Started on	Thursday, 15 May 2025, 1:35 PM
State	Finished
Completed on	Thursday, 15 May 2025, 1:36 PM
Time taken	1 min 34 secs
Grade	**80.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Create a python program to for the following problem statement.

You are given an `n x n grid` representing a field of cherries, each cell is one of three possible integers.

- `0` means the cell is empty, so you can pass through,
- `1` means the cell contains a cherry that you can pick up and pass through, or
- `-1` means the cell contains a thorn that blocks your way.

Return *the maximum number of cherries you can collect by following the rules below*:

- Starting at the position `(0, 0)` and reaching `(n - 1, n - 1)` by moving right or down through valid path cells (cells with value `0` or `1`).
- After reaching `(n - 1, n - 1)`, returning to `(0, 0)` by moving left or up through valid path cells.
- When passing through a path cell containing a cherry, you pick it up, and the cell becomes an empty cell `0`.
- If there is no valid path between `(0, 0)` and `(n - 1, n - 1)`, then no cherries can be collected.

For example:

Test	Result
obj.cherryPickup(grid)	5

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
class Solution:
    def cherryPickup(self, grid):
        n = len(grid)
        ### add code here
        dp=[[[-1]*n for _ in range(n)] for _ in range(n)]
        def f(x1,y1,x2):
            y2=x1+y1-x2
            if x1<0 or y1<0 or x2<0 or y2<0 or grid[x1][y1]==-1 or grid[x2][y2]==-1:
                return float('-inf')
            if x1==0 and y1==0 and x2==0 and y2==0:
                return grid[0][0]
            if dp[x1][y1][x2]!=-1:
                return dp[x1][y1][x2]
            cherries=grid[x1][y1]
            if x1!=x2 or y1!=y2:
                cherries+=grid[x2][y2]
```

	Test	Expected	Got	
✔	obj.cherryPickup(grid)	5	5	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **2**

Correct

Mark 20.00 out of 20.00

Create a python program using brute force method of searching for the given substring in the main string.

For example:

Test	Input	Result
match(str1,str2)	AABAACAADAABAABA AABA	Found at index 0 Found at index 9 Found at index 12

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
def match(string,sub):
    l=len(string)
    l2=len(sub)
    for i in range(l-l2+1):
        if string[i:i+l2]==sub:
            print("Found at index",i)
str1=input()
str2=input()
```

	Test	Input	Expected	Got	
✔	match(str1,str2)	AABAACAADAABAABA AABA	Found at index 0 Found at index 9 Found at index 12	Found at index 0 Found at index 9 Found at index 12	✔
✔	match(str1,str2)	saveetha savee	Found at index 0	Found at index 0	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **3**

Correct

Mark 20.00 out of 20.00

Create a python program for 0/1 knapsack problem using naive recursion method

For example:

Test	Input	Result
knapSack(W, wt, val, n)	3 3 50 60 100 120 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 220

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
def knapSack(W, wt, val, n):
    if n==0 or W==0:
        return 0
    if wt[n-1]>W:
        return knapSack(W, wt, val, n-1)
    return max(val[n-1]+knapSack(W-wt[n-1], wt, val, n-1),knapSack(W, wt, val, n-1))

x=int(input())
y=int(input())
W=int(input())
val=[]
wt=[]
for i in range(x):
    val.append(int(input()))
for y in range(y):
    wt.append(int(input()))
```

	Test	Input	Expected	Got	
✔	knapSack(W, wt, val, n)	3 3 50 60 100 120 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 220	The maximum value that can be put in a knapsack of capacity W is: 220	✔

	Test	Input	Expected	Got	
✔	knapSack(W, wt, val, n)	3 3 55 65 115 125 15 25 35	The maximum value that can be put in a knapsack of capacity W is: 190	The maximum value that can be put in a knapsack of capacity W is: 190	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **4**

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 values.

For example:

Input	Result
7 33 42 9 37 8 47 5	left: [33] Right: [42] left: [9] Right: [37] left: [8] Right: [47] left: [5] Right: [] left: [33, 42] Right: [9, 37] left: [8, 47] Right: [5] left: [9, 33, 37, 42] Right: [5, 8, 47] [5, 8, 9, 33, 37, 42, 47]
6 10 3 5 61 74 92	left: [10] Right: [3] left: [5] Right: [61] left: [74] Right: [92] left: [3, 10] Right: [5, 61] left: [74, 92] Right: [] left: [3, 5, 10, 61] Right: [74, 92] [3, 5, 10, 61, 74, 92]

Answer: (penalty regime: 0 %)

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

Question **5**

Correct

Mark 20.00 out of 20.00

Given a 2D matrix **tsp[][]**, where each row has the array of distances from that indexed city to all the other cities and **-1** denotes that there doesn't exist a path between those two indexed cities. The task is to print minimum cost in TSP cycle.

tsp[][] = {{-1, 30, 25, 10},
{15, -1, 20, 40},
{10, 20, -1, 25},
{30, 10, 20, -1}};

Answer: (penalty regime: 0 %)

Reset answer

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
from sys import maxsize
from itertools import permutations
V = 4


def travellingSalesmanProblem(graph, s):

   #Write your code
    v=[]
    for i in range(V):
        if i!=s:
            v.append(i)
    mp=maxsize
    np=permutations(v)
    for i in np:
        k=s
        cp=0
```

	Expected	Got	
✔	Minimum Cost is : 50	Minimum Cost is : 50	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.