152. Write a program that finds the convex hull of a set of 2D points using the brute force approach.

Input:

A list or array of points represented by coordinates (x, y).

Points: [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]

Output:

The list of points that form the convex hull in counter-clockwise order.

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

AIM: To find the convex hull of a set of 2D points using the brute force

PROGRAM:

```
def cross_product_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 1
    elif val < 0:
        return -1
    else:
        return 0

def is_convex(points):
    n = len(points)
    if n < 3:
        return False

    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                orientation = cross_product_orientation(points[i], points[j], points[k])
                if orientation == 0:
                    continue
                else:
                    for m in range(n):
                        if m != i and m != j and m != k:
                            if cross_product_orientation(points[i], points[j], points[m]) == orientation:
                                return False
    return True

def convex_hull_brute_force(points):
    n = len(points)
    if n < 3:
        return []

    convex_hull = []

    for i in range(n):
        for j in range(i + 1, n):
            for k in range(j + 1, n):
                if is_convex([points[i], points[j], points[k]]):
```

```
                if points[i] not in convex_hull:
                    convex_hull.append(points[i])
                if points[j] not in convex_hull:
                    convex_hull.append(points[j])
                if points[k] not in convex_hull:
                    convex_hull.append(points[k])

    return convex_hull

points = [(1, 1), (4, 6), (8, 1), (0, 0), (3, 3)]
convex_hull = convex_hull_brute_force(points)
print("Convex Hull:", convex_hull)
```

OUTPUT:

```
Convex Hull: [(1, 1), (4, 6), (8, 1), (0, 0),
    (3, 3)]
```

TIME COMPLEXITY: O(n^3)