DAY-3 PROGRAMS

1. Write a program to perform the following o An empty list o A list with one element o A list with all identical elements o A list with negative numbers Test Cases: 1. Input: [] o Expected Output: [] 2. Input: [1] o Expected Output: [1] 3. Input: [7, 7, 7, 7] o Expected Output: [7, 7, 7, 7] 4. Input: [-5, -1, -3, -2, -4] o Expected Output: [-5, -4, -3, -2, -1].

main.py | Share | Run | Output

```python
def process_list(input_list):
    sorted_list = sorted(input_list)
    return sorted_list
test_cases = [
    ([], []),
    ([1], [1]),
    ([7, 7, 7, 7], [7, 7, 7, 7]),
    ([-5, -1, -3, -2, -4], [-5, -4, -3, -2, -1]),
]
for i, (input_list, expected_output) in enumerate(test_cases):
    output = process_list(input_list)
    print(f"Test Case {i+1}:")
    print(f"Input: {input_list}")
    print(f"Expected Output: {expected_output}")
    print(f"Actual Output: {output}")
    print(f"Pass: {output == expected_output}")
    print("-" * 30)

```

```
Test Case 1:
Input: []
Expected Output: []
Actual Output: []
Pass: True
------------------------------
Test Case 2:
Input: [1]
Expected Output: [1]
Actual Output: [1]
Pass: True
------------------------------
Test Case 3:
Input: [7, 7, 7, 7]
Expected Output: [7, 7, 7, 7]
Actual Output: [7, 7, 7, 7]
Pass: True
------------------------------
```

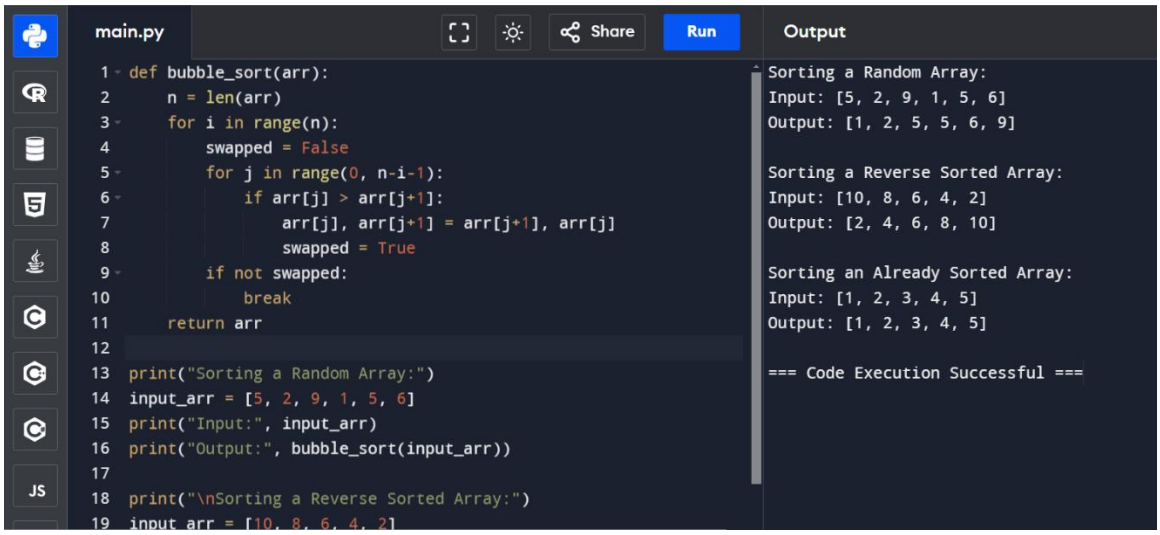
2. Describe the Selection Sort algorithm's process of sorting an array. Selection Sort works by dividing the array into a sorted and an unsorted region. Initially, the sorted region is empty, and the unsorted region contains all elements. The algorithm repeatedly selects the smallest element from the unsorted region and swaps it with the leftmost unsorted element, then moves the boundary of the sorted region one element to the right. Explain why Selection Sort is simple to understand and implement but is inefficient for large datasets. Provide examples to illustrate step-by-step how Selection Sort rearranges the elements into ascending order, ensuring clarity in your explanation of the algorithm's mechanics and effectiveness. Sorting a Random Array: Input: [5, 2, 9, 1, 5, 6] Output: [1, 2, 5, 5, 6, 9] Sorting a Reverse Sorted Array: Input: [10, 8, 6, 4, 2] Output: [2, 4, 6, 8, 10] Sorting an Already Sorted Array: Input: [1, 2, 3, 4, 5] Output: [1, 2, 3, 4, 5].

main.py | Share | Run | Output

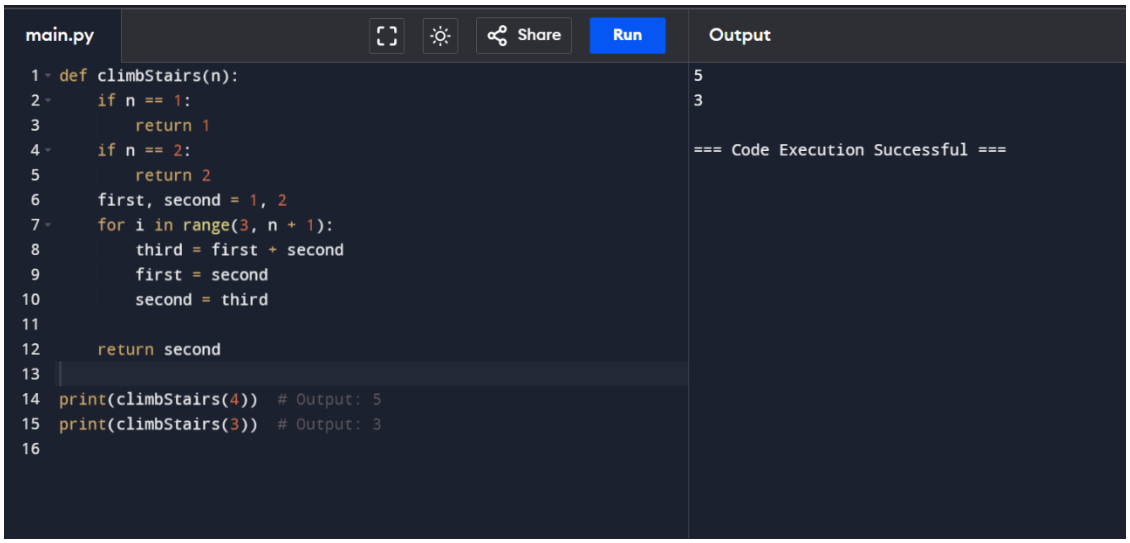
```python
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
print("Sorting a Random Array:")
input_arr = [5, 2, 9, 1, 5, 6]
print("Input:", input_arr)
print("Output:", selection_sort(input_arr))

print("\nSorting a Reverse Sorted Array:")
input_arr = [10, 8, 6, 4, 2]
print("Input:", input_arr)
print("Output:", selection_sort(input_arr))

print("\nSorting an Already Sorted Array:")
```

```
Sorting a Random Array:
Input: [5, 2, 9, 1, 5, 6]
Output: [1, 2, 5, 5, 6, 9]

Sorting a Reverse Sorted Array:
Input: [10, 8, 6, 4, 2]
Output: [2, 4, 6, 8, 10]

Sorting an Already Sorted Array:
Input: [1, 2, 3, 4, 5]
Output: [1, 2, 3, 4, 5]

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

3. Write code to modify bubble_sort function to stop early if the list becomes sorted before all passes are completed.

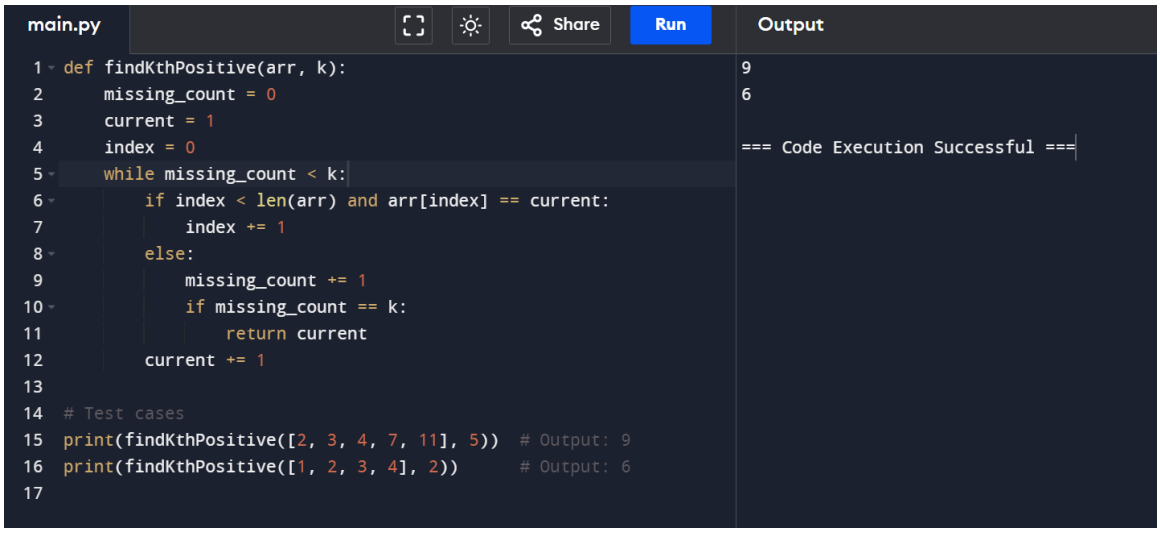


4. Test Cases: • Test your optimized function with the following lists: 1. Input: [64, 25, 12, 22, 11] ♣ Expected Output: [11, 12, 22, 25, 64] 2. Input: [29, 10, 14, 37, 13] ♣ Expected Output: [10, 13, 14, 29, 37] 3. Input: [3, 5, 2, 1, 4] ♣ Expected Output: [1, 2, 3, 4, 5] 4. Input: [1, 2, 3, 4, 5] (Already sorted list) ♣ Expected Output: [1, 2, 3, 4, 5] 5. Input: [5, 4, 3, 2, 1] (Reverse sorted list) ♣ Expected Output: [1, 2, 3, 4, 5] 1. Write code for Insertion Sort that manages arrays with duplicate elements during the sorting process. Ensure the algorithm's behavior when encountering duplicate values, including whether it preserves the relative order of duplicates and how it affects the overall sorting outcome. Examples: 1. Array with Duplicates: o Input: [3, 1, 4, 1, 5, 9, 2, 6, 5, 3] o Output: [1, 1, 2, 3, 3, 4, 5, 5, 6, 9] 2. All Identical Elements: o Input: [5, 5, 5, 5, 5] o Output: [5, 5, 5, 5, 5] 3. Mixed Duplicates: o Input: [2, 3, 1, 3, 2, 1, 1, 3] o Output: [1, 1, 1, 2, 2, 3, 3, 3].

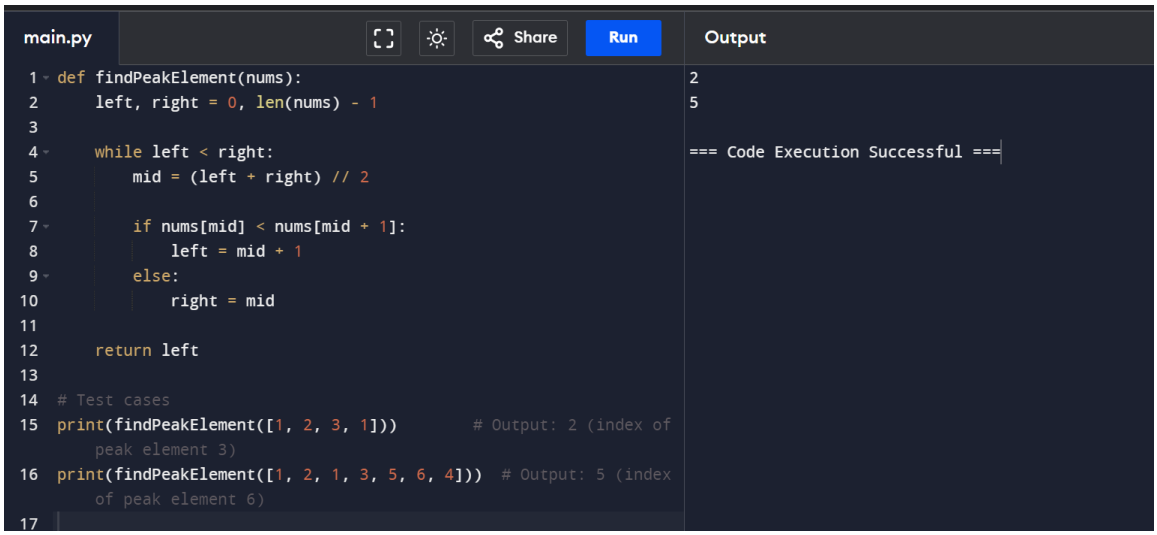


5. Given an array arr of positive integers sorted in a strictly increasing order, and an integer k. return the kth positive integer that is missing from this array. Example 1: Input: arr = [2,3,4,7,11], k = 5 Output: 9 Explanation: The missing positive integers are [1,5,6,8,9,10,12,13,...]. The 5th missing positive integer is 9. Example 2: Input: arr = [1,2,3,4],

k = 2 Output: 6 Explanation: The missing positive integers are [5,6,7,...]. The 2nd missing positive integer is 6.



6. A peak element is an element that is strictly greater than its neighbors. Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array. You must write an algorithm that runs in O(log n) time. Example 1: Input: nums = [1,2,3,1] Output: 2 Explanation: 3 is a peak element and your function should return the index number 2. Example 2: Input: nums = [1,2,1,3,5,6,4] Output: 5 Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.



7. Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack. Example 1: Input: haystack = "sadbutsad", needle = "sad" Output: 0 Explanation: "sad" occurs at index 0 and 6. The first occurrence is at index 0, so we return 0. Example 2: Input: haystack = "leetcode", needle = "leeto" Output: -1 Explanation: "leeto" did not occur in "leetcode", so we return -1.

main.py Share Run Output

```python
def strStr(haystack, needle):
    if not needle:
        return 0
    h_len, n_len = len(haystack), len(needle)
    for i in range(h_len - n_len + 1):
        if haystack[i:i + n_len] == needle:
            return i

    return -1

# Test cases
print(strStr("sadbutsad", "sad"))  # Output: 0
print(strStr("leetcode", "leeto"))  # Output: -1
```

```
0
-1

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

8. Given an array of string words, return all strings in words that is a substring of another word. You can return the answer in any order. A substring is a contiguous sequence of characters within a string Example 1: Input: words = ["mass","as","hero","superhero"] Output: ["as","hero"] Explanation: "as" is substring of "mass" and "hero" is substring of "superhero". ["hero","as"] is also a valid answer. Example 2: Input: words = ["leetcode","et","code"] Output: ["et","code"] Explanation: "et", "code" are substring of "leetcode". Example 3: Input: words = ["blue","green","bu"] Output: [] Explanation: No string of words is substring of another string.

main.py Share Run Output

```python
def find_substrings(words):
    substrings = set()
    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                substrings.add(words[i])

    return list(substrings)

# Example usage
words1 = ["mass", "as", "hero", "superhero"]
print(find_substrings(words1))  # Output: ["as", "hero"]

words2 = ["leetcode", "et", "code"]
print(find_substrings(words2))  # Output: ["et", "code"]

words3 = ["blue", "green", "bu"]
print(find_substrings(words3))  # Output: []
```

```
['as', 'hero']
['et', 'code']
[]

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