**Bubble Sort**

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

# Last i elements are already sorted, no need to check them

for j in range(0, n - i - 1):

# Swap if the element found is greater than the next element

if arr[j] > arr[j + 1]:

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

# Get user input for the array

input\_str = input("Enter space-separated integers for the array: ")

arr = list(map(int, input\_str.split()))

# Display the unsorted array

print("Unsorted array:", arr)

# Apply Bubble Sort

bubble\_sort(arr)

# Display the sorted array

print("Sorted array:", arr)

**Selection Sort**

def selection\_sort(arr):

n = len(arr)

for i in range(n):

# Display the unsorted part of the array

print(f"Pass {i+1}:")

print("Unsorted array:", arr[i:])

# Find the minimum element in the unsorted part of the array

min\_index = i

for j in range(i + 1, n):

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

min\_index = j

# Display the swapping process

print(f"Swapping {arr[i]} and {arr[min\_index]}")

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

# Display the sorted part of the array

print("Sorted array:", arr[:i+1])

print()

# Get user input for the array

input\_str = input("Enter space-separated integers for the array: ")

arr = list(map(int, input\_str.split()))

# Apply Selection Sort

selection\_sort(arr)

# Display the final sorted array

print("Sorted array:", arr)

**Insertion Sort**

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 to one position ahead of their current position

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

arr[j + 1] = arr[j]

j -= 1

arr[j + 1] = key

# Display the current state of the array after each iteration

print(f"Iteration {i}: {arr}")

# Get user input for the array

input\_str = input("Enter space-separated integers for the array: ")

arr = list(map(int, input\_str.split()))

# Display the unsorted array

print("Unsorted array:", arr)

# Apply Insertion Sort and display each step

insertion\_sort(arr)

# Display the sorted array

print("Sorted array:", arr)

**Quick Sort**

def quick\_sort(arr):

if len(arr) <= 1:

return arr

else:

pivot = arr[0]

less\_than\_pivot = [x for x in arr[1:] if x <= pivot]

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

sorted\_partition = quick\_sort(less\_than\_pivot) + [pivot] + quick\_sort(greater\_than\_pivot)

print(f"Sorted partition: {sorted\_partition}")

return sorted\_partition

# Get user input for the array

input\_str = input("Enter space-separated integers for the array: ")

arr = list(map(int, input\_str.split()))

# Display the unsorted array

print("Unsorted array:", arr)

# Apply Quick Sort and display each step

sorted\_arr = quick\_sort(arr)

# Display the sorted array

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

**Merge Sort**

def merge\_sort(arr):

print(f"Splitting: {arr}")

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

# Recursive call on the left and right halves

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

# Merge the sorted halves

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

# Check for any remaining elements in the left and right halves

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

print(f"Merging: {arr}")

# Get user input for the array

input\_str = input("Enter space-separated integers for the array: ")

arr = list(map(int, input\_str.split()))

# Display the unsorted array

print("Unsorted array:", arr)

# Apply Merge Sort and display each step

merge\_sort(arr)

# Display the sorted array

print("Sorted array:", arr)

**Binary Search**

def binary\_search(arr, target):

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

while left <= right:

mid = left + (right - left) // 2

# Check if target is present at the mid

if arr[mid] == target:

return mid

# If target is greater, ignore the left half

elif arr[mid] < target:

left = mid + 1

# If target is smaller, ignore the right half

else:

right = mid - 1

# If the target is not present in the array

return -1

# Example usage:

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

target = 2

result = binary\_search(arr, target)

if result != -1:

print(f"Element {target} is present at index {result}.")

else:

print(f"Element {target} is not present in the array.")

**Linear Search**

def linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i # Return the index if the target is found

return -1 # Return -1 if the target is not present in the array

# Example usage:

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

target = 7

result = linear\_search(arr, target)

if result != -1:

print(f"Element {target} is present at index {result}.")

else:

print(f"Element {target} is not present in the array.")

**Ternary Search**

def ternary\_search(arr, target):

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

while left <= right:

# Divide the array into three parts

mid1 = left + (right - left) // 3

mid2 = right - (right - left) // 3

# Check if the target is present at mid1 or mid2

if arr[mid1] == target:

return mid1

elif arr[mid2] == target:

return mid2

# If the target is in the left third

elif target < arr[mid1]:

right = mid1 - 1

# If the target is in the right third

elif target > arr[mid2]:

left = mid2 + 1

# If the target is in the middle third

else:

left = mid1 + 1

right = mid2 - 1

return -1 # Return -1 if the target is not present in the array

# Example usage:

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

target = 5

result = ternary\_search(arr, target)

if result != -1:

print(f"Element {target} is present at index {result}.")

else:

print(f"Element {target} is not present in the array.")