QUESTION 1:

**PROBLEM ANALYSIS**

Given is an array of numbers representing daily temperatures. For each day, we need to determine how many days must pass before a warmer temperature occurs. In other words, we need to find how far the next greater temperature is from the current index. If no higher temperature occurs, the output should default to zero.

**ALGORITHM DESIGN**

A brute-force approach would be to start from each element in the array and compare it with the values to its right. However, this solution is inefficient for large arrays due to its high time complexity.

Instead, we will use a stack data structure to optimize the solution.

STEP 1:Initialize the output array with zeros. This array will store the number of days until the next warmer day for each temperature.

STEP 2: Initialize the stack to keep track of the indices of the temperatures as follows:

index[0] = 73

index[1] = 74

index[2] = 75

index[3] = 71

index[4] = 69

index[5] = 72

index[6] = 76

index[7] = 73

STEP 3: Add the index 0 to the stack:

A. Compare the value at `index 1` with the value at `index 0` (i.e., is 74 > 73?).

If this condition is true:

i) Subtract 'index 1 - index 0` and store the result in the output array.

ii) Pop the element from the stack.

STEP 4: Continue iterating through the array:

A. For each temperature, compare the current value with the value at the index stored at the top of the stack.

STEP 5: If the condition is false (i.e., we haven’t found a warmer day):

i) Push the current index onto the stack without popping anything.

STEP 6: After the iteration is complete, exit the loop and print the output array.

**IMPLEMENT THE SOLUTION**

def dailyTemperatures(temperatures):

stack = []

# initialize an empty stack to store indices of temperatures

output = [0] \* len(temperatures)

# initialize output array with 0s, the same length as temperatures

print("Input:", user\_input)

# print the initial state of the output array (all zeros)

for i in range(len(temperatures)):

# loop through each day (index) in the temperatures list

while stack and temperatures[stack[-1]] < temperatures[i]:

# while the stack is not empty and the current temperature is higher

# than the temperature at the index stored at the top of the stack

ind = stack.pop()

# pop the index from the stack (this is the day with a cooler temperature)

output[ind] = i - ind

# update the output for that day with the number of days until a warmer temperature

stack.append(i)

# add the current index (day) to the stack

print("Output:", output)

# print the final output after all calculations are done

return output

# Given array

user\_input = [98, 102, 100, 97, 104, 101, 106, 107]

result = dailyTemperatures(user\_input)

**TEST CASE:**

A.

Input: [73, 74, 75, 71, 69, 72, 76, 73]

Output: [1, 1, 4, 2, 1, 1, 0, 0]

B.

Input: [98, 102, 100, 97, 104, 101, 106, 107]

Output: [1, 3, 2, 1, 2, 1, 1, 0]

**QUESTION 2:**

**PROBLEM ANALYSIS**

Given is an array of numbers representing book prices. Re-arrange the prices in ascending order.

**ALGORITHM DESIGN**

Use the bubble sort algorithm to sort the array. In other words, we will keep shuffling the elements until the price are reflected in current order i.e. ascending.

STEP 1. The iteration starts from the beginning of the unsorted array.

STEP 2. Compare each book prices of adjacent elements.

STEP 3. If the element on the left is greater than the element on the right, swap them.

STEP 4. If the elements are already in the correct order, no swap is performed.

STEP 5. Once the iteration reaches the end of the list, start a new pass from the beginning.

STEP 6. The swapping continues over and over, until all the numbers are sorted.

**IMPLEMENT THE SOLUTION**

"""

Bubble Sort Algorithm for Sorting Book Prices

"""

# Define the bubble sort function

def bubble\_sort(book\_prices):

print('Input (Book Prices):', book\_prices)

# book\_prices is the list of book prices that needs to be sorted

indexing\_length = len(book\_prices) - 1

# Length minus 1 because comparisons stop before the last element

sorted = False

# sorted is used to break the loop once the list is fully sorted

# Perform sorting using a while loop

while not sorted:

# Continue looping as long as the list is not fully sorted

sorted = True

for i in range(0, indexing\_length):

if book\_prices[i] > book\_prices[i + 1]:

# Compare adjacent elements in the list

sorted = False

# Swap if the current price is greater than the next price

book\_prices[i], book\_prices[i + 1] = book\_prices[i + 1], book\_prices[i]

# Return the sorted list of book prices

return book\_prices

# Call the bubble\_sort function with a list of book prices

book\_prices = [50.15, 12.49, 66.09, 7.00, 90.99]

sorted\_prices = bubble\_sort(book\_prices)

# Adding $ symbol in the output

print('Output (Sorted Book Prices):', ['${:.2f}'.format(price) for price in sorted\_prices])

**TEST CASE:**

A.

Input (Book Prices): [29.99, 19.99, 39.99, 9.99, 24.99]

Output (Sorted Book Prices): [9.99, 19.99, 24.99, 29.99, 39.99]

B.

*Adding['${:.2f}'.format(price) for price in sorted\_prices]) to print the output with $ symbol*

Input (Book Prices): [50.15, 12.49, 66.09, 7.0, 90.99]

Output (Sorted Book Prices): ['$7.00', '$12.49', '$50.15', '$66.09', '$90.99']

QUESTION 3:

**PROBLEM ANALYSIS**

This program will calculate the rental price from the list of desired vehicles . The car should have attributes such as make, model, year and rental price per day. Plus, this should have methods to display the car details and calculate the total cost for rental.

**ALGORITHM DESIGN**

STEP 1: We will create a class name 'Car'. It will have attributes are make, model, year and rental price.

STEP 2: Create an initializer method that takes the above attributes as parameters.

STEP 3: Define a method to display car details

STEP 4: Define a method to calculate the rental cost

STEP 5: Create required number of car objects and create object's instance variable

STEP 6: Create test cases

**IMPLEMENT THE SOLUTION**

class Car:

# Creating a class Car

def \_\_init\_\_(self, brand, model, year, rental\_price):

# Defining the initializer and passing the arguments

self.brand = brand

self.model = model

self.year = year

self.rental\_price = rental\_price

def display\_details(self):

# Method to display car details

print(self.brand, self.model, self.year, self.rental\_price)

def calculate\_rental(self, days):

# Method to calculate the rental cost based on the passed number of days

invoice = self.rental\_price \* days

return invoice

# Creating car1 object

print("Test Case: 1")

car1 = Car("Toyota", "Camry", 2020, 40)

# Accessing car attributes using display\_details methods

car1.display\_details()

# Rental cost for 5 days, passing the days argument to calculate\_rental function

days = 5

car1.calculate\_rental(days)

print(f"Total rental cost for {days} days is ${car1.calculate\_rental(days)}")

print("- "\*20)

# Creating car2 object

print("Test Case: 2")

car2 = Car("Honda", "Civic", 2019, 35)

car2.display\_details()

# Rental cost for 7 days,

days = 7

car2.calculate\_rental(days)

print(f"Total rental cost for {days} days is ${car2.calculate\_rental(days)}")

print("- "\*20)

# Similarly

print("Test Case: 3")

car3 = Car("Tesla", "Model Y", 2024, 100)

car3.display\_details()

days = 10

car3.calculate\_rental(days)

print(f"Total rental cost for {days} days is ${car3.calculate\_rental(days)}")

print("- "\*20)

**TEST CASES**

Test Case: A

Toyota Camry 2020 40 $/day

Total rental cost for 5 is $200

- - - - - - - - - - - - - - - - - - - -

Test Case: B

Honda Civic 2019 35 $/day

- - - - - - - - - - - - - - - - - - - -

Test Case: C

Tesla Model Y 2024 100 $/day

Question 4

**PROBLEM ANALYSIS**

This e-commerce features calculating the total price of items including discount for items on sale. We are given a Python code to debug the code error. Given are some variables such as name, price, quantity and a Boolean representing if the item is on sale.

**ALGORITHM DESIGN**

STEP 1: Investigate the code to VS code editor

STEP 2: Identifying the inputs, output and logical requirements.

Inputs are the item name, price, quantity and value if its set to on sale.

STEP 3: Look for syntax and indentation error

STEP 4: The Item class is correctly initialized with \_\_init\_\_ method and attributes.

The calculate\_total function is defined correctly.

The logic for calculation is correct for applying the 10%

STEP 4: The Item instances are correctly created with proper arguments to pass

STEP 5: Double checking the mathematical calculation

Laptop $1000 \* 1 = $1000 (not on sale)

Mouse $50 \*2 \* 0.9 = $90

Keyboard $80 \*1 \*0.9 = $ 72

Total = $1162

**IMPLEMENT THE SOLUTION**To handle the edge cases such as cart is empty or zero quantity or negative prices.

Output the result with $ symbol

Setting the on-sale value false by default

class Item:

def \_\_init\_\_(self, name, price, quantity, on\_sale: bool=False):

**# setting the on sale value to false by default make it clear**

if price < 0 or quantity < 0:

raise ValueError ("Price & Quantity should be non negative")

**# adding edge case handling**

self.name = name

self.price = price

self.quantity = quantity

self.on\_sale = on\_sale

def calculate\_total(cart):

if not cart:

return 0.0

**# immediately return zero if the cart list is empty**

total = 0

for item in cart:

if item.on\_sale:

total += item.price \* item.quantity \* 0.9

else:

total += item.price \* item.quantity

return total

# Example usage

cart = [

Item("Laptop", 1000, 1),

Item("Mouse", 50, 2, True),

Item("Keyboard", 80, 1, True)

]

print(f"Total price: ${calculate\_total(cart):.2f}")

**# output the result with $ symbol**

#print("Total price:", calculate\_total(cart))

**TEST CASE**

**A.**

Item("Laptop", 1000, 1),

Item("Mouse", 50, 2, True),

Item("Keyboard", 80, 1, True)

Output: Total price: 1162.0

**B.**

Missing to set Boolean value  
 Item("Laptop", 1000, 1),

Item("Mouse", 50, 2, True),

Item("Keyboard", 80, 1, True)

Output: Total price: 1162.0

**C.**

Setting the Keyboard quantity 0

Item("Laptop", 1000, 1),

Item("Mouse", 50, 2, True),

Item("Keyboard", 80, 0, True)

Output: Total price: $1090.00

**D.**

Setting the Mouse quantity -2

Output: raise ValueError ("Price & Quantity should be non-negative")