Assignment 1

Meet Desai (202311031)

Divide and Conquer:

Divide and conquer algorithms solve problems by dividing them into smaller instances, solving each instance recursively and merging the corresponding results to a complete solution. Further, asserts that all instances have exactly the same structure as the original problem and can be solved independently from each other, and so can easily be distributed over a number of parallel processes or threads. These algorithms exploit the fact that solutions to smaller problems can be used to solve larger problems.

The key idea is that is we have two convex hull then, they can be merged in linear time to get a convex hull of a larger set of points.

Convex Hull Algorithm:

The convex hull algorithm locates the smallest convex shape in a 2D or 3D space that encompasses all input points. The outer boundary is created by tightly encircling the points with an elastic band. Convex hull is the name given to this form. It can be used in robotics, collision detection, and computer graphics. To effectively locate the convex hull, there are numerous techniques available, including Gift Wrapping, Graham's Scan, and Jarvis March.

Here is step by step Explanation:

Step 1: Input a set of points represented as (x, y) coordinates.

Step 2: Find the point with the lowest y-coordinate. If there are ties, choose the leftmost point among them. Let's call this point "p0."

Step 3: Sort the remaining points based on their polar angles relative to point "p0." The polar angle is the angle formed by the line segment from point "p0" to the other points with respect to the positive x-axis.

Step 4: Remove collinear points. If two or more points have the same polar angle, keep only the farthest point and remove the others.

Step 5: Initialize an empty stack.

Step 6: Push the first two points from the sorted list onto the stack.

Step 7: Iterate through the sorted points (starting from the third point):

Step 8: Check the orientation of the last two points on the stack (top and second-top) and the current point in the iteration.

Step 9: While the orientation is not counterclockwise (i.e., the last two points and the current point make a clockwise angle or are collinear), pop the top point from the stack.

Step 10: Push the current point onto the stack.

Step 11: Repeat steps 8 to 10 until all points are processed.

Step 12: The points remaining in the stack after the iteration form the vertices of the convex hull in counterclockwise order.

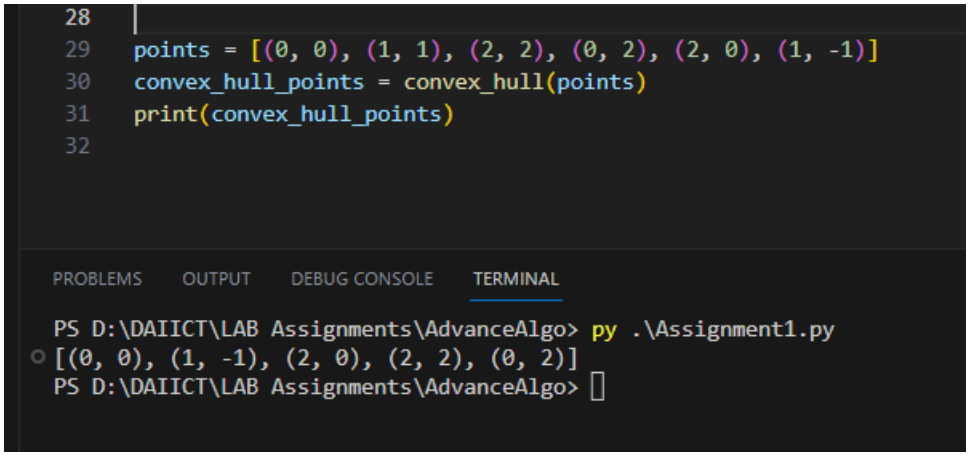
Step 13: Output the points in the stack as the vertices of the convex hull.

Python Code:

```
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 2
def convex_hull(points):
# Function using Divide and Conquer.
    def divide_and_conquer(points_list):
        if len(points_list) <= 1:
            return points_list
        points_list.sort(key=lambda p: (p[0], p[1]))

        def build_hull(stack, point):
            while len(stack) >= 2 and orientation(stack[-2], stack[-1], point)
!= 2:
                stack.pop()
            stack.append(point)

        lower_hull = []

        for point in points_list:
            build_hull(lower_hull, point)
        upper_hull = []

        for point in reversed(points_list):
            build_hull(upper_hull, point)
        return lower_hull[:-1] + upper_hull[:-1]

    if len(points) < 3:
        return points
```

```
    mid = len(points) // 2
    left_half_hull = convex_hull(points[:mid])
    right_half_hull = convex_hull(points[mid:])

    return divide_and_conquer(left_half_hull + right_half_hull)
```

OUTPUT:



Time Complexity Analysis:

In this implementation, the main function calls the recursive function, which performs the divide and conquer step. The function divides the points into smaller halves and then merges the convex hulls of the two halves. This process continues until all points are considered.

Sorting, it will take O(n logn)

Dividing into halves and merging will take T(n/2) time each hull.

T(n)<= O(n long) + 2T(n/2)

=2T(n/2)+c n logn Using induction case 2 log=k

log2(2)=1

k=1

p=1

Gives :- O(n log2n)

So overall running time complexity for the above method is **O(n logn)**.