**DIVIDE AND CONQUER LAB PROGRAMS**

**1.MERGE SORT**

def merge\_sort(arr):

if len(arr)>1:

mid=len(arr)//2

left=arr[:mid]

right=arr[mid:]

merge\_sort(left)

merge\_sort(right)

merge(arr,left,right)

def merge(arr,left,right):

i=j=k=0

while i<len(left) and j<len(right):

if left[i]<right[j]:

arr[k]=left[i]

i+=1

else:

arr[k]=right[j]

j+=1

k+=1

while i<len(left):

arr[k]=left[i]

i+=1

k+=1

while j<len(right):

arr[k]=right[j]

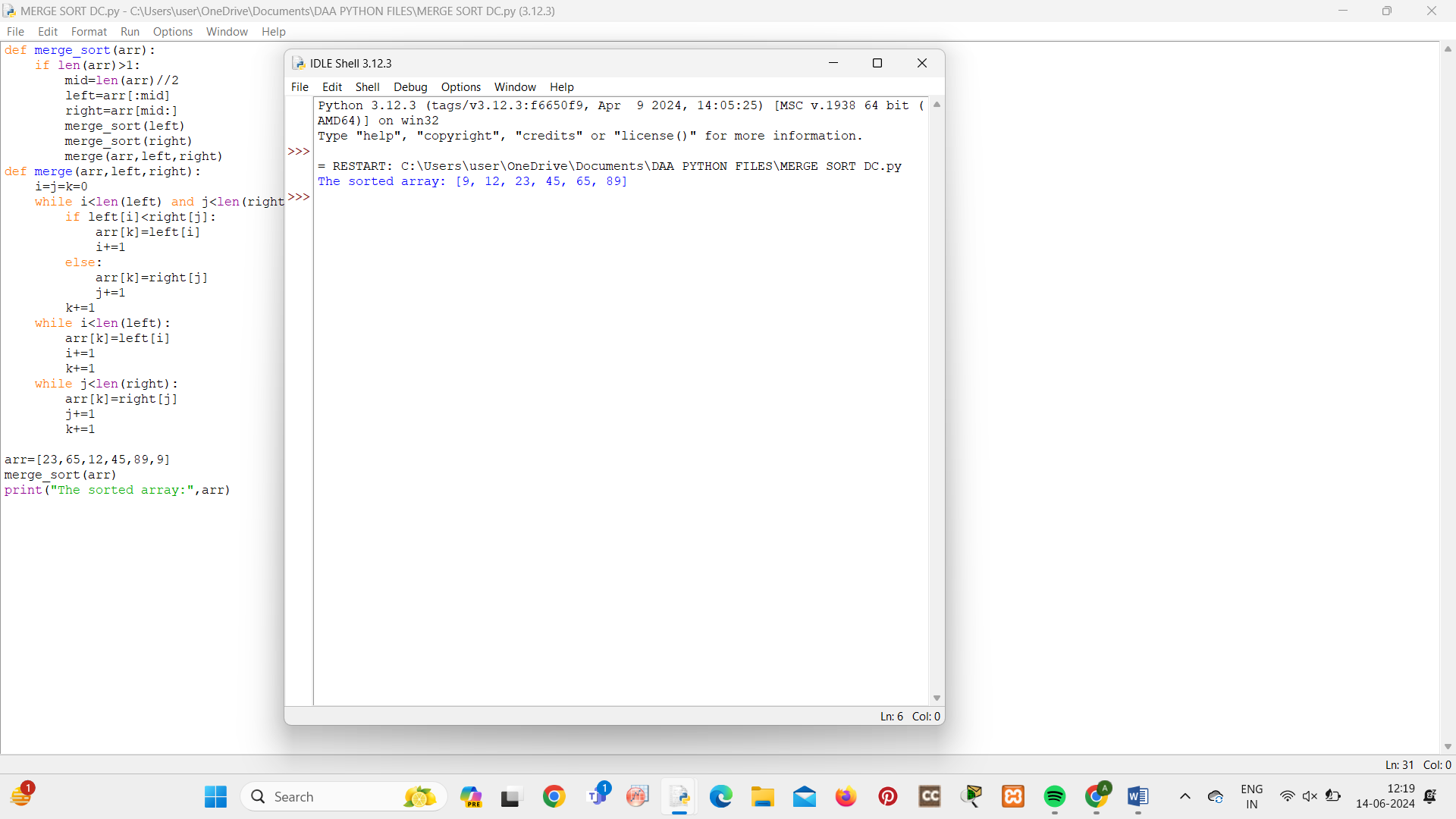
j+=1

k+=1

arr=[23,65,12,45,89,9]

merge\_sort(arr)

print("The sorted array:",arr)



**2.QUICK SORT**

def quick\_sort(arr):

if len(arr)<=1:

return arr

else:

pivot=arr[0]

lesser=[x for x in arr[1:] if x<=pivot]

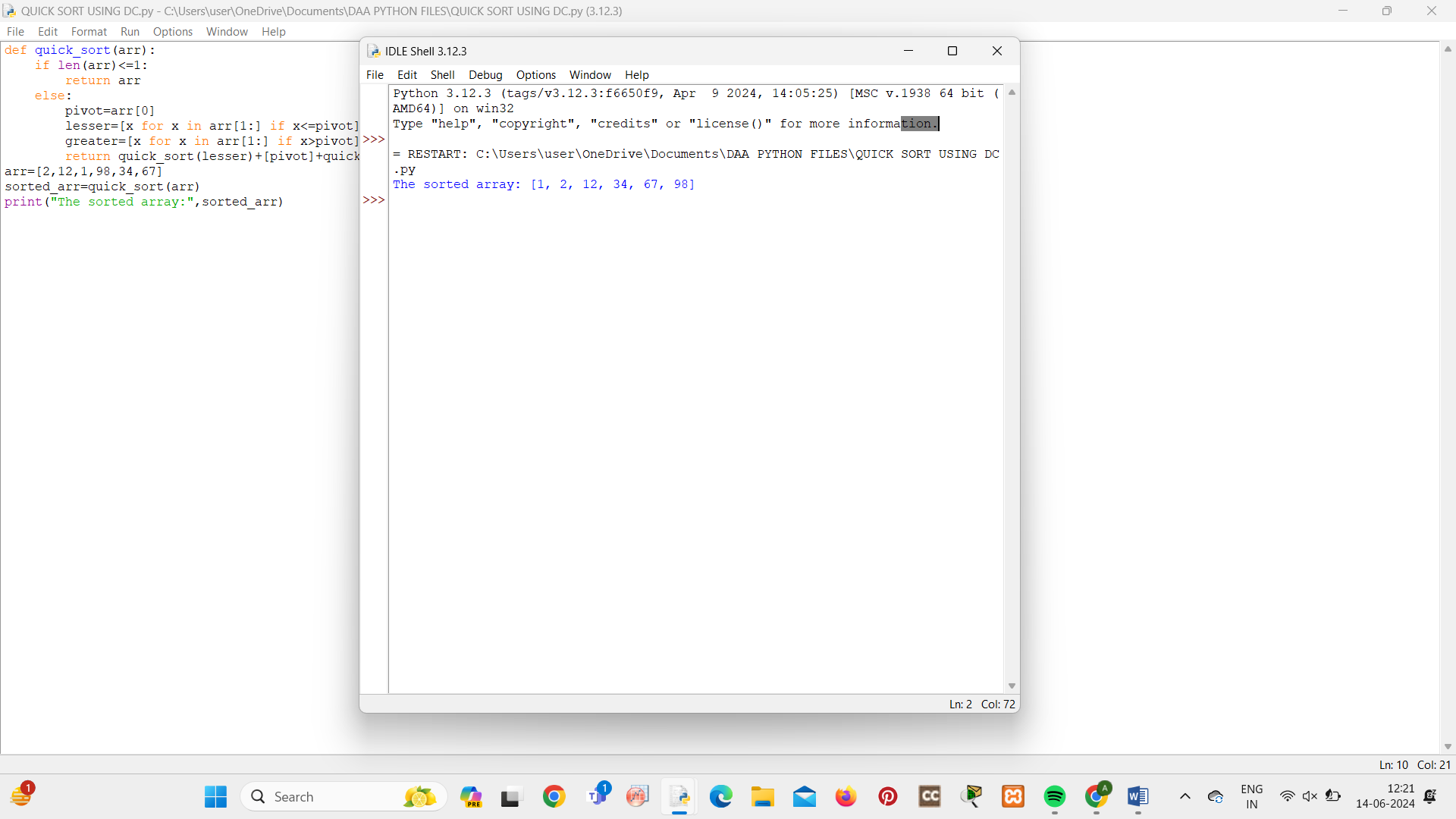
greater=[x for x in arr[1:] if x>pivot]

return quick\_sort(lesser)+[pivot]+quick\_sort(greater)

arr=[2,12,1,98,34,67]

sorted\_arr=quick\_sort(arr)

print("The sorted array:",sorted\_arr)



**3.FINDING MAXIMUM AND MINIMUM**

def max\_min(arr):

if len(arr)==1:

return (arr[0],arr[0])

elif len(arr)==2:

if arr[0]<arr[1]:

return (arr[1],arr[0])

else:

return (arr[0],arr[1])

else:

mid=len(arr)//2

max1,min1=max\_min(arr[:mid])

max2,min2=max\_min(arr[mid:])

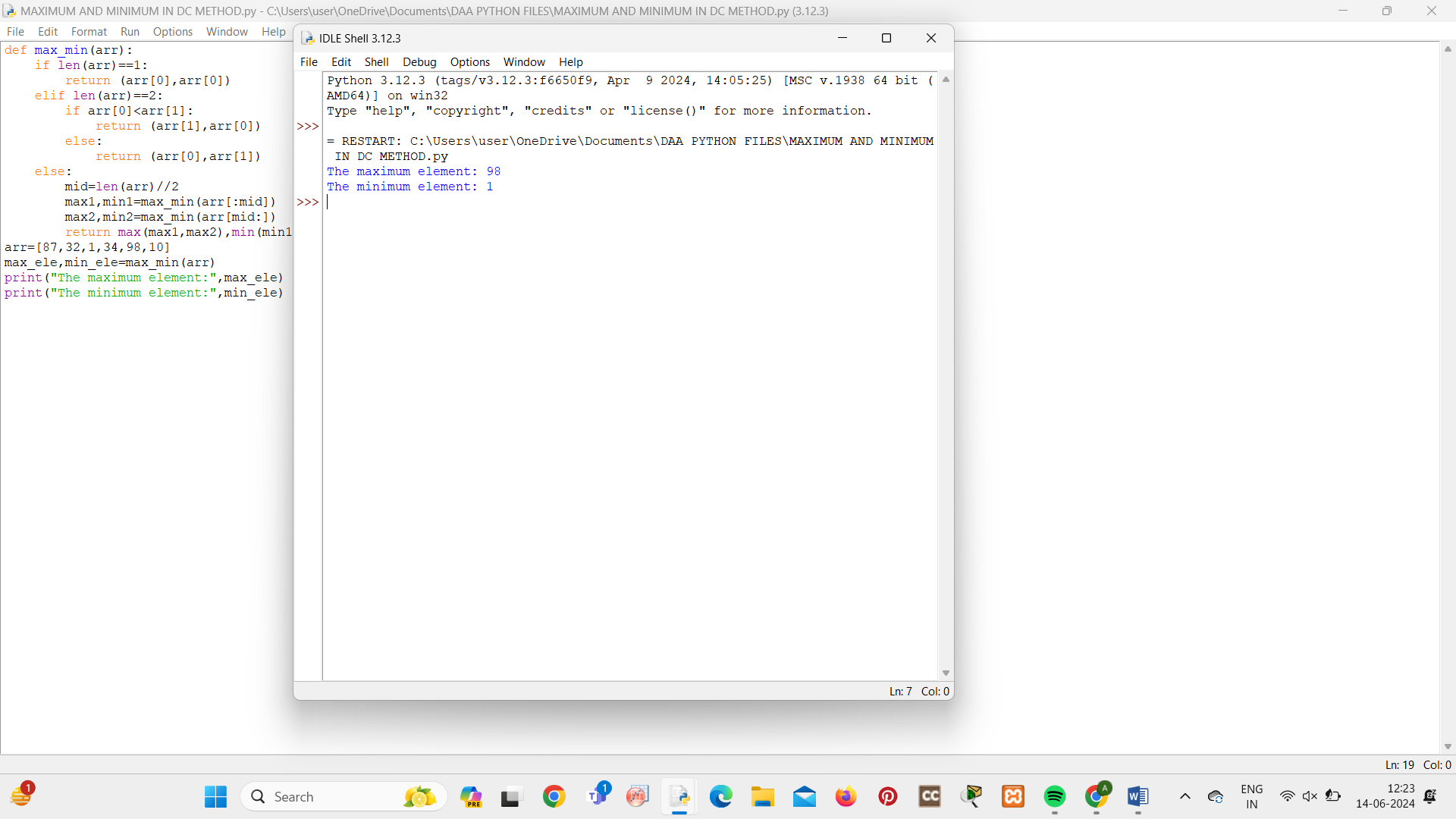
return max(max1,max2),min(min1,min2)

arr=[87,32,1,34,98,10]

max\_ele,min\_ele=max\_min(arr)

print("The maximum element:",max\_ele)

print("The minimum element:",min\_ele)



**4.BINARY SEARCH**

def bin\_search(arr,target):

low=0

high=len(arr)-1

while low<=high:

mid=(low+high)//2

if arr[mid]==target:

return mid

elif arr[mid]<target:

low=mid+1

else:

high=mid-1

return -1

arr=[1,2,3,4,5,6]

target=5

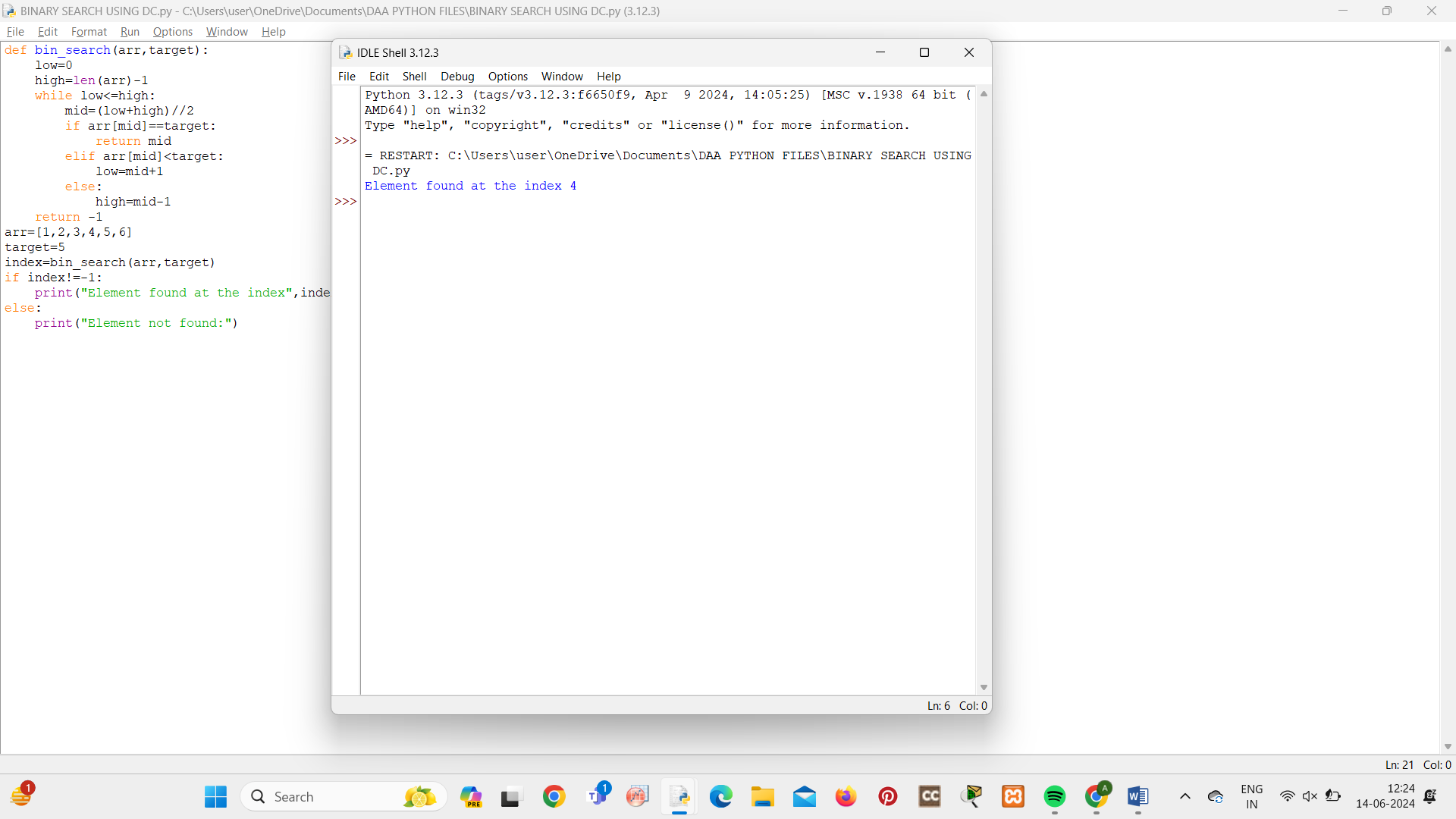
index=bin\_search(arr,target)

if index!=-1:

print("Element found at the index",index)

else:

print("Element not found:")



**5.STRASSENS MATRIX MULTIPLICATION**

def strassen\_matrix\_multiply(A, B):

n = len(A)

if n == 1:

return [[A[0][0] \* B[0][0]]]

a11, a12, a21, a22 = split\_matrix(A)

b11, b12, b21, b22 = split\_matrix(B)

p1 = strassen\_matrix\_multiply(add\_matrices(a11, a22), add\_matrices(b11, b22))

p2 = strassen\_matrix\_multiply(add\_matrices(a21, a22), b11)

p3 = strassen\_matrix\_multiply(a11, subtract\_matrices(b12, b22))

p4 = strassen\_matrix\_multiply(a22, subtract\_matrices(b21, b11))

p5 = strassen\_matrix\_multiply(add\_matrices(a11, a12), b22)

p6 = strassen\_matrix\_multiply(subtract\_matrices(a21, a11), add\_matrices(b11, b12))

p7 = strassen\_matrix\_multiply(subtract\_matrices(a12, a22), add\_matrices(b21, b22))

c11 = add\_matrices(subtract\_matrices(add\_matrices(p1, p4), p5), p7)

c12 = add\_matrices(p3, p5)

c21 = add\_matrices(p2, p4)

c22 = add\_matrices(subtract\_matrices(add\_matrices(p1, p3), p2), p6)

C = combine\_matrices(c11, c12, c21, c22)

return C

def split\_matrix(A):

n = len(A)

mid = n // 2

a11 = [row[:mid] for row in A[:mid]]

a12 = [row[mid:] for row in A[:mid]]

a21 = [row[:mid] for row in A[mid:]]

a22 = [row[mid:] for row in A[mid:]]

return a11, a12, a21, a22

def add\_matrices(A, B):

return [[A[i][j] + B[i][j] for j in range(len(A))] for i in range(len(A))]

def subtract\_matrices(A, B):

return [[A[i][j] - B[i][j] for j in range(len(A))] for i in range(len(A))]

def combine\_matrices(a11, a12, a21, a22):

n = len(a11)

result = [[0] \* (2 \* n) for \_ in range(2 \* n)]

for i in range(n):

for j in range(n):

result[i][j] = a11[i][j]

result[i][j + n] = a12[i][j]

result[i + n][j] = a21[i][j]

result[i + n][j + n] = a22[i][j]

return result

A = [[1, 2, 3, 4],

[5, 6, 7, 8],

[9, 10, 11, 12],

[13, 14, 15, 16]]

B = [[17, 18, 19, 20],

[21, 22, 23, 24],

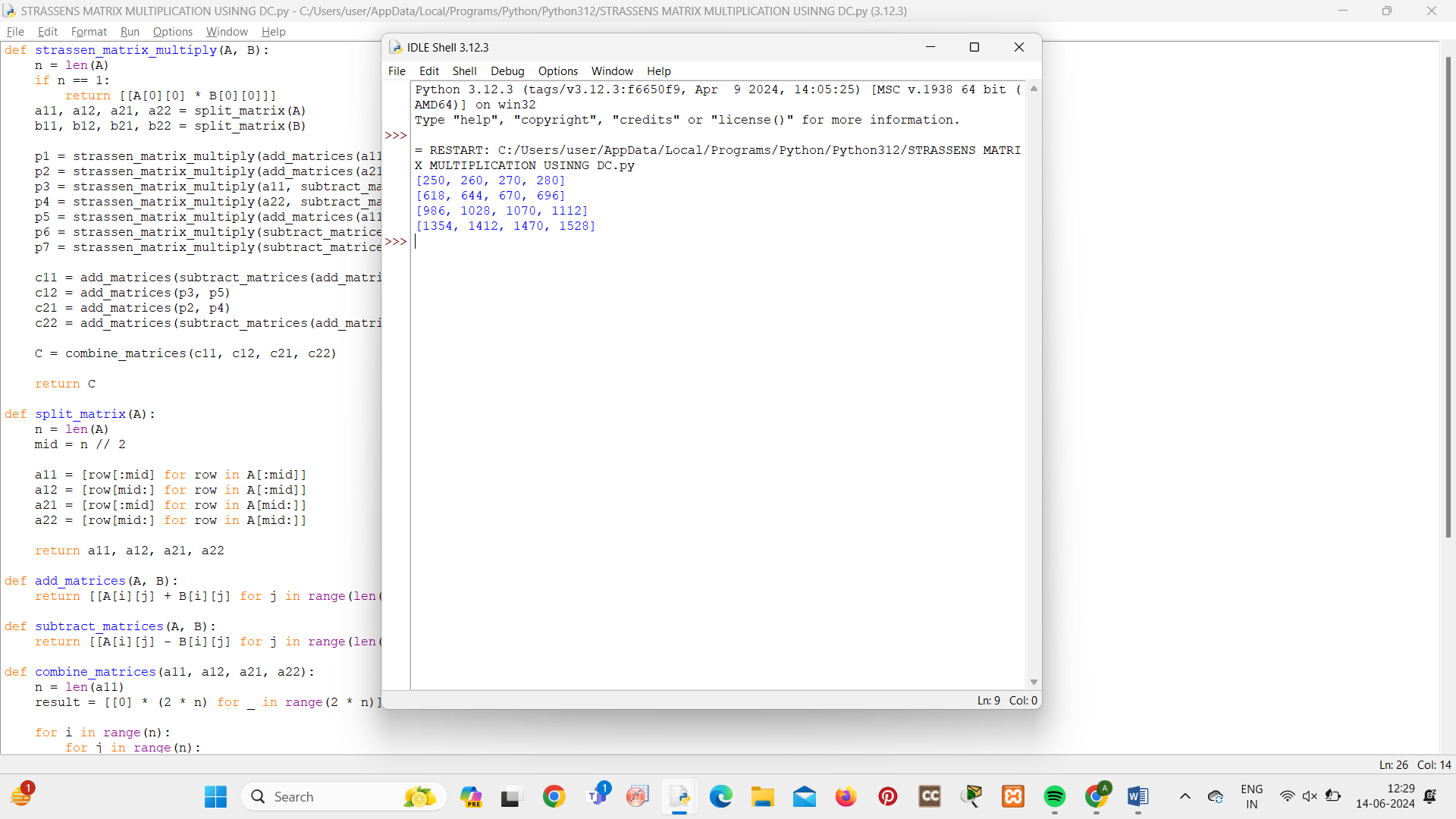
[25, 26, 27, 28],

[29, 30, 31, 32]]

result = strassen\_matrix\_multiply(A, B)

for row in result:

print(row)



**6.KARATSUPA MULTIPLICATION**

def karatsuba\_multiply(x, y):

str\_x = str(x)

str\_y = str(y)

if len(str\_x) == 1 or len(str\_y) == 1:

return x \* y

max\_len = max(len(str\_x), len(str\_y))

str\_x = str\_x.zfill(max\_len)

str\_y = str\_y.zfill(max\_len)

m = max\_len // 2

a = int(str\_x[:m])

b = int(str\_x[m:])

c = int(str\_y[:m])

d = int(str\_y[m:])

ac = karatsuba\_multiply(a, c)

bd = karatsuba\_multiply(b, d)

ad\_plus\_bc = karatsuba\_multiply(a + b, c + d) - ac - bd

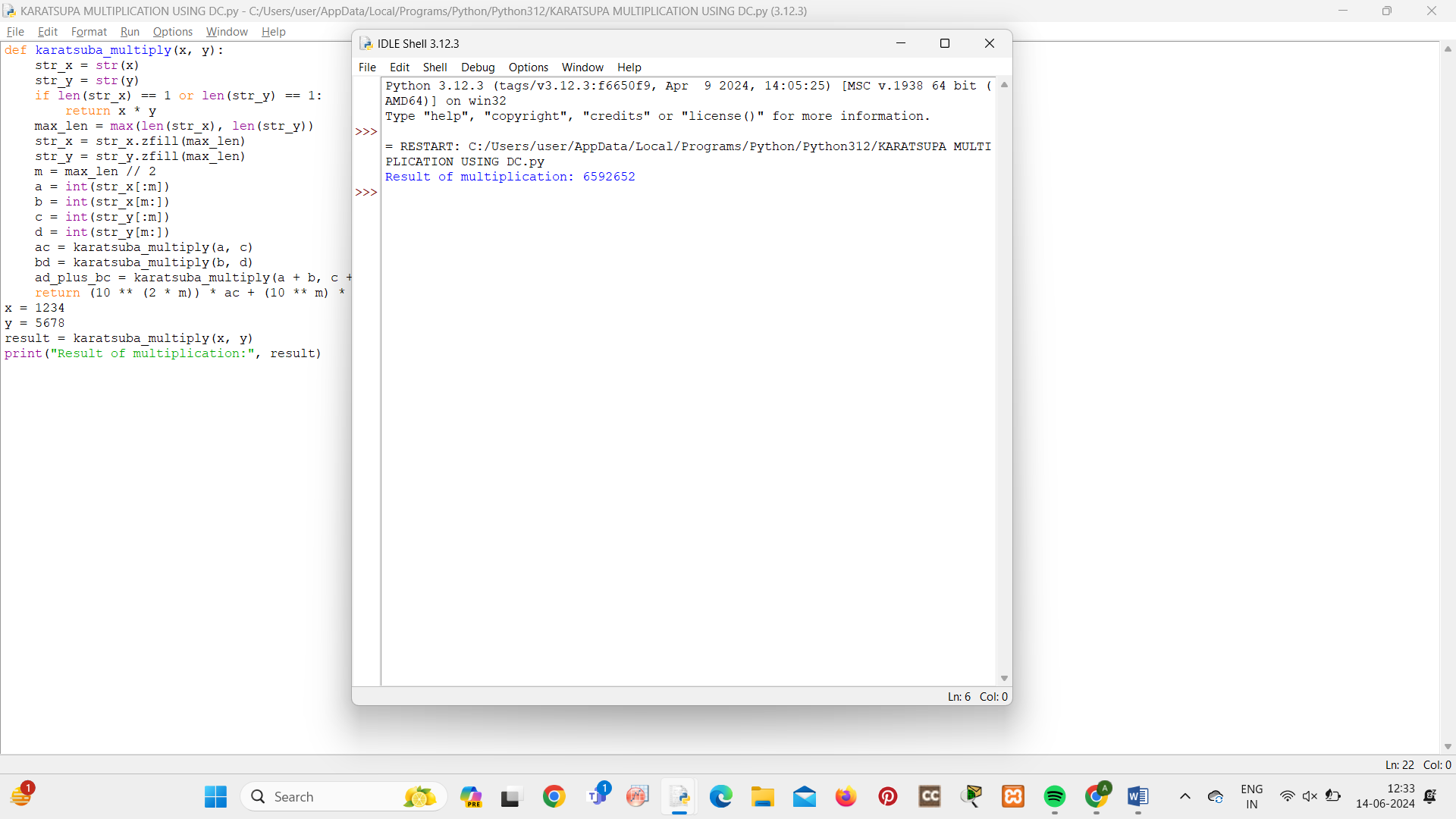
return (10 \*\* (2 \* m)) \* ac + (10 \*\* m) \* ad\_plus\_bc + bd

x = 1234

y = 5678

result = karatsuba\_multiply(x, y)

print("Result of multiplication:", result)



**7.CLOSEST PAIR OF POINTS**

import math

def euclidean\_distance(p1, p2):

return math.sqrt((p1[0] - p2[0]) \*\* 2 + (p1[1] - p2[1]) \*\* 2)

def brute\_force\_closest\_pair(points):

min\_dist = float('inf')

closest\_pair = None

for i in range(len(points)):

for j in range(i + 1, len(points)):

dist = euclidean\_distance(points[i], points[j])

if dist < min\_dist:

min\_dist = dist

closest\_pair = (points[i], points[j])

return min\_dist, closest\_pair

def closest\_pair\_strip(strip, d):

min\_dist = d

closest\_pair = None

strip.sort(key=lambda point: point[1])

for i in range(len(strip)):

j = i + 1

while j < len(strip) and (strip[j][1] - strip[i][1]) < min\_dist:

dist = euclidean\_distance(strip[i], strip[j])

if dist < min\_dist:

min\_dist = dist

closest\_pair = (strip[i], strip[j])

j += 1

return min\_dist, closest\_pair

def closest\_pair\_divide\_and\_conquer(points):

n = len(points)

if n <= 3:

return brute\_force\_closest\_pair(points)

mid = n // 2

mid\_point = points[mid]

left\_half = points[:mid]

right\_half = points[mid:]

left\_dist, left\_pair = closest\_pair\_divide\_and\_conquer(left\_half)

right\_dist, right\_pair = closest\_pair\_divide\_and\_conquer(right\_half)

min\_dist = min(left\_dist, right\_dist)

strip = [point for point in points if abs(point[0] - mid\_point[0]) < min\_dist]

strip\_dist, strip\_pair = closest\_pair\_strip(strip, min\_dist)

if strip\_dist < min\_dist:

return strip\_dist, strip\_pair

elif left\_dist <= right\_dist:

return left\_dist, left\_pair

else:

return right\_dist, right\_pair

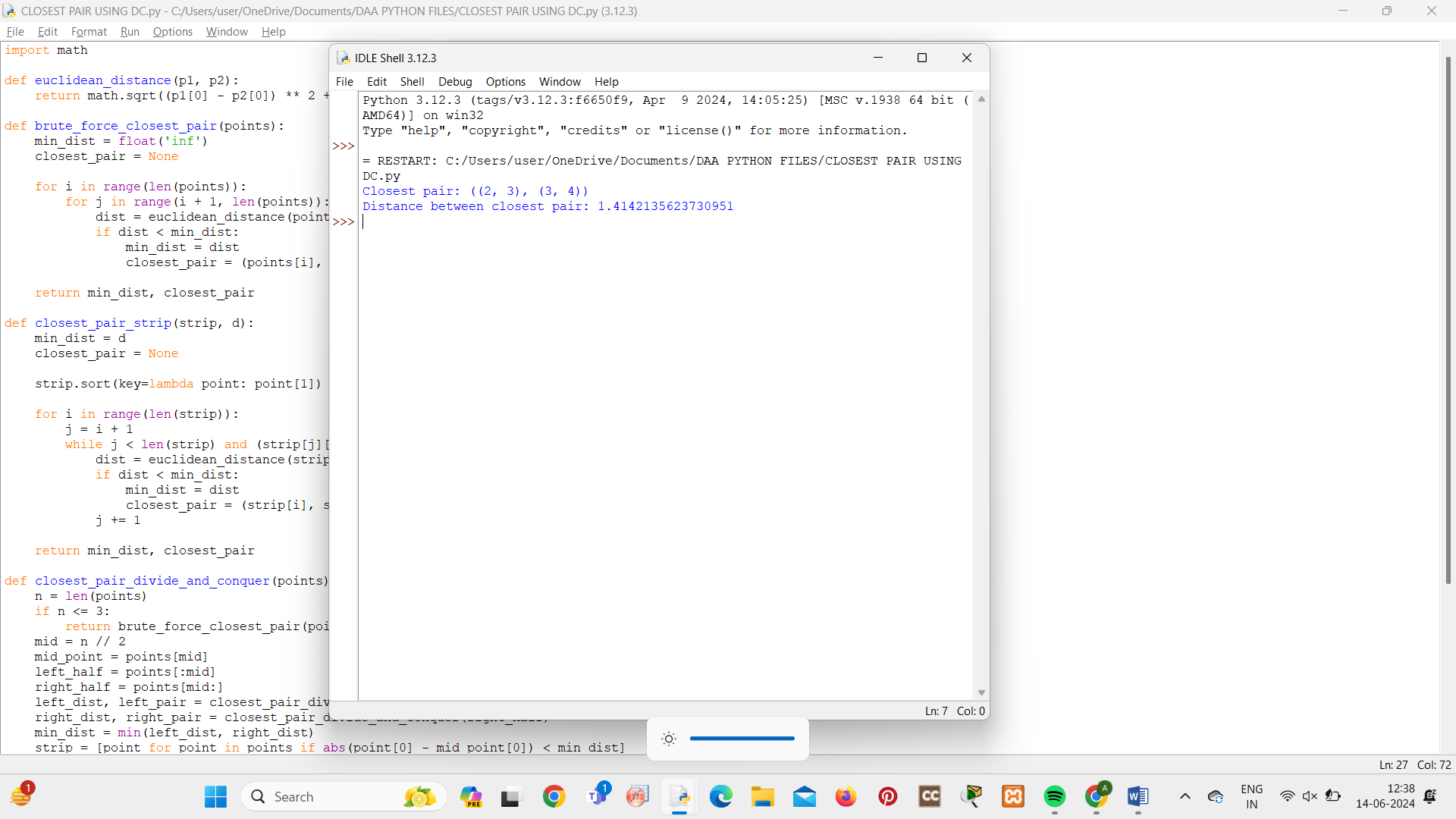
points = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]

points.sort()

closest\_dist, closest\_pair = closest\_pair\_divide\_and\_conquer(points)

print("Closest pair:", closest\_pair)

print("Distance between closest pair:", closest\_dist)



**8.MEDIAN OF MEDIANS**

import statistics

def median\_of\_medians(arr, k):

if len(arr) == 1:

return arr[0]

sublists = [arr[i:i + 5] for i in range(0, len(arr), 5)]

medians = [statistics.median(sublist) for sublist in sublists]

pivot = median\_of\_medians(medians, len(medians) // 2)

left = [x for x in arr if x < pivot]

right = [x for x in arr if x > pivot]

pivot\_count = arr.count(pivot)

if k < len(left):

return median\_of\_medians(left, k)

elif k < len(left) + pivot\_count:

return pivot

else:

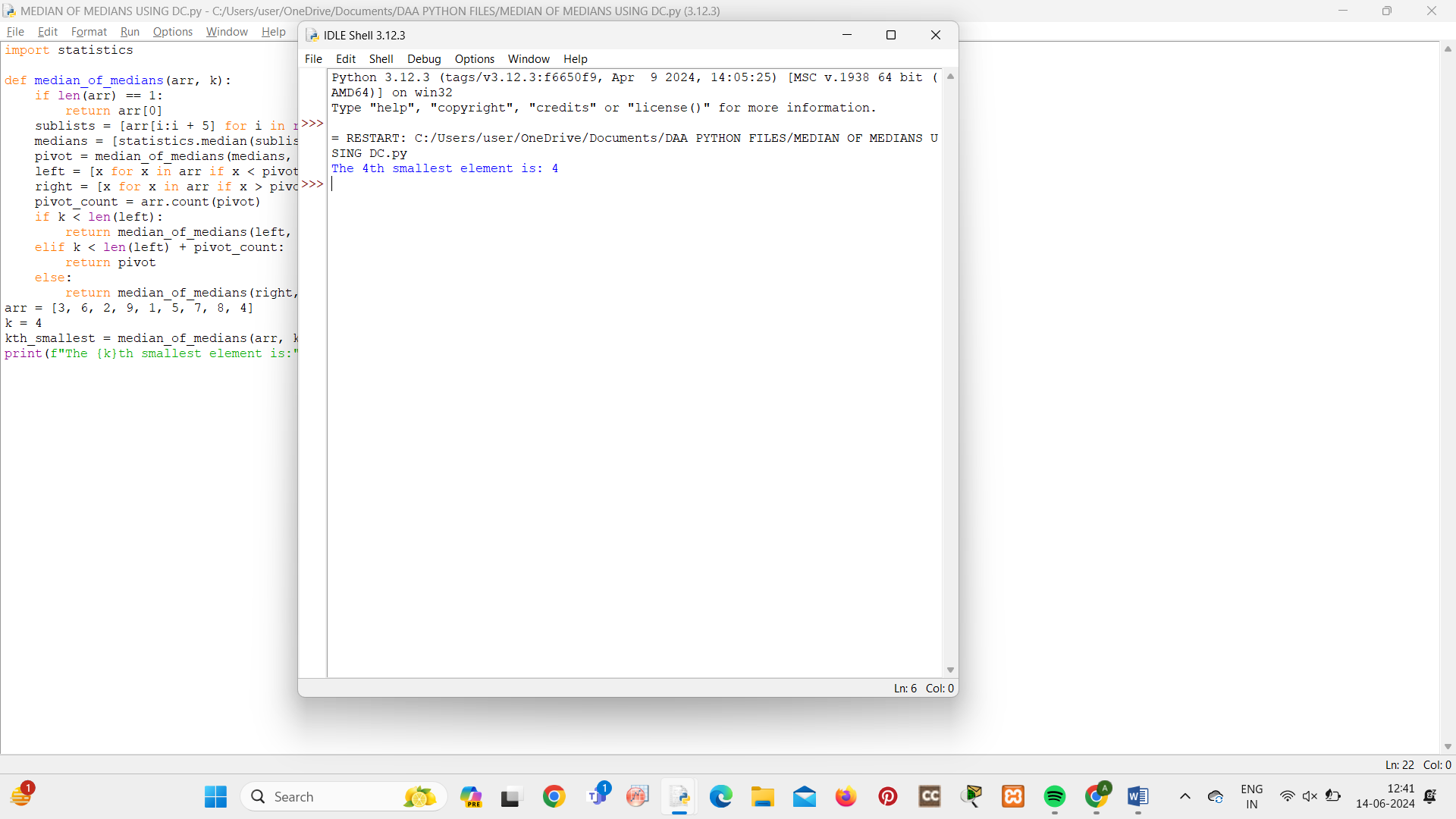
return median\_of\_medians(right, k - len(left) - pivot\_count)

arr = [3, 6, 2, 9, 1, 5, 7, 8, 4]

k = 4

kth\_smallest = median\_of\_medians(arr, k - 1)

print(f"The {k}th smallest element is:", kth\_smallest)



**9.MEET IN MIDDLE TECHNIQUE**

def subsets\_sum(arr, target):

def generate\_subsets(subarr, index, path, target, result):

if sum(path) == target:

result.append(path)

if index == len(subarr) or sum(path) >= target:

return

generate\_subsets(subarr, index + 1, path + [subarr[index]], target, result)

generate\_subsets(subarr, index + 1, path, target, result)

result = []

generate\_subsets(arr, 0, [], target, result)

return result

arr = [1, 2, 3, 4, 5]

target = 7

result = subsets\_sum(arr, target)

print("Subsets that sum up to", target, ":", result)

