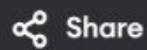


main.py



Run

Output

```
1 def find_min_max(arr):
2
3     min_val = min(arr)
4     max_val = max(arr)
5     return min_val, max_val
6
7 arr1 = [5, 7, 3, 4, 9, 12, 6, 2]
8 min_val1, max_val1 = find_min_max(arr1)
9 print(f"Input: {arr1} \nOutput: Min = {min_val1}, Max =
    {max_val1}")
10
11 arr2 = [1, 3, 5, 7, 9, 11, 13, 15, 17]
12 min_val2, max_val2 = find_min_max(arr2)
13 print(f"Input: {arr2} \nOutput: Min = {min_val2}, Max =
    {max_val2}")
14
15 arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
16 min_val3, max_val3 = find_min_max(arr3)
17 print(f"Input: {arr3} \nOutput: Min = {min_val3}, Max =
    {max_val3}")
18
```

```
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

Output

Clear

```
1 def find_min_max(arr):
2
3     min_val = arr[0]
4     max_val = arr[-1]
5     return min_val, max_val
6
7 arr1 = [2, 4, 6, 8, 10, 12, 14, 18]
8 min_val1, max_val1 = find_min_max(arr1)
9 print(f"Input: {arr1} \nOutput: Min = {min_val1}, Max =
    {max_val1}")
10
11 arr2 = [11, 13, 15, 17, 19, 21, 23, 35, 37]
12 min_val2, max_val2 = find_min_max(arr2)
13 print(f"Input: {arr2} \nOutput: Min = {min_val2}, Max =
    {max_val2}")
14
15 arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
16 min_val3, max_val3 = find_min_max(sorted(arr3))
17 print(f"Input: {arr3} (Sorted: {sorted(arr3)}) \nOutput: Min =
    {min_val3}, Max = {max_val3}")
18
```

Input: [2, 4, 6, 8, 10, 12, 14, 18]

Output: Min = 2, Max = 18

Input: [11, 13, 15, 17, 19, 21, 23, 35, 37]

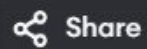
Output: Min = 11, Max = 37

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



Run

Output

```
1 def merge_sort(arr):
2     if len(arr) > 1:
3         mid = len(arr) // 2
4         left_half = arr[:mid]
5         right_half = arr[mid:]
6         merge_sort(left_half)
7         merge_sort(right_half)
8         i = j = k = 0
9         while i < len(left_half) and j < len(right_half):
10             if left_half[i] < right_half[j]:
11                 arr[k] = left_half[i]
12                 i += 1
13             else:
14                 arr[k] = right_half[j]
15                 j += 1
16             k += 1
17         while i < len(left_half):
18             arr[k] = left_half[i]
19             i += 1
20             k += 1
21         while j < len(right_half):
22             arr[k] = right_half[j]
23             j += 1
24             k += 1
```

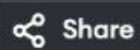
Sorted array (Test Case 1): [11, 15, 21, 23, 27, 28, 31, 35]

=== Code Execution Successful ===

1/3

219%

main.py



Share

Run

Output

Clear

```
1 def merge_sort(arr):
2     comparison_count = [0]
3     def merge(arr, left_half, right_half):
4         i = j = k = 0
5         while i < len(left_half) and j < len(right_half):
6             comparison_count[0] += 1
7             if left_half[i] < right_half[j]:
8                 arr[k] = left_half[i]
9                 i += 1
10            else:
11                arr[k] = right_half[j]
12                j += 1
13            k += 1
14
15        while i < len(left_half):
16            arr[k] = left_half[i]
17            i += 1
18            k += 1
19
20        while j < len(right_half):
21            arr[k] = right_half[j]
22            j += 1
23            k += 1
```

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

=== Code Execution Successful ===

main.py



Share

Run

```
1 def quick_sort(arr, low, high):
2     if low < high:
3         pi = partition(arr, low, high)
4         print(f"Array after partitioning with pivot {arr[pi]}:
           {arr}")
5         quick_sort(arr, low, pi - 1)
6         quick_sort(arr, pi + 1, high)
7 def partition(arr, low, high):
8     pivot = arr[low]
9     i = low + 1
10    j = high
11    while True:
12        while i <= j and arr[i] <= pivot:
13            i += 1
14        while i <= j and arr[j] > pivot:
15            j -= 1
16        if i <= j:
17            arr[i], arr[j] = arr[j], arr[i]
18        else:
19            break
20    arr[low], arr[j] = arr[j], arr[low]
21    return j
22 arr = [10, 16, 8, 12, 15, 6, 3, 9, 5]
```

Output

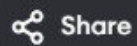
Clear

```
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

Output

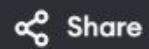
Clear

```
1 def quick_sort(arr, low, high):
2     if low < high:
3         pi = partition(arr, low, high)
4         print(f"Array after partitioning with pivot {arr[pi]}:
          {arr}")
5         quick_sort(arr, low, pi - 1)
6         quick_sort(arr, pi + 1, high)
7 def partition(arr, low, high):
8     mid = (low + high) // 2
9     pivot = arr[mid]
10    arr[mid], arr[low] = arr[low], arr[mid]
11    i = low + 1
12    j = high
13    while True:
14        while i <= j and arr[i] <= pivot:
15            i += 1
16        while i <= j and arr[j] > pivot:
17            j -= 1
18        if i <= j:
19            arr[i], arr[j] = arr[j], arr[i]
20        else:
21            break
22    arr[low], arr[i] = arr[i], arr[low]
```

```
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

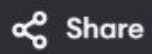
Output

```
1 def binary_search(arr, low, high, key):
2     comparison_count = 0
3     while low <= high:
4         comparison_count += 1
5         mid = (low + high) // 2
6         if arr[mid] == key:
7             return mid, comparison_count
8         elif arr[mid] < key:
9             low = mid + 1
10        else:
11            high = mid - 1
12        return -1, comparison_count
13 arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]
14 key = 20
15 index, comparisons = binary_search(arr, 0, len(arr) - 1, key)
16 print(f"Element found at index: {index}, Comparisons made:
17     {comparisons}")
```

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

=== Code Execution Successful ===

main.py



Run

Output

```
1 import heapq
2
3 def k_closest_points(points, k):
4     max_heap = []
5     for point in points:
6         distance = point[0]**2 + point[1]**2
7         heapq.heappush(max_heap, (-distance, point))
8         if len(max_heap) > k:
9             heapq.heappop(max_heap)
10    closest_points = [point for _, point in max_heap]
11    return closest_points
12 points = [[1, 3], [-2, 2], [5, 8], [0, 1]]
13 k = 2
14 k_closest = k_closest_points(points, k)
15 print(k_closest)
16
```

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

=== Code Execution Successful ===



main.py



Run

Output

```
1 from collections import Counter
2 def four_sum_count(A, B, C, D):
3     sum_ab = Counter(a + b for a in A for b in B)
4     count = 0
5     for c in C:
6         for d in D:
7             count += sum_ab[-(c + d)]
8     return count
9 A = [1, 2]
10 B = [-2, -1]
11 C = [-1, 2]
12 D = [0, 2]
13 result = four_sum_count(A, B, C, D)
14 print(result)
15
```

2

=== Code Execution Successful ===