1. PROGRAM:

```
def unique_elements(input_list):
    seen = set()
    unique_list = []
    for num in input_list:
        if num not in seen:
            unique_list.append(num)
            seen.add(num)

    return unique_list
print(unique_elements([3, 7, 3, 5, 2, 5, 9, 2]))
print(unique_elements([-1, 2, -1, 3, 2, -2]))
print(unique_elements([1000000, 999999, 1000000]))
```

OUTPUT:

```
[3, 7, 5, 2, 9]
[-1, 2, 3, -2]
[1000000, 999999]
```

3. PROGRAM:

```
import math

def closest_pair_brute_force(points):
    min_distance = float('inf')
    closest_pair = (None, None)
    for i in range(len(points)):
        for j in range(i + 1, len(points)):
            distance = math.sqrt((points[i][0] - points[j][0]) ** 2 + (points[i]

            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])

    return closest_pair, min_distance
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_pair, min_distance = closest_pair_brute_force(points)

print(f"Closest pair: {closest_pair[0]} - {closest_pair[1]}")
print(f"Minimum distance: {min_distance}")
```

OUTPUT:

```
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.2360679774997979
```

4. PROGRAM:

```
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, 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

def is_counter_clockwise(p1, p2, p3):
    return (p2[1] - p1[1]) * (p3[0] - p2[0]) - (p2[0] - p1[0]) * (p3[1] - p2[1]) > 0

def are_collinear(p1, p2, p3):
    return (p2[1] - p1[1]) * (p3[0] - p2[0]) == (p2[0] - p1[0]) * (p3[1] - p2[1])
```

```
def convex_hull_brute_force(points):
    hull_points = []
    for i in range(len(points)):
        for j in range(len(points)):
            if i == j:
                continue
            is_hull_edge = True
            collinear_points = []
            for k in range(len(points)):
                if k == i or k == j:
                    continue
                if not is_counter_clockwise(points[i], points[j], points[k]):
                    if are_collinear(points[i], points[j], points[k]):
                        collinear_points.append(points[k])
                    else:
                        is_hull_edge = False
                        break
            if is_hull_edge:
                if points[i] not in hull_points:
                    hull_points.append(points[i])
                if points[j] not in hull_points:
                    hull_points.append(points[j])
                for point in collinear_points:
                    if point not in hull_points:
                        hull_points.append(point)
    return hull_points
```

```
points = [(1, 2), (4, 5), (7, 8), (3, 1)]
closest_pair, min_distance = closest_pair_brute_force(points)
print(f"Closest pair: {closest_pair[0]} - {closest_pair[1]}")
print(f"Minimum distance: {min_distance}")

points_convex_hull = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5), (7.5, 4.5)]
hull_points = convex_hull_brute_force(points_convex_hull)
hull_points.sort(key=lambda p: (p[0], p[1]))
print("Convex Hull Points:", hull_points)
```

OUTPUT:

```
Closest pair: (1, 2) - (3, 1)
Minimum distance: 2.23606797749979
Convex Hull Points: [(5, 3), (6, 6.5), (10, 0), (12.5, 7), (15, 3)]
```

8. PROGRAM:

```
def gameOfLife(board):
    m, n = len(board), len(board[0])
    directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]

    for i in range(m):
        for j in range(n):
            live_neighbors = 0
            for direction in directions:
                r, c = i + direction[0], j + direction[1]
                if 0 <= r < m and 0 <= c < n and abs(board[r][c]) == 1:
                    live_neighbors += 1

            if board[i][j] == 1 and (live_neighbors < 2 or live_neighbors > 3):
                board[i][j] = 2
            elif board[i][j] == 0 and live_neighbors == 3:
                board[i][j] = 3

    for i in range(m):
        for j in range(n):
            if board[i][j] == 2:
                board[i][j] = 0
            elif board[i][j] == 3:
                board[i][j] = 1

    return board
```

OUTPUT:

```
[[0, 0, 0], [0, 0, 1], [0, 1, 1], [0, 0, 0]]
```

9. PROGRAM:

```
def brute_force_string_matching(text, pattern):
    m = len(pattern)
    n = len(text)
    total_comparisons = 0

    for i in range(n - m + 1):
        j = 0
        while j < m and text[i + j] == pattern[j]:
            j += 1
            total_comparisons += 1
        if j == m:
            print(f"Pattern found at index {i}")
        else:
            total_comparisons += m - j

    return total_comparisons

text = "ACGTACGTACGT"
pattern = "ACG"
comparisons = brute_force_string_matching(text, pattern)
print("Total Comparisons (Brute-Force):", comparisons)
```

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr, 0, 0

    mid = len(arr) // 2
    left, left_comp, left_merges = merge_sort(arr[:mid])
    right, right_comp, right_merges = merge_sort(arr[mid:])
    merged, merge_comp, merge_merges = merge(left, right)

    total_comp = left_comp + right_comp + merge_comp
    total_merges = left_merges + right_merges + merge_merges

    return merged, total_comp, total_merges
```

```
def merge(left, right):
    sorted_list = []
    i = j = 0
    comparisons = 0
    merges = 1

    while i < len(left) and j < len(right):
        comparisons += 1
        if left[i] <= right[j]:
            sorted_list.append(left[i])
            i += 1
        else:
            sorted_list.append(right[j])
            j += 1

    while i < len(left):
        sorted_list.append(left[i])
        i += 1

    while j < len(right):
        sorted_list.append(right[j])
        j += 1

    return sorted_list, comparisons, merges
```

```
array = [38, 27, 43, 3, 9, 82, 10]
sorted_array, comparisons, merges = merge_sort(array)
print("Sorted Array:", sorted_array)
print("Total Comparisons (Merge Sort):", comparisons)
print("Total Merges (Merge Sort):", merges)
```

OUTPUT:

```
Pattern found at index 0
Pattern found at index 4
Pattern found at index 8
Total Comparisons (Brute-Force): 30
Sorted Array: [3, 9, 10, 27, 38, 43, 82]
Total Comparisons (Merge Sort): 13
Total Merges (Merge Sort): 6
```