Started on	Saturday, 24 May 2025, 1:25 PM
State	Finished
Completed on	Saturday, 24 May 2025, 2:36 PM
Time taken	1 hour 11 mins
Grade	**80.00** out of 100.00

Question **1**

Not answered

Mark 0.00 out of 20.00

Write a python program to implement quick sort on the given array values.

For example:

Input	Result
5 21 40 50 30 13	left: [] right: [] left: [] right: [] left: [] right: [] left: [30] right: [50] left: [13] right: [30, 40, 50] [13, 21, 30, 40, 50]
6 7 5 21 63 4 9	left: [] right: [] left: [4] right: [] left: [] right: [] left: [] right: [] left: [9] right: [63] left: [4, 5] right: [9, 21, 63] [4, 5, 7, 9, 21, 63]

Answer: (penalty regime: 0 %)

```
1
```

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

```
import re #Import this package
def match(str1,str2):
    ############ Add your code here #######
    #Start here
    pattern = re.compile(str2)
    r = pattern.search(str1)
    while r:
        print("Found at index {}".format(r.start()))
        r = pattern.search(str1,r.start() + 1)
    #End here
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

```
def knapSack(W, wt, val, n):
    ############### Add your code here ##############
    #Start here
    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, va
    #End here
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
```

	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

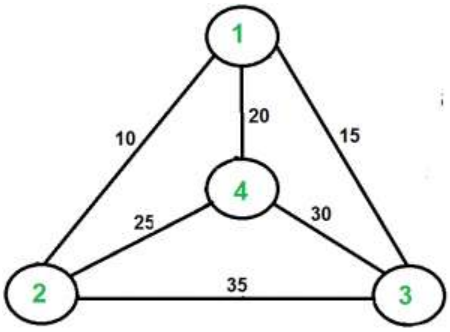
Marks for this submission: 20.00/20.00.

Question **4**

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):
    #Start here
    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)
```

	Expected	Got	
✔	80	80	✔

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.

Question **5**

Correct

Mark 20.00 out of 20.00

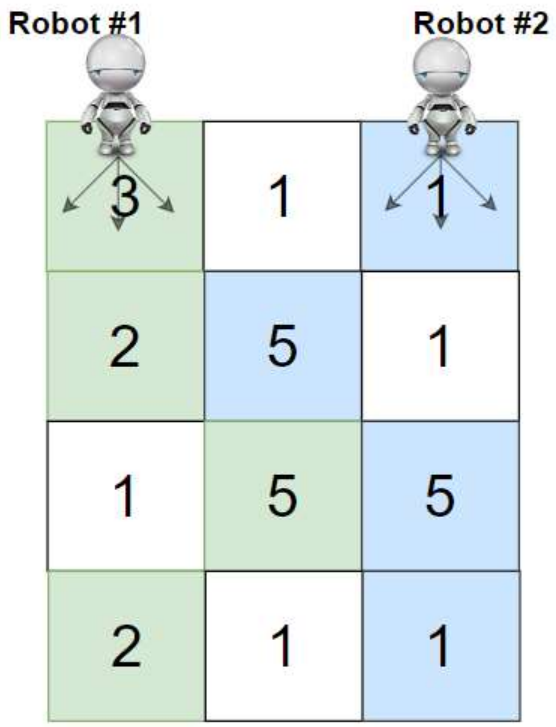
You are given a `rows x cols` matrix `grid` representing a field of cherries where `grid[i][j]` represents the number of cherries that you can collect from the `(i, j)` cell.

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):
        def dp(k):
            #########  Add your code here ##########
            #Start here
            if k == ROW_NUM - 1:
                return [[grid[-1][i] if i == j else grid[-1][i] + grid[-1][j] for
                         for i in range(COL_NUM)]
            row = grid[k]
```

```
            ans = [[0] * COL_NUM for i in range(COL_NUM)]
            next_dp = dp(k + 1)
            for i in range(COL_NUM):
                for j in range(i, COL_NUM):
                    for di in [-1, 0, 1]:
                        for dj in [-1, 0, 1]:
                            if 0 <= i + di < COL_NUM and 0 <= j + dj < COL_NUM:
                                if i == j:
                                    ans[i][j] = max(ans[i][j], next_dp[i + di][j
                                else:
                                    ans[i][j] = max(ans[i][j], next_dp[i + di][j
            return ans
```

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

Passed all tests! ✔

Correct

Marks for this submission: 20.00/20.00.