main.py

Share Run

```python
def selection_sort(lst):
    n = len(lst)
    for i in range(n):
        min_index = i
        for j in range(i+1, n):
            if lst[j] < lst[min_index]:
                min_index = j
        lst[i], lst[min_index] = lst[min_index], lst[i]
    return lst

# Test cases
test_cases = [
    [5, 2, 9, 1, 5, 6],
    [10, 8, 6, 4, 2],
    [1, 2, 3, 4, 5]
]

# Running test cases
for i, test in enumerate(test_cases, 1):
    print(f"Test Case {i}: Input: {test} | Output: {selection_sort(test)}")
```

Output

Clear

```
Test Case 1: Input: [5, 2, 9, 1, 5, 6] | Output: [1, 2, 5, 5, 6, 9]
Test Case 2: Input: [10, 8, 6, 4, 2] | Output: [2, 4, 6, 8, 10]
Test Case 3: Input: [1, 2, 3, 4, 5] | Output: [1, 2, 3, 4, 5]

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

main.py

Share Run

Output

```python
def bubble_sort(lst):
    n = len(lst)
    for i in range(n):
        swapped = False
        for j in range(n - i - 1):
            if lst[j] > lst[j + 1]:
                lst[j], lst[j + 1] = lst[j + 1], lst[j]
                swapped = True
        if not swapped:
            break
    return lst
test_cases = [
    [5, 2, 9, 1, 5, 6],
    [10, 8, 6, 4, 2],
    [1, 2, 3, 4, 5]
]
for i, test in enumerate(test_cases, 1):
    print(f"Test Case {i}: Input: {test} | Output: {bubble_sort(test)}")

```

```
Test Case 1: Input: [5, 2, 9, 1, 5, 6] | Output: [1, 2, 5, 5, 6, 9]
Test Case 2: Input: [10, 8, 6, 4, 2] | Output: [2, 4, 6, 8, 10]
Test Case 3: Input: [1, 2, 3, 4, 5] | Output: [1, 2, 3, 4, 5]

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

main.py

Share Run

```python
def find_peak_element(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = (left + right) // 2
        if nums[mid] > nums[mid + 1]:
            right = mid
        else:
            left = mid + 1
    return left

test_cases = [
    [1, 2, 3, 1],
    [1, 2, 1, 3, 5, 6, 4]
]

for i, nums in enumerate(test_cases, 1):
    print(f"Test Case {i}: Input: {nums} | Output: {find_peak_element(nums)}")
```

Output

Clear

```
Test Case 1: Input: [1, 2, 3, 1] | Output: 2
Test Case 2: Input: [1, 2, 1, 3, 5, 6, 4] | Output: 5

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

```python
def find_substrings(words):
    result = []
    for word in words:
        for other in words:
            if word != other and word in other:
                result.append(word)
                break
    return result

test_cases = [
    ["mass", "as", "hero", "superhero"],
    ["leetcode", "et", "code"],
    ["blue", "green", "bu"]
]

for i, words in enumerate(test_cases, 1):
    print(f"Test Case {i}: Input: {words} | Output: {find_substrings(words)}")

```

```
Test Case 1: Input: ['mass', 'as', 'hero', 'superhero'] | Output: ['as', 'hero']
Test Case 2: Input: ['leetcode', 'et', 'code'] | Output: ['et', 'code']
Test Case 3: Input: ['blue', 'green', 'bu'] | Output: []

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

Programiz Python Online Compiler

Programiz PRO >

main.py

Share Run

```python
def process_list(lst):
    return sorted(lst)
test_cases = [
    [],
    [1],
    [7, 7, 7, 7],
    [-5, -1, -3, -2, -4]
]
for i, test in enumerate(test_cases, 1):
    print(f"Test Case {i}: Input: {test} | Output: {process_list(test
        )}")

```

Output

Clear

```
Test Case 1: Input: [] | Output: []
Test Case 2: Input: [1] | Output: [1]
Test Case 3: Input: [7, 7, 7, 7] | Output: [7, 7, 7, 7]
Test Case 4: Input: [-5, -1, -3, -2, -4] | Output: [-5, -4, -3, -2, -1]

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

```python
def insertion_sort(lst):
    n = len(lst)
    for i in range(1, n):
        key = lst[i]
        j = i - 1
        while j >= 0 and lst[j] > key:
            lst[j + 1] = lst[j]
            j -= 1
        lst[j + 1] = key
    return lst

test_cases = [
    [3, 1, 4, 1, 5, 9, 2, 6, 5, 3],
    [5, 5, 5, 5, 5],
    [2, 3, 1, 3, 2, 1, 1, 3]
]

for i, test in enumerate(test_cases, 1):
    print(f"Test Case {i}: Input: {test} | Output: {insertion_sort(test)}")
```

```
Test Case 1: Input: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3] | Output: [1, 1, 2, 3, 3, 4, 5, 5, 6,
    9]
Test Case 2: Input: [5, 5, 5, 5, 5] | Output: [5, 5, 5, 5, 5]
Test Case 3: Input: [2, 3, 1, 3, 2, 1, 1, 3] | Output: [1, 1, 1, 2, 2, 3, 3, 3]

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

```python
def str_str(haystack, needle):
    if not needle:
        return 0

test_cases = [
    ("sadbutsad", "sad"),
    ("leetcode", "leeto")
]

for i, (haystack, needle) in enumerate(test_cases, 1):
    print(f"Test Case {i}: Input: haystack='{haystack}', needle='{needle}' | Output:
        {str_str(haystack, needle)}")

```

```
Test Case 1: Input: haystack='sadbutsad', needle='sad' | Output: None
Test Case 2: Input: haystack='leetcode', needle='leeto' | Output: None

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

main.py

Share Run Output

```python
def find_kth_missing(arr, k):
    missing_count = 0
    current = 1
    index = 0

    while missing_count < k:
        if index < len(arr) and arr[index] == current:
            index += 1
        else:
            missing_count += 1
            if missing_count == k:
                return current
        current += 1

test_cases = [
    ([2, 3, 4, 7, 11], 5),
    ([1, 2, 3, 4], 2)
]

for i, (arr, k) in enumerate(test_cases, 1):
    print(f"Test Case {i}: Input: {arr}, k={k} | Output: {find_kth_missing(arr, k)}"
        )

```

```
Test Case 1: Input: [2, 3, 4, 7, 11], k=5 | Output: 9
Test Case 2: Input: [1, 2, 3, 4], k=2 | Output: 6

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