DAA Lab 2

Code for linear search:

def linear\_search(arr, target):

for index, value in enumerate(arr):

if value == target:

return index

return -1

# Sample Test Cases

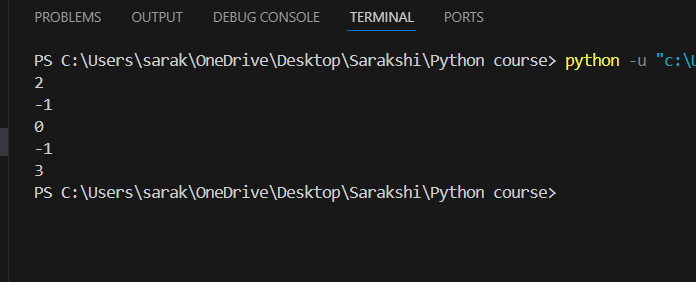
print(linear\_search([2, 4, 6, 8, 10], 6)) # Output: 2

print(linear\_search([1, 3, 5, 7, 9], 4)) # Output: -1

print(linear\_search([10, 20, 30, 40], 10)) # Output: 0

print(linear\_search([5, 15, 25, 35], 50)) # Output: -1

print(linear\_search([3, 6, 9, 12], 12)) # Output: 3



Code for binary search:

def recursive\_binary\_search(arr, left, right, target):

if left > right:

return -1

# Calculate the middle index

mid = (left + right) // 2

# Compare the middle element with the target

if arr[mid] == target:

return mid

elif arr[mid] > target:

return recursive\_binary\_search(arr, left, mid - 1, target)

else:

return recursive\_binary\_search(arr, mid + 1, right, target)

def binary\_search(arr, target):

return recursive\_binary\_search(arr, 0, len(arr) - 1, target)

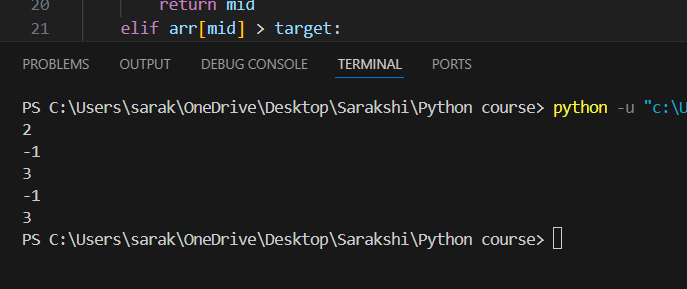
print(binary\_search([1, 3, 5, 7, 9], 5)) # Output: 2

print(binary\_search([2, 4, 6, 8, 10], 3)) # Output: -1

print(binary\_search([1, 2, 3, 4, 5], 4)) # Output: 3

print(binary\_search([1, 2, 3, 4, 5], 6)) # Output: -1

print(binary\_search([2, 4, 6, 8, 10], 8)) # Output: 3



Conclusion:

In this exercise, we explored two fundamental search algorithms: Linear Search and Binary Search. Linear Search is simple to implement and works on both sorted and unsorted arrays, but it may be inefficient for large datasets due to its O(n) time complexity. Binary Search is more efficient with a time complexity of O(log n) but requires the array to be sorted.