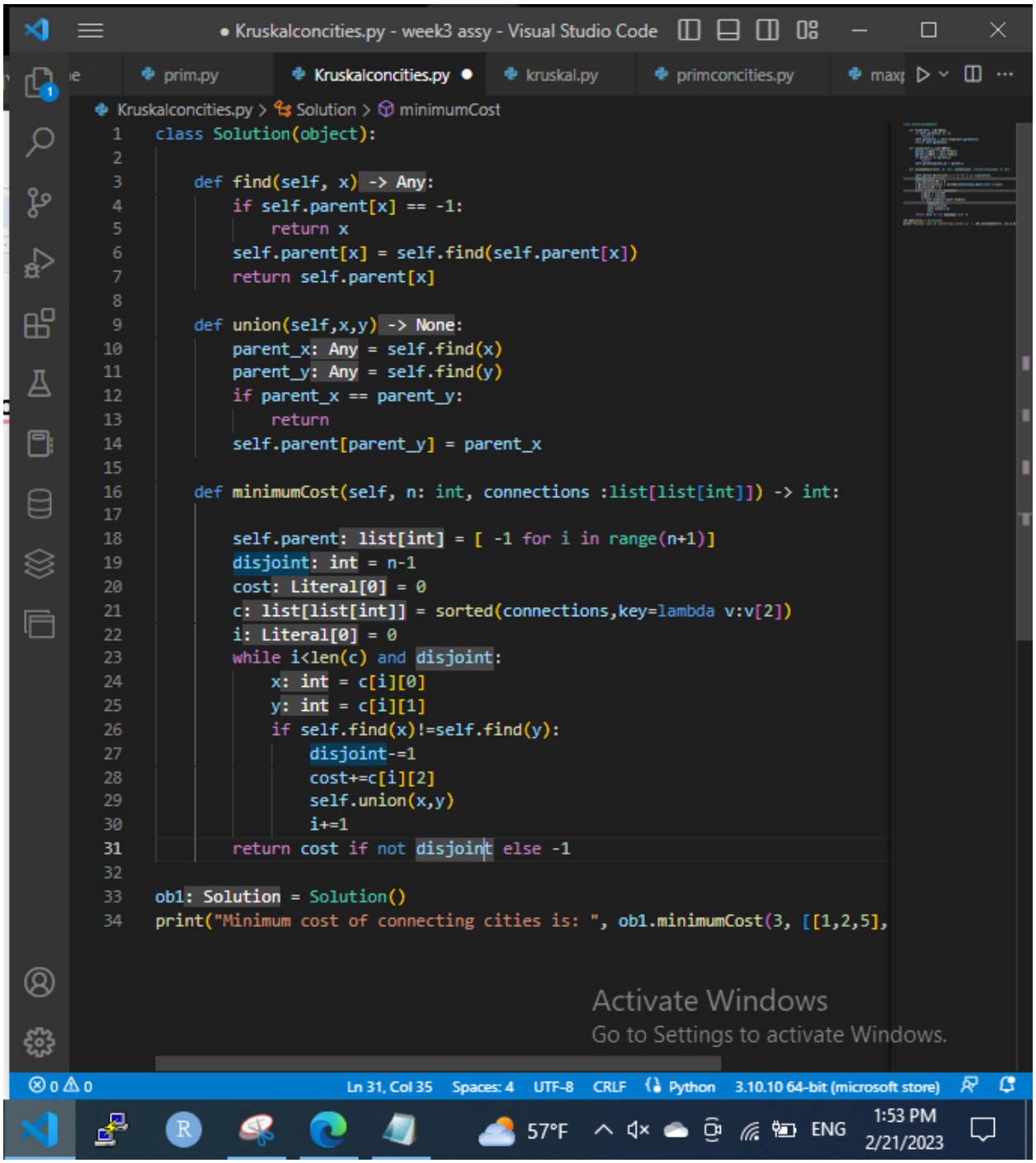
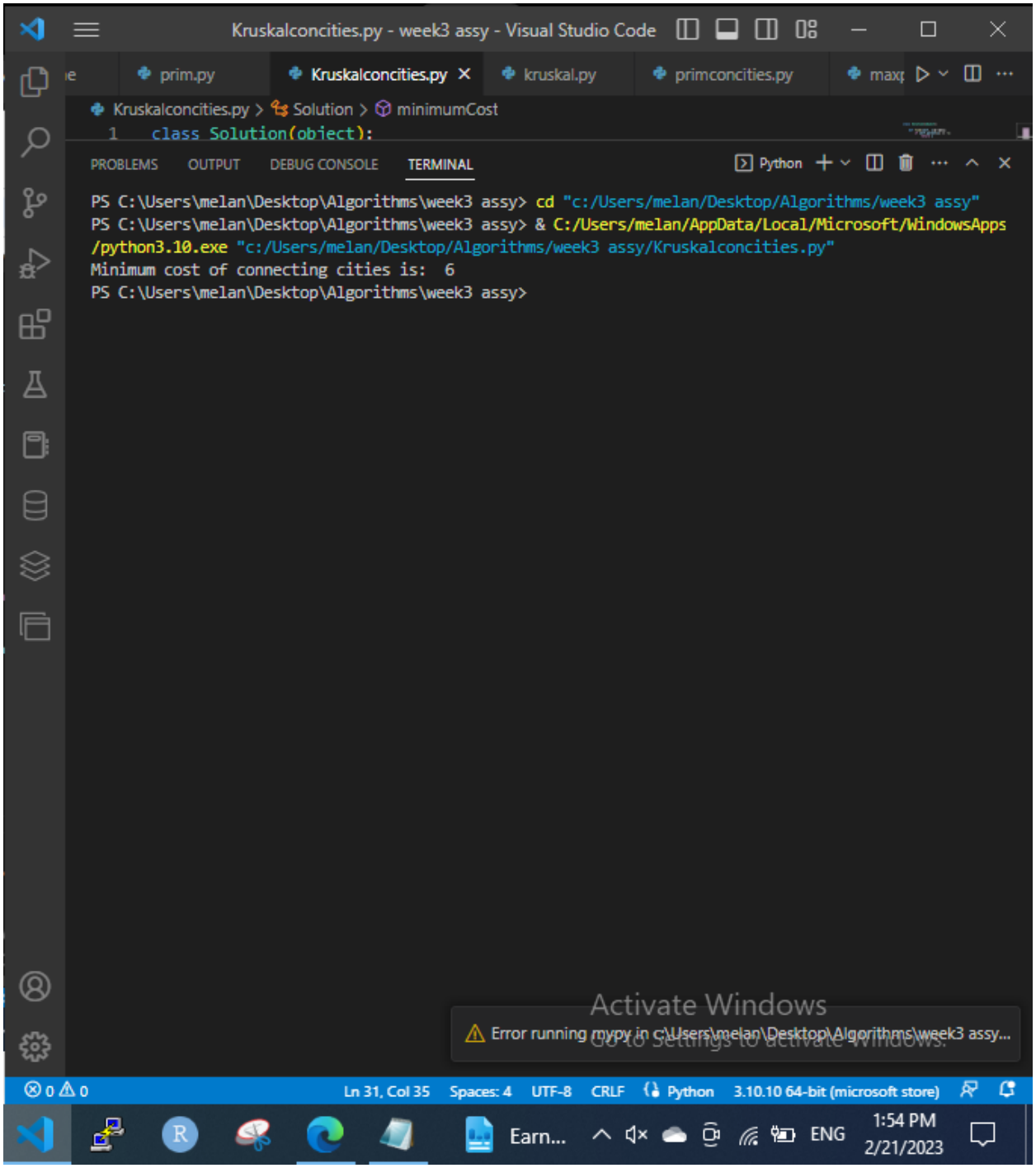
Week 5 Homework 2: Greedy Algorithm : Minimum Spanning Tree (Kruskal) - LC

Step 1

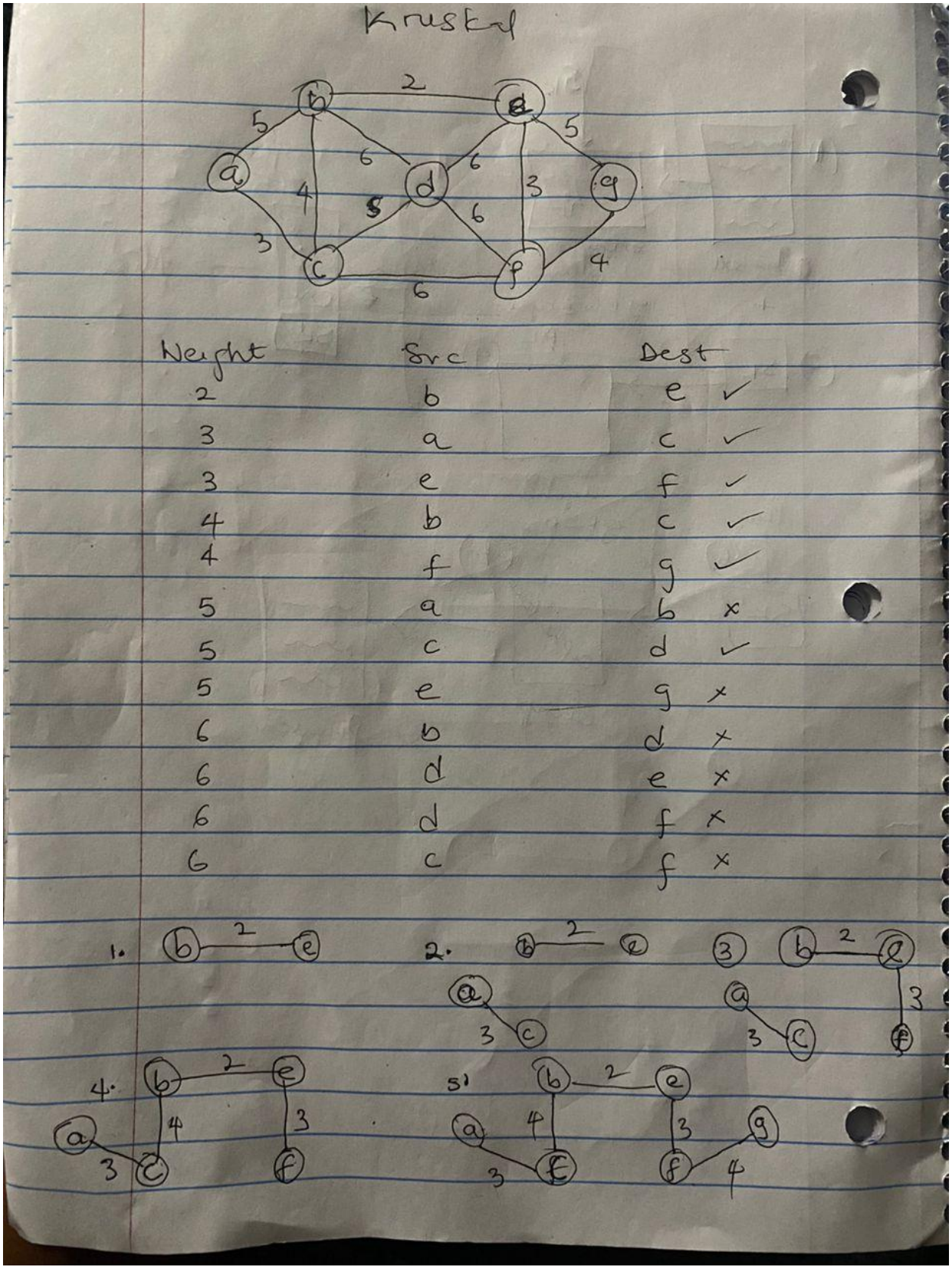



Kruskalconcities.py - week3 assy - Visual Studio Code
prim.py
Kruskalconcities.py
kruskal.py
primconcities.py
Kruskalconcities.py > Solution > minimumCost
1 class Solution(object):
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
Python
PS C:\Users\melan\Desktop\Algorithms\week3 assy> cd "c:/Users/melan/Desktop/Algorithms/week3 assy"
PS C:\Users\melan\Desktop\Algorithms\week3 assy> & C:/Users/melan/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/melan/Desktop/Algorithms/week3 assy/Kruskalconcities.py"
Minimum cost of connecting cities is: 6
PS C:\Users\melan\Desktop\Algorithms\week3 assy>
Activate Windows
Error running mypy in c:\Users\melan\Desktop\Algorithms\week3 assy...
Ln 31, Col 35
Spaces: 4
UTF-8
CRLF
Python
3.10.10 64-bit (microsoft store)
Earn...
ENG
1:54 PM
2/21/2023

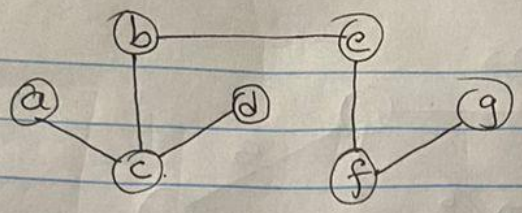
Code

```
class Solution(object):

    def find(self, x):
        if self.parent[x] == -1:
            return x
        self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self,x,y):
        parent_x = self.find(x)
        parent_y = self.find(y)
        if parent_x == parent_y:
            return
        self.parent[parent_y] = parent_x

    def minimumCost(self, n: int, connections :list[list[int]]) -> int:

        self.parent = [ -1 for i in range(n+1)]
        disjoint = n-1
        cost = 0
        c = sorted(connections,key=lambda v:v[2])
        i = 0
        while i<len(c) and disjoint:
            x = c[i][0]
            y = c[i][1]
            if self.find(x)!=self.find(y):
                disjoint-=1
                cost+=c[i][2]
                self.union(x,y)
                i+=1
        return cost if not disjoint else -1

ob1 = Solution()

print("Minimum cost of connecting cities is: ", ob1.minimumCost(3, [[1,2,5], [1,3,6], [2,3,1]]))
```

STEP 2


Kruskal
Weight
Src
Dest
2 b e ✓
3 a c ✓
3 e f ✓
4 b c ✓
4 f g ✓
5 a b ×
5 c d ✓
5 e g ×
6 b d ×
6 d e ×
6 d f ×
6 c f ×
1.
2.
4.
5.

6.

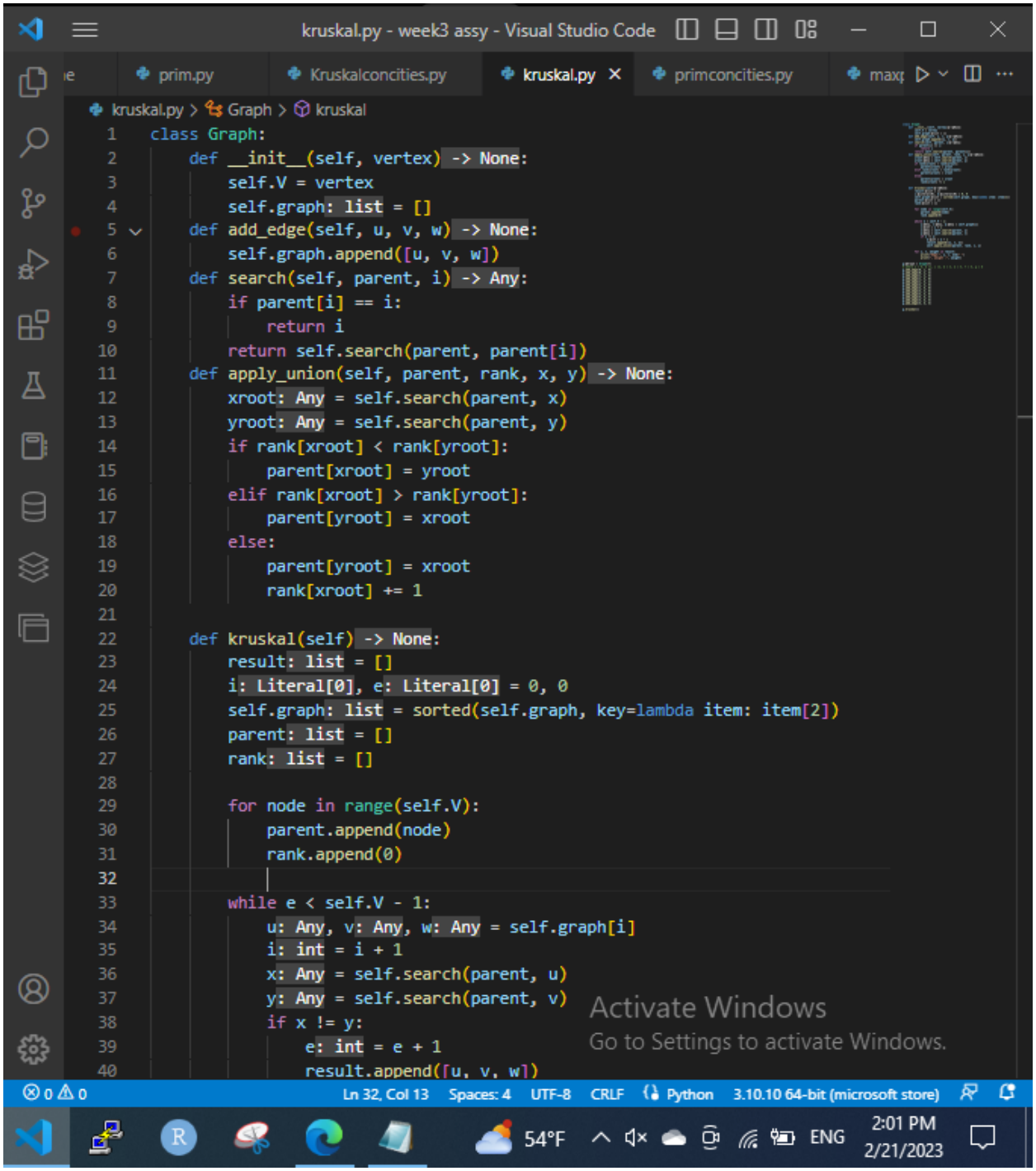


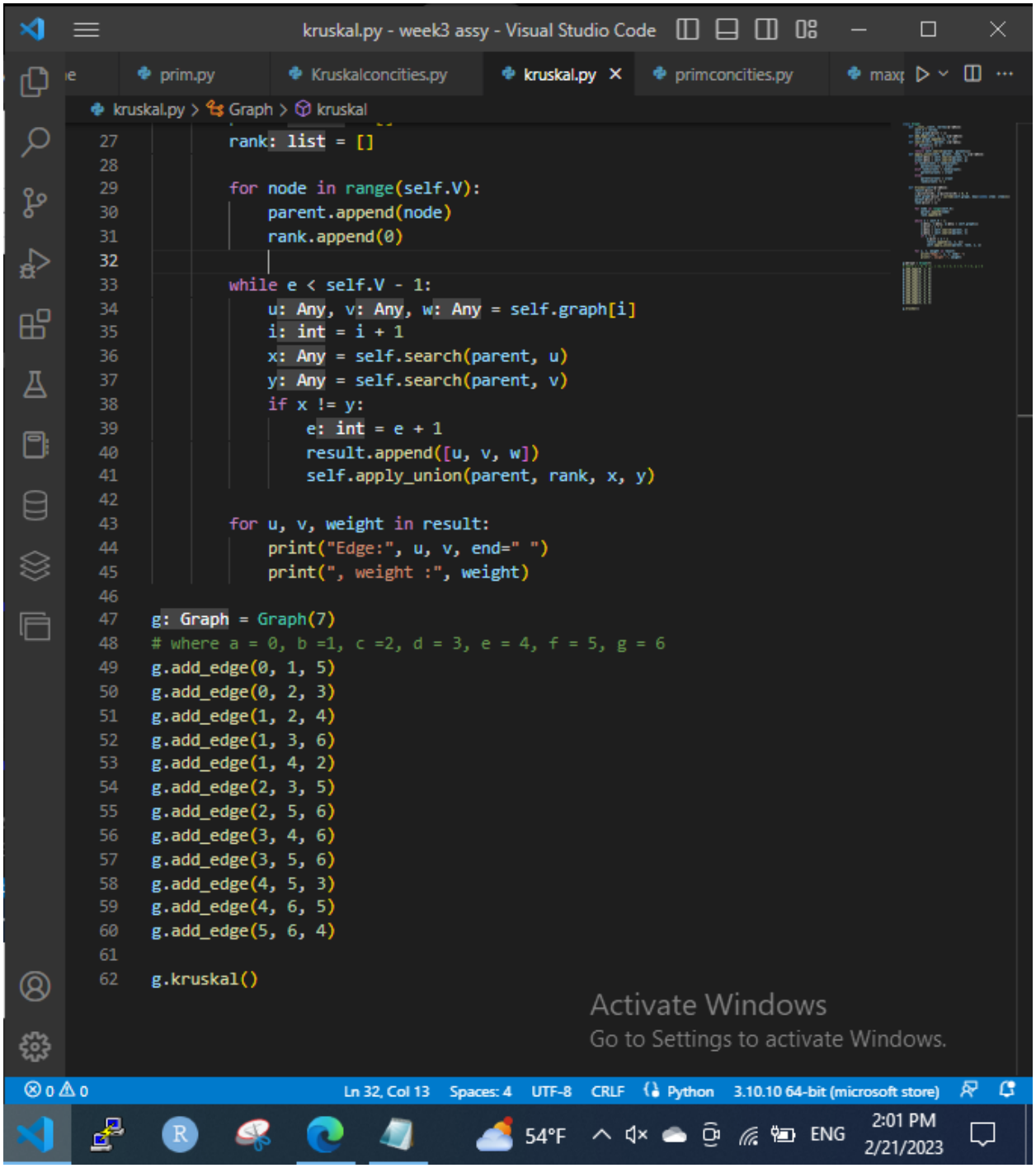
Every other step results in a loop being formed so we skip them.

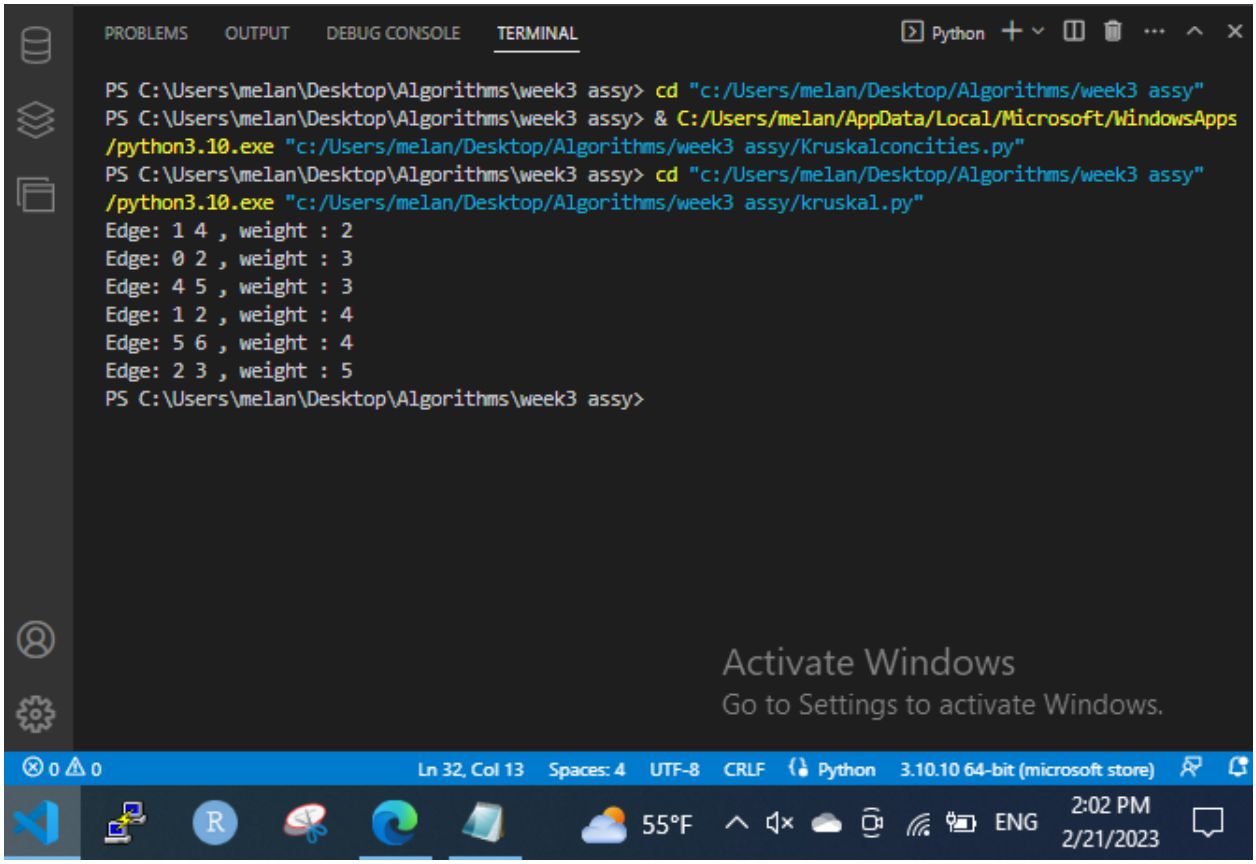
Step 6 is the final answer

Time complexity = $O(E \log V)$ because in Kruskal's algorithm, most time consuming operation is sorting because of the total complexity of the disjoint set operations.

STEP 3


kruskal.py - week3 assy - Visual Studio Code
prim.py
Kruskalconcities.py
kruskal.py
primconcities.py
kruskal.py > Graph > kruskal
class Graph:
def __init__(self, vertex) -> None:
self.V = vertex
self.graph: list = []
def add_edge(self, u, v, w) -> None:
self.graph.append([u, v, w])
def search(self, parent, i) -> Any:
if parent[i] == i:
return i
return self.search(parent, parent[i])
def apply_union(self, parent, rank, x, y) -> None:
xroot: Any = self.search(parent, x)
yroot: Any = self.search(parent, y)
if rank[xroot] < rank[yroot]:
parent[xroot] = yroot
elif rank[xroot] > rank[yroot]:
parent[yroot] = xroot
else:
parent[yroot] = xroot
rank[xroot] += 1
def kruskal(self) -> None:
result: list = []
i: Literal[0], e: Literal[0] = 0, 0
self.graph: list = sorted(self.graph, key=lambda item: item[2])
parent: list = []
rank: list = []
for node in range(self.V):
parent.append(node)
rank.append(0)
while e < self.V - 1:
u: Any, v: Any, w: Any = self.graph[i]
i: int = i + 1
x: Any = self.search(parent, u)
y: Any = self.search(parent, v)
if x != y:
e: int = e + 1
result.append([u, v, w])
Activate Windows
Go to Settings to activate Windows.
Ln 32, Col 13
Spaces: 4
UTF-8
CRLF
Python
3.10.10 64-bit (microsoft store)
54°F
ENG
2:01 PM
2/21/2023


kruskal.py - week3 assy - Visual Studio Code
prim.py
Kruskalconcities.py
kruskal.py
primconcities.py
kruskal.py > Graph > kruskal
rank: list = []
for node in range(self.V):
parent.append(node)
rank.append(0)
while e < self.V - 1:
u: Any, v: Any, w: Any = self.graph[i]
i: int = i + 1
x: Any = self.search(parent, u)
y: Any = self.search(parent, v)
if x != y:
e: int = e + 1
result.append([u, v, w])
self.apply_union(parent, rank, x, y)
for u, v, weight in result:
print("Edge:", u, v, end=" ")
print(", weight :", weight)
g: Graph = Graph(7)
# where a = 0, b =1, c =2, d = 3, e = 4, f = 5, g = 6
g.add_edge(0, 1, 5)
g.add_edge(0, 2, 3)
g.add_edge(1, 2, 4)
g.add_edge(1, 3, 6)
g.add_edge(1, 4, 2)
g.add_edge(2, 3, 5)
g.add_edge(2, 5, 6)
g.add_edge(3, 4, 6)
g.add_edge(3, 5, 6)
g.add_edge(4, 5, 3)
g.add_edge(4, 6, 5)
g.add_edge(5, 6, 4)
g.kruskal()
Activate Windows
Go to Settings to activate Windows.
Ln 32, Col 13
Spaces: 4
UTF-8
CRLF
Python
3.10.10 64-bit (microsoft store)
54°F
ENG
2:01 PM
2/21/2023



CODE

```
class Graph:
    def __init__(self, vertex):
        self.V = vertex
        self.graph = []
    def add_edge(self, u, v, w):
        self.graph.append([u, v, w])
    def search(self, parent, i):
        if parent[i] == i:
            return i
        return self.search(parent, parent[i])
    def apply_union(self, parent, rank, x, y):
        xroot = self.search(parent, x)
        yroot = self.search(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
```

```
            rank[xroot] += 1

    def kruskal(self):
        result = []
        i, e = 0, 0
        self.graph = sorted(self.graph, key=lambda item: item[2])
        parent = []
        rank = []

        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.search(parent, u)
            y = self.search(parent, v)
            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.apply_union(parent, rank, x, y)

        for u, v, weight in result:
            print("Edge:", u, v, end=" ")
            print(", weight :", weight)

g = Graph(7)
# where a = 0, b =1, c =2, d = 3, e = 4, f = 5, g = 6
g.add_edge(0, 1, 5)
g.add_edge(0, 2, 3)
g.add_edge(1, 2, 4)
g.add_edge(1, 3, 6)
g.add_edge(1, 4, 2)
g.add_edge(2, 3, 5)
g.add_edge(2, 5, 6)
g.add_edge(3, 4, 6)
g.add_edge(3, 5, 6)
g.add_edge(4, 5, 3)
g.add_edge(4, 6, 5)
g.add_edge(5, 6, 4)

g.kruskal()
```