Started on	Monday, 17 March 2025, 3:07 PM
State	Finished
Completed on	Monday, 17 March 2025, 3:29 PM
Time taken	21 mins 45 secs
Grade	**100.00** out of 100.00

Question **1**

Correct

Mark 20.00 out of 20.00

Create a python program using dynamic programming for 0/1 knapsack problem.

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

```
def knapSack(W, wt, val, n):
    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=[]
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	✔
✔	knapSack(W, wt, val, n)	3 3 40 50 90 110 10 20 30	The maximum value that can be put in a knapsack of capacity W is: 160	The maximum value that can be put in a knapsack of capacity W is: 160	✔

Question **2**

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 %)

```
def mergesort(li):
    if len(li) < 2:
        return li

    mid = len(li) // 2
    y = mergesort(li[:mid])
    z = mergesort(li[mid:])
    result = []

    i = 0
    j= 0

    while i < len(y) and j < len(z):
        if y[i] > z[j]:
            result.append(z[j])
            j+=1
        else:
            result.append(y[i])
            i+=1
    result += y[i:]
    result += z[j:]
    return result
```

	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]	✔
✔	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]	✔

	Test	Input	Expected	Got	
✔	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.

Question **3**

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

```
from typing import DefaultDict
INT_MAX = 2147483647
def findMinRoute(tsp):
    sum = 0
    counter = 0
    j = 0
    i = 0
    min = INT_MAX
    visitedRouteList = DefaultDict(int)

    visitedRouteList[0] = 1
    route = [0] * len(tsp)
    while i < len(tsp) and j < len(tsp[i]):
        if counter >= len(tsp[i]) - 1:
            break
        if j != i and (visitedRouteList[j] == 0):
            if tsp[i][j] < min:
                min = tsp[i][j]
                route[counter] = j + 1

        j += 1
        if i == len(tsp[i]):
```

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

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 maximum value in linear search.

For example:

Test	Input	Result
find_maximum(test_scores)	10 88 93 75 100 80 67 71 92 90 83	Maximum value is 100

Answer: (penalty regime: 0 %)

Reset answer

```
def find_maximum(test_scores):
    if not test_scores:
        return None
    mv=test_scores[0]
    for x in test_scores:
        if x>mv:
            mv=x
    return mv
test_scores = []
n=int(input())
for i in range(n):
    test_scores.append(int(input()))
print("Maximum value is ",find_maximum(test_scores))
```

	Test	Input	Expected	Got	
✔	find_maximum(test_scores)	10 88 93 75 100 80 67 71 92 90 83	Maximum value is 100	Maximum value is 100	✔
✔	find_maximum(test_scores)	5 45 86 95 76 28	Maximum value is 95	Maximum value is 95	✔

Question **5**

Correct

Mark 20.00 out of 20.00

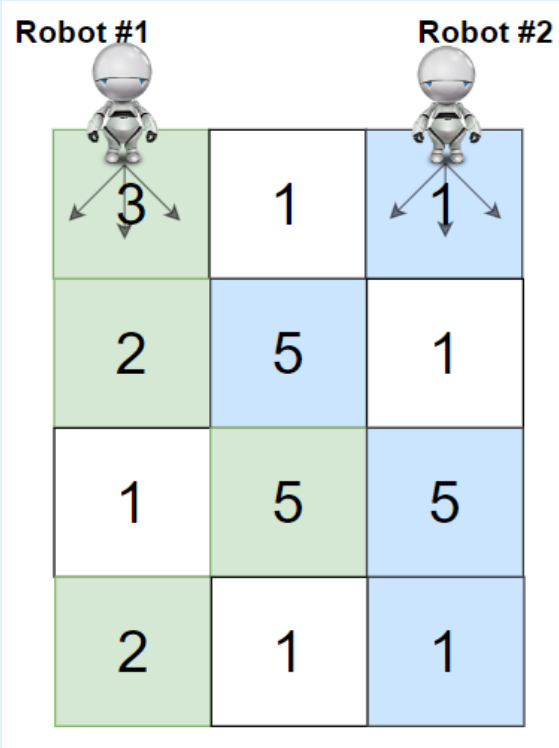
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

You have two robots that can collect cherries for you:

- **Robot #1** is located at the **top-left corner** `(0, 0)`, and
- **Robot #2** is located at the **top-right corner** `(0, cols - 1)`.

Return *the maximum number of cherries collection using both robots by following the rules below*:

- From a cell `(i, j)`, robots can move to cell `(i + 1, j - 1)`, `(i + 1, j)`, or `(i + 1, j + 1)`.
- When any robot passes through a cell, It picks up all cherries, and the cell becomes an empty cell.
- When both robots stay in the same cell, only one takes the cherries.
- Both robots cannot move outside of the grid at any moment.
- Both robots should reach the bottom row in `grid`.



For example:

Test	Result
`ob.cherryPickup(grid)`	24

Answer: (penalty regime: 0 %)

Reset answer

```
class Solution(object):
    def cherryPickup(self, grid):
        r = len(grid)
        c = len(grid[0])
        dp = [[[float('-inf')] * c for _ in range(c)] for _ in range(r)]
        dp[0][0][c - 1] = grid[0][0] + grid[0][c - 1]
        for i in range(1, r):
            for j1 in range(c):
                for j2 in range(c):
                    curr_cherries = grid[i][j1]

                    if j1 != j2:
                        curr_cherries += grid[i][j2]

                    for prev_j1 in range(j1 - 1, j1 + 2):
```

```
                    for prev_j2 in range(j2 - 1, j2 + 2):
                        if 0 <= prev_j1 < c and 0 <= prev_j2 < c:
                            prev_cherries = dp[i - 1][prev_j1][prev_j2]
                            dp[i][j1][j2] = max(dp[i][j1][j2], curr_cherries + prev_cherries)

        return max(0, dp[r - 1][0][c - 1])
grid=[[3,1,1]
```

	Test	Expected	Got	
✔	ob.cherryPickup(grid)	24	24	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.