Assignment – 12.06.24

1. Convert the Temperature.

Program:

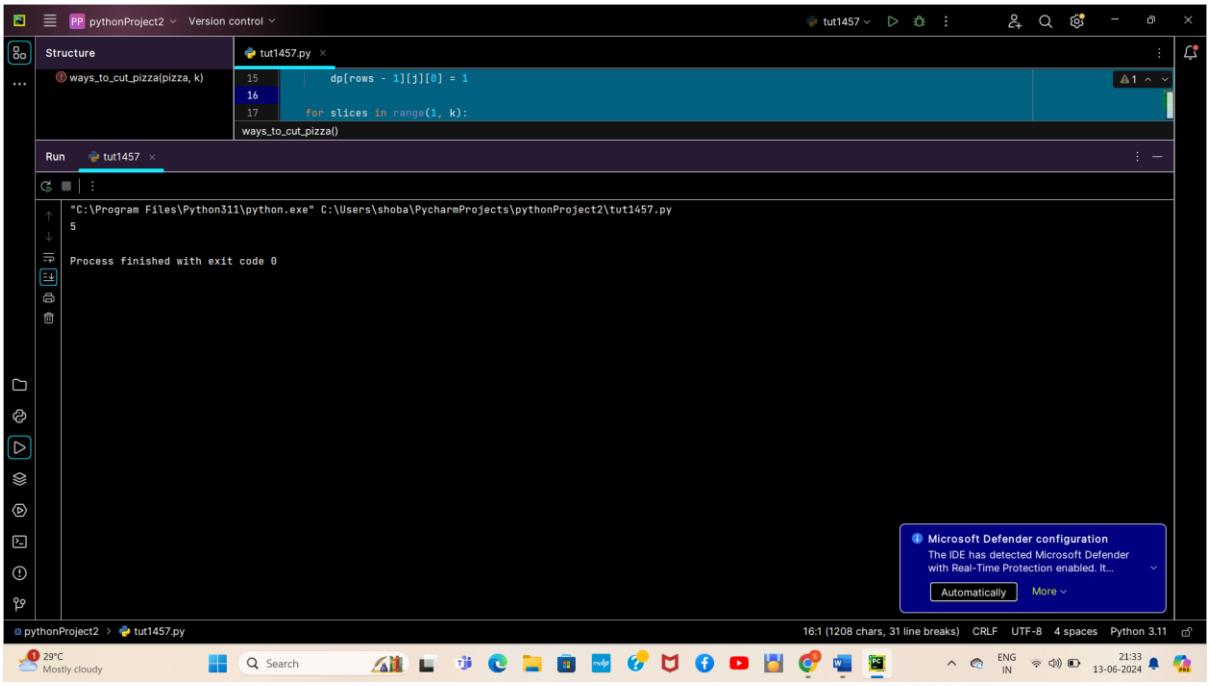
```
def celsius_to_fahrenheit(celsius):
    return (celsius * 9/5) + 32

def
fahrenheit_to_celsius(fahrenheit):
    return (fahrenheit - 32) * 5/9

def celsius_to_kelvin(celsius):
    return celsius + 273.15

def kelvin_to_celsius(kelvin):
    return kelvin - 273.15

def fahrenheit_to_kelvin(fahrenheit):
```

```
    celsius =
fahrenheit_to_celsius(fahrenheit)
    return celsius_to_kelvin(celsius)

def kelvin_to_fahrenheit(kelvin):
    celsius = kelvin_to_celsius(kelvin)
    return
celsius_to_fahrenheit(celsius)

def convert_temperature(value,
from_scale, to_scale):
    if from_scale == "Celsius":
        if to_scale == "Fahrenheit":
            return
celsius_to_fahrenheit(value)
        elif to_scale == "Kelvin":
            return celsius_to_kelvin(value)
        else:
            return value
    elif from_scale == "Fahrenheit":
```

```
        if to_scale == "Celsius":
            return
fahrenheit_to_celsius(value)
        elif to_scale == "Kelvin":
            return
fahrenheit_to_kelvin(value)
        else:
            return value
    elif from_scale == "Kelvin":
        if to_scale == "Celsius":
            return kelvin_to_celsius(value)
        elif to_scale == "Fahrenheit":
            return
kelvin_to_fahrenheit(value)
        else:
            return value
    else:
        raise ValueError("Invalid
temperature scale")
```

```
# Example usage
value = 100
from_scale = "Celsius"
to_scale = "Fahrenheit"
converted_value =
convert_temperature(value,
from_scale, to_scale)
print(f"{value} degrees {from_scale}
is {converted_value} degrees
{to_scale}.")
```

Output:



2. Number of Subarrays With LCM Equal to K

Program:

```
import math
from typing import List

def lcm(a: int, b: int) -> int:
    return abs(a * b) // math.gcd(a, b)

def subarrays_with_lcm_equal_to_k(arr: List[int], K: int) -> int:
    n = len(arr)
    count = 0

    for i in range(n):
        current_lcm = arr[i]
        for j in range(i, n):
            current_lcm = lcm(current_lcm, arr[j])
            if current_lcm == K:
                count += 1
            if current_lcm > K:  # If the current LCM exceeds K, no need to proceed further
                break

    return count
```

 # Example usage

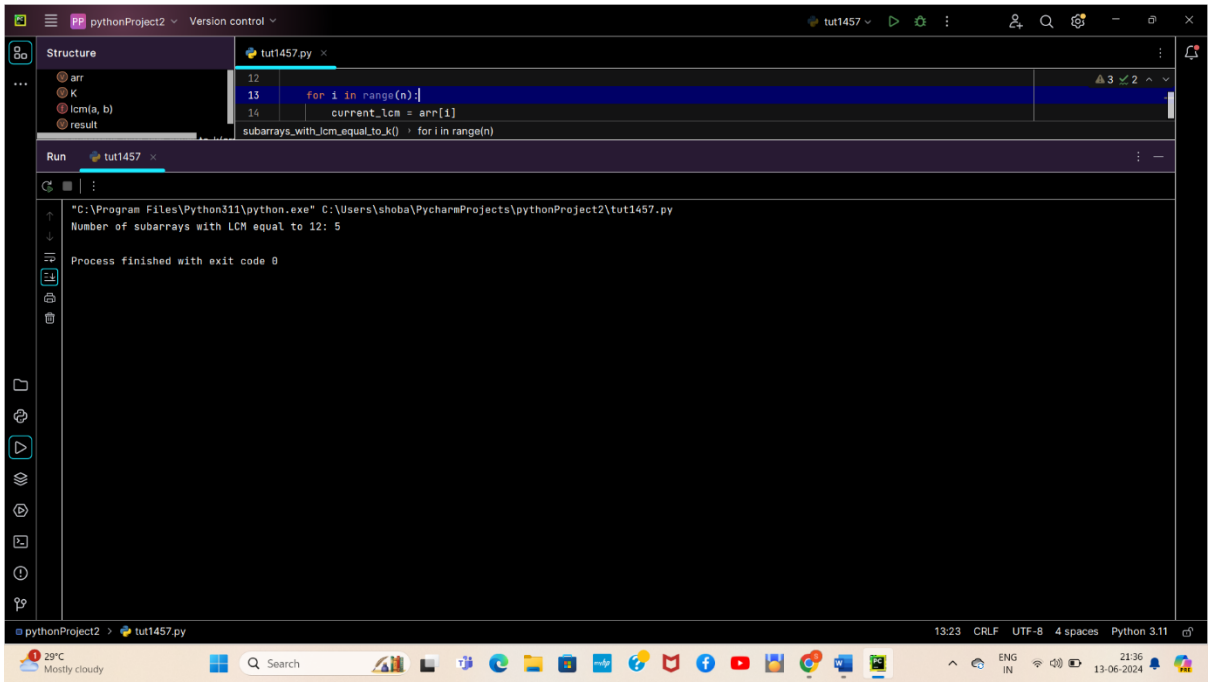
arr = [2, 3, 4, 6]

K = 12

result = subarrays_with_lcm_equal_to_k(arr, K)

print(f"Number of subarrays with LCM equal to {K}: {result}")

output:



3. Minimum Number of Operations to Sort a Binary Tree by Level.

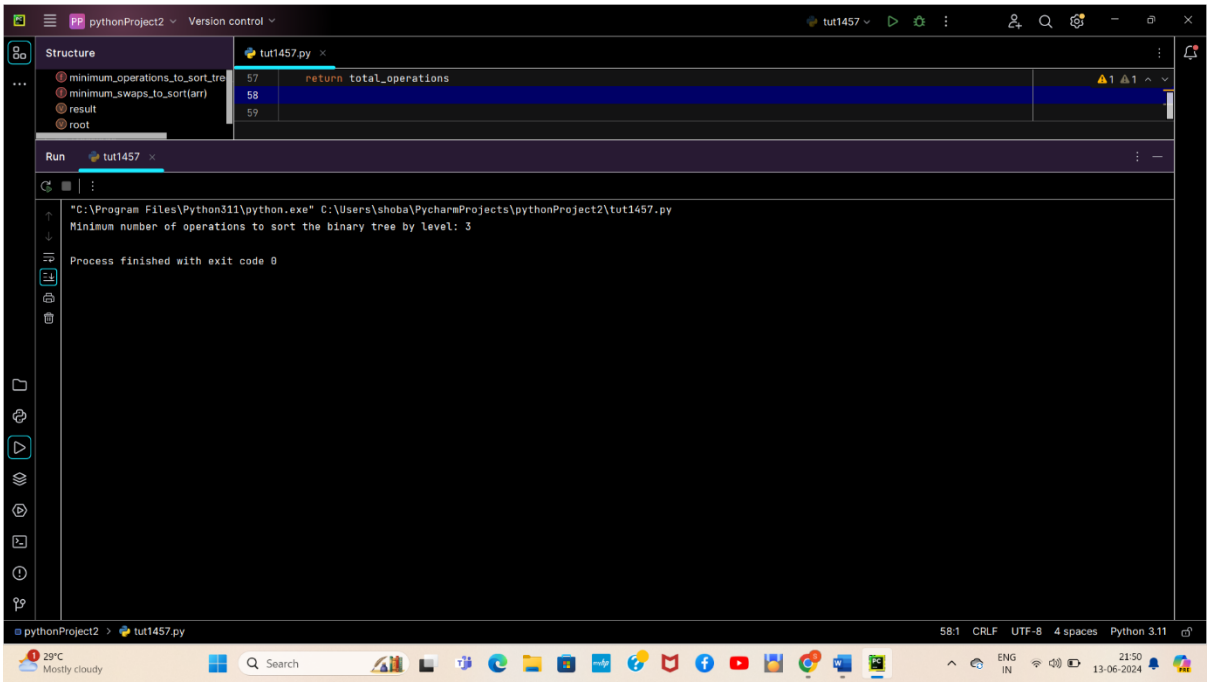
Program:

```
from collections import deque, defaultdict


class TreeNode:
```

```
    def __init__(self, val=0, left=None,
right=None):
        self.val = val
        self.left = left
        self.right = right

def minimum_swaps_to_sort(arr):
    n = len(arr)
    sorted_arr = sorted(arr)
    index_map = {value: index for index, value in
enumerate(arr)}
    swaps = 0
    visited = [False] * n

    for i in range(n):
        if visited[i] or arr[i] == sorted_arr[i]:
            continue

        cycle_size = 0
        x = i
        while not visited[x]:
            visited[x] = True
            x = index_map[sorted_arr[x]]
            cycle_size += 1

        if cycle_size > 0:
```

```
            swaps += (cycle_size - 1)

    return swaps

def
minimum_operations_to_sort_tree_by_level(ro
ot):
    if not root:
        return 0

    queue = deque([root])
    total_operations = 0

    while queue:
        level_size = len(queue)
        current_level = []

        for _ in range(level_size):
            node = queue.popleft()
            current_level.append(node.val)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
```

```
        total_operations +=
minimum_swaps_to_sort(current_level)

    return total_operations

# Example usage
root = TreeNode(1)
root.left = TreeNode(3)
root.right = TreeNode(2)
root.left.left = TreeNode(7)
root.left.right = TreeNode(6)
root.right.left = TreeNode(5)
root.right.right = TreeNode(4)

result =
minimum_operations_to_sort_tree_by_level(ro
ot)
print(f"Minimum number of operations to sort
the binary tree by level: {result}")
```
Output:



4. Maximum Number of Non overlapping Palindrome Substring.

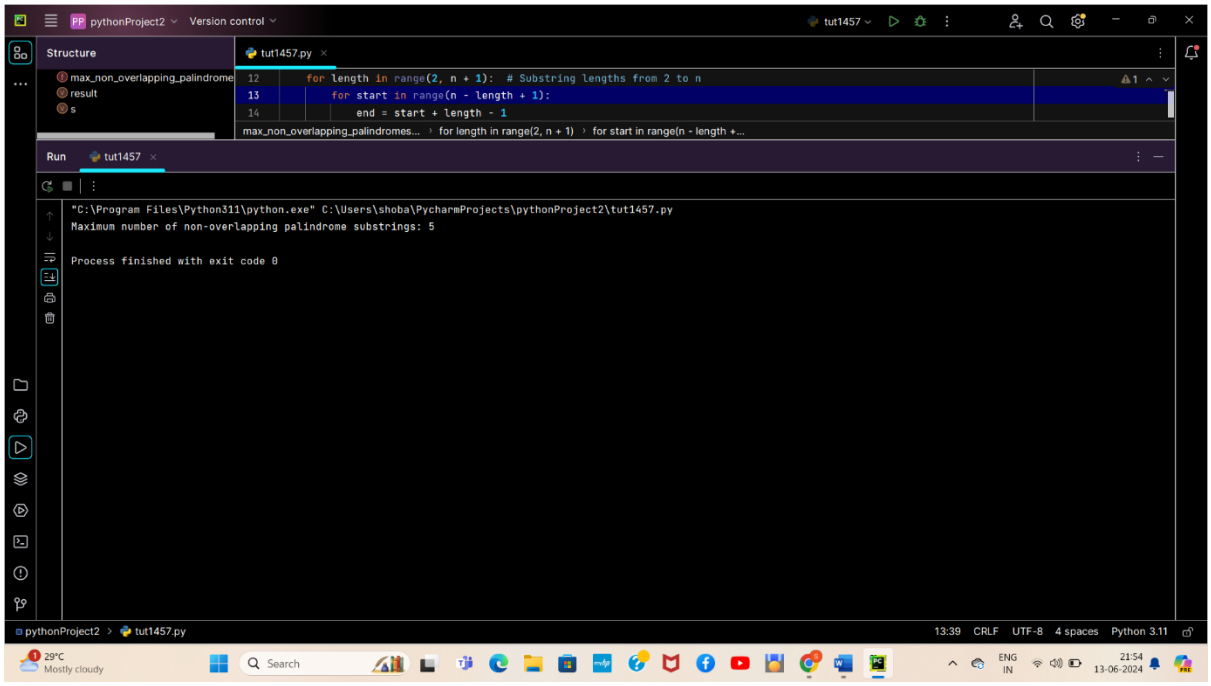
Program:

```
def max_non_overlapping_palindromes(s: str) -> int:
    n = len(s)
    if n == 0:
        return 0

    # Step 1: Create a 2D DP array to check for palindromes
    dp = [[False] * n for _ in range(n)]
```

```
    for i in range(n):
        dp[i][i] = True  # Every single character is a palindrome

    for length in range(2, n + 1):  # Substring lengths from 2 to n
        for start in range(n - length + 1):
            end = start + length - 1
            if length == 2:
                dp[start][end] = (s[start] == s[end])
            else:
                dp[start][end] = (s[start] == s[end] and dp[start + 1][end - 1])

    # Step 2: Use a 1D DP array to find the max number of non-overlapping palindromes
    max_palindromes = [0] * n

    for end in range(n):
        for start in range(end + 1):
```

```
            if dp[start][end]:
                if start == 0:
                    max_palindromes[end] =
max(max_palindromes[end], 1)
                else:
                    max_palindromes[end] =
max(max_palindromes[end], max_palindromes[start -
1] + 1)

    return max_palindromes[-1]

# Example usage
s = "ababa"
result = max_non_overlapping_palindromes(s)
print(f"Maximum number of non-overlapping
palindrome substrings: {result}")
```

Output:



5. Minimum Cost to Buy Apples.

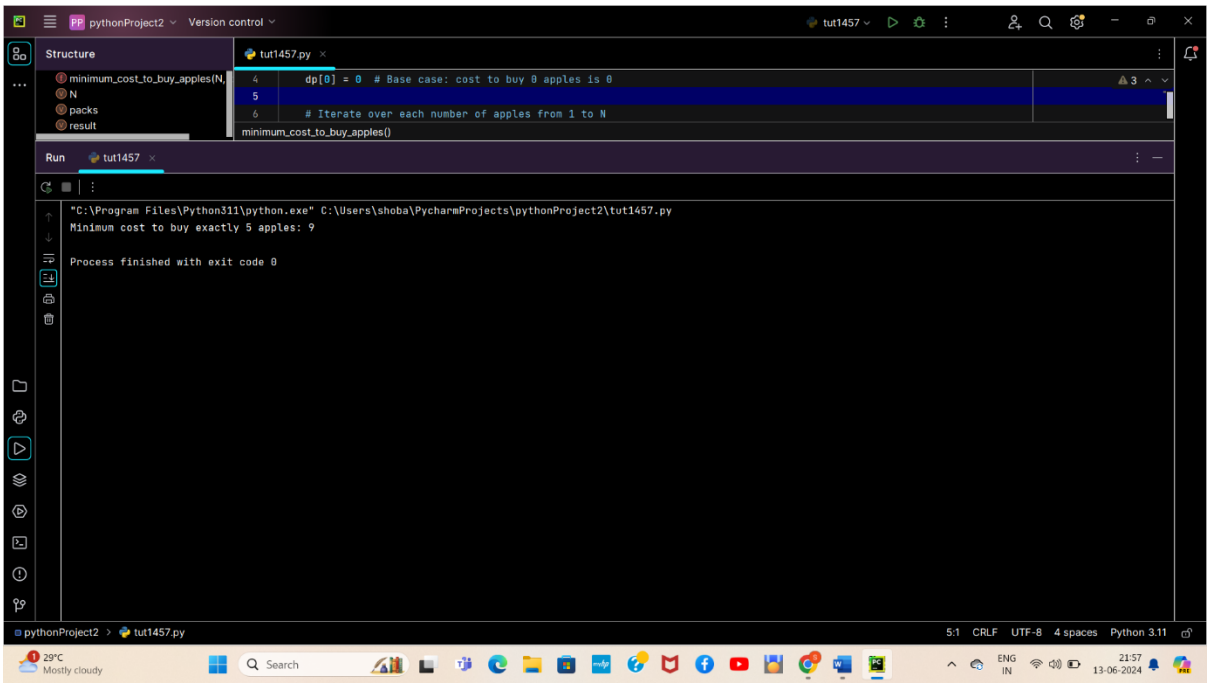
Program:

```
def
minimum_cost_to_buy_apples(N,
packs):
    # Initialize dp array with infinity
    dp = [float('inf')] * (N + 1)
    dp[0] = 0  # Base case: cost to buy
0 apples is 0
```

```
    # Iterate over each number of
apples from 1 to N
    for i in range(1, N + 1):
        for pack_size, cost in packs:
            if i >= pack_size:
                dp[i] = min(dp[i], dp[i -
pack_size] + cost)

    # If dp[N] is still infinity, it means
it's impossible to buy exactly N
apples
    return dp[N] if dp[N] != float('inf')
else -1

# Example usage
packs = [(1, 3), (3, 8), (4, 6)]  # Each
tuple is (pack_size, cost)
N = 5
```

```
result =
minimum_cost_to_buy_apples(N,
packs)
print(f"Minimum cost to buy exactly
{N} apples: {result}")
```

output:



6. Customers With Strictly Increasing Purchases.

Program:

```
import pandas as pd

# Load the data into a DataFrame
```

```
# Assume the data is in a CSV file named 'purchases.csv'
# The CSV file should have columns: CustomerId, PurchaseAmount, PurchaseDate

# Example CSV content:
# CustomerId,PurchaseAmount,PurchaseDate
# 1,100,2023-01-01
# 1,200,2023-02-01
# 1,300,2023-03-01
# 2,150,2023-01-01
# 2,140,2023-02-01
# 3,50,2023-01-01
# 3,60,2023-02-01
# 3,70,2023-03-01

df = pd.read_csv('purchases.csv', parse_dates=['PurchaseDate'])
```

```
# Sort the DataFrame by CustomerId and
PurchaseDate
df = df.sort_values(by=['CustomerId',
'PurchaseDate'])

# Function to check if a list is strictly
increasing
def is_strictly_increasing(lst):
    return all(x < y for x, y in zip(lst,
lst[1:]))

# Group by CustomerId and apply the
check
result = df.groupby('CustomerId').agg(
    Purchases=('PurchaseAmount', list)
).reset_index()

result['IsStrictlyIncreasing'] =
result['Purchases'].apply(is_strictly_incr
easing)
```

```
# Filter the customers with strictly increasing purchases
customers_with_increasing_purchases = result[result['IsStrictlyIncreasing']]['CustomerId']

print(f"Customers with strictly increasing purchases: {customers_with_increasing_purchases.tolist()}")
```

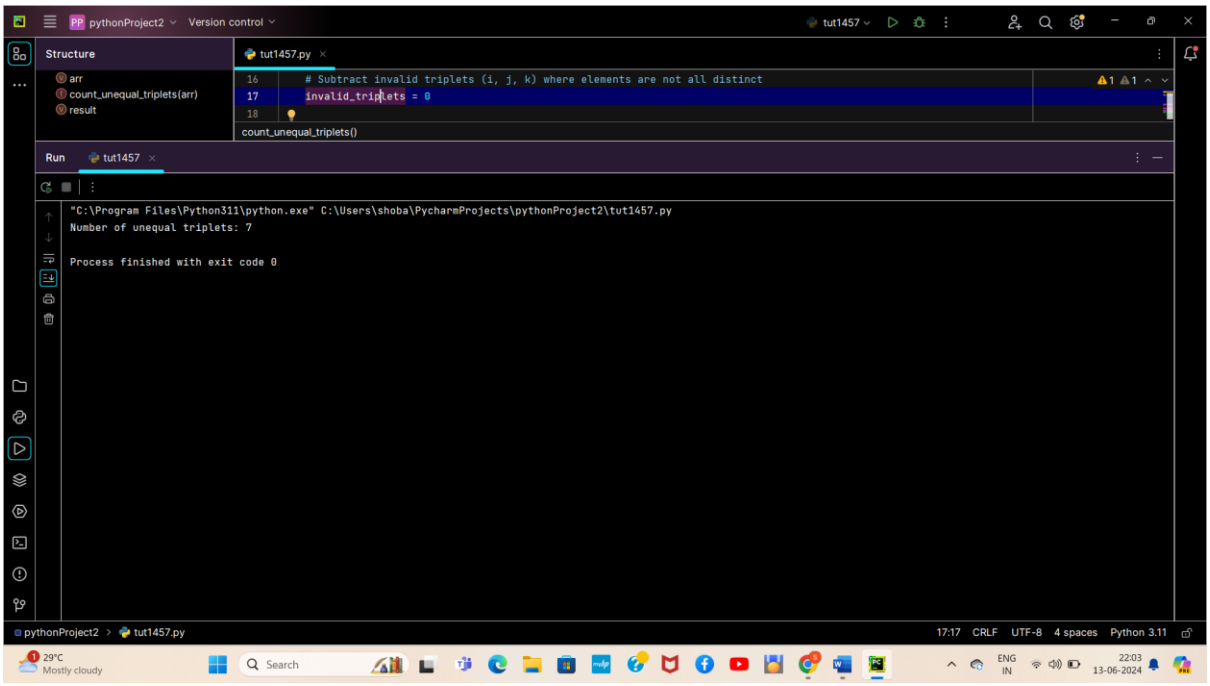
7. Number of Unequal Triplets in Array.

Program:

```
from collections import Counter
from itertools import combinations

def count_unequal_triplets(arr):
    n = len(arr)
    if n < 3:
        return 0

    # Total number of triplets (i, j, k) with i < j < k
    total_triplets = n * (n - 1) * (n - 2) // 6
```

```
    # Count frequency of each element
    freq = Counter(arr)

    # Subtract invalid triplets (i, j, k) where
elements are not all distinct
    invalid_triplets = 0

    for key, count in freq.items():
        if count >= 2:
            # Choose 2 out of count and any other
element
            invalid_triplets += count * (count - 1) // 2
* (n - count)
        if count >= 3:
            # Choose 3 out of count (all same
elements)
            invalid_triplets += count * (count - 1) *
(count - 2) // 6

    return total_triplets - invalid_triplets

# Example usage
arr = [1, 2, 2, 3, 4]
result = count_unequal_triplets(arr)
print(f"Number of unequal triplets: {result}")
```

Output:



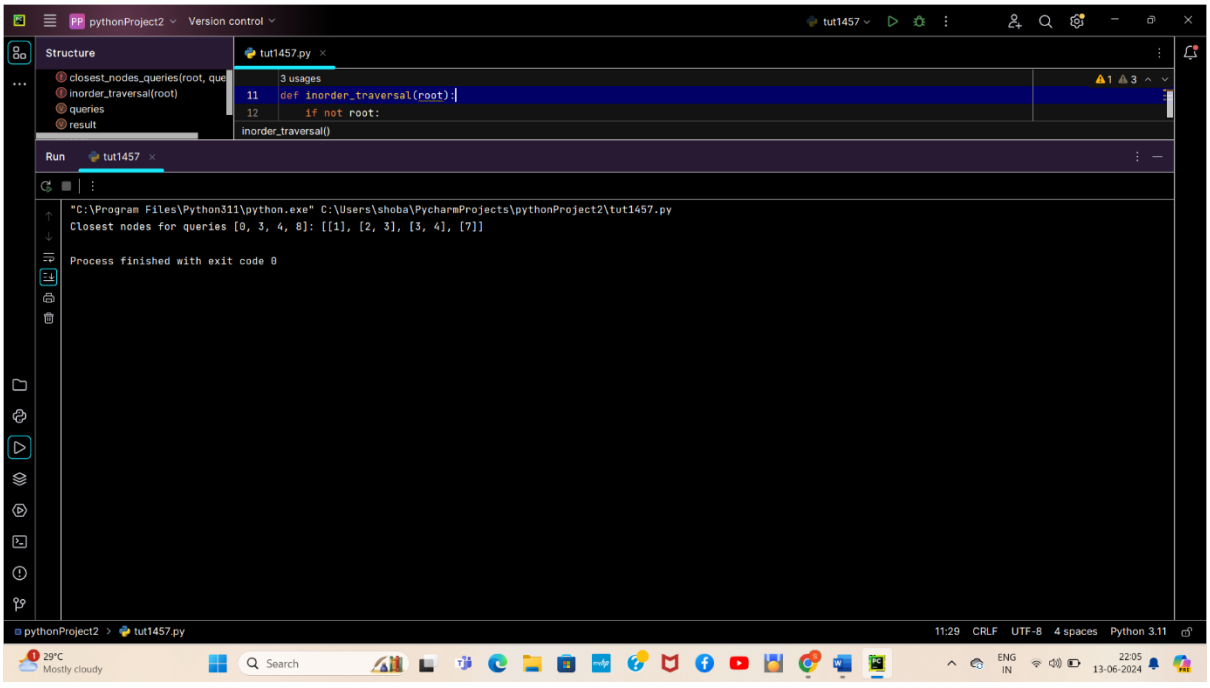
8. Closest Nodes Queries in a Binary Search Tree .

Program:

```
from bisect import bisect_left, bisect_right


class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right


def inorder_traversal(root):
    if not root:
        return []
    return inorder_traversal(root.left) + [root.val] +
inorder_traversal(root.right)


def closest_nodes_queries(root, queries):
    # Get the sorted list of node values using in-order traversal
    sorted_values = inorder_traversal(root)

    results = []
    for query in queries:
```

```
        # Find the position to insert the query value in the sorted list
        pos = bisect_left(sorted_values, query)

        # Determine the closest values
        closest_values = []
        if pos > 0:
            closest_values.append(sorted_values[pos - 1])
        if pos < len(sorted_values):
            closest_values.append(sorted_values[pos])

        results.append(closest_values)

    return results

root = TreeNode(4)
root.left = TreeNode(2, TreeNode(1), TreeNode(3))
root.right = TreeNode(6, TreeNode(5), TreeNode(7))

queries = [0, 3, 4, 8]
result = closest_nodes_queries(root, queries)
print(f"Closest nodes for queries {queries}: {result}")
```

Output:

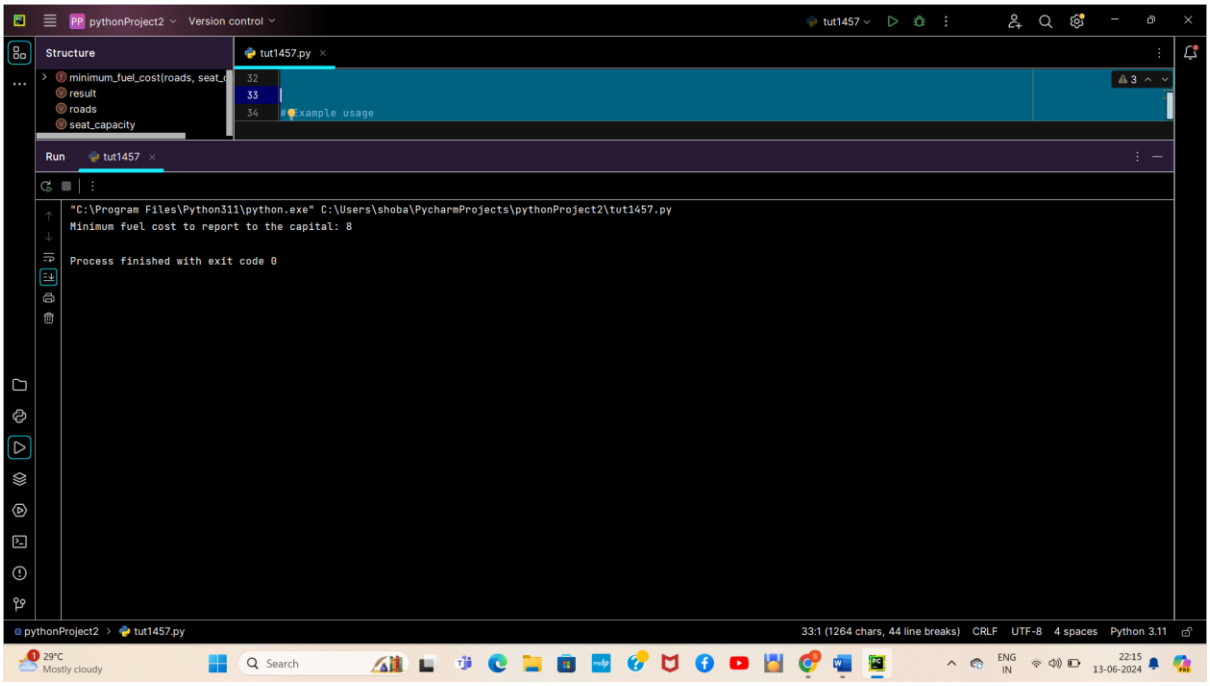


9. . Minimum Fuel Cost to Report to the Capital.

Program:

```python
def minimum_fuel_cost(roads, seat_capacity):
    from collections import defaultdict

    # Build the tree as an adjacency list
    tree = defaultdict(list)
    for u, v in roads:
        tree[u].append(v)
        tree[v].append(u)

    def dfs(node, parent):
        # Total representatives (initially 1 for the current node)
        total_representatives = 1
        total_fuel_cost = 0

        for neighbor in tree[node]:
            if neighbor != parent:
                reps, cost = dfs(neighbor, node)
                total_representatives += reps
                total_fuel_cost += cost

        # Calculate trips needed from this node to the parent
        if node != 0:  # Ignore the root node for the cost calculation to
itself
            trips = (total_representatives + seat_capacity - 1) //
seat_capacity
            total_fuel_cost += trips

        return total_representatives, total_fuel_cost

    # Start DFS from the root node (capital), assumed to be node 0
    _, total_fuel_cost = dfs(0, -1)

    return total_fuel_cost


# Example usage
roads = [
    (0, 1),
    (1, 2),
    (1, 3),
    (2, 4),
    (2, 5)
]
seat_capacity = 2
result = minimum_fuel_cost(roads, seat_capacity)
print(f"Minimum fuel cost to report to the capital: {result}")
```

Output:



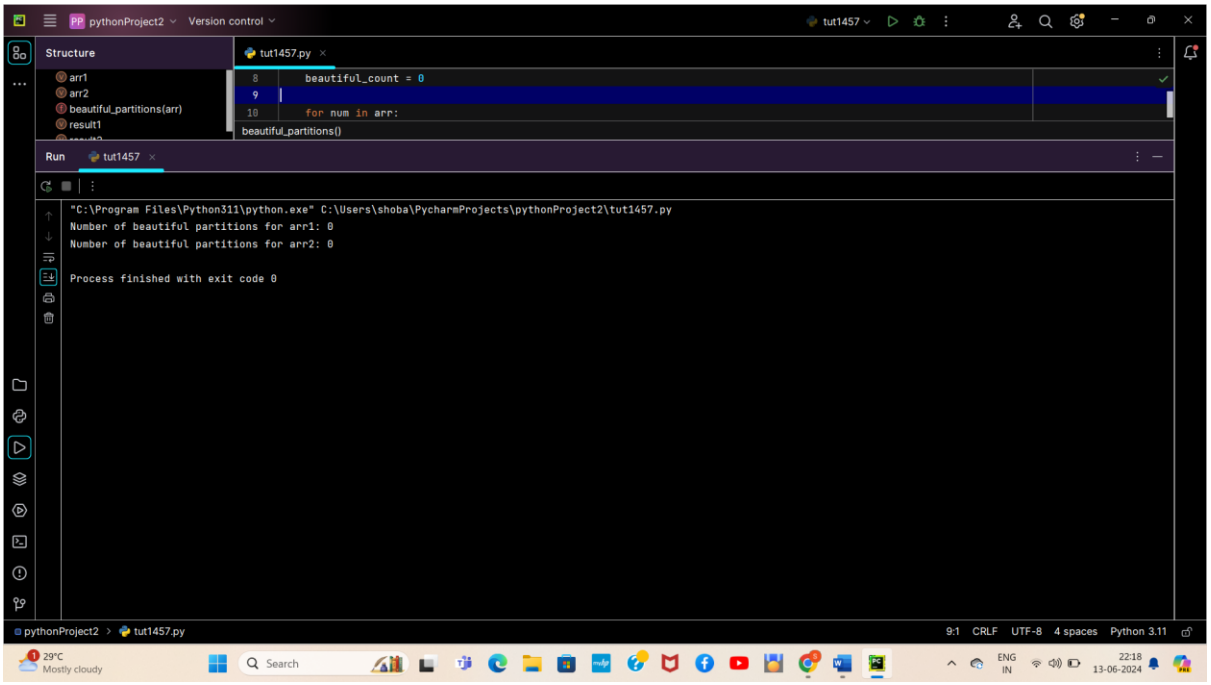
10. Number of Beautiful Partitions.

Program:

```
def beautiful_partitions(arr):
    total_sum = sum(arr)
    if total_sum % 2 != 0:
        return 0  # If total sum is odd,
no beautiful partitions possible

    target_sum = total_sum // 2
    prefix = 0
```

```
    beautiful_count = 0

    for num in arr:
        prefix += num
        if prefix == target_sum:
            beautiful_count += 1

    return beautiful_count

# Example usage
arr1 = [1, 2, 3, 4, 5]
arr2 = [2, 1, 2, 3, 4, 1]

result1 = beautiful_partitions(arr1)
result2 = beautiful_partitions(arr2)

print(f"Number of beautiful
partitions for arr1: {result1}")
print(f"Number of beautiful
partitions for arr2: {result2}")
```

Output:


pythonProject2
Version control
tut1457
Structure
arr1
arr2
beautiful_partitions(arr)
result1
tut1457.py
8 beautiful_count = 0
9
10 for num in arr:
beautiful_partitions()
Run
tut1457
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Number of beautiful partitions for arr1: 0
Number of beautiful partitions for arr2: 0
Process finished with exit code 0
pythonProject2 > tut1457.py
9:1 CRLF UTF-8 4 spaces Python 3.11
29°C
Mostly cloudy
Search
ENG
IN
22:18
13-06-2024