```python
import heapq

def kClosest(points, k):
    heap = []
    for x, y in points:
        dist = -(x*x + y*y)
        if len(heap) == k:
            heapq.heappushpop(heap, (dist, x, y))
        else:
            heapq.heappush(heap, (dist, x, y))
    return [(x, y) for (dist, x, y) in heap]

# Example 1
points = [[1, 3], [-2, 2]]
k = 1
print(kClosest(points, k))

# Example 2
points = [[1.3], [-2.2], [5.8], [0, 1]]
k = 2
print(kClosest(points, k))
```

```
[1, 2, 5, 5, 6, 9]

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

```python
def findKthPositive(arr, k):
    missing = []
    num = 1
    while len(missing) < k:
        if num not in arr:
            missing.append(num)
        num += 1
    return missing[-1]
arr = [2, 3, 4, 7, 11]
k = 5
output = findKthPositive(arr, k)
print(output)
```

```
9

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

```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
arr = [5, 10, 15, 20, 25, 30, 35, 40, 45]
target = 20
result = binary_search(arr, target)

if result != -1:
    print(f"Element found at index: {result}")
else:
    print("Element not found in the array.")
```

```
Element found at index: 3

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

```python
def combinationSum(candidates, target):
    def backtrack(start, path, target):
        if target == 0:
            res.append(path[:])
            return
        for i in range(start, len(candidates)):
            if candidates[i] > target:
                continue
            path.append(candidates[i])
            backtrack(i, path, target - candidates[i])
            path.pop()
    candidates.sort()
    res = []
    backtrack(0, [], target)
    return res
candidates = [2, 3, 6, 7]
target = 7
print(combinationSum(candidates, target))
```

```
[[2, 2, 3], [7]]

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

```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
arr1 = [31, 23, 35, 27, 11, 21, 15, 28]
merge_sort(arr1)
print("Output:", arr1)
```

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

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

```python
import heapq
def kClosest(points, k):
    heap = []
    for x, y in points:
        dist = -(x*x + y*y)
        if len(heap) == k:
            heapq.heappushpop(heap, (dist, x, y))
        else:
            heapq.heappush(heap, (dist, x, y))
    return [(x, y) for (dist, x, y) in heap]
points = [[1, 3], [-2, 2]]
k = 1
print(kClosest(points, k))
```

```
[(-2, 2)]

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

```python
def graph_coloring(adjacency_list):
    colors = {}
    colored_regions = 0
    for region in adjacency_list:
        used_colors = set()
        for neighbor in adjacency_list[region]:
            if neighbor in colors:
                used_colors.add(colors[neighbor])
        for color in range(len(adjacency_list)):
            if color not in used_colors:
                colors[region] = color
                colored_regions += 1
                break
    return colored_regions
adjacency_list = {
    0: [1, 2, 3],
    1: [0, 2],
    2: [1, 3, 0],
    3: [2, 0]
}
max_regions_colored = graph_coloring(adjacency_list)
print("Maximum number of regions colored:", max_regions_colored)
```

```
Maximum number of regions colored: 4

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

```python
a = [11, 13, 15, 17, 19, 21, 23, 35, 37]
min_val = min(a)
max_val = max(a)
print("minimum value in an array is:",min_val)
print("Maximum value in an array is:",max_val)
```

```
minimum value in an array is: 11
Maximum value in an array is: 37

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

```python
def rob(nums):
    def rob_range(start, end):
        rob_next, rob_curr = 0, 0
        for i in range(start, end):
            rob_next, rob_curr = max(rob_curr + nums[i], rob_next), rob_next
        return rob_next
    if len(nums) == 1:
        return nums[0]
    return max(rob_range(0, len(nums) - 1), rob_range(1, len(nums)))
nums = [2, 3, 2]
print(rob(nums))
```

```
3

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

```python
import sys

def dijkstra(graph, source):
    n = len(graph)
    dist = [sys.maxsize] * n
    dist[source] = 0
    visited = [False] * n

    for _ in range(n):
        u = min_distance(dist, visited)
        visited[u] = True

        for v in range(n):
            if not visited[v] and graph[u][v] and dist[u] + graph[u][v] < dist[v]:
                dist[v] = dist[u] + graph[u][v]

    return dist

def min_distance(dist, visited):
    min_dist = sys.maxsize
    min_index = -1

    for v in range(len(dist)):
        if dist[v] < min_dist and not visited[v]:
            min_dist = dist[v]
            min_index = v

    return min_index

# Test Case
n = 5
graph = [[0, 10, 3, sys.maxsize, sys.maxsize],
         [sys.maxsize, 0, 1, 2, sys.maxsize],
         [sys.maxsize, 4, 0, 8, 2],
         [sys.maxsize, sys.maxsize, sys.maxsize, 0, 7],
         [sys.maxsize, sys.maxsize, sys.maxsize, 9, 0]]
source = 0

print(dijkstra(graph, source))
```

```
[0, 7, 3, 9, 5]


...Program finished with exit code 0
Press ENTER to exit console.
```