1.

main.py

```python
def find_min_max(arr):
    return min(arr), max(arr)
a = [5, 7, 3, 4, 9, 12, 6, 2]
min_val, max_val = find_min_max(a)
print(f"Min = {min_val}, Max = {max_val}")

```

Output

```
Min = 2, Max = 12

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

2

main.py

```python
arr = [2, 4, 6, 8, 10, 12, 14, 18]
min_val, max_val = arr[0], arr[-1]
print(f"Min = {min_val}, Max = {max_val}")

```

Output

```
Min = 2, Max = 18

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

3

main.py

```python
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1
    return arr
arr = [31, 23, 35, 27, 11, 21, 15, 28]
sorted_arr = merge_sort(arr)
```

Output

```
[11, 15, 21, 23, 27, 28, 31, 35]

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

4

main.py

Share Run

Output

```
def merge_sort(arr):
    comparisons = 0

    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        comparisons += merge_sort(left_half)
        comparisons += 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
            comparisons += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1
```

```
Number of comparisons: 16
Sorted Array: [1, 4, 12, 23, 45, 67, 78, 89]

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

main.py

Share Run

Output

```
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1
            comparisons += 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

    return comparisons

arr = [12, 4, 78, 23, 45, 67, 89, 1]
comparisons = merge_sort(arr)

print(f"Number of comparisons: {comparisons}")
print(f"Sorted Array: {arr}")
```

```
Number of comparisons: 16
Sorted Array: [1, 4, 12, 23, 45, 67, 78, 89]

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

5

main.py | Share | Run | Output

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        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)
N = 9
a = [10, 16, 8, 12, 15, 6, 3, 9, 5]
sorted_array = quick_sort(a)
print(sorted_array)

```

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

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

6

main.py | Share | Run | Output

```
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)
input_array = [19, 72, 35, 46, 58, 91, 22, 31]
sorted_array = quick_sort(input_array)
print(sorted_array)

```

```
[19, 22, 31, 35, 46, 58, 72, 91]

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

7

main.py | Share | Run | Output

```
def binary_search(arr, x):
    low = 0
    high = len(arr) - 1
    count = 0
    while low <= high:
        mid = (low + high) // 2
        count += 1
        if arr[mid] < x:
            low = mid + 1
        elif arr[mid] > x:
            high = mid - 1
        else:
            return mid, count
    return -1, count
arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]
search_key = 20
index, comparisons = binary_search(arr, search_key)
if index != -1:
    print(f"Element found at index {index} with {comparisons} comparisons.")
else:
    print("Element not found.")

```

```
Element found at index 3 with 4 comparisons.

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

8

main.py

```python
def binary_search(arr, target):
    low = 0
    high = len(arr) - 1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1
N = 9
a = [3, 9, 14, 19, 25, 31, 42, 47, 53]
search_key = 31
print(binary_search(a, search_key))

```

Output

```
5

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

9

main.py

```python
import heapq
def kClosest(points, k):
    return heapq.nsmallest(k, points, key=lambda x: x[0]**2 + x[1]**2)
points1 = [[1, 3], [-2, 2], [5, 8], [0, 1]]
k1 = 2
print(kClosest(points1, k1))
points2 = [[1, 3], [-2, 2]]
k2 = 1
print(kClosest(points2, k2))

```

Output

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

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

10

main.py

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

```

Output

```
2

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