In [1]:
```
#1.
# An empty list
empty_list = []
print("Empty list:", empty_list)

# A list with one element
one_element_list = [42]
print("List with one element:", one_element_list)

# A list with all identical elements
identical_elements_list = [5, 5, 5, 5, 5]
print("List with all identical elements:", identical_elements_list)

# A list with negative numbers
negative_numbers_list = [-1, -2, -3, -4, -5]
print("List with negative numbers:", negative_numbers_list)
```

```
Empty list: []
List with one element: [42]
List with all identical elements: [5, 5, 5, 5, 5]
List with negative numbers: [-1, -2, -3, -4, -5]
```

In [2]:
```
#2.
def selection_sort(arr):
    n = len(arr)
    for i in range(n):
        # Find the minimum element in the remaining unsorted array
        min_idx = i
        for j in range(i+1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        # Swap the found minimum element with the first element of the unsor
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr

# Example usage
array = [64, 25, 12, 22, 11]
sorted_array = selection_sort(array)
print("Sorted array:", sorted_array)
```

```
Sorted array: [11, 12, 22, 25, 64]
```

In [3]:

```
#3.
def bubble_sort_optimized(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr

# Test cases
print(bubble_sort_optimized([64, 25, 12, 22, 11]))  # Expected Output: [11,
print(bubble_sort_optimized([29, 10, 14, 37, 13]))  # Expected Output: [10,
print(bubble_sort_optimized([3, 5, 2, 1, 4]))       # Expected Output: [1, 
print(bubble_sort_optimized([1, 2, 3, 4, 5]))       # Expected Output: [1, 
print(bubble_sort_optimized([5, 4, 3, 2, 1]))       # Expected Output: [1, 
```

```
[11, 12, 22, 25, 64]
[10, 13, 14, 29, 37]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

In [4]:

```
#4.
def insertion_sort(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
print(insertion_sort([3, 1, 4, 1, 5, 9, 2, 6, 5, 3]))  # Expected Output: [
print(insertion_sort([5, 5, 5, 5, 5]))                 # Expected Output: [5,
print(insertion_sort([2, 3, 1, 3, 2, 1, 1, 3]))       # Expected Output: [1,
```

```
[1, 1, 2, 3, 3, 4, 5, 5, 6, 9]
[5, 5, 5, 5, 5]
[1, 1, 1, 2, 2, 3, 3, 3]
```

In [5]:
```python
#5.
def find_kth_missing(arr, k):
    missing_count = 0
    current = 1
    idx = 0
    while missing_count < k:
        if idx < len(arr) and arr[idx] == current:
            idx += 1
        else:
            missing_count += 1
            if missing_count == k:
                return current
        current += 1

# Test cases
print(find_kth_missing([2, 3, 4, 7, 11], 5))  # Expected Output: 9
print(find_kth_missing([1, 2, 3, 4], 2))     # Expected Output: 6
```

```
9
6
```

In [6]:
```python
#6.
def find_peak_element(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = (left + right) // 2
        if nums[mid] < nums[mid + 1]:
            left = mid + 1
        else:
            right = mid
    return left

# Test cases
print(find_peak_element([1, 2, 3, 1]))          # Expected Output: 2
print(find_peak_element([1, 2, 1, 3, 5, 6, 4])) # Expected Output: 5 (or 1)
```

```
2
5
```

In [7]:
```python
#7.
def str_str(haystack, needle):
    if not needle:
        return 0
    for i in range(len(haystack) - len(needle) + 1):
        if haystack[i:i+len(needle)] == needle:
            return i
    return -1

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

```
0
-1
```

In [8]:
```
#8.
def find_substrings(words):
    result = []
    for i in range(len(words)):
        for j in range(len(words)):
            if i != j and words[i] in words[j]:
                result.append(words[i])
                break
    return result

# Test cases
print(find_substrings(["mass", "as", "hero", "superhero"]))  # Expected Out
print(find_substrings(["leetcode", "et", "code"]))           # Expected Out
print(find_substrings(["blue", "green", "bu"]))              # Expected Out
```

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

In [9]:
```
#9.
import math

def euclidean_distance(point1, point2):
    return math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**

def closest_pair(points):
    min_distance = float('inf')
    closest_points = 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_points = (points[i], points[j])
    return closest_points, min_distance

# Test case
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_points, min_distance = closest_pair(points)
print(f"Closest pair: {closest_points} Minimum distance: {min_distance}")  #
```

```
Closest pair: ((1, 2), (3, 1)) Minimum distance: 2.23606797749979
```

In [10]:
```
#10.
import itertools

def cross_product(o, a, b):
    return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) * (b[0] - o[0])

def convex_hull(points):
    points = sorted(points)
    lower = []
    for p in points:
        while len(lower) >= 2 and cross_product(lower[-2], lower[-1], p) <=
            lower.pop()
        lower.append(p)
    upper = []
    for p in reversed(points):
        while len(upper) >= 2 and cross_product(upper[-2], upper[-1], p) <=
            upper.pop()
        upper.append(p)
    return lower[:-1] + upper[:-1]

# Test case
points = [(10,0), (11,5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (
convex_hull_points = convex_hull(points)
print(f"Convex Hull: {convex_hull_points}")  # Expected Output: [(5, 3), (9,
```

```
Convex Hull: [(5, 3), (10, 0), (15, 3), (12.5, 7), (6, 6.5)]
```

In [11]:
```
#11.
def convex_hull(points):
    points = sorted(set(points))

    if len(points) <= 1:
        return points

    def cross(o, a, b):
        return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) * (b[0] - o[0]

    lower = []
    for p in points:
        while len(lower) >= 2 and cross(lower[-2], lower[-1], p) <= 0:
            lower.pop()
        lower.append(p)

    upper = []
    for p in reversed(points):
        while len(upper) >= 2 and cross(upper[-2], upper[-1], p) <= 0:
            upper.pop()
        upper.append(p)

    return lower[:-1] + upper[:-1]

# Test case
points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
convex_hull_points = convex_hull(points)
print(f"Convex Hull: {convex_hull_points}")  # Expected Output: [(0, 0), (1,
```

```
Convex Hull: [(0, 0), (8, 1), (4, 6)]
```

In [16]:

```
#12.
import itertools

def distance(city1, city2):
    return math.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)

def tsp(cities):
    n = len(cities)
    min_path = None
    min_distance = float('inf')

    for perm in itertools.permutations(range(1, n)):
        current_path = [0] + list(perm) + [0]
        current_distance = sum(distance(cities[current_path[i]], cities[curr
        if current_distance < min_distance:
            min_distance = current_distance
            min_path = current_path

    return min_distance, [cities[i] for i in min_path]

# Test cases
cities1 = [(1, 2), (4, 5), (7, 1), (3, 6)]
min_distance, min_path = tsp(cities1)
print(f"Shortest Distance: {min_distance}")
print(f"Shortest Path: {min_path}")

cities2 = [(2, 4), (8, 1), (1, 7), (6, 3), (5, 9)]
min_distance, min_path = tsp(cities2)
print(f"Shortest Distance: {min_distance}")
print(f"Shortest Path: {min_path}")
```

```
Shortest Distance: 16.969112047670894
Shortest Path: [(1, 2), (7, 1), (4, 5), (3, 6), (1, 2)]
Shortest Distance: 23.12995011084934
Shortest Path: [(2, 4), (6, 3), (8, 1), (5, 9), (1, 7), (2, 4)]
```

In [17]:
```
#13.
def total_cost(assignment, cost_matrix):
    return sum(cost_matrix[i][assignment[i]] for i in range(len(assignment))

def assignment_problem(cost_matrix):
    n = len(cost_matrix)
    min_cost = float('inf')
    min_assignment = None

    for perm in itertools.permutations(range(n)):
        current_cost = total_cost(perm, cost_matrix)
        if current_cost < min_cost:
            min_cost = current_cost
            min_assignment = perm

    return min_assignment, min_cost

# Test cases
cost_matrix1 = [
    [3, 10, 7],
    [8, 5, 12],
    [4, 6, 9]
]
assignment, cost = assignment_problem(cost_matrix1)
print(f"Optimal Assignment: {assignment}")
print(f"Total Cost: {cost}")

cost_matrix2 = [
    [15, 9, 4],
    [8, 7, 18],
    [6, 12, 11]
]
assignment, cost = assignment_problem(cost_matrix2)
print(f"Optimal Assignment: {assignment}")
print(f"Total Cost: {cost}")
```

```
Optimal Assignment: (2, 1, 0)
Total Cost: 16
Optimal Assignment: (2, 1, 0)
Total Cost: 17
```

In [18]:

```
#14.
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(weights, values, capacity):
    n = len(weights)
    max_value = 0
    best_combination = []

    for r in range(1, n + 1):
        for combination in itertools.combinations(range(n), r):
            if is_feasible(combination, weights, capacity):
                current_value = total_value(combination, values)
                if current_value > max_value:
                    max_value = current_value
                    best_combination = combination

    return best_combination, max_value

# Test cases
weights1 = [2, 3, 1]
values1 = [4, 5, 3]
capacity1 = 4
optimal_selection, total_val = knapsack(weights1, values1, capacity1)
print(f"Optimal Selection: {optimal_selection} Total Value: {total_val}")

weights2 = [1, 2, 3, 4]
values2 = [2, 4, 6, 3]
capacity2 = 6
optimal_selection, total_val = knapsack(weights2, values2, capacity2)
print(f"Optimal Selection: {optimal_selection} Total Value: {total_val}")
```

```
Optimal Selection: (1, 2) Total Value: 8
Optimal Selection: (0, 1, 2) Total Value: 12
```

In [ ]: