main.py

Share Run

```python
def find_min_max(arr):

    min_val = min(arr)
    max_val = max(arr)
    return min_val, max_val

arr1 = [5, 7, 3, 4, 9, 12, 6, 2]
min_val1, max_val1 = find_min_max(arr1)
print(f"Input: {arr1} \nOutput: Min = {min_val1}, Max =
    {max_val1}")

arr2 = [1, 3, 5, 7, 9, 11, 13, 15, 17]
min_val2, max_val2 = find_min_max(arr2)
print(f"Input: {arr2} \nOutput: Min = {min_val2}, Max =
    {max_val2}")

arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
min_val3, max_val3 = find_min_max(arr3)
print(f"Input: {arr3} \nOutput: Min = {min_val3}, Max =
    {max_val3}")

```

Output

```
Input: [5, 7, 3, 4, 9, 12, 6, 2]
Output: Min = 2, Max = 12
Input: [1, 3, 5, 7, 9, 11, 13, 15, 17]
Output: Min = 1, Max = 17
Input: [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
Output: Min = 12, Max = 67

=== Code Execution Successful ===
```

main.py

Share Run

```python
def find_min_max(arr):

    min_val = arr[0]
    max_val = arr[-1]
    return min_val, max_val

arr1 = [2, 4, 6, 8, 10, 12, 14, 18]
min_val1, max_val1 = find_min_max(arr1)
print(f"Input: {arr1} \nOutput: Min = {min_val1}, Max =
    {max_val1}")

arr2 = [11, 13, 15, 17, 19, 21, 23, 35, 37]
min_val2, max_val2 = find_min_max(arr2)
print(f"Input: {arr2} \nOutput: Min = {min_val2}, Max =
    {max_val2}")

arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
min_val3, max_val3 = find_min_max(sorted(arr3))
print(f"Input: {arr3} (Sorted: {sorted(arr3)}) \nOutput: Min =
    {min_val3}, Max = {max_val3}")

```

Output

Clear

```
Input: [2, 4, 6, 8, 10, 12, 14, 18]
Output: Min = 2, Max = 18
Input: [11, 13, 15, 17, 19, 21, 23, 35, 37]
Output: Min = 11, Max = 37
Input: [22, 34, 35, 36, 43, 67, 12, 13, 15, 17] (Sorted: [12, 13, 15,
    17, 22, 34, 35, 36, 43, 67])
Output: Min = 12, Max = 67

=== Code Execution Successful ===
```

main.py

Share Run

```python
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]
            i += 1
```

Output

```
Sorted array (Test Case 1): [11, 15, 21, 23, 27, 28, 31, 35]

=== Code Execution Successful ===
```

main.py

Share Run

Output Clear

```python
def merge_sort(arr):
    comparison_count = [0]
    def merge(arr, left_half, right_half):
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            comparison_count[0] += 1
            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
```

```
Sorted array: [1, 4, 12, 23, 45, 67, 78, 89], Comparisons made: 16

=== Code Execution Successful ===
```

main.py

Share Run

```python
def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        print(f"Array after partitioning with pivot {arr[pi]}:
            {arr}")
        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)
def partition(arr, low, high):
    pivot = arr[low]
    i = low + 1
    j = high
    while True:
        while i <= j and arr[i] <= pivot:
            i += 1
        while i <= j and arr[j] > pivot:
            j -= 1
        if i <= j:
            arr[i], arr[j] = arr[j], arr[i]
        else:
            break
    arr[low], arr[j] = arr[j], arr[low]
    return j
arr = [10, 16, 8, 12, 15, 6, 3, 9, 5]
```

Output Clear

```
Initial array: [10, 16, 8, 12, 15, 6, 3, 9, 5]
Array after partitioning with pivot 10: [6, 5, 8, 9, 3, 10, 15, 12, 16]
Array after partitioning with pivot 6: [3, 5, 6, 9, 8, 10, 15, 12, 16]
Array after partitioning with pivot 3: [3, 5, 6, 9, 8, 10, 15, 12, 16]
Array after partitioning with pivot 9: [3, 5, 6, 8, 9, 10, 15, 12, 16]
Array after partitioning with pivot 15: [3, 5, 6, 8, 9, 10, 12, 15, 16]
Sorted array: [3, 5, 6, 8, 9, 10, 12, 15, 16]

=== Code Execution Successful ===
```

main.py

Share Run

```python
def quick_sort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
        print(f"Array after partitioning with pivot {arr[pi]}:
            {arr}")
        quick_sort(arr, low, pi - 1)
        quick_sort(arr, pi + 1, high)
def partition(arr, low, high):
    mid = (low + high) // 2
    pivot = arr[mid]
    arr[mid], arr[low] = arr[low], arr[mid]
    i = low + 1
    j = high
    while True:
        while i <= j and arr[i] <= pivot:
            i += 1
        while i <= j and arr[j] > pivot:
            j -= 1
        if i <= j:
            arr[i], arr[j] = arr[j], arr[i]
        else:
            break
    arr[low], arr[j] = arr[j], arr[low]
```

Output Clear

```
Initial array: [19, 72, 35, 46, 58, 91, 22, 31]
Array after partitioning with pivot 46: [22, 31, 35, 19, 46, 91, 58,
    72]
Array after partitioning with pivot 31: [19, 22, 31, 35, 46, 91, 58,
    72]
Array after partitioning with pivot 19: [19, 22, 31, 35, 46, 91, 58,
    72]
Array after partitioning with pivot 58: [19, 22, 31, 35, 46, 58, 91,
    72]
Array after partitioning with pivot 91: [19, 22, 31, 35, 46, 58, 72,
    91]
Sorted array: [19, 22, 31, 35, 46, 58, 72, 91]

=== Code Execution Successful ===
```

main.py

Share Run

```python
def binary_search(arr, low, high, key):
    comparison_count = 0
    while low <= high:
        comparison_count += 1
        mid = (low + high) // 2
        if arr[mid] == key:
            return mid, comparison_count
        elif arr[mid] < key:
            low = mid + 1
        else:
            high = mid - 1
        return -1, comparison_count
arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]
key = 20
index, comparisons = binary_search(arr, 0, len(arr) - 1, key)
print(f"Element found at index: {index}, Comparisons made:
    {comparisons}")

```

Output

```
Element found at index: -1, Comparisons made: 1

=== Code Execution Successful ===
```

main.py

Share Run

Output

```python
import heapq

def k_closest_points(points, k):
    max_heap = []
    for point in points:
        distance = point[0]**2 + point[1]**2
        heapq.heappush(max_heap, (-distance, point))
        if len(max_heap) > k:
            heapq.heappop(max_heap)
    closest_points = [point for _, point in max_heap]
    return closest_points
points = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k = 2
k_closest = k_closest_points(points, k)
print(k_closest)

```

```
[[-2, 2], [0, 1]]

=== Code Execution Successful ===
```

main.py

Share Run

Output

```python
from collections import Counter
def four_sum_count(A, B, C, D):
    sum_ab = Counter(a + b for a in A for b in B)
    count = 0
    for c in C:
        for d in D:
            count += sum_ab[-(c + d)]
        return count
A = [1, 2]
B = [-2, -1]
C = [-1, 2]
D = [0, 2]
result = four_sum_count(A, B, C, D)
print(result)

```

```
2

=== Code Execution Successful ===
```