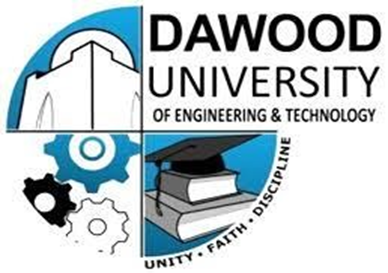
**Data Structures & Algorithms**

**(Practical Manual)**



**5th Semester, 3rd Year**  **BATCH -2022**

# **BS ARTIFICIAL INTELLIGENCE**

**DAWOOD UNIVERSITY OF ENGINEERING & TECHNOLOGY, KARACHI**



## **CERTIFICATE**

This is to certify that Mr./Ms. **Muhammad Sohaib** with Roll # **22F-BSAI-40** of Batch 2023 has successfully completed all the labs prescribed for the course “**Data Structures & Algorithms**”.

**Engr. Hamza Farooqui**

Lecturer

Department of AI

**Lab No. 01**

## [Introduction to Programming in Python]

**Task No. 01**

Write a program which can generate the following

Input a number: 10

10 x 1 = 10

10 x 2 = 20

10 x 3 = 30

10 x 4 = 40

10 x 5 = 50

10 x 6 = 60

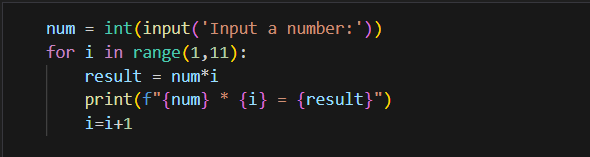
10 x 7 = 70

10 x 8 = 80

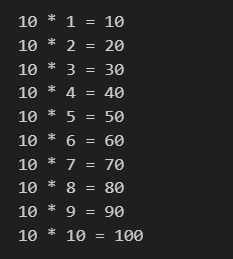
10 x 9 = 90

10 x 10 = 100

**CODE**



**OUTPUT**



**Task No. 02**

Write a program to prompt for a score between 0.0 and 1.0. If the score is out of range, print an error message. If the score is between 0.0 and 1.0, print a grade using the following table:

>= 0.9 A

>= 0.8 B

>>= 0.7 C

>= 0.6 D < 0.6 F

Enter score: 0.95 A

Enter score: perfect Bad score

Enter score: 10.0

Bad score

Enter score: 0.75 C

Enter score: 0.5

F

**CODE**

def get\_grade(score):

if score < 0.0 or score > 1.0:

return "Bad score"

elif score >= 0.9:

return "A"

elif score >= 0.8:

return "B"

elif score >= 0.7:

return "C"

elif score >= 0.6:

return "D"

else:

return "F"

while True:

score\_input = input("Enter score (or type 'exit' to quit): ")

if score\_input.lower() == "exit":

break

if score\_input.lower() == "perfect":

print("Bad score")

continue

try:

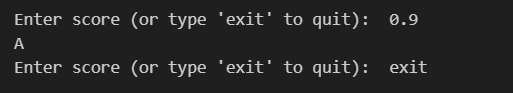
score = float(score\_input)

print(get\_grade(score))

except ValueError:

print("Bad score")

**OUTPUT**



**Task No. 03**

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.

Example 1:

Input: nums = [2,7,11,15], target = 9

Output: [0,1]

Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4], target = 6

Output: [1,2]

**CODE**

class TwoSum:

def \_\_init\_\_(self, list1, target):

self.list1 = list1

self.target = target

def solution(self):

length = len(list1)

for i in range(length-1):

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

if list1[i]+list1[j] == self.target:

new\_list = i, j

return list(new\_list)

return -1

list1 = [2,7,11,15]

target = 9

obj = TwoSum(list1, target)

print(obj.solution())

**OUTPUT**



**Lab No. 02**

**[Implementing Stack Data Structure in Python]**

**Task No. 01**

Execute the above code and observe its output

**Code**

def create\_stack(): # creating a stack

stack = []

return stack

# creating an function to check stack is empty or not:

def check\_empty(stack):

return len(stack) == 0

# adding an items into the stack:

def push(stack,item):

stack.append(item)

print('pushed item:',item)

# removing an element from the stack:

def pop(stack):

if(check\_empty(stack)):

return 'stack is empty'

else:

return stack.pop()

stack = create\_stack()

push(stack, str(1))

push(stack, str(2))

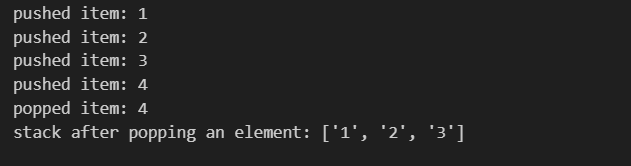
push(stack, str(3))

push(stack, str(4))

print("popped item: " + pop(stack))

print("stack after popping an element: " + str(stack))

**OUTPUT**



**Task No. 02**

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.

2. Open brackets must be closed in the correct order.

3. Every close bracket has a corresponding open bracket of the same type.

Example 1:

Input: s = "()"

Output: true

Example 2:

Input: s = "()[]{}"

Output: true

Example 3:

Input: s = "(]"

Output: false

Example 4:

Input: s = "([])"

Output: true

**Code**

def ispar(s):

stack = []

for char in s:

# Opening bracket

if char in '({[':

stack.append(char)

# Closing Bracket

elif char in ')}]':

# closing bracket without opening

if not stack:

return False

# Else pop an item check for matching

top = stack.pop()

if (top == '(' and char != ')') or \

(top == '{' and char != '}') or \

(top == '[' and char != ']'):

return False

# If an opening bracket without closing

return len(stack) == 0

s = '{()}[]'

if ispar(s):

print("true")

else:

print("false")

**OUTPUT**



**Lab No. 03**

**[Building and Utilizing Queues in Python]**

**Task No. 01**

Execute the above code and observe its output.

**Code**

class Queue:

def \_\_init\_\_(self):

self.queue = []

# add an element

def enqueue(self,item):

self.queue.append(item)

# remove an element

def dequeue(self):

if len(self.queue)<1:

return None

else:

return self.queue.pop(0)

# display the queue

def display(self):

print(self.queue)

def size(self):

return len(self.queue)

q = Queue()

q.enqueue(1)

q.enqueue(2)

q.enqueue(3)

q.enqueue(4)

q.enqueue(5)

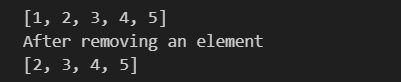
q.display()

q.dequeue()

print("After removing an element")

q.display()

**OUTPUT**



**Task No. 02**

There are n people in a line queuing to buy tickets, where the 0th person is at the front of the line and the (n - 1)th person is at the back of the line.

You are given a 0-indexed integer array tickets of length n where the number of tickets that the ith person would like to buy is tickets[i].

Each person takes exactly 1 second to buy a ticket. A person can only buy 1 ticket at a time and has to go back to the end of the line (which happens instantaneously) in order to buy more tickets. If a person does not have any tickets left to buy, the person will leave the line.

Return the time taken for the person initially at position k (0-indexed) to finish buying tickets.

Example 1:

Input: tickets = [2,3,2], k = 2

Output: 6

Explanation:

• The queue starts as [2,3,2], where the kth person is underlined.

• After the person at the front has bought a ticket, the queue becomes [3,2,1] at 1 second.

• Continuing this process, the queue becomes [2,1,2] at 2 seconds.

• Continuing this process, the queue becomes [1,2,1] at 3 seconds.

• Continuing this process, the queue becomes [2,1] at 4 seconds. Note: the person at the front left the queue.

• Continuing this process, the queue becomes [1,1] at 5 seconds.

• Continuing this process, the queue becomes [1] at 6 seconds. The kth person has bought all their tickets, so return 6.

Example 2:

Input: tickets = [5,1,1,1], k = 0

Output: 8

Explanation:

• The queue starts as [5,1,1,1], where the kth person is underlined.

• After the person at the front has bought a ticket, the queue becomes [1,1,1,4] at 1 second.

• Continuing this process for 3 seconds, the queue becomes [4] at 4 seconds.

• Continuing this process for 4 seconds, the queue becomes [] at 8 seconds. The kth person has bought all their tickets, so return 8.

**Code**

from typing import List

class Solution:

def timeRequiredToBuy(self, tickets: List[int], k: int) -> int:

# Initialize the total time required to 0

total\_time = 0

# Iterate over the ticket queue to simulate the time passing

for index, tickets\_at\_this\_position in enumerate(tickets):

# If the current position is before or at the target position k

if index <= k:

# Add the minimum of the target tickets and tickets at the current position

# It ensures we do not count the extra tickets the target person doesn't need

total\_time += min(tickets[k], tickets\_at\_this\_position)

else:

# After the target person has bought their tickets, they will not buy more

# Thus, for the people after the target, we consider one less ticket for the target

# Person at position k would have already bought their ticket when turn comes to later positions

total\_time += min(tickets[k] - 1, tickets\_at\_this\_position)

# Return the calculated total time

return total\_time

# Example usage:

sol = Solution()

print(sol.timeRequiredToBuy([2, 3, 2], 2)) # This would output 6, the total time to buy tickets

**OUTPUT**



**Lab No. 04**

**[Working with Linked Lists and Node Insertion]**

**Task No. 01**

Implement LinkedList Data Structure in Python.

**Code**

class Node: # Creating a node

def \_\_init\_\_(self, item):

self.item = item

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

linked\_list = LinkedList()

# Assign item values

linked\_list.head = Node(1)

second = Node(2)

third = Node(3)

# Connect nodes

linked\_list.head.next = second

second.next = third

# Print the linked list item

while linked\_list.head != None:

print(linked\_list.head.item, end=" ")

linked\_list.head = linked\_list.head.next

**OUTPUT**



**Task No. 02**

Insert a node at Head, and End of the LinkedList

**Code**

# Linked List - Python

class Node:

def \_\_init\_\_(self,data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def insertHead(self,newNode):

temporaryNode = self.head

self.head = newNode

self.head.next = temporaryNode

del temporaryNode

def insertAt(self,newNode,position):

if position < 0 or position > self.listLength():

print("Invalid Position")

return

if position == 0:

self.insertHead(newNode)

previousNode = None

return

currentNode = self.head

currentPosition = 0

while True:

if currentPosition == position:

previousNode.next = newNode

newNode.next = currentNode

break

previousNode = currentNode

currentNode = currentNode.next

currentPosition += 1

def insertEnd(self,newNode):

if self.head is None:

self.head = newNode

else:

lastNode = self.head

while True:

if lastNode.next is None:

break

lastNode = lastNode.next

lastNode.next = newNode

def listLength(self):

currentNode = self.head

length = 0

while currentNode is not None:

length += 1

currentNode = currentNode.next

return length

def printList(self):

if self.head is None:

print("List is empty")

return

currentNode = self.head

while True:

if currentNode is None:

break

print(currentNode.data)

currentNode = currentNode.next

firstNode = Node(10)

secondNode = Node(20)

thirdNode = Node(30)

fourthNode = Node(40)

linkedList =LinkedList()

# insert at head

linkedList.insertHead(firstNode)

linkedList.insertHead(secondNode)

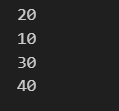
# insert at end

linkedList.insertEnd(thirdNode)

linkedList.insertEnd(fourthNode)

linkedList.printList()

**OUTPUT**



**Task No. 03**

Insert a new node in between two nodes passing the index where the new node is to be inserted.

**Code**

class Node:

def \_\_init\_\_(self,data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def insertHead(self,newNode):

temporaryNode = self.head

self.head = newNode

self.head.next = temporaryNode

del temporaryNode

def insertAt(self,newNode,position):

if position < 0 or position > self.listLength():

print("Invalid Position")

return

if position == 0:

self.insertHead(newNode)

previousNode = None

return

currentNode = self.head

currentPosition = 0

while True:

if currentPosition == position:

previousNode.next = newNode

newNode.next = currentNode

break

previousNode = currentNode

currentNode = currentNode.next

currentPosition += 1

def insertEnd(self,newNode):

if self.head is None:

self.head = newNode

else:

lastNode = self.head

while True:

if lastNode.next is None:

break

lastNode = lastNode.next

lastNode.next = newNode

def listLength(self):

currentNode = self.head

length = 0

while currentNode is not None:

length += 1

currentNode = currentNode.next

return length

def printList(self):

if self.head is None:

print("List is empty")

return

currentNode = self.head

while True:

if currentNode is None:

break

print(currentNode.data)

currentNode = currentNode.next

firstNode = Node(10)

secondNode = Node(20)

thirdNode = Node(30)

fourthNode = Node(40)

linkedList =LinkedList()

# insert at end

linkedList.insertAt(firstNode,0)

linkedList.insertAt(secondNode,1)

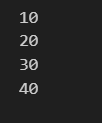
# insert at end

linkedList.insertEnd(thirdNode)

linkedList.insertEnd(fourthNode)

linkedList.printList()

**OUTPUT**



**Lab No. 05**

**[Manipulating Linked Lists: Deletion and Merging]**

**Task No. 01**

Implement Deletion of a node from LinkedList using the three ways explained in python.

**Code**

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def insertHead(self, newNode):

temporaryNode = self.head

self.head = newNode

self.head.next = temporaryNode

del temporaryNode

def insertAt(self, newNode, position):

if position < 0 or position > self.listLength():

print("Invalid Position")

return

if position == 0:

self.insertHead(newNode)

return

currentNode = self.head

currentPosition = 0

while currentNode is not None:

if currentPosition == position:

previousNode.next = newNode

newNode.next = currentNode

return

previousNode = currentNode

currentNode = currentNode.next

currentPosition += 1

def insertEnd(self, newNode):

if self.head is None:

self.head = newNode

else:

lastNode = self.head

while lastNode.next is not None:

lastNode = lastNode.next

lastNode.next = newNode

def deleteHead(self):

if self.head is None:

print("List is empty")

return

self.head = self.head.next # Move head to the next node

def deleteNode(self, key):

# Case 1: Deleting the head node

if self.head is None:

print("List is empty")

return

if self.head.data == key:

self.head = self.head.next

return

# Case 2: Deleting a node from the middle or end

currentNode = self.head

while currentNode is not None:

if currentNode.next and currentNode.next.data == key:

currentNode.next = currentNode.next.next

return

currentNode = currentNode.next

print("Node with value", key, "not found.")

def deleteAt(self, position):

if position < 0 or position >= self.listLength():

print("Invalid Position")

return

# Case 1: Delete at position 0 (head)

if position == 0:

self.deleteHead()

return

currentNode = self.head

currentPosition = 0

while currentNode is not None:

if currentPosition == position - 1:

currentNode.next = currentNode.next.next # Skip the node to delete

return

currentNode = currentNode.next

currentPosition += 1

def listLength(self):

currentNode = self.head

length = 0

while currentNode is not None:

length += 1

currentNode = currentNode.next

return length

def printList(self):

if self.head is None:

print("List is empty")

return

currentNode = self.head

while currentNode is not None:

print(currentNode.data)

currentNode = currentNode.next

# Example usage:

firstNode = Node(10)

secondNode = Node(20)

thirdNode = Node(30)

fourthNode = Node(40)

linkedList = LinkedList()

# Insert nodes

linkedList.insertAt(firstNode, 0) # Insert first node at the head

linkedList.insertAt(secondNode, 1) # Insert second node at position 1

linkedList.insertEnd(thirdNode) # Insert third node at the end

linkedList.insertEnd(fourthNode) # Insert fourth node at the end

print("Original List:")

linkedList.printList()

# Deleting the head node

linkedList.deleteHead()

print("\nList after deleting head node:")

linkedList.printList()

# Deleting a specific node by value

linkedList.deleteNode(30)

print("\nList after deleting node with value 30:")

linkedList.printList()

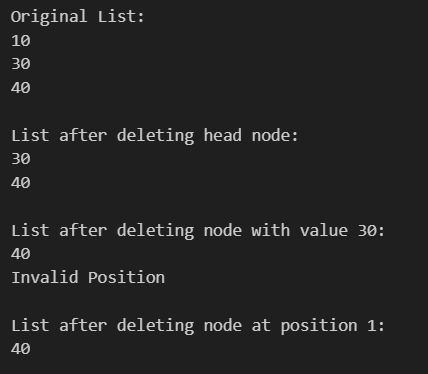
# Deleting node at a specific position

linkedList.deleteAt(1) # Deleting the second node (value 40)

print("\nList after deleting node at position 1:")

linkedList.printList()

**OUTPUT**

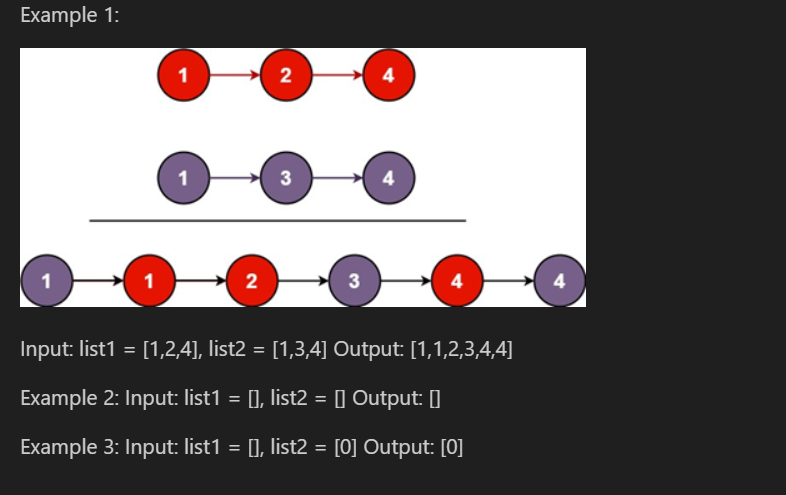


**Task No. 02**

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

Merge the two lists into 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.



**Code**

class ListNode:

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

self.val = val

self.next = next

class Solution:

def mergeTwoLists(self, list1, list2):

if list1 is None or list2 is None:

return list1 or list2

if list1.val <= list2.val:

list1.next = self.mergeTwoLists(list1.next, list2)

return list1

else:

list2.next = self.mergeTwoLists(list1, list2.next)

return list2

# Helper function to convert a Python list into a linked list

def list\_to\_linked\_list(arr):

if not arr:

return None

head = ListNode(arr[0])

current = head

for val in arr[1:]:

current.next = ListNode(val)

current = current.next

return head

# Helper function to convert a linked list back to a Python list

def linked\_list\_to\_list(head):

result = []

while head:

result.append(head.val)

head = head.next

return result

# Test cases

test\_cases = [

([1, 2, 4], [1, 3, 4]), # Expected Output: [1,1,2,3,4,4]

([], []), # Expected Output: []

([], [0]) # Expected Output: [0]

]

# Running the test cases

solution = Solution()

for list1, list2 in test\_cases:

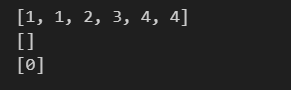
l1 = list\_to\_linked\_list(list1)

l2 = list\_to\_linked\_list(list2)

merged\_head = solution.mergeTwoLists(l1, l2)

print(linked\_list\_to\_list(merged\_head))

**OUTPUT**



**Lab No. 06**

**[Exploring Recursion for Problem Solving]**

**Task No. 01**

Implement python code for factorial of a number.

**Code**

# Python also accepts function recursion, which means a defined function can call itself.

def factorial(n):

if(n==0 or n==1):

return 1

else:

return n \* factorial(n-1) #==> called the factorial(n-1) function inside factorial(n)

print(factorial(5))

# it go inside 'else'.

# It will ne told to calculate 5 \* factorial(n==4) so,

# 5 \* factorial(4)

# 5 \* 4 \* factorial(3)

# 5 \* 4 \* 3 \* factorial(2)

# 5 \* 4 \* 3 \* 2 \* factorial(1) -> this is n==1 it go inside 'if' & print 1

# 5 \* 4 \* 3 \* 2 \* 1

**OUTPUT**



**Task No. 02**

Given an integer n, return true if it is a power of two. Otherwise, return false. An integer n is a power of two, if there exists an integer x such that n == 2x.

Example 1:

Input: n = 1

Output: true

Explanation: 20 = 1

Example 2:

Input: n = 16

Output: true

Explanation: 24 = 16

Example 3:

Input: n = 3

Output: false

**Code**

def ispowerofTwo(n):

if (n == 0):

return False

while (n != 1):

if (n % 2 != 0):

return False

n = n // 2

return True

ispowerofTwo(3)

**OUTPUT**



**Task No. 03**

Given an integer n, return true if it is a power of three. Otherwise, return false. An integer n is a power of three, if there exists an integer x such that n == 3x.

Example 1:

Input: n = 27

Output: true

Explanation: 27 = 33

Example 2:

Input: n = 0

Output: false

Explanation: There is no x where 3x = 0.

Example 3:

Input: n = -1

Output: false

Explanation: There is no x where 3x = (-1).

**Code**

def isPowerofThree(n):

if n <= 0:

return False

while n % 3 == 0:

n /= 3

return n == 1

isPowerofThree(3)

**OUTPUT**



**Task No. 04**

Given an integer n, return true if it is a power of four. Otherwise, return false. An integer n is a power of four, if there exists an integer x such that n == 4x.

Example 1:

Input: n = 16

Output: true

Example 2:

Input: n = 5

Output: false

Example 3:

Input: n = 1

Output: true

**Code**

def isPowerOfFour(n):

# Check for non-positive numbers

if n <= 0:

return False

# Continuously divide by 4 until n is no longer divisible

while n % 4 == 0:

n /= 4

# If n is 1, it's a power of four

return n == 1

isPowerOfFour(4)

**OUTPUT**



**Lab No. 07**

**[Understanding and Applying Basic Sorting Algorithms]**

**Task No. 01**

Develop Python programs for Bubble Sort, Selection Sort, and Insertion Sort.

**Code**

# bubble sort

def bubblesort(array):

n = len(array)

for i in range(n-1):

swapped = False

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

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

array[j],array[j+1]=array[j+1],array[j]

swapped = True

if (not swapped):

return

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

bubblesort(array)

print('sorted array in Ascending order:')

print(array)

**OUTPUT**



**Selection Sort**

def selection\_sort(array):

length = len(array)

for i in range(length-1):

minIndex = i

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

if array[j]<array[minIndex]:

minIndex = j

array[i], array[minIndex] = array[minIndex], array[i]

return array

array = [21,6,9,33,3]

print("The sorted array is: ", selection\_sort(array))

**OUTPUT**



**Insertion Sort**

def insertionSort(array):

for step in range(1, len(array)):

key = array[step]

j = step - 1

# Compare key with each element on the left of it until an element smaller than it is found

# For descending order, change key<array[j] to key>array[j].

while j >= 0 and key < array[j]:

array[j + 1] = array[j]

j = j - 1

# Place key at after the element just smaller than it.

array[j + 1] = key

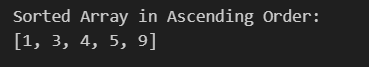
data = [9, 5, 1, 4, 3]

insertionSort(data)

print('Sorted Array in Ascending Order:')

print(data)

**OUTPUT**



**Task No. 02**

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Example 1:

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

Output: true

Explanation: The element 1 occurs at the indices 0 and 3.

Example 2:

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

Output: false

Explanation: All elements are distinct.

Example 3:

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

Output: true

**Code**

def containsDuplicate(nums):

# Create an empty set to store unique elements

unique\_set = set()

# Iterate through the array

for num in nums:

# If the element is already in the set, it's a duplicate

if num in unique\_set:

return True

# Otherwise, add the element to the set

unique\_set.add(num)

# If the loop completes without returning, there are no duplicates

return False

containsDuplicate([1,2,22,4])

**OUTPUT**



**Task No. 03**

Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

Example 1:

Input: nums = [3,0,1]

Output: 2

Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

Example 2:

Input: nums = [0,1]

Output: 2

Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Example 3:

Input: nums = [9,6,4,2,3,5,7,0,1]

Output: 8

Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.

**Code**

from typing import List

class Solution:

def missingNumber(self, nums: List[int]) -> int:

nums.sort()

# ensure n is at the last index

if nums[-1] != len(nums):

return len(nums)

# ensure 0 is at the first index

elif nums[0] != 0:

return 0

# otherwise, the missing number is in the range (0, n)

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

expected\_num = nums[i - 1] + 1

if nums[i] != expected\_num:

return expected\_num

nums = [3, 0, 1]

solution = Solution()

print(solution.missingNumber(nums))

**OUTPUT**



**Lab No. 08**

**[Applying the Divide-and-Conquer Approach to Sorting]**

**Task No. 01**

Write python implementations on Merge Sort & Quick Sort

**Code**

def mergesort(arr):

if len(arr) <= 1:

return

mid = len(arr) // 2 # [1,3,5,2,9,4] (6/2--- mid = 3)

left = arr[:mid] # 0-(mid-1) or 0-2

right = arr[mid:] # 3-last tak

# left = [1,3,5]

# right = [2,9,4]

mergesort(left)

mergesort(right)

MergeTwosortList(arr, left, right)

def MergeTwosortList(arr, left, right):

a = len(left)

b = len(right)

i = j = k = 0

while i < a and j < b:

if left[i] < right[j]:

arr[k] = left[i]

i += 1

else:

arr[k] = right[j]

j += 1

k += 1

while i < a:

arr[k] = left[i]

i += 1

k += 1

while j < b:

arr[k] = right[j]

j += 1

k += 1

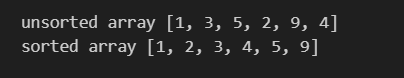
arr = [1,3,5,2,9,4]

print('unsorted array',arr)

mergesort(arr) # Sorting is done in place

print('sorted array',arr) # Print the sorted array

**OUTPUT**



**Code**

# quick sort

def QuickSort(arr, low, high):

if low < high:

pivot = partition(arr, low, high)

QuickSort(arr, low, pivot - 1)

QuickSort(arr, pivot + 1, high)

def partition(arr, low, high):

pivot = arr[high] #pivot = 5

i = low - 1 #i=1

for j in range(low, high):

if arr[j] < pivot:

i += 1 # i=0(index)

arr[i], arr[j] = arr[j], arr[i] #swapping

# 11 , 1 = 1 , 11

i += 1

arr[i], arr[high] = arr[high], arr[i]

return i

# [1, 7, 8, 9, 11, 5]

# low high

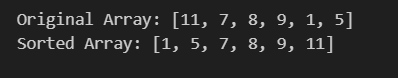
arr = [11, 7, 8, 9, 1, 5]

print("Original Array:", arr)

QuickSort(arr, 0, len(arr) - 1)

print("Sorted Array:", arr)

**OUTPUT**



**Task No. 02**

You are given an integer array score of size n, where score[i] is the score of the ith athlete in a competition. All the scores are guaranteed to be unique.

The athletes are placed based on their scores, where the 1st place athlete has the highest score, the 2nd place athlete has the 2nd highest score, and so on. The placement of each athlete determines their rank:

• The 1st place athlete's rank is "Gold Medal".

• The 2nd place athlete's rank is "Silver Medal".

• The 3rd place athlete's rank is "Bronze Medal".

• For the 4th place to the nth place athlete, their rank is their placement number (i.e., the xth place athlete's rank is "x").

Return an array answer of size n where answer[i] is the rank of the ith athlete.

Example 1:

Input: score = [5,4,3,2,1]

Output: ["Gold Medal","Silver Medal","Bronze Medal","4","5"]

Explanation: The placements are [1st, 2nd, 3rd, 4th, 5th].

Example 2:

Input: score = [10,3,8,9,4]

Output: ["Gold Medal","5","Bronze Medal","Silver Medal","4"]

Explanation: The placements are [1st, 5th, 3rd, 2nd, 4th].

**Code**

# Given scores of N athletes, find their relative ranks and the people with the top three highest scores, who will be awarded medals: "Gold Medal", "Silver Medal" and "Bronze Medal".

def findrelativeRanks(score):

sorted\_score = sorted(score, reverse=True)

rank\_map = {}

for i, value in enumerate(sorted\_score):

if i == 0:

rank\_map[value] = "Gold Medal"

elif i == 1:

rank\_map[value] = "Silver Medal"

elif i == 2:

rank\_map[value] = "Bronze Medal"

else:

rank\_map[value] = str(i+1)

result = []

for s in score:

result.append(rank\_map[s])

return result

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

findrelativeRanks(score)

**OUTPUT**



**Task No. 03**

Given two strings s and t, return true if t is an anagram of s, and false otherwise.

Example 1:

Input: s = "anagram", t = "nagaram"

Output: true

Example 2:

Input: s = "rat", t = "car"

Output: false

**Code**

from collections import Counter

class Solution:

def isAnagram(self,s:str,t:str) -> bool:

if len(s) != len(t):

return False

s\_dict = Counter(s)

t\_dict = Counter(t)

return s\_dict == t\_dict

s1= Solution()

print(s1.isAnagram("anagram","nagaram"))

**OUTPUT**



**Task No. 04**

Given an integer array nums, return the maximum difference between two successive elements in its sorted form. If the array contains less than two elements, return 0.

You must write an algorithm that runs in linear time and uses linear extra space.

Example 1:

Input: nums = [3,6,9,1]

Output: 3

Explanation: The sorted form of the array is [1,3,6,9], either (3,6) or (6,9) has the maximum difference 3.

Example 2:

Input: nums = [10]

Output: 0

Explanation: The array contains less than 2 elements, therefore return 0.

**Code**

def maximumGap(nums):

# sort the array

nums.sort()

# find the max gap

max\_gap = 0

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

max\_gap = max(nums[i] - nums[i-1], max\_gap)

return max\_gap

maximumGap([3,6,9,1])

**OUTPUT**



**Lab No. 09**

**[Utilizing HashMaps for Efficient Data Storage and Retrieval]**

**Task No. 01**

Implement a Python function for hashing with collision handling using chaining.

**Code**

class HashTable:

def \_\_init\_\_(self, size):

self.size = size

self.table = [[] for \_ in range(size)] # Initialize the table with empty lists (chaining)

def hash\_function(self, key):

return hash(key) % self.size # Simple modulo-based hash function

def insert(self, key, value):

index = self.hash\_function(key)

# Check if the key already exists, update it

for pair in self.table[index]:

if pair[0] == key:

pair[1] = value

return

# If key doesn't exist, append the new key-value pair

self.table[index].append([key, value])

def get(self, key):

index = self.hash\_function(key)

for pair in self.table[index]:

if pair[0] == key:

return pair[1]

return None # Key not found

def remove(self, key):

index = self.hash\_function(key)

for i, pair in enumerate(self.table[index]):

if pair[0] == key:

del self.table[index][i]

return True

return False # Key not found

def display(self):

for i, bucket in enumerate(self.table):

print(f"Index {i}: {bucket}")

# Example Usage

hash\_table = HashTable(5)

hash\_table.insert("apple", 10)

hash\_table.insert("banana", 20)

hash\_table.insert("grape", 30)

hash\_table.insert("orange", 40)

hash\_table.insert("lemon", 50)

print("Hash Table after insertions:")

hash\_table.display()

print("\nRetrieving values:")

print("apple:", hash\_table.get("apple"))

print("banana:", hash\_table.get("banana"))

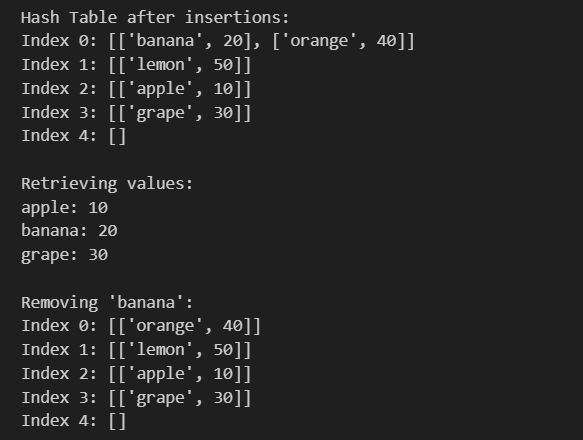
print("grape:", hash\_table.get("grape"))

print("\nRemoving 'banana':")

hash\_table.remove("banana")

hash\_table.display()

**OUTPUT**



**Task No. 02**

Given two strings s and t, determine if they are isomorphic.

Two strings s and t are isomorphic if the characters in s can be replaced to get t.

All occurrences of a character must be replaced with another character while preserving the order of characters. No two characters may map to the same character, but a character may map to itself.

Example 1:

Input: s = "egg", t = "add"

Output: true

Explanation:

The strings s and t can be made identical by:

• Mapping 'e' to 'a'.

• Mapping 'g' to 'd'.

Example 2:

Input: s = "foo", t = "bar"

Output: false

Explanation:

The strings s and t can not be made identical as 'o' needs to be mapped to both 'a' and 'r'.

Example 3:

Input: s = "paper", t = "title"

Output: true

**Code**

def isIsomorphicHelper(s: str, t: str) -> bool:

if len(s) != len(t):

return False

s\_to\_t = {}

t\_to\_s = {}

for char\_s, char\_t in zip(s, t):

# Check if the current s character is already mapped

if char\_s in s\_to\_t:

if s\_to\_t[char\_s] != char\_t:

return False

else:

# Check if the current t character is already mapped by another s character

if char\_t in t\_to\_s:

return False

# Create new mappings

s\_to\_t[char\_s] = char\_t

t\_to\_s[char\_t] = char\_s

return True

# Main function to call helper

def isIsomorphic(s: str, t: str) -> bool:

return isIsomorphicHelper(s, t)

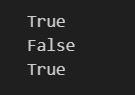
# Test cases

print(isIsomorphic("egg", "add")) # Output: True

print(isIsomorphic("foo", "bar")) # Output: False

print(isIsomorphic("paper", "title")) # Output: True

**OUTPUT**



**Lab No. 10**

**[Implementing Binary Search Trees for Efficient Searching]**

**Task No. 01**

Implement Binary Search Trees (BSTs) in Python with Inorder Traversal, a function to add nodes, and a searching function.

**Code**

class Node:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

class BST:

def \_\_init\_\_(self):

self.root = None

def insert(self, key):

new\_node = Node(key)

if self.root is None:

self.root = new\_node

return

current = self.root

while True:

if key < current.key:

if current.left is None:

current.left = new\_node

return

else:

current = current.left

else:

if current.right is None:

current.right = new\_node

return

else:

current = current.right

def search(self, key):

current = self.root

while current is not None:

if key == current.key:

return True

elif key < current.key:

current = current.left

else:

current = current.right

return False

def inorder\_traversal(self):

result = []

self.\_inorder\_helper(self.root, result)

return result

def \_inorder\_helper(self, node, result):

if node is not None:

self.\_inorder\_helper(node.left, result)

result.append(node.key)

self.\_inorder\_helper(node.right, result)

new\_node = Node(10) # Creates a node with key 10

print(new\_node.key) # Output: 10

print(new\_node.left) # Output: None

print(new\_node.right) # Output: None

root = Node(15) # Root node

root.left = Node(10) # Left child of root

root.right = Node(20) # Right child of root

root.left.left = Node(5) # Left child of 10

root.left.right = Node(12) # Right child of 10

**OUTPUT**

