Started on	Saturday, 10 May 2025, 2:19 PM
State	Finished
Completed on	Saturday, 10 May 2025, 3:00 PM
Time taken	40 mins 36 secs
Grade	**80.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Create a python program to find the longest palindromic substring using optimal algorithm Expand around center.

For example:

Test	Input	Result
`findLongestPalindromicSubstring(s)`	`samsunggnusgnusam`	`sunggnus`

Answer: (penalty regime: 0 %)

Reset answer

```
def expand(s, low, high):
    length = len(s)

    while low >= 0 and high < length and s[low] == s[high]:
        low = low - 1
        high = high + 1

    return s[low + 1:high]


def findLongestPalindromicSubstring(s):

    dp = [[False for i in range(len(s))] for i in range(len(s))]
    for i in range(len(s)):
        dp[i][i] = True
        max_length = 1
        start = 0
        for l in range(2,len(s)+1):
            for i in range(len(s)-l+1):
                end = i+l
                if l==2:
```

	Test	Input	Expected	Got	
✔	`findLongestPalindromicSubstring(s)`	`samsunggnusgnusam`	`sunggnus`	`sunggnus`	✔
✔	`findLongestPalindromicSubstring(s)`	`welcomeindiaaidni`	`indiaaidni`	`indiaaidni`	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **2**

Not answered

Mark 0.00 out of 20.00

Write a python program to implement merge sort without using recursive function on the given list of float values.

For example:

Input	Result
5 6.2 4.1 3.2 5.6 7.4	left: [6.2] Right: [4.1] left: [3.2] Right: [5.6] left: [7.4] Right: [] left: [4.1, 6.2] Right: [3.2, 5.6] left: [7.4] Right: [] left: [3.2, 4.1, 5.6, 6.2] Right: [7.4] [3.2, 4.1, 5.6, 6.2, 7.4]
6 3.2 8.9 4.5 6.2 1.5 8.0	left: [3.2] Right: [8.9] left: [4.5] Right: [6.2] left: [1.5] Right: [8.0] left: [3.2, 8.9] Right: [4.5, 6.2] left: [1.5, 8.0] Right: [] left: [3.2, 4.5, 6.2, 8.9] Right: [1.5, 8.0] [1.5, 3.2, 4.5, 6.2, 8.0, 8.9]

Answer: (penalty regime: 0 %)

```
1
```

Question **3**

Correct

Mark 20.00 out of 20.00

To Write a Python Program to find longest common subsequence using Dynamic Programming

For example:

Input	Result
abcbdab bdcaba	bdab

Answer: (penalty regime: 0 %)

```
def lcs(u, v):
    c = [[-1]*(len(v) + 1) for _ in range(len(u) + 1)]
    for i in range(len(u) + 1):
        c[i][len(v)] = 0
    for j in range(len(v)):
        c[len(u)][j] = 0

    for i in range(len(u) - 1, -1, -1):
        for j in range(len(v) - 1, -1, -1):
            if u[i] == v[j]:
                c[i][j] = 1 + c[i + 1][j + 1]
            else:
                c[i][j] = max(c[i + 1][j], c[i][j + 1])
    return c

def print_lcs(u, v, c):
    i = j = 0
    while not (i == len(u) or j == len(v)):
        if u[i] == v[j]:
            print(u[i], end='')
            i += 1
```

	Input	Expected	Got	
✔	abcbdab bdcaba	bdab	bdab	✔
✔	treehouse elephant	eeh	eeh	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **4**

Correct

Mark 20.00 out of 20.00

Create a python program to find the Edit distance between two strings using dynamic programming.

For example:

Input	Result
Cats Rats	No. of Operations required : 1

Answer: (penalty regime: 0 %)

Reset answer

```
def LD(s, t):
    if s == "":
        return len(t)
    if t == "":
        return len(s)
    if s[-1] == t[-1]:
        cost = 0
    else:
        cost = 1
    res = min([LD(s[:-1], t)+1,
               LD(s, t[:-1])+1,
               LD(s[:-1], t[:-1]) + cost])
    return res

str1=input()
str2=input()
print("No. of Operations required :",LD(str1,str2))

```

	Input	Expected	Got	
✔	Cats Rats	No. of Operations required : 1	No. of Operations required : 1	✔
✔	Saturday Sunday	No. of Operations required : 3	No. of Operations required : 3	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

Create a Python program to find longest common substring or subword (LCW) of two strings using dynamic programming with top-down approach or memoization.

Problem Description

A string r is a substring or subword of a string s if r is contained within s. A string r is a common substring of s and t if r is a substring of both s and t. A string r is a longest common substring or subword (LCW) of s and t if there is no string that is longer than r and is a common substring of s and t. The problem is to find an LCW of two given strings.

For example:

Test	Input	Result
`lcw(u, v)`	potato tomato	`Longest Common Subword: ato`

Answer: (penalty regime: 0 %)

Reset answer

```
def lcw(u, v):
    c = [[-1]*(len(v) + 1) for _ in range(len(u) + 1)]
    lcw_i = lcw_j = -1
    length_lcw = 0
    for i in range(len(u)):
        for j in range(len(v)):
            temp = lcw_starting_at(u, v, c, i, j)
            if length_lcw < temp:
                length_lcw = temp
                lcw_i = i
                lcw_j = j
    return length_lcw, lcw_i, lcw_j
def lcw_starting_at(u, v, c, i, j):
    ############ Add your code here ############

    if i >= len(u) or j >= len(v):
        return 0
    if c[i][j] != -1:
        return c[i][j]
    if u[i] == v[j]:
        c[i][j] = 1 + lcw_starting_at(u, v, c, i + 1, j + 1)
```

	Test	Input	Expected	Got	
✔	`lcw(u, v)`	potato tomato	`Longest Common Subword: ato`	`Longest Common Subword: ato`	✔
✔	`lcw(u, v)`	snakegourd bottlegourd	`Longest Common Subword: egourd`	`Longest Common Subword: egourd`	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.