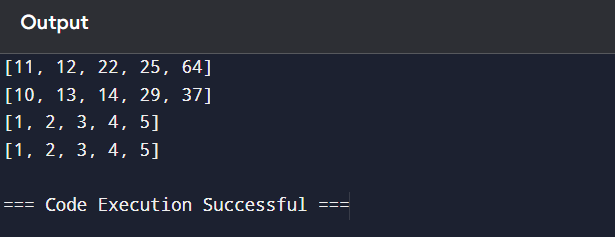
if arr[j] < arr[min\_index]:

min\_index = j

arr[i], arr[min\_index] = arr[min\_index], arr[i]

print(arr)

output:

vdfdf

5. Given an array arr of positive integers sorted in a strictly increasing order, and an integer k. return the kth positive integer that is missing from this array.

Example :

Input: arr = [2,3,4,7,11], k = 5

Output: 9

Code:

arr = [2, 3, 4, 7, 11]

k = 5

missing\_count = 0

current = 1

idx = 0

while missing\_count < k:

if idx < len(arr) and arr[idx] == current:

idx += 1

else:

missing\_count += 1

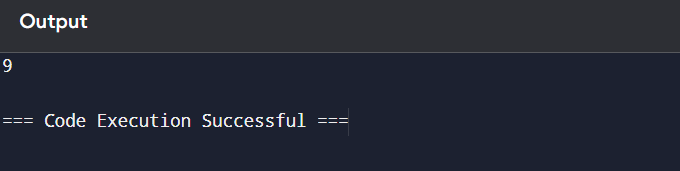
if missing\_count == k:

print(current)

break

current+=1

output;



6. A peak element is an element that is strictly greater than its neighbors. Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array. You must write an algorithm that runs in O(log n) time.

Example :

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

Output: 2

Code:

nums = [1, 2, 3, 1]

n = len(nums)

left, right = 0, n - 1

while left < right:

mid = (left + right) // 2

if nums[mid] < nums[mid + 1]:

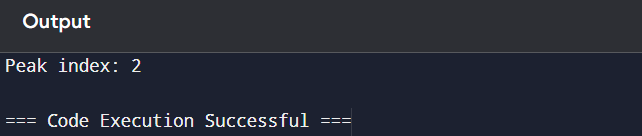
left = mid + 1

else:

right = mid

print("Peak index:", left)

output:

cvdvb

7. Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Example 1:

Input: haystack = "sadbutsad", needle = "sad"

Output: 0

Code:

haystack = "sadbutsad"

needle = "sad"

n\_len = len(needle)

h\_len = len(haystack)

index = -1

for i in range(h\_len - n\_len + 1):

if haystack[i:i + n\_len] == needle:

index = i

break

print("First occurrence index:", index)

output:

