|  |
| --- |
| 1. |
|  |  | # An empty list |
|  |  | empty\_list = [] |
|  |  |  |
|  |  | # A list with one element |
|  |  | single\_element\_list = [1] |
|  |  |  |
|  |  | # A list with all identical elements |
|  |  | identical\_elements\_list = [7, 7, 7, 7] |
|  |  |  |
|  |  | # A list with negative numbers |
|  |  | negative\_numbers\_list = [-5, -1, -3, -2, -4] |
|  |  |  |
|  |  | # Test Cases |
|  |  | # Input: [] |
|  |  | # Expected Output: [] |
|  |  | assert empty\_list == [] |
|  |  |  |
|  |  | # Input: [1] |
|  |  | # Expected Output: [1] |
|  |  | assert single\_element\_list == [1] |
|  |  |  |
|  |  | # Input: [7, 7, 7, 7] |
|  |  | # Expected Output: [7, 7, 7, 7] |
|  |  | assert identical\_elements\_list == [7, 7, 7, 7] |
|  |  |  |
|  |  | # Input: [-5, -1, -3, -2, -4] |
|  |  | # Expected Output: [-5, -4, -3, -2, -1] |
|  |  | assert negative\_numbers\_list == [-5, -4, -3, -2, -1] |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 2. |
|  |  | 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 |
|  |  |  |
|  |  | # Sorting a Random Array |
|  |  | random\_array = [5, 2, 9, 1, 5, 6] |
|  |  | sorted\_random\_array = selection\_sort(random\_array) |
|  |  | print("Random Array Sorted:", sorted\_random\_array) |
|  |  |  |
|  |  | # Sorting a Reverse Sorted Array |
|  |  | reverse\_sorted\_array = [10, 8, 6, 4, 2] |
|  |  | sorted\_reverse\_array = selection\_sort(reverse\_sorted\_array) |
|  |  | print("Reverse Sorted Array Sorted:", sorted\_reverse\_array) |
|  |  |  |
|  |  | # Sorting an Already Sorted Array |
|  |  | already\_sorted\_array = [1, 2, 3, 4, 5] |
|  |  | sorted\_already\_sorted\_array = selection\_sort(already\_sorted\_array) |
|  |  | print("Already Sorted Array Sorted:", sorted\_already\_sorted\_array) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 3. |
|  |  | def bubble\_sort(arr): |
|  |  | n = len(arr) |
|  |  | for i in range(n): |
|  |  | already\_sorted = True |
|  |  | for j in range(n - i - 1): |
|  |  | if arr[j] > arr[j + 1]: |
|  |  | arr[j], arr[j + 1] = arr[j + 1], arr[j] |
|  |  | already\_sorted = False |
|  |  | if already\_sorted: |
|  |  | break |
|  |  | return arr |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 4. |
|  |  | def insertion\_sort\_with\_duplicates(arr): |
|  |  | for i in range(1, len(arr)): |
|  |  | key = arr[i] |
|  |  | j = i - 1 |
|  |  | while j >= 0 and key < arr[j]: |
|  |  | arr[j + 1] = arr[j] |
|  |  | j -= 1 |
|  |  | arr[j + 1] = key |
|  |  | return arr |
|  |  |  |
|  |  | # Test Cases |
|  |  | test\_cases = [ |
|  |  | [64, 25, 12, 22, 11], |
|  |  | [29, 10, 14, 37, 13], |
|  |  | [3, 5, 2, 1, 4], |
|  |  | [1, 2, 3, 4, 5], |
|  |  | [5, 4, 3, 2, 1], |
|  |  | [3, 1, 4, 1, 5, 9, 2, 6, 5, 3], |
|  |  | [5, 5, 5, 5, 5], |
|  |  | [2, 3, 1, 3, 2, 1, 1, 3] |
|  |  | ] |
|  |  |  |
|  |  | for idx, test in enumerate(test\_cases): |
|  |  | sorted\_arr = insertion\_sort\_with\_duplicates(test) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 5. |
|  |  | def findKthPositive(arr, k): |
|  |  | missing = [] |
|  |  | i = 1 |
|  |  | while len(missing) < k: |
|  |  | if i not in arr: |
|  |  | missing.append(i) |
|  |  | i += 1 |
|  |  | return missing[-1] |
|  |  |  |
|  |  | # Example 1 |
|  |  | arr1 = [2, 3, 4, 7, 11] |
|  |  | k1 = 5 |
|  |  | output1 = findKthPositive(arr1, k1) |
|  |  | print(output1) # Output: 9 |
|  |  |  |
|  |  | # Example 2 |
|  |  | arr2 = [1, 2, 3, 4] |
|  |  | k2 = 2 |
|  |  | output2 = findKthPositive(arr2, k2) |
|  |  | print(output2) # Output: 6 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 6. |
|  |  | class Solution: |
|  |  | def findPeakElement(self, nums: List[int]) -> int: |
|  |  | left, right = 0, len(nums) - 1 |
|  |  | while left < right: |
|  |  | mid = left + (right - left) // 2 |
|  |  | if nums[mid] < nums[mid + 1]: |
|  |  | left = mid + 1 |
|  |  | else: |
|  |  | right = mid |
|  |  | return left |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 7. |
|  |  | def strStr(haystack, needle): |
|  |  | if needle in haystack: |
|  |  | return haystack.index(needle) |
|  |  | else: |
|  |  | return -1 |
|  |  |  |
|  |  | # Example 1 |
|  |  | haystack = "sadbutsad" |
|  |  | needle = "sad" |
|  |  | print(strStr(haystack, needle)) # Output: 0 |
|  |  |  |
|  |  | # Example 2 |
|  |  | haystack = "leetcode" |
|  |  | needle = "leeto" |
|  |  | print(strStr(haystack, needle)) # Output: -1 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 8. |
|  |  | def stringMatching(words): |
|  |  | return [word for word in words if any(other\_word != word and other\_word.find(word) != -1 for other\_word in words)] |
|  |  |  |
|  |  | # Example 1 |
|  |  | words1 = ["mass", "as", "hero", "superhero"] |
|  |  | print(stringMatching(words1)) # Output: ["as", "hero"] |
|  |  |  |
|  |  | # Example 2 |
|  |  | words2 = ["leetcode", "et", "code"] |
|  |  | print(stringMatching(words2)) # Output: ["et", "code"] |
|  |  |  |
|  |  | # Example 3 |
|  |  | words3 = ["blue", "green", "bu"] |
|  |  | print(stringMatching(words3)) # Output: [] |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 9. |
|  |  | import math |
|  |  |  |
|  |  | def euclidean\_distance(point1, point2): |
|  |  | return math.sqrt((point1[0] - point2[0])\*2 + (point1[1] - point2[1])\*2) |
|  |  |  |
|  |  | def closest\_pair\_of\_points(points): |
|  |  | min\_distance = float('inf') |
|  |  | closest\_pair = None |
|  |  |  |
|  |  | for i in range(len(points)): |
|  |  | for j in range(i + 1, len(points)): |
|  |  | distance = euclidean\_distance(points[i], points[j]) |
|  |  | if distance < min\_distance: |
|  |  | min\_distance = distance |
|  |  | closest\_pair = (points[i], points[j]) |
|  |  |  |
|  |  | return closest\_pair, min\_distance |
|  |  |  |
|  |  | # Input points |
|  |  | points = [(1, 2), (4, 5), (7, 8), (3, 1)] |
|  |  |  |
|  |  | # Find the closest pair of points and minimum distance |
|  |  | closest\_pair, min\_distance = closest\_pair\_of\_points(points) |
|  |  |  |
|  |  | # Output the result |
|  |  | print(f"Closest pair: {closest\_pair[0]} - {closest\_pair[1]} Minimum distance: {min\_distance}") |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 10. |
|  |  | import math |
|  |  |  |
|  |  | def euclidean\_distance(point1, point2): |
|  |  | return math.sqrt((point1[0] - point2[0])\*2 + (point1[1] - point2[1])\*2) |
|  |  |  |
|  |  | def closest\_pair\_brute\_force(points): |
|  |  | min\_distance = float('inf') |
|  |  | closest\_pair = None |
|  |  | for i in range(len(points)): |
|  |  | for j in range(i + 1, len(points)): |
|  |  | distance = euclidean\_distance(points[i], points[j]) |
|  |  | if distance < min\_distance: |
|  |  | min\_distance = distance |
|  |  | closest\_pair = (points[i], points[j]) |
|  |  | return closest\_pair |
|  |  |  |
|  |  | # Sample set of points |
|  |  | sample\_points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (7.5, 4.5)] |
|  |  |  |
|  |  | # Find the closest pair of points |
|  |  | closest\_pair = closest\_pair\_brute\_force(sample\_points) |
|  |  | print(closest\_pair) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 11. |
|  |  | from itertools import combinations |
|  |  |  |
|  |  | def orientation(p, q, r): |
|  |  | val = (q[1] - p[1]) \* (r[0] - q[0]) - (q[0] - p[0]) \* (r[1] - q[1]) |
|  |  | if val == 0: |
|  |  | return 0 |
|  |  | return 1 if val > 0 else -1 |
|  |  |  |
|  |  | def convex\_hull(points): |
|  |  | n = len(points) |
|  |  | if n < 3: |
|  |  | return points |
|  |  |  |
|  |  | hull = [] |
|  |  | for p, q in combinations(points, 2): |
|  |  | side = [r for r in points if orientation(p, q, r) == 1] |
|  |  | if len(side) == n - 2: |
|  |  | hull.extend(side) |
|  |  |  |
|  |  | return hull |
|  |  |  |
|  |  | # Input points |
|  |  | points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)] |
|  |  |  |
|  |  | # Calculate Convex Hull |
|  |  | convex\_hull\_points = convex\_hull(points) |
|  |  |  |
|  |  | print("Convex Hull:", convex\_hull\_points) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 12. |
|  |  | from itertools import combinations |
|  |  |  |
|  |  | def orientation(p, q, r): |
|  |  | val = (q[1] - p[1]) \* (r[0] - q[0]) - (q[0] - p[0]) \* (r[1] - q[1]) |
|  |  | if val == 0: |
|  |  | return 0 |
|  |  | return 1 if val > 0 else -1 |
|  |  |  |
|  |  | def convex\_hull(points): |
|  |  | n = len(points) |
|  |  | if n < 3: |
|  |  | return points |
|  |  |  |
|  |  | hull = [] |
|  |  | for p, q in combinations(points, 2): |
|  |  | side = [r for r in points if orientation(p, q, r) == 1] |
|  |  | if len(side) == n - 2: |
|  |  | hull.extend(side) |
|  |  |  |
|  |  | return hull |
|  |  |  |
|  |  | # Input points |
|  |  | points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)] |
|  |  |  |
|  |  | # Calculate Convex Hull |
|  |  | convex\_hull\_points = convex\_hull(points) |
|  |  |  |
|  |  | print("Convex Hull:", convex\_hull\_points) |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  | 13. |
|  |  | def total\_value(items, values): |
|  |  | return sum(values[i] for i in items) |
|  |  |  |
|  |  | def is\_feasible(items, weights, capacity): |
|  |  | return sum(weights[i] for i in items) <= capacity |
|  |  |  |
|  |  | def knapsack\_01\_exhaustive\_search(items, weights, values, capacity): |
|  |  | n = len(items) |
|  |  | max\_value = 0 |
|  |  | optimal\_selection = [] |
|  |  |  |
|  |  | for i in range(1 << n): |
|  |  | selected\_items = [j for j in range(n) if (i & (1 << j))] |
|  |  |  |
|  |  | if is\_feasible(selected\_items, weights, capacity): |
|  |  | total = total\_value(selected\_items, values) |
|  |  | if total > max\_value: |
|  |  | max\_value = total |
|  |  | optimal\_selection = selected\_items |
|  |  |  |
|  |  | return optimal\_selection, max\_value |
|  |  |  |
|  |  | # Test Case 1 |
|  |  | items\_1 = [0, 1, 2] |
|  |  | weights\_1 = [2, 3, 1] |
|  |  | values\_1 = [4, 5, 3] |
|  |  | capacity\_1 = 4 |
|  |  | optimal\_selection\_1, total\_value\_1 = knapsack\_01\_exhaustive\_search(items\_1, weights\_1, values\_1, capacity\_1) |
|  |  | print("Test Case 1:") |
|  |  | print("Optimal Selection:", optimal\_selection\_1) |
|  |  | print("Total Value:", total\_value\_1) |
|  |  |  |
|  |  | # Test Case 2 |
|  |  | items\_2 = [0, 1, 2, 3] |
|  |  | weights\_2 = [1, 2, 3, 4] |
|  |  | values\_2 = [2, 4, 6, 3] |
|  |  | capacity\_2 = 6 |
|  |  | optimal\_selection\_2, total\_value\_2 = knapsack\_01\_exhaustive\_search(items\_2, weights\_2, values\_2, capacity\_2) |
|  |  | print("\nTest Case 2:") |
|  |  | print("Optimal Selection:", optimal\_selection\_2) |
|  |  | print("Total Value:", total\_value\_2) |