Started on	Tuesday, 20 May 2025, 2:28 PM
State	Finished
Completed on	Monday, 26 May 2025, 4:04 PM
Time taken	6 days 1 hour
Overdue	5 days 23 hours
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(object):
    def cherryPickup(self, grid):
        def dp(i, j, k):
            if (i, j, k) in memo:
                return memo[(i, j, k)]

            if i == ROW_NUM - 1:
                return grid[i][j] + (grid[i][k] if j != k else 0)

            cherries = grid[i][j] + (grid[i][k] if j != k else 0)

            max_cherries = 0
            for dj in [-1, 0, 1]:
                for dk in [-1, 0, 1]:
                    next_j, next_k = j + dj, k + dk
                    if 0 <= next_j < COL_NUM and 0 <= next_k < COL_NUM:
                        max_cherries = max(max_cherries, dp(i + 1, next_j, next_k))
```

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

Passed all tests! ✔

Correct

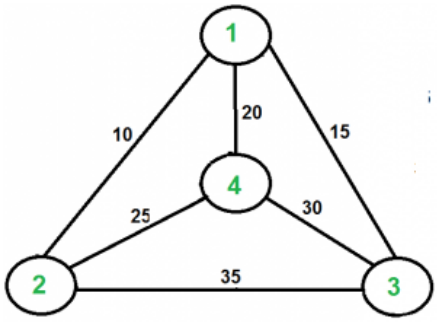
Marks for this submission: 20.00/20.00.

Question **2**

Correct

Mark 20.00 out of 20.00

Solve Travelling Sales man Problem for the following graph



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
    vertex = []
    for i in range(V):
        if i != s:
            vertex.append(i)
    min_path = maxsize
    next_permutation=permutations(vertex)
    for i in next_permutation:

        current_pathweight = 0
        k = s
        for j in i:
```

	Expected	Got	
✔	80	80	✔

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):
    ########## Add your code here #########
    K = [[0 for x in range(W + 1)] for x in range(n + 1)]
    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                K[i][w] = 0
            elif wt[i-1] <= w:
                K[i][w] = max(val[i-1]+ K[i-1][w-wt[i-1]],K[i-1][w])
            else:
                K[i][w] = K[i-1][w]

    return K[n][W]

x=int(input())
y=int(input())
W=int(input())
val=[]
```

	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**

Incorrect

Mark 0.00 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(s1,s2):
    i = 0
    j = 0
    while(i < len(s1) and j < len(s2)):
        if(s1[i] ==  s2[j]):
            i += 1
            j += 1
        else:
            i = i - j + 1
            j = 0
    if(j >= len(s2)):
        return i - len(s2)
    else:
        return 0
if __name__ == "__main__":
    str1=input()
    str2=input()
    b=match(str1,str2)
```

	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	✖

Your code must pass all tests to earn any marks. Try again.

Show differences

Incorrect

Marks for this submission: 0.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

Write a recursive python function to perform merge sort on the unsorted list of float values.

For example:

Test	Input	Result
mergesort(li)	5 3.2 1.5 1.6 1.7 8.9	[1.5, 1.6, 1.7, 3.2, 8.9]
mergesort(li)	6 3.1 2.3 6.5 4.5 7.8 9.2	[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]

Answer: (penalty regime: 0 %)

Ace editor not ready. Perhaps reload page?

Falling back to raw text area.

```
def mergesort(inp_arr):
    if len(inp_arr)>1:
        mid=len(inp_arr)//2
        L=inp_arr[:mid]
        R=inp_arr[mid:]
        mergesort(L)
        mergesort(R)
        i=j=k=0
        while i<len(L) and j<len(R):
            if L[i]<R[j]:
                inp_arr[k]=L[i]
                i+=1
            else:
                inp_arr[k]=R[j]
                j+=1;
            k+=1
        while i<len(L):
            inp_arr[k]=L[i]
```

	Test	Input	Expected	Got	
✔	mergesort(li)	5 3.2 1.5 1.6 1.7 8.9	[1.5, 1.6, 1.7, 3.2, 8.9]	[1.5, 1.6, 1.7, 3.2, 8.9]	✔

	Test	Input	Expected	Got	
✔	`mergesort(li)`	6 3.1 2.3 6.5 4.5 7.8 9.2	`[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]`	`[2.3, 3.1, 4.5, 6.5, 7.8, 9.2]`	✔
✔	`mergesort(li)`	4 3.1 2.3 6.5 4.1	`[2.3, 3.1, 4.1, 6.5]`	`[2.3, 3.1, 4.1, 6.5]`	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.