DAY – 08

1.Finding the Maximum and Minimum

def min\_max\_sequence(numbers):

for num\_list in numbers:

print(f"List: {num\_list}")

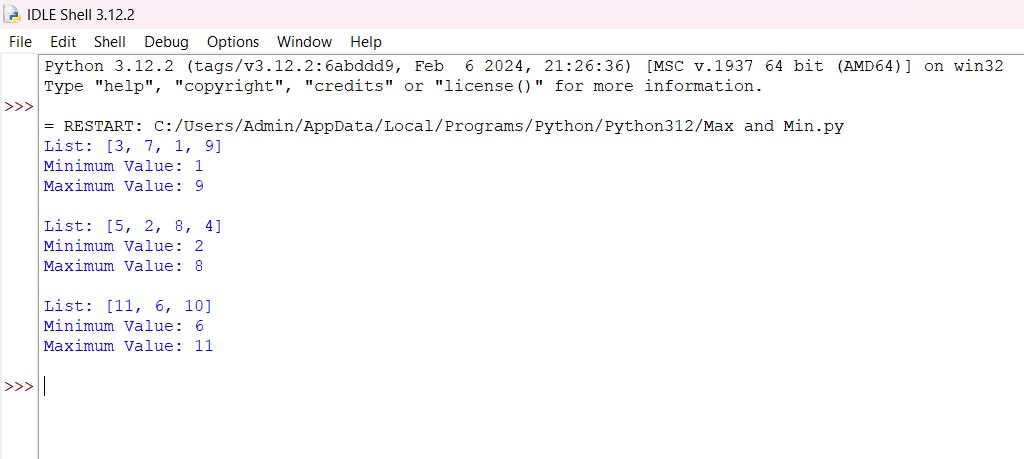
print(f"Minimum Value: {min(num\_list)}")

print(f"Maximum Value: {max(num\_list)}")

print()

numbers\_list = [[3, 7, 1, 9], [5, 2, 8, 4], [11, 6, 10]]

min\_max\_sequence(numbers\_list)



2. Merge Sort

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

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

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

def print\_list(arr):

for i in range(len(arr)):

print(arr[i], end=" ")

print()

if \_\_name\_\_ == "\_\_main\_\_":

arr = [12, 11, 13, 5, 6, 7]

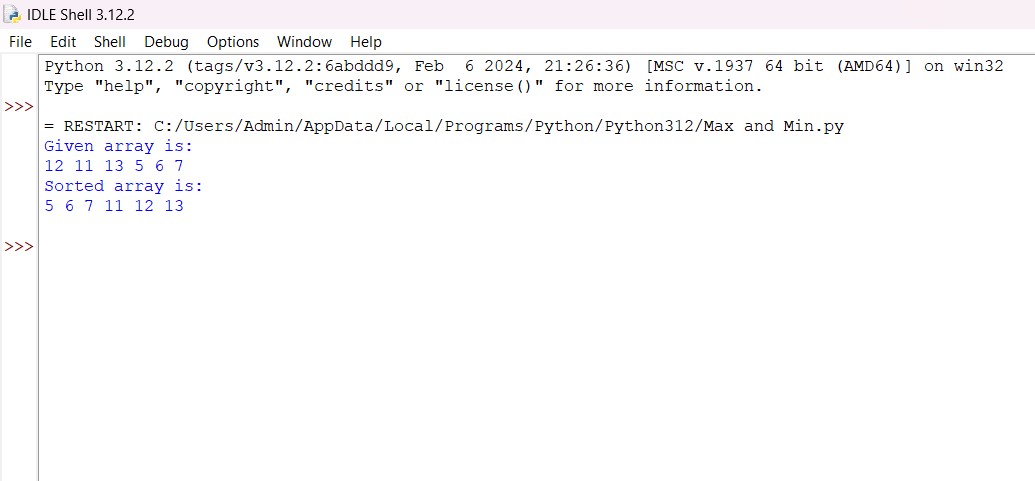
print("Given array is:")

print\_list(arr)

merge\_sort(arr)

print("Sorted array is:")

print\_list(arr)



3. Quick Sort

def quick\_sort(arr):

if len(arr) <= 1:

return arr

else:

pivot = arr[len(arr) // 2]

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

middle = [x for x in arr if x == pivot]

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

return quick\_sort(left) + middle + quick\_sort(right)

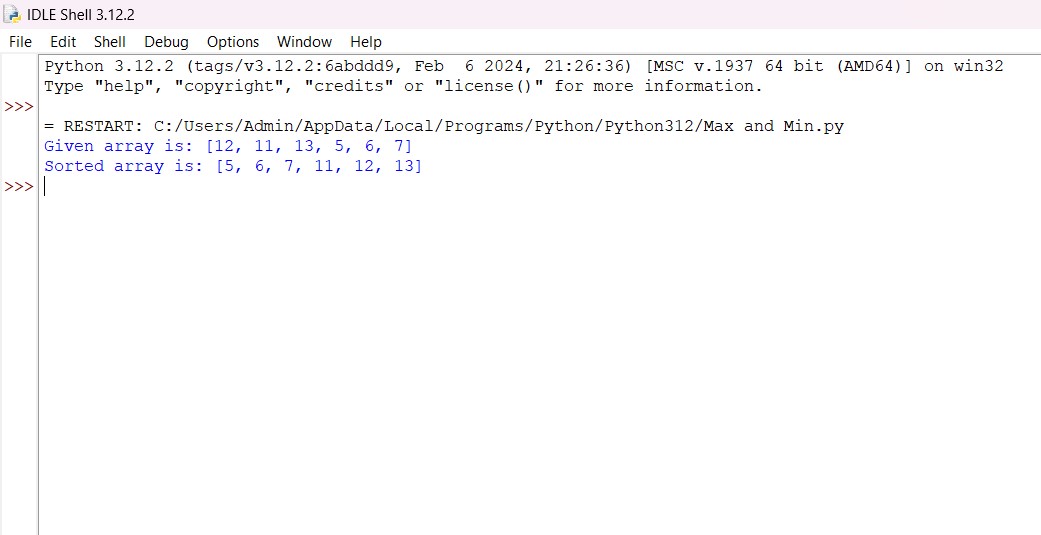
if \_\_name\_\_ == "\_\_main\_\_":

arr = [12, 11, 13, 5, 6, 7]

print("Given array is:", arr)

sorted\_arr = quick\_sort(arr)

print("Sorted array is:", sorted\_arr)



4. Binary Search

def binary\_search(arr, target):

left, right = 0, len(arr) - 1

while left <= right:

mid = (left + right) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

left = mid + 1

else:

right = mid - 1

return -1

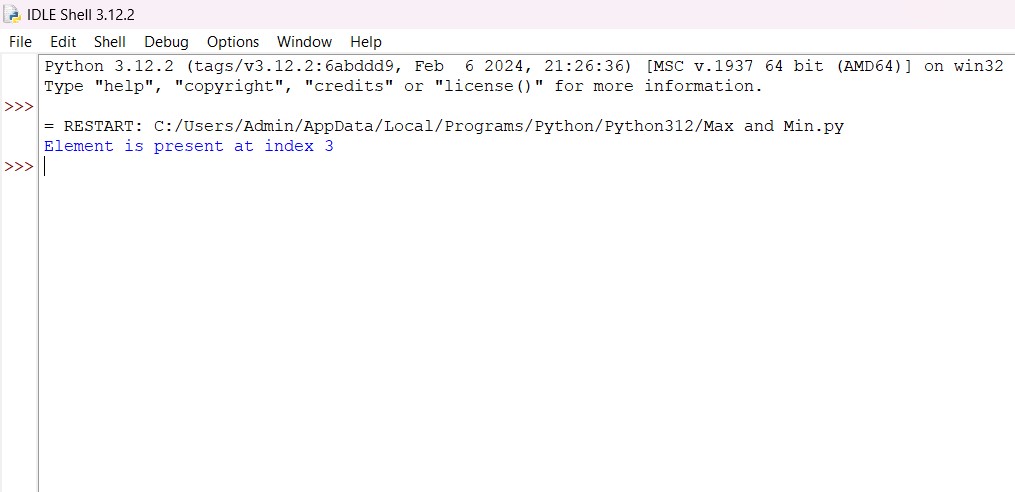
if \_\_name\_\_ == "\_\_main\_\_":

arr = [2, 3, 4, 10, 40]

target = 10

result = binary\_search(arr, target)

print("Element is present at index" if result != -1 else "Element is not present in array", result)



5. Strassens Matrix Multiplication

import math

from typing import List, Tuple

def default\_matrix\_multiplication(a: List, b: List) -> List:

"""

Multiplication only for 2x2 matrices

"""

if len(a) != 2 or len(a[0]) != 2 or len(b) != 2 or len(b[0]) != 2:

raise Exception("Matrices are not 2x2")

new\_matrix = [

[a[0][0] \* b[0][0] + a[0][1] \* b[1][0], a[0][0] \* b[0][1] + a[0][1] \* b[1][1]],

[a[1][0] \* b[0][0] + a[1][1] \* b[1][0], a[1][0] \* b[0][1] + a[1][1] \* b[1][1]],

]

return new\_matrix

def matrix\_addition(matrix\_a: List, matrix\_b: List):

return [[matrix\_a[row][col] + matrix\_b[row][col] for col in range(len(matrix\_a[row]))]

for row in range(len(matrix\_a))]

def matrix\_subtraction(matrix\_a: List, matrix\_b: List):

return [[matrix\_a[row][col] - matrix\_b[row][col] for col in range(len(matrix\_a[row]))]

for row in range(len(matrix\_a))]

def split\_matrix(a: List,) -> Tuple[List, List, List, List]:

"""

Given an even length matrix, returns the top\_left, top\_right, bot\_left, bot\_right

quadrant.

>>> split\_matrix([[4,3,2,4],[2,3,1,1],[6,5,4,3],[8,4,1,6]])

([[4, 3], [2, 3]], [[2, 4], [1, 1]], [[6, 5], [8, 4]], [[4, 3], [1, 6]])

>>> split\_matrix([

... [4,3,2,4,4,3,2,4],[2,3,1,1,2,3,1,1],[6,5,4,3,6,5,4,3],[8,4,1,6,8,4,1,6],

... [4,3,2,4,4,3,2,4],[2,3,1,1,2,3,1,1],[6,5,4,3,6,5,4,3],[8,4,1,6,8,4,1,6]

... ]) # doctest: +NORMALIZE\_WHITESPACE

([[4, 3, 2, 4], [2, 3, 1, 1], [6, 5, 4, 3], [8, 4, 1, 6]], [[4, 3, 2, 4],

[2, 3, 1, 1], [6, 5, 4, 3], [8, 4, 1, 6]], [[4, 3, 2, 4], [2, 3, 1, 1],

[6, 5, 4, 3], [8, 4, 1, 6]], [[4, 3, 2, 4], [2, 3, 1, 1], [6, 5, 4, 3],

[8, 4, 1, 6]])

"""

if len(a) % 2 != 0 or len(a[0]) % 2 != 0:

raise Exception("Odd matrices are not supported!")

matrix\_length = len(a)

mid = matrix\_length // 2

top\_right = [[a[i][j] for j in range(mid, matrix\_length)] for i in range(mid)]

bot\_right = [[a[i][j] for j in range(mid, matrix\_length)] for i in range(mid, matrix\_length)]

top\_left = [[a[i][j] for j in range(mid)] for i in range(mid)]

bot\_left = [[a[i][j] for j in range(mid)] for i in range(mid, matrix\_length)]

return top\_left, top\_right, bot\_left, bot\_right

def matrix\_dimensions(matrix: List) -> Tuple[int, int]:

return len(matrix), len(matrix[0])

def print\_matrix(matrix: List) -> None:

for i in range(len(matrix)):

print(matrix[i])

def actual\_strassen(matrix\_a: List, matrix\_b: List) -> List:

"""

Recursive function to calculate the product of two matrices, using the Strassen

Algorithm. It only supports even length matrices.

"""

if matrix\_dimensions(matrix\_a) == (2, 2):

return default\_matrix\_multiplication(matrix\_a, matrix\_b)

a, b, c, d = split\_matrix(matrix\_a)

e, f, g, h = split\_matrix(matrix\_b)

t1 = actual\_strassen(a, matrix\_subtraction(f, h))

t2 = actual\_strassen(matrix\_addition(a, b), h)

t3 = actual\_strassen(matrix\_addition(c, d), e)

t4 = actual\_strassen(d, matrix\_subtraction(g, e))

t5 = actual\_strassen(matrix\_addition(a, d), matrix\_addition(e, h))

t6 = actual\_strassen(matrix\_subtraction(b, d), matrix\_addition(g, h))

t7 = actual\_strassen(matrix\_subtraction(a, c), matrix\_addition(e, f))

top\_left = matrix\_addition(matrix\_subtraction(matrix\_addition(t5, t4), t2), t6)

top\_right = matrix\_addition(t1, t2)

bot\_left = matrix\_addition(t3, t4)

bot\_right = matrix\_subtraction(matrix\_subtraction(matrix\_addition(t1, t5), t3), t7)

new\_matrix = []

for i in range(len(top\_right)):

new\_matrix.append(top\_left[i] + top\_right[i])

for i in range(len(bot\_right)):

new\_matrix.append(bot\_left[i] + bot\_right[i])

return new\_matrix

def strassen(matrix1: List, matrix2: List) -> List:

"""

>>> strassen([[2,1,3],[3,4,6],[1,4,2],[7,6,7]], [[4,2,3,4],[2,1,1,1],[8,6,4,2]])

[[34, 23, 19, 15], [68, 46, 37, 28], [28, 18, 15, 12], [96, 62, 55, 48]]

>>> strassen([[3,7,5,6,9],[1,5,3,7,8],[1,4,4,5,7]], [[2,4],[5,2],[1,7],[5,5],[7,8]])

[[139, 163], [121, 134], [100, 121]]

"""

if matrix\_dimensions(matrix1)[1] != matrix\_dimensions(matrix2)[0]:

raise Exception(f"Unable to multiply these matrices, please check the dimensions. \n"

f"Matrix A:{matrix1} \nMatrix B:{matrix2}")

dimension1 = matrix\_dimensions(matrix1)

dimension2 = matrix\_dimensions(matrix2)

if dimension1[0] == dimension1[1] and dimension2[0] == dimension2[1]:

return matrix1, matrix2

maximum = max(max(dimension1), max(dimension2))

maxim = int(math.pow(2, math.ceil(math.log2(maximum))))

new\_matrix1 = matrix1

new\_matrix2 = matrix2

for i in range(0, maxim):

if i < dimension1[0]:

for j in range(dimension1[1], maxim):

new\_matrix1[i].append(0)

else:

new\_matrix1.append([0] \* maxim)

if i < dimension2[0]:

for j in range(dimension2[1], maxim):

new\_matrix2[i].append(0)

else:

new\_matrix2.append([0] \* maxim)

final\_matrix = actual\_strassen(new\_matrix1, new\_matrix2)

for i in range(0, maxim):

if i < dimension1[0]:

for j in range(dimension2[1], maxim):

final\_matrix[i].pop()

else:

final\_matrix.pop()

return final\_matrix

if \_\_name\_\_ == "\_\_main\_\_":

matrix1 = [[2, 3, 4, 5],

[6, 4, 3, 1],

[2, 3, 6, 7],

[3, 1, 2, 4],

[2, 3, 4, 5],

[6, 4, 3, 1],

[2, 3, 6, 7],

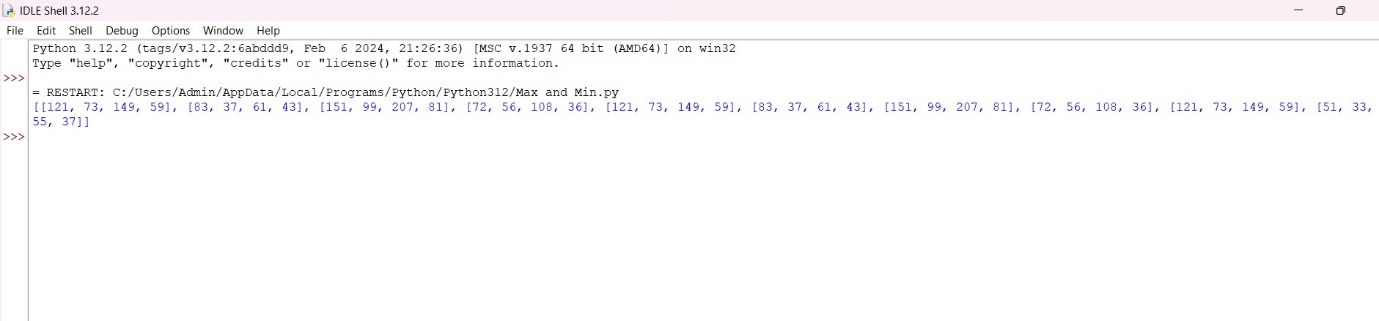
[3, 1, 2, 4],

[2, 3, 4, 5],

[6, 2, 3, 1],]

matrix2 = [[0, 2, 1, 1], [16, 2, 3, 3], [2, 2, 7, 7], [13, 11, 22, 4]]

print(strassen(matrix1, matrix2))



6. Karatsuba algorithm for multiplication

def karatsuba(x, y):

if x < 10 or y < 10:

return x \* y

n = max(len(str(x)), len(str(y)))

m = n // 2

high1, low1 = divmod(x, 10\*\*m)

high2, low2 = divmod(y, 10\*\*m)

z0 = karatsuba(low1, low2)

z1 = karatsuba((low1 + high1), (low2 + high2))

z2 = karatsuba(high1, high2)

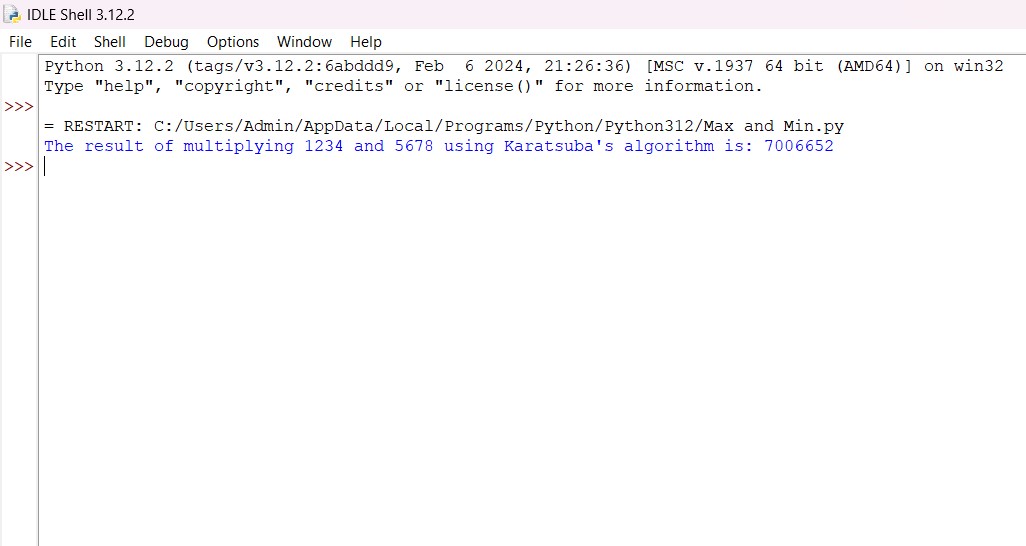
return (z2 \* 10\*\*(2\*m)) + ((z1 - z2 - z0) \* 10\*\*m) + z0

x = 1234

y = 5678

result = karatsuba(x, y)

print(f"The result of multiplying {x} and {y} using Karatsuba's algorithm is: {result}")



7. Closest pair of points using divide and conquer

import math

def distance(p1, p2):

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

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

min\_dist = d

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:

min\_dist = distance(strip[i], strip[j])

j += 1

return min\_dist

def closest\_pair\_rec(points\_sorted\_x, points\_sorted\_y):

if len(points\_sorted\_x) <= 3:

return brute\_force(points\_sorted\_x)

mid = len(points\_sorted\_x) // 2

midpoint = points\_sorted\_x[mid]

left\_half\_x = points\_sorted\_x[:mid]

right\_half\_x = points\_sorted\_x[mid:]

left\_half\_y = list(filter(lambda point: point[0] <= midpoint[0], points\_sorted\_y))

right\_half\_y = list(filter(lambda point: point[0] > midpoint[0], points\_sorted\_y))

d1 = closest\_pair\_rec(left\_half\_x, left\_half\_y)

d2 = closest\_pair\_rec(right\_half\_x, right\_half\_y)

d = min(d1, d2)

strip = [point for point in points\_sorted\_y if abs(point[0] - midpoint[0]) < d]

return min(d, closest\_in\_strip(strip, d))

def brute\_force(points):

min\_dist = float('inf')

for i in range(len(points)):

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

if distance(points[i], points[j]) < min\_dist:

min\_dist = distance(points[i], points[j])

return min\_dist

def closest\_pair(points):

points\_sorted\_x = sorted(points, key=lambda point: point[0])

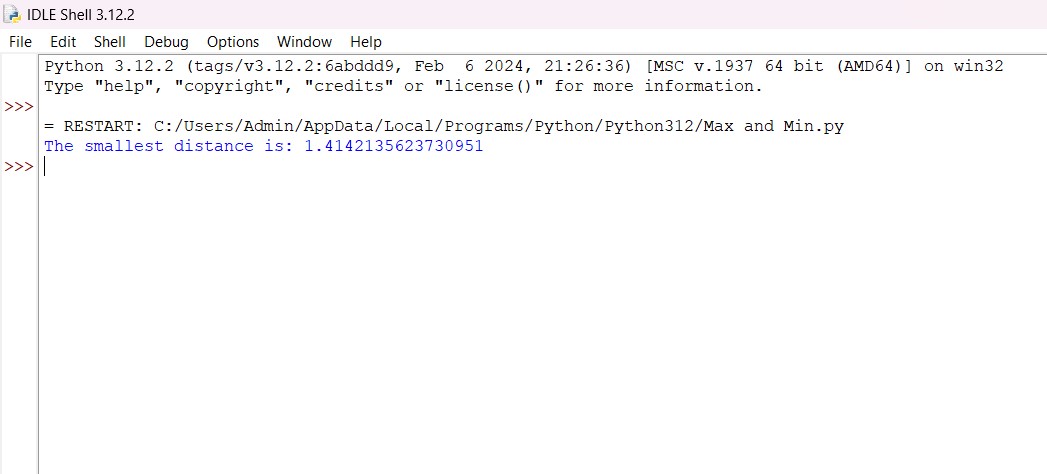
points\_sorted\_y = sorted(points, key=lambda point: point[1])

return closest\_pair\_rec(points\_sorted\_x, points\_sorted\_y)

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

result = closest\_pair(points)

print(f"The smallest distance is: {result}")



8. Median of medians

def find\_median(arr):

arr.sort()

return arr[len(arr) // 2]

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

if len(arr) <= 5:

return sorted(arr)[k]

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

medians = [find\_median(sublist) for sublist in sublists]

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

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

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

pivot\_count = len(arr) - len(low) - len(high)

if k < len(low):

return median\_of\_medians(low, k)

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

return pivot

else:

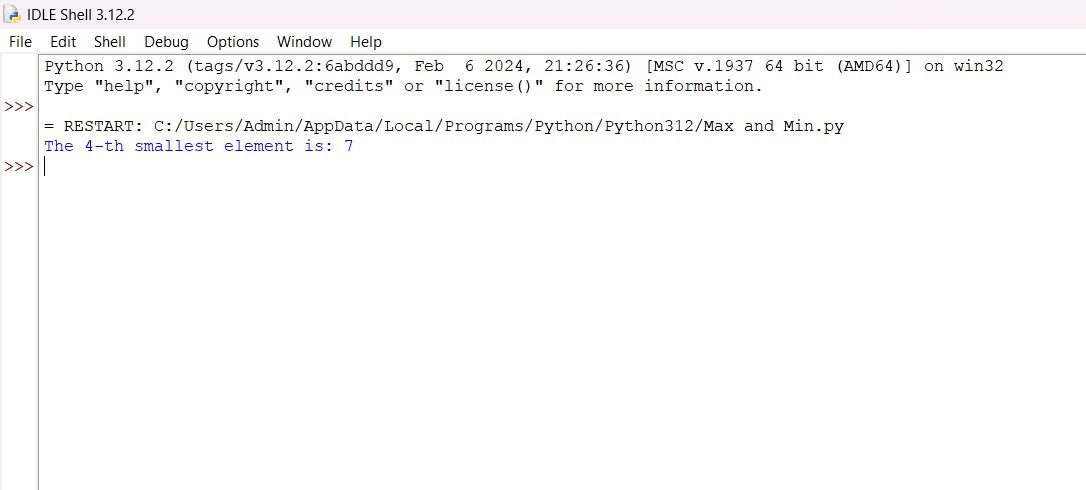
return median\_of\_medians(high, k - len(low) - pivot\_count)

arr = [12, 3, 5, 7, 4, 19, 26]

k = 3

result = median\_of\_medians(arr, k)

print(f"The {k+1}-th smallest element is: {result}")



9. Median of Medians

from itertools import combinations

def subset\_sums(nums):

"""Generate all possible subset sums of a list of numbers."""

sums = []

for r in range(len(nums) + 1):

for comb in combinations(nums, r):

sums.append(sum(comb))

return sums

def meet\_in\_the\_middle(nums, target):

n = len(nums)

left\_part = nums[:n // 2]

right\_part = nums[n // 2:]

left\_sums = subset\_sums(left\_part)

right\_sums = subset\_sums(right\_part)

right\_sums\_set = set(right\_sums)

for s in left\_sums:

if target - s in right\_sums\_set:

return True

return False

nums = [3, 34, 4, 12, 5, 2]

target = 9

result = meet\_in\_the\_middle(nums, target)

print(f"Is there a subset that sums up to {target}? {'Yes' if result else 'No'}")

