1. def find\_min\_max(arr):

if not arr:

return None, None

min\_val = max\_val = arr[0]

for num in arr:

if num < min\_val:

min\_val = num

if num > max\_val:

max\_val = num

return min\_val, max\_val

arr1 = [5, 7, 3, 4, 9, 12, 6, 2]

arr2 = [1, 3, 5, 7, 9, 11, 13, 15, 17]

arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]

print("Test Case 1:", find\_min\_max(arr1))

print("Test Case 2:", find\_min\_max(arr2))

print("Test Case 3:", find\_min\_max(arr3))

Output: Test Case 1: (2, 12)

Test Case 2: (1, 17)

Test Case 3: (12, 67)

2. def find\_min\_max\_sorted(arr):

if not arr:

return None, None

return arr[0], arr[-1]

arr1 = [2, 4, 6, 8, 10, 12, 14, 18]

arr2 = [11, 13, 15, 17, 19, 21, 23, 35, 37]

arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]

print("Test Case 1:", find\_min\_max\_sorted(arr1))

print("Test Case 2:", find\_min\_max\_sorted(arr2))

print("Test Case 3:", find\_min\_max\_sorted(sorted(arr3)))

Output: Test Case 1: (2, 18)

Test Case 2: (11, 37)

Test Case 3: (12, 67)

3. def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

merge\_sort(left)

merge\_sort(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

arr1 = [31, 23, 35, 27, 11, 21, 15, 28]

arr2 = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]

merge\_sort(arr1)

merge\_sort(arr2)

print("Test Case 1:", arr1)

print("Test Case 2:", arr2)

Output: Test Case 1: [11, 15, 21, 23, 27, 28, 31, 35]

Test Case 2: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67].

4. def merge\_sort\_with\_count(arr):

comparisons = 0

def merge\_sort\_helper(arr):

nonlocal comparisons

if len(arr) > 1:

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

merge\_sort\_helper(left)

merge\_sort\_helper(right)

i = j = k = 0

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

comparisons += 1

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

merge\_sort\_helper(arr)

return arr, comparisons

arr1 = [12, 4, 78, 23, 45, 67, 89, 1]

arr2 = [38, 27, 43, 3, 9, 82, 10]

sorted\_arr1, comparisons1 = merge\_sort\_with\_count(arr1)

sorted\_arr2, comparisons2 = merge\_sort\_with\_count(arr2)

print("Test Case 1:", sorted\_arr1, "Comparisons:", comparisons1)

print("Test Case 2:", sorted\_arr2, "Comparisons:", comparisons2)

Output: Test Case 1: [1, 4, 12, 23, 45, 67, 78, 89] Comparisons: 16

Test Case 2: [3, 9, 10, 27, 38, 43, 82] Comparisons: 13.

5. def quick\_sort(arr):

if len(arr) <= 1:

return arr

pivot = arr[0]

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

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

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

arr1 = [10, 16, 8, 12, 15, 6, 3, 9, 5]

arr2 = [12, 4, 78, 23, 45, 67, 89, 1]

arr3 = [38, 27, 43, 3, 9, 82, 10]

print("Test Case 1:", quick\_sort(arr1))

print("Test Case 2:", quick\_sort(arr2))

print("Test Case 3:", quick\_sort(arr3))

Output: Test Case 1: [3, 5, 6, 8, 9, 10, 12, 15, 16]

Test Case 2: [1, 4, 12, 23, 45, 67, 78, 89]

Test Case 3: [3, 9, 10, 27, 38, 43, 82]

6. def quick\_sort\_middle(arr):

if len(arr) <= 1:

return arr

pivot\_index = len(arr) // 2

pivot = arr[pivot\_index]

less = [x for i, x in enumerate(arr) if x <= pivot and i != pivot\_index]

greater = [x for i, x in enumerate(arr) if x > pivot and i != pivot\_index]

return quick\_sort\_middle(less) + [pivot] + quick\_sort\_middle(greater)

arr1 = [19, 72, 35, 46, 58, 91, 22, 31]

arr2 = [31, 23, 35, 27, 11, 21, 15, 28]

arr3 = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]

print("Test Case 1:", quick\_sort\_middle(arr1))

print("Test Case 2:", quick\_sort\_middle(arr2))

print("Test Case 3:", quick\_sort\_middle(arr3))

Output: Test Case 1: [19, 22, 31, 35, 46, 58, 72, 91]

Test Case 2: [11, 15, 21, 23, 27, 28, 31, 35]

Test Case 3: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]

7. def binary\_search(arr, target):

comparisons = 0

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

while left <= right:

comparisons += 1

mid = (left + right) // 2

if arr[mid] == target:

return mid, comparisons

elif arr[mid] < target:

left = mid + 1

else:

right = mid - 1

return -1, comparisons

arr1 = [5, 10, 15, 20, 25, 30, 35, 40, 45]

arr2 = [10, 20, 30, 40, 50, 60]

arr3 = [21, 32, 40, 54, 65, 76, 87]

index1, comparisons1 = binary\_search(arr1, 20)

index2, comparisons2 = binary\_search(arr2, 50)

index3, comparisons3 = binary\_search(arr3, 32)

print("Test Case 1:", index1, "Comparisons:", comparisons1)

print("Test Case 2:", index2, "Comparisons:", comparisons2)

print("Test Case 3:", index3, "Comparisons:", comparisons3)

Output: Test Case 1: 3 Comparisons: 4

Test Case 2: 4 Comparisons: 2

Test Case 3: 1 Comparisons: 2

8. def binary\_search\_with\_steps(arr, target):

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

steps = []

while left <= right:

mid = (left + right) // 2

steps.append(mid)

if arr[mid] == target:

return mid, steps

elif arr[mid] < target:

left = mid + 1

else:

right = mid - 1

return -1, steps

# Test Cases

arr1 = [3, 9, 14, 19, 25, 31, 42, 47, 53]

arr2 = [13, 19, 24, 29, 35, 41, 42]

arr3 = [20, 40, 60, 80, 100, 120]

index1, steps1 = binary\_search\_with\_steps(arr1, 31)

index2, steps2 = binary\_search\_with\_steps(arr2, 42)

index3, steps3 = binary\_search\_with\_steps(arr3, 60)

print("Test Case 1:", index1, "Steps:", steps1)

print("Test Case 2:", index2, "Steps:", steps2)

print("Test Case 3:", index3, "Steps:", steps3)

Output: Output: 5, [4, 6, 5]

Output: 6, [3, 5, 6]

Output: 2, [2]

9. import math

def k\_closest\_points(points, k):

points.sort(key=lambda x: math.sqrt(x[0]\*\*2 + x[1]\*\*2))

return points[:k]

# Test Cases

points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]

points2 = [[1, 3], [-2, 2]]

points3 = [[3, 3], [5, -1], [-2, 4]]

print("Test Case 1:", k\_closest\_points(points1, 2))

print("Test Case 2:", k\_closest\_points(points2, 1))

print("Test Case 3:", k\_closest\_points(points3, 2))

Output: Test Case 1: [[0, 1], [-2, 2]]

Test Case 2: [[-2, 2]]

Test Case 3: [[3, 3], [-2, 4]]

10. from itertools import product

def count\_tuples(A, B, C, D):

count = 0

for a, b, c, d in product(A, B, C, D):

if a + b + c + d == 0:

count += 1

return count

# Test Cases

A1, B1, C1, D1 = [1, 2], [-2, -1], [-1, 2], [0, 2]

A2, B2, C2, D2 = [0], [0], [0], [0]

print("Test Case 1:", count\_tuples(A1, B1, C1, D1))

print("Test Case 2:", count\_tuples(A2, B2, C2, D2))

Output: Test Case 1: 2

Test Case 2: 1