**Create virtual environment**

>python –m venv myenv

**Install matplot lib in terminal**

>pip install matplotlib

1. Linear Search

import time

import matplotlib.pyplot as plt

def linear\_search(arr, x):

    for i in range(len(arr)):

        if arr[i] == x:

         return i

    return -1

# Manually enter the array

arr = [int(x) for x in input("Enter the array elements separated by spaces: ").split()]

x = int(input("Enter the element to search: "))

result=linear search(arr,x)

if result != -1:

  print("Element is present at index", str(result))

else:

  print("Element is not present in array")

def measure\_time\_complexity(arr, x):

    start\_time = time.time()

    linear\_search(arr, x)

    end\_time = time.time()

    return end\_time - start\_time

# Generate different array sizes and measure time

array\_sizes = [10, 100, 1000, 50000]

execution\_times = []

for size in array\_sizes:

    arr = [i for i in range(size)]

    x = size - 1  # Worst-case scenario

    time\_taken = measure\_time\_complexity(arr, x)

    execution\_times.append(time\_taken)

total\_time = sum(execution\_times)

print(f"Total time complexity across all array sizes: {total\_time:.4f} seconds")

# Plot the results

plt.plot(array\_sizes, execution\_times)

plt.xlabel("Array Size")

plt.ylabel("Execution Time (seconds)")

plt.title("Time Complexity of Linear Search")

plt.show()

output:

Enter the array elements separated by spaces: 3 5 7 2 8

Enter the element to search: 5

Element is present at index 1

Total time complexity across all array sizes: 0.0041 seconds

2)Bubble Sort

import time

import matplotlib.pyplot as plt

def bubble\_sort(arr):

    n = len(arr)

    for i in range(n):

        for j in range(0, n-i-1):

            if arr[j] > arr[j+1]:

                arr[j], arr[j+1] = arr[j+1], arr[j]

    return arr

arr = [int(x) for x in input("Enter the array elements separated by spaces: ").split()]

result=bubble\_sort(arr)

print(result)

# Generate different array sizes

array\_sizes = [10, 100, 1000, 5000]  # Adjusted for practical execution

execution\_times = []

# Measure time complexity for different array sizes

for size in array\_sizes:

    # Create a reversed array (worst case)

    arr = [i for i in range(size, 0, -1)]

    start\_time = time.time()

    bubble\_sort(arr.copy())  # Avoid modifying the original array

    end\_time = time.time()

    execution\_times.append(end\_time - start\_time)

total\_time = sum(execution\_times)

print(f"Total time complexity across all array sizes: {total\_time:.4f} seconds")

# Plot the results

plt.plot(array\_sizes, execution\_times, marker='o')

plt.xlabel("Array Size")

plt.ylabel("Execution Time (seconds)")

plt.title("Time Complexity of Bubble Sort (Worst Case)")

plt.grid(True)

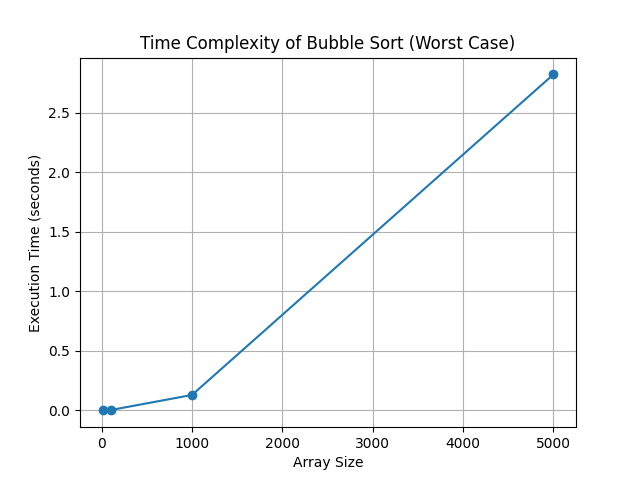
plt.show()

output

>Enter the array elements separated by spaces: 4 7 9 2 3

[2, 3, 4, 7, 9]

Total time complexity across all array sizes: 2.9531 seconds



**3)Insertion Sort**

import time

import matplotlib.pyplot as plt

def insertion\_sort(arr):

    for i in range(1, len(arr)):

        key = arr[i]

        j = i - 1

        # Move elements of arr[0..i-1] that are greater than key

        while j >= 0 and arr[j] > key:

            arr[j + 1] = arr[j]

            j -= 1

        arr[j + 1] = key

    return arr

arr = [int(x) for x in input("Enter the array elements separated by spaces: ").split()]

result=insertion\_sort(arr)

print(result)

# Generate different array sizes

array\_sizes = [10, 100, 1000, 5000]  # Adjusted for practical execution

execution\_times = []

# Measure time complexity for different array sizes

for size in array\_sizes:

    # Create a reversed array (worst case)

    arr = [i for i in range(size, 0, -1)]

    start\_time = time.time()

    insertion\_sort(arr.copy())  # Avoid modifying the original array

    end\_time = time.time()

    execution\_times.append(end\_time - start\_time)

total\_time = sum(execution\_times)

print(f"Total time complexity: {total\_time:.4f} seconds")

# Plot the results

plt.plot(array\_sizes, execution\_times, marker='o')

plt.xlabel("Array Size")

plt.ylabel("Execution Time (seconds)")

plt.title("Time Complexity of insertion Sort (Worst Case)")

plt.grid(True)

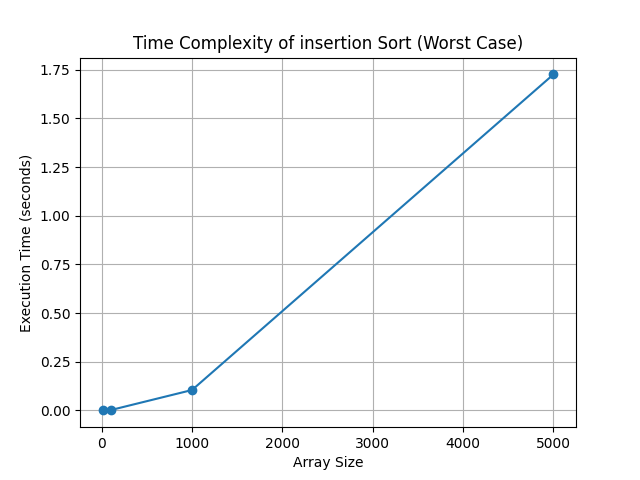
plt.show()

output

>Enter the array elements separated by spaces: 4 6 8 2 9

[2, 4, 6, 8, 9]

Total time complexity: 1.7832 seconds



**4)Selection sort**

import time

import matplotlib.pyplot as plt

def selection\_sort(arr):

    n = len(arr)

    for i in range(n):

        # Find the minimum element in the remaining unsorted array

        min\_index = i

        for j in range(i+1, n):

            if arr[j] < arr[min\_index]:

                min\_index = j

        arr[i], arr[min\_index] = arr[min\_index], arr[i]  # Swap

    return arr

arr = [int(x) for x in input("Enter the array elements separated by spaces: ").split()]

result=selection\_sort(arr)

print(result)

# Generate different array sizes

array\_sizes = [10, 100, 1000, 5000]  # Adjusted for practical execution

execution\_times = []

# Measure time complexity for different array sizes

for size in array\_sizes:

    # Create a reversed array (worst case)

    arr = [i for i in range(size, 0, -1)]

    start\_time = time.time()

    selection\_sort(arr.copy())  # Avoid modifying the original array

    end\_time = time.time()

    execution\_times.append(end\_time - start\_time)

total\_time = sum(execution\_times)

print(f"Total time complexity sizes: {total\_time:.4f} seconds")

# Plot the results

plt.plot(array\_sizes, execution\_times, marker='o')

plt.xlabel("Array Size")

plt.ylabel("Execution Time (seconds)")

plt.title("Time Complexity of selection Sort (Worst Case)")

plt.grid(True)

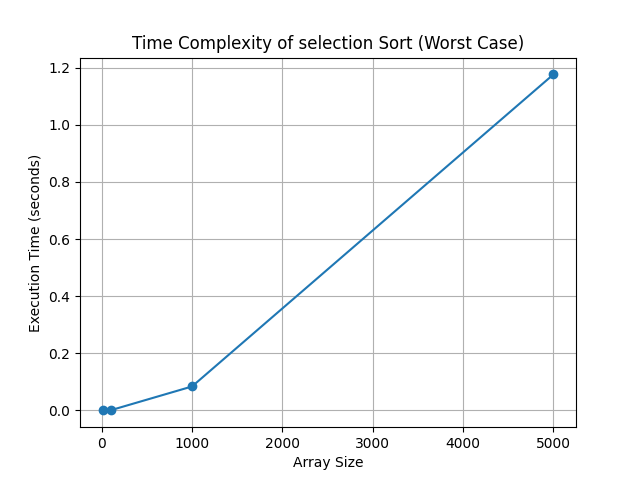
plt.show()

**output**

> Enter the array elements separated by spaces: 4 7 8 2 9

[2, 4, 7, 8, 9]

Total time complexity sizes: 1.2612 seconds



5)Binary Search

import matplotlib.pyplot as plt

import time

import sys

#binary search on sorted array

def binary\_search(arr, target):

    low, high = 0, len(arr) - 1

    while low <= high:

        mid = (low + high) // 2  # Find the middle index

        if arr[mid] == target:

            return mid  # Target found, return its index

        elif arr[mid] < target:

            low = mid + 1  # Discard the left half

        else:

            high = mid - 1  # Discard the right half

    return -1  # Target not found

# Input and Test the Binary Search

arr = [int(x) for x in input("Enter a sorted list of numbers separated by spaces: ").split()]

arr\_copy=arr[:]

print(arr\_copy)

arr\_copy.sort()

if(arr\_copy != arr):

    print("array is not sorted")

    sys.exit(1)

else:

   target = int(input("Enter the number to search for: "))

   result = binary\_search(arr, target)

   if result != -1:

    print(f"Element found at index {result}.")

   else:

    print("Element not found.")

'''

def calculate\_array\_space(arr):

    total\_size = sys.getsizeof(arr)  # Base size of the array object

    total\_size += sum(sys.getsizeof(item) for item in arr)  # Size of all elements

    return total\_size

'''

# Evaluate and Plot Time Complexity

array\_sizes = [10, 100, 1000, 5000,10000,20000]

execution\_times = []

for size in array\_sizes:

    test\_array = list(range(size, 0, -1))  # Create reverse-sorted array

    start\_time = time.time()

    binary\_search(test\_array,target)

    end\_time = time.time()

    execution\_times.append(end\_time - start\_time)

# Display Time Complexity Results

total\_time = sum(execution\_times)

print(f"Total time complexity sizes: {total\_time:.4f} seconds")

#space\_used=calculate\_array\_space(arr)

#print("space used=",space\_used)

plt.plot(array\_sizes, execution\_times, marker='o', color='b', label='Execution Time')

plt.xlabel("Array Size")

plt.ylabel("Execution Time (seconds)")

plt.title("Time Complexity of Binary Sort")

plt.legend()

plt.grid(True)

plt.show()

**output**

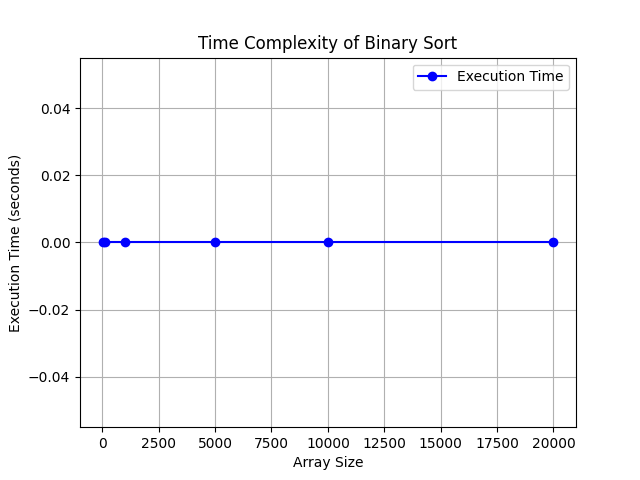
>Enter a sorted list of numbers separated by spaces: 5 6 7 8 9

[5, 6, 7, 8, 9]

Enter the number to search for: 8

Element found at index 3.

Total time complexity sizes: 0.0000 seconds



6) Merge Sort:

from matplotlib import pyplot as plt

import time

import sys

# Merge Sort Function

def mergesort(arr):

    if len(arr) > 1:

        mid = len(arr) // 2

        left\_half = arr[:mid]

        right\_half = arr[mid:]

        mergesort(left\_half)

        mergesort(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

# Input and Sort

arr = [int(x) for x in input("Enter the array elements separated by spaces: ").split()]

print(f"Original Array: {arr}")

mergesort(arr)

print(f"Sorted Array: {arr}")

# Evaluate and Plot Time Complexity

array\_sizes = [10, 100, 1000, 5000]  # Array sizes for testing

execution\_times = []

for size in array\_sizes:

    test\_array = [i for i in range(size, 0, -1)]  # Worst-case input: reverse-sorted array

    start\_time = time.time()

    mergesort(test\_array)

    end\_time = time.time()

    execution\_times.append(end\_time - start\_time)

# Display Time Complexity Results

total\_time = sum(execution\_times)

print(f"Total time complexity sizes: {total\_time:.4f} seconds")

# Plotting

plt.plot(array\_sizes, execution\_times, marker='o', color='b', label='Execution Time')

plt.xlabel("Array Size")

plt.ylabel("Execution Time (seconds)")

plt.title("Time Complexity of Merge Sort (Worst Case)")

plt.legend()

plt.grid(True)

plt.show()

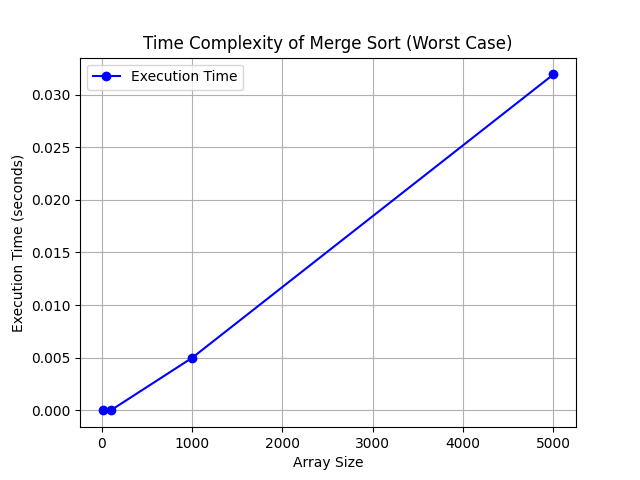
output

Enter the array elements separated by spaces: 5 7 8 3 4

Original Array: [5, 7, 8, 3, 4]

Sorted Array: [3, 4, 5, 7, 8]

Total time complexity sizes: 0.0369 seconds



7 )**Quick Sort**

import random

import matplotlib.pyplot as plt

import time

def quick\_sort(arr):

    if len(arr) <= 1:

        return arr  # Base case: return if the list has one or no elements

    pivot = arr[len(arr) // 2]  # Choosing the middle element as the pivot

    left = [x for x in arr if x < pivot]  # Elements smaller than pivot

    middle = [x for x in arr if x == pivot]  # Elements equal to pivot

    right = [x for x in arr if x > pivot]  # Elements greater than pivot

    return quick\_sort(left) + middle + quick\_sort(right)  # Recursively sorting and merging

# Example usage

arr = [int(x) for x in input("Enter the array elements separated by spaces: ").split()]

print(arr)

sorted\_arr = quick\_sort(arr)

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

sizes = [100, 500, 1000, 5000, 50000]

quicksort\_times = []

#mergesort\_times = []

for size in sizes:

    arr1 = [random.randint(0, 100) for \_ in range(size)]

    arr2 = list(arr1)

    quick\_arr = list(arr1)  # Copy list

    start = time.time()

    quick\_sort(arr2)

    quicksort\_times.append(time.time() - start)

plt.plot(sizes, quicksort\_times, label="Quicksort", marker='o')

#plt.plot(sizes, mergesort\_times, label="Mergesort", marker='s')

plt.xlabel("Input Size")

plt.ylabel("Time (seconds)")

plt.title("Quicksort vs Mergesort Performance")

plt.legend()

plt.grid()

plt.show()

**Enter the array elements separated by spaces: 2 8 3 4 1 9**

**[2, 8, 3, 4, 1, 9]**

**Sorted array: [1, 2, 3, 4, 8, 9]**

