Started on	Thursday, 22 May 2025, 9:22 AM
State	Finished
Completed on	Saturday, 24 May 2025, 1:29 PM
Time taken	2 days 4 hours
Overdue	2 days 2 hours
Grade	**100.00** out of 100.00

Question **1**

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(lst):
    max=None
    for i in lst:
        if max== None or i>max:
            max=i
    return max

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	✔

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 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

```python
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)
    else:
        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()))

n = len(val)
print('The maximum value that can be put in a knapsack of capacity W is: ',knapSack(W, wt, val, n))
```

	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	✔

Passed all tests! ✔

Correct

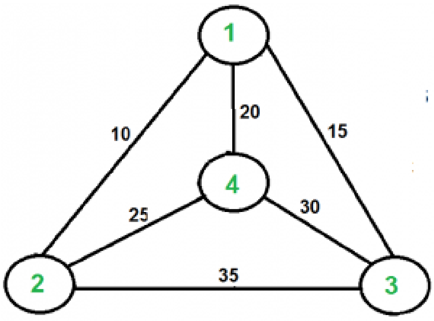
Marks for this submission: 20.00/20.00.

Question **3**

Correct

Mark 20.00 out of 20.00

Solve Travelling Sales man Problem for the following graph



Answer: (penalty regime: 0 %)

Reset answer

```
from sys import maxsize
from itertools import permutations
V = 4
def travellingSalesmanProblem(graph, s):
    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:
            current_pathweight += graph[k][j]
            k = j
        current_pathweight += graph[k][s]
        min_path = min(min_path, current_pathweight)

    return min_path


```

	Expected	Got	
✔	80	80	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question 4

Correct

Mark 20.00 out of 20.00

Write a Python program to sort unsorted numbers using Multi-key quicksort

For example:

Test	Input	Result
quick_sort_3partition(nums, 0, len(nums)-1)	5 4 3 5 1 2	Original list: [4, 3, 5, 1, 2] After applying Random Pivot Quick Sort the said list becomes: [1, 2, 3, 4, 5]
quick_sort_3partition(nums, 0, len(nums)-1)	6 21 10 3 65 4 8	Original list: [21, 10, 3, 65, 4, 8] After applying Random Pivot Quick Sort the said list becomes: [3, 4, 8, 10, 21, 65]

Answer: (penalty regime: 0 %)

```python
def quick_sort_3partition(sorting: list, left: int, right: int) -> None:
    if right <= left:
        return
    a = i = left
    b = right
    pivot = sorting[left]
    while i <= b:
        if sorting[i] < pivot:
            sorting[a], sorting[i] = sorting[i], sorting[a]
            a += 1
            i += 1
        elif sorting[i] > pivot:
            sorting[b], sorting[i] = sorting[i], sorting[b]
            b -= 1
        else:
            i += 1
    quick_sort_3partition(sorting, left, a - 1)
    quick_sort_3partition(sorting, b + 1, right)
def three_way_radix_quicksort(sorting: list) -> list:
    if len(sorting) <= 1:
        return sorting
    return (
```

	Test	Input	Expected	Got	
✔	quick_sort_3partition(nums, 0, len(nums)-1)	5 4 3 5 1 2	Original list: [4, 3, 5, 1, 2] After applying Random Pivot Quick Sort the said list becomes: [1, 2, 3, 4, 5]	Original list: [4, 3, 5, 1, 2] After applying Random Pivot Quick Sort the said list becomes: [1, 2, 3, 4, 5]	✔
✔	quick_sort_3partition(nums, 0, len(nums)-1)	6 21 10 3 65 4 8	Original list: [21, 10, 3, 65, 4, 8] After applying Random Pivot Quick Sort the said list becomes: [3, 4, 8, 10, 21, 65]	Original list: [21, 10, 3, 65, 4, 8] After applying Random Pivot Quick Sort the said list becomes: [3, 4, 8, 10, 21, 65]	✔

	Test	Input	Expected	Got	
✔	`quick_sort_3partition(nums, 0, len(nums)-1)`	4 21 3 10 4	Original list: [21, 3, 10, 4] After applying Random Pivot Quick Sort the said list becomes: [3, 4, 10, 21]	Original list: [21, 3, 10, 4] After applying Random Pivot Quick Sort the said list becomes: [3, 4, 10, 21]	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

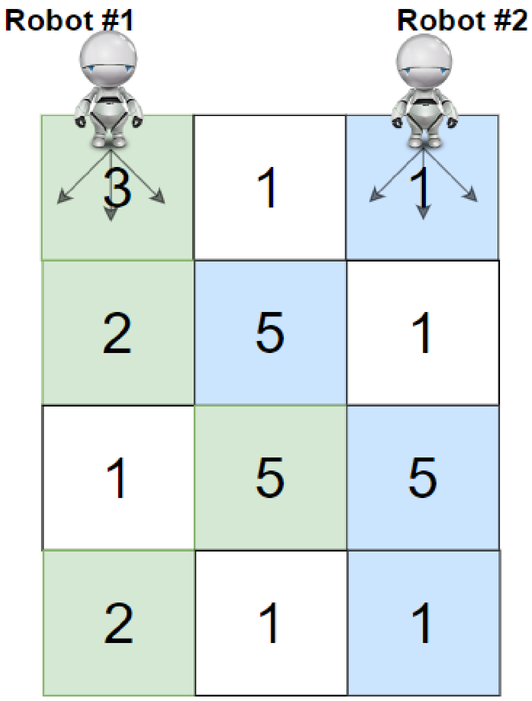
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

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):
        dp = [[0 for i in range(len(grid))] for j in range(len(grid))]
        for i in range(len(grid)):
            for j in range(len(grid)):
                dp[i][j] = grid[i-1][j-1]
        res = len(grid)*6
        ROW_NUM = len(grid)
        COL_NUM = len(grid[0])
        return dp[0][COL_NUM - 1]*res

grid=[[3,1,1],
      [2,5,1],
      [1,5,5],
      [2,1,1]]
ob=Solution()
print(ob.cherryPickup(grid))
```

```
print(ob.cherryPickup(grid))
```

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

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.