Name : Pratik Suryawanshi

ASU ID: 1213231238

ASURITE : psuryawa

Problem 1:

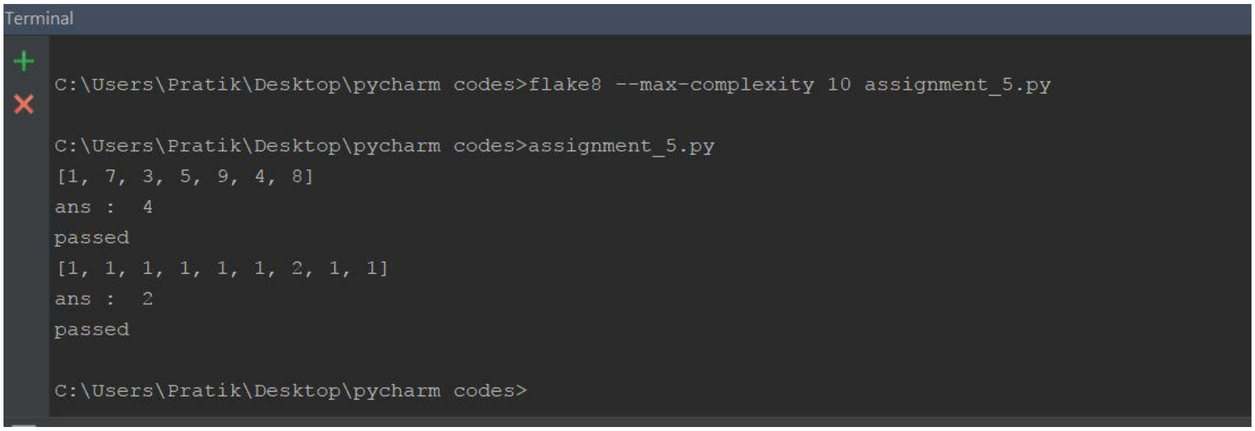
```
def longest_ordered_subsequence(l):
    n = len(l)
    dp = [1] * n
    if n == 0:
        return 0
    result = 1
    for i in range(1, n):
        for j in range(0, i):
            if l[i] > l[j] and dp[i] < dp[j] + 1:
                dp[i] = dp[j] + 1
                result = max(result, dp[i])
    return result
```

Explanation: This algorithm uses dynamic programming approach to solve the problem to find longest ordered subsequence.

Test cases :

1. arr = [1, 7, 3, 5, 9, 4, 8]
2. arr = [1, 1, 1, 1, 1, 1, 2, 1, 1]

Outout :



```
C:\Users\Pratik\Desktop\pycharm codes>flake8 --max-complexity 10 assignment_5.py

C:\Users\Pratik\Desktop\pycharm codes>assignment_5.py
[1, 7, 3, 5, 9, 4, 8]
ans :  4
passed
[1, 1, 1, 1, 1, 1, 2, 1, 1]
ans :  2
passed

C:\Users\Pratik\Desktop\pycharm codes>
```

Problem 2

```
class Graph:
    def __init__(self, rows, coloumns, g):
        self.R = rows
        self.C = coloumns
        self.graph = g

    def checkAdj(self, i, j, isVisited):
        return (0 <= i < self.R and 0 <= j < self.C and
                not isVisited[i][j] and self.graph[i][j] == '#')

    def DFS(self, i, j, isVisited):
        isVisited[i][j] = True
        if self.checkAdj(i-1, j-1, isVisited):
            self.DFS(i-1, j-1, isVisited)
        if self.checkAdj(i-1, j, isVisited):
            self.DFS(i-1, j, isVisited)
        if self.checkAdj(i-1, j+1, isVisited):
            self.DFS(i-1, j+1, isVisited)
        if self.checkAdj(i, j-1, isVisited):
            self.DFS(i, j-1, isVisited)
        if self.checkAdj(i, j+1, isVisited):
            self.DFS(i, j+1, isVisited)
        if self.checkAdj(i+1, j-1, isVisited):
            self.DFS(i+1, j-1, isVisited)
        if self.checkAdj(i+1, j, isVisited):
            self.DFS(i+1, j, isVisited)
        if self.checkAdj(i+1, j+1, isVisited):
            self.DFS(i+1, j+1, isVisited)

    def countPonds(self):
        isVisited = [[False for j in range(self.C)] for i in range(self.R)]
        pondNum = 0
        for i in range(self.R):
            for j in range(self.C):
                if not isVisited[i][j] and self.graph[i][j] == '#':
                    self.DFS(i, j, isVisited)
                    pondNum += 1
        return pondNum


def count_ponds(G):
    x = len(G)
    y = len(G[0])
    g = Graph(x, y, G)
    res = g.countPonds()
    return res
```

Explaination :

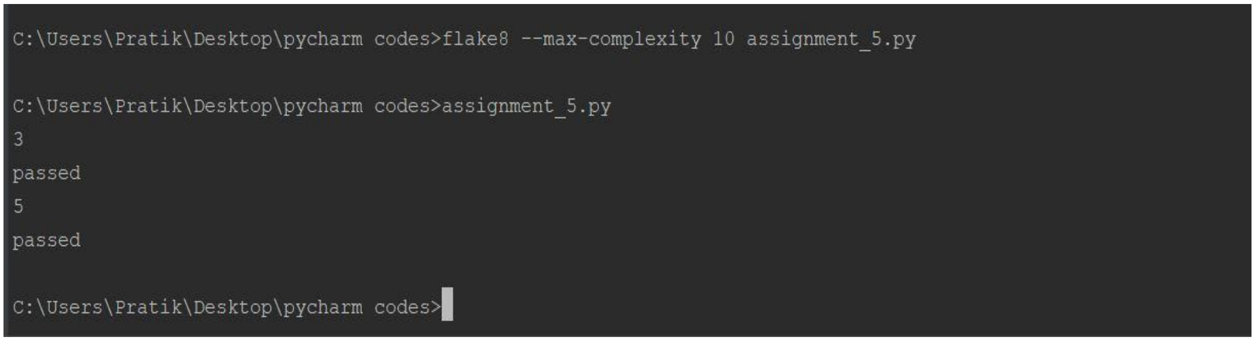
This problem uses connected componets logic and can be solved using DFS. We check for adjacent vertices which are not visited and check if its water then add it to path and continue. Maintaining count gives number of ponds

Test Condition:

```
1.      arr = ["#--------##-",
            "-###-----###",
            "----##---##-",
            "---------##-",
            "---------#--",
            "--#------#--",
            "-#-#-----##-",
            "#-#-#-----#-",
            "-#-#------#-",
            "--#-------#-"]
```

```
1.    arr = ["#--------##-",
            "-##------###",
2.          "---###------",
3.          "---#-----##-",
4.          "--##--------",
5.          "--#------#--",
6.          "-#-#-----##-",
7.          "#-#-#-----#-",
8.          "-#-#--------",
9.          "--#-------#-"]
```

Output screenshot



```
C:\Users\Pratik\Desktop\pycharm codes>flake8 --max-complexity 10 assignment_5.py

C:\Users\Pratik\Desktop\pycharm codes>assignment_5.py
3
passed
5
passed

C:\Users\Pratik\Desktop\pycharm codes>
```

Problem 3

```
def getMaxVal(T, time, val, n):
    Val = [[0 for x in range(T + 1)] for x in range(n + 1)]
    for i in range(n + 1):
        for t in range(T + 1):
            if i == 0 or t == 0:
                Val[i][t] = 0
            elif time[i - 1] >= t:
                Val[i][t] = max(val[i - 1] + Val[i - 1][t - 1], Val[i - 1][t])
            else:
                Val[i][t] = Val[i][t - 1]

    return Val[n][T]



def supermarket(items):
    n = len(items)
    items.sort(key=lambda tup: tup[1])
    val = []
    time = []
    maxTime = 0
    for item in items:
        val.append(item[0])
        time.append((item[1]))
        maxTime = max(maxTime, item[1])
    return getMaxVal(maxTime, time, val, n)
```

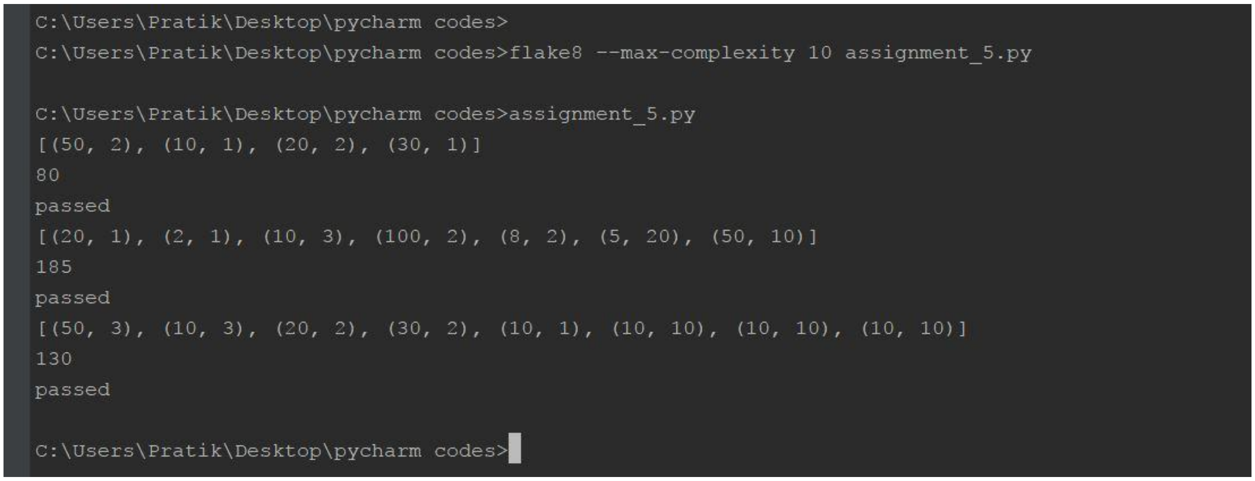
Explaination :

In this problem we want max profit with following the deadline of sale data. We can do that by sorting the given data based on deadline and then using the dynamic programming approach like Knapsack, except here we maintain the condition of check whether deadline is feasible instead of checking if weight is feasible (i.e time[i - 1] >= t). In condition to check if current sale is considered or not this condition can be used and max profit sale is chosen by comparing previous value. There are more approaches to solve this problem, but to use dynamic programming approach here this approach is chosen.

Test conditions:

1. arr = [(50, 2), (10, 1), (20, 2), (30, 1)]
2. arr = [(20, 1), (2, 1), (10, 3), (100, 2), (8, 2), (5, 20), (50, 10)]
3. arr = [(50, 3), (10, 3), (20, 2), (30, 2), (10, 1), (10, 10), (10, 10), (10, 10)]

Output screenshot:



```
C:\Users\Pratik\Desktop\pycharm codes>
C:\Users\Pratik\Desktop\pycharm codes>flake8 --max-complexity 10 assignment_5.py

C:\Users\Pratik\Desktop\pycharm codes>assignment_5.py
[(50, 2), (10, 1), (20, 2), (30, 1)]
80
passed
[(20, 1), (2, 1), (10, 3), (100, 2), (8, 2), (5, 20), (50, 10)]
185
passed
[(50, 3), (10, 3), (20, 2), (30, 2), (10, 1), (10, 10), (10, 10), (10, 10)]
130
passed

C:\Users\Pratik\Desktop\pycharm codes>
```

Problem 4:

```
def subset_sum_solver(S, n):
    totalSum = sum(S)
    if n > totalSum or totalSum == 0:
        return False
    temp = []
    for i in range(len(S)):
        temp.append(S[i])
    temp.append(totalSum + n)
    temp.append((2 * totalSum) - n)
    if partition_set_solver(temp):
        return True
    else:
        return False
```

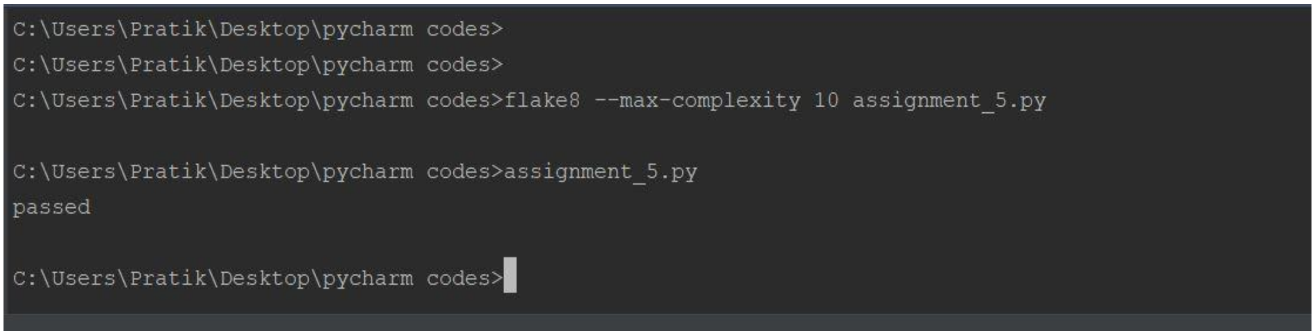
Explaination :

Given solution shows the use of reducibility in problem to check for subset sum

Test Case:

[1, 3, 2, 4], 8

Output:



```
C:\Users\Pratik\Desktop\pycharm codes>
C:\Users\Pratik\Desktop\pycharm codes>
C:\Users\Pratik\Desktop\pycharm codes>flake8 --max-complexity 10 assignment_5.py

C:\Users\Pratik\Desktop\pycharm codes>assignment_5.py
passed

C:\Users\Pratik\Desktop\pycharm codes>
```