Convex Hull Brute Force

Quick Hull Elimation ALgorithm for Finding Convex Hull

The Quick Hull algorithm is a divide-and-conquer algorithm used to find the convex hull of a set of points in a 2D space. It works similarly to the QuickSort algorithm, hence the name. Here's a step-by-step explanation of how the Quick Hull algorithm operates:

Steps of the Quick Hull Algorithm

1. **Find the Extremes**:
   - Identify the points with the minimum and maximum x-coordinates. These points will always be part of the convex hull.
2. **Divide the Points**:
   - Divide the remaining points into two subsets: those that lie above the line formed by the two extreme points and those that lie below.
3. **Recursively Find Hull Points**:
   - For each subset, find the point that is farthest from the line formed by the two extreme points. This point will also be part of the convex hull.
   - Split the remaining points into two new subsets: points above and below the line formed by the new point and one of the extreme points.
   - Recursively repeat this process for each subset until no points are left.
4. **Combine the Hull Points**:
   - Combine all the points identified in the previous steps to form the convex hull.

Pseudocode

Here's a simple pseudocode representation of the Quick Hull algorithm:

```plaintext
function quickHull(points):
    if number of points < 3:
        return the points as the hull

    minPoint = point with minimum x-coordinate
    maxPoint = point with maximum x-coordinate

    hull = []
    hull.add(minPoint)
    hull.add(maxPoint)

    pointsAbove = []
    pointsBelow = []

    for each point in points:
        if point is above line (minPoint, maxPoint):
            pointsAbove.add(point)
        else if point is below line (minPoint, maxPoint):
            pointsBelow.add(point)

    findHull(pointsAbove, minPoint, maxPoint, hull)
    findHull(pointsBelow, maxPoint, minPoint, hull)

    return hull

function findHull(points, p1, p2, hull):
    if no points left:
        return

    farthestPoint = point farthest from line (p1, p2)
    hull.add(farthestPoint)

    pointsLeftOfLine1 = []
    pointsLeftOfLine2 = []

    for each point in points:
        if point is left of line (p1, farthestPoint):
            pointsLeftOfLine1.add(point)
        else if point is left of line (farthestPoint, p2):
            pointsLeftOfLine2.add(point)

    findHull(pointsLeftOfLine1, p1, farthestPoint, hull)
    findHull(pointsLeftOfLine2, farthestPoint, p2, hull)
```

Complexity

- **Time Complexity**: O(n log n) in the average case; O(n^2) in the worst case (when all points are collinear).
- **Space Complexity**: O(n) for storing the points.

Implementation

Here's a simple implementation in Python:

```python
def orientation(p, q, r):
    """Return the orientation of the triplet (p, q, r).
    0 -> p, q and r are collinear
    1 -> Clockwise
    2 -> Counterclockwise
    """
    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 quick_hull(points):
    if len(points) < 3:
        return points

    # Find the points with minimum and maximum x-coordinates
    min_point = min(points, key=lambda p: p[0])
    max_point = max(points, key=lambda p: p[0])

    hull = [min_point, max_point]

    # Split the points into above and below the line
    points_above = [p for p in points if orientation(min_point, max_point, p) == 2]
    points_below = [p for p in points if orientation(min_point, max_point, p) == 1]

    # Recursive calls to find the hull
    find_hull(points_above, min_point, max_point, hull)
    find_hull(points_below, max_point, min_point, hull)

    return hull

def find_hull(points, p1, p2, hull):
    if not points:
        return

    farthest_point = max(points, key=lambda p: distance_to_line(p1, p2, p))
    hull.append(farthest_point)

    points_left1 = [p for p in points if orientation(p1, farthest_point, p) == 2]
    points_left2 = [p for p in points if orientation(farthest_point, p2, p) == 2]

    find_hull(points_left1, p1, farthest_point, hull)
    find_hull(points_left2, farthest_point, p2, hull)

def distance_to_line(p1, p2, p):
    # Calculate the distance from point p to the line formed by points p1 and p2
    return abs((p2[1] - p1[1]) * p[0] - (p2[0] - p1[0]) * p[1] + p2[0] * p1[1] - p2[1] * p
1[0]) / ((p2[1] - p1[1])**2 + (p2[0] - p1[0])**2) ** 0.5

# Example usage:
points = [(0, 3), (2, 2), (1, 1), (2, 1), (3, 0), (0, 0), (3, 3)]
hull = quick_hull(points)
print("Convex Hull Points:", hull)
```

Conclusion

The Quick Hull algorithm is an efficient way to compute the convex hull of a set of points, especially in average cases. The key lies in its divide-and-conquer strategy, making it a popular choice in computational geometry.

provide a complete step by step solution of numerical question for creating convex hull through quick hull elimination method

Let’s work through a step-by-step example to find the convex hull of a set of points using the Quick Hull algorithm.

Given Points

Consider the following set of points in a 2D space:

$$P = \{(0,0), (1,2), (2,1), (3,3), (2,2), (4,0), (3,1)\}$$

Step-by-Step Quick Hull Algorithm

Step 1: Find Extremes

1. Identify the point with the minimum and maximum x-coordinates.
    - **Minimum point (minPoint)**: $(0, 0)$
    - **Maximum point (maxPoint)**: $(4, 0)$

Step 2: Initialize the Hull

2. Start the convex hull with the extreme points:
    - Hull = { (0, 0), (4, 0) }

Step 3: Divide Points

3. Divide the remaining points into two subsets based on their position relative to the line formed by (0, 0) and (4, 0):
    - **Above the line**:
        - (1, 2)
        - (2, 1)
        - (3, 3)
        - (2, 2)
        - (3, 1)
    - **Below the line**:
        - (None in this case)

Step 4: Find the Farthest Point Above the Line

4. From the points above the line, find the point that is farthest from the line segment between (0, 0) and (4, 0).

    To find the distance from a point to a line, use the formula:

    $$\text{Distance} = \frac{|(y_2 - y_1) \cdot x - (x_2 - x_1) \cdot y + x_2 \cdot y_1 - y_2 \cdot x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}$$

    - For the line segment (0, 0) to (4, 0):
        - $(x_1, y_1) = (0, 0)$
        - $(x_2, y_2) = (4, 0)$
    - Compute distances for each point above the line:
        - For (1, 2):

        $$\text{Distance} = \frac{|(0 - 0) \cdot 1 - (4 - 0) \cdot 2 + 4 \cdot 0 - 0 \cdot 0|}{\sqrt{(0 - 0)^2 + (4 - 0)^2}} = \frac{8}{4} = 2$$

        - For (2, 1):

        $$\text{Distance} = \frac{|(0 - 0) \cdot 2 - (4 - 0) \cdot 1 + 4 \cdot 0 - 0 \cdot 0|}{\sqrt{(0 - 0)^2 + (4 - 0)^2}} = \frac{4}{4} = 1$$

        - For (3, 3):

        $$\text{Distance} = \frac{|(0 - 0) \cdot 3 - (4 - 0) \cdot 3 + 4 \cdot 0 - 0 \cdot 0|}{\sqrt{(0 - 0)^2 + (4 - 0)^2}} = \frac{12}{4} = 3$$

- For (2, 2):

$$\text{Distance} = \frac{|(0-0)\cdot 2-(4-0)\cdot 2+4\cdot 0-0\cdot 0|}{\sqrt{(0-0)^2+(4-0)^2}} = \frac{8}{4} = 2$$

- For (3, 1):

$$\text{Distance} = \frac{|(0-0)\cdot 3-(4-0)\cdot 1+4\cdot 0-0\cdot 0|}{\sqrt{(0-0)^2+(4-0)^2}} = \frac{4}{4} = 1$$

- **Farthest point**: $(3, 3)$

Step 5: Update Hull

5. Add (3, 3) to the hull:
   - Hull = { (0, 0), (4, 0), (3, 3) }

Step 6: Divide Again

6. Now, divide the remaining points into two new subsets based on the line segments (0, 0) to (3, 3) and (3, 3) to (4, 0):
   - **Above line (0, 0) to (3, 3)**:
     - (1, 2)
     - (2, 1)
     - (2, 2)
   - **Above line (3, 3) to (4, 0)**:
     - (None)

Step 7: Find the Farthest Point for (0, 0) to (3, 3)

7. Calculate distances for the points (1, 2), (2, 1), and (2, 2):
   - For (1, 2):

$$\text{Distance} = \frac{|(3-0)\cdot 1-(3-0)\cdot 2+3\cdot 0-3\cdot 0|}{\sqrt{(3-0)^2+(3-0)^2}} = \frac{3}{3\sqrt{2}} = \frac{1}{\sqrt{2}} \approx 0.707$$

   - For (2, 1):

$$\text{Distance} = \frac{|(3-0)\cdot 2-(3-0)\cdot 1+3\cdot 0-3\cdot 0|}{\sqrt{(3-0)^2+(3-0)^2}} = \frac{6}{3\sqrt{2}} = \frac{2}{\sqrt{2}} = \sqrt{2} \approx 1.414$$

   - For (2, 2):

$$\text{Distance} = \frac{|(3-0)\cdot 2-(3-0)\cdot 2+3\cdot 0-3\cdot 0|}{\sqrt{(3-0)^2+(3-0)^2}} = 0$$

   - **Farthest point**: $(2, 1)$

Step 8: Update Hull

8. Add (2, 1) to the hull:
    - Hull = { (0, 0), (4, 0), (3, 3), (2, 1) }

Step 9: Check Remaining Points

9. Now we check the remaining points:
    - **For (0, 0) to (2, 1)**, the points left are (1, 2) and (2, 2):
        - Calculate distances to find the farthest point.
    - For (1, 2):

$$\text{Distance} = \frac{|(2-0)\cdot 1-(1-0)\cdot 2+2\cdot 0-1\cdot 0|}{\sqrt{(2-0)^2+(1-0)^2}} = \frac{0}{\sqrt{5}} = 0$$

    - For (2, 2):

$$\text{Distance} = \frac{|(2-0)\cdot 2-(1-0)\cdot 2+2\cdot 0-1\cdot 0|}{\sqrt{(2-0)^2+(1-0)^2}} = 0$$

   Both distances are 0, indicating that there are no additional points to add.

10. Finally, check the other segments. For (2, 1) to (3, 3), the remaining point is (2, 2):
    - Distance check shows (2, 2) is collinear and does not contribute to the hull.

Final Convex Hull

The final convex hull is:

$$\{(0,0),(4,0),(3,3),(2,1)\}$$

Conclusion

The points that form the convex hull of the given set using the Quick Hull algorithm are ((0,

Quick elimination for convex hull creation

The **Quick Hull** algorithm is a method to compute the convex hull of a set of points in a 2D space. It uses a divide-and-conquer approach similar to QuickSort. Here's a detailed breakdown of how to perform a Quick Hull operation to create a convex hull.

Step-by-Step Quick Hull Algorithm

Example Points

Let's consider the following set of points:

$$P = \{(0,0),(1,1),(2,2),(3,1),(1,3),(4,0),(3,3)\}$$

Step 1: Find Extremes

1. Identify the points with the minimum and maximum x-coordinates. These points will always be part of the convex hull.
   - **Minimum point (minPoint)**: $(0,0)$
   - **Maximum point (maxPoint)**: $(4,0)$

Step 2: Initialize the Hull

2. Start the convex hull with these extreme points:
   - Hull = { (0, 0), (4, 0) }

Step 3: Divide Points

3. Divide the remaining points into two subsets based on their position relative to the line segment formed by (0, 0) and (4, 0):
   - **Above the line**:
     - (1, 1)
     - (3, 1)
     - (1, 3)
     - (3, 3)
   - **Below the line**:
     - (2, 2)

Step 4: Find the Farthest Point Above the Line

4. For each point above the line segment, calculate the distance to the line. The point that is farthest from the line will also be part of the convex hull.

Distance Calculation

Using the formula for the distance from a point (x_0, y_0) to a line defined by two points (x_1, y_1) and (x_2, y_2):

$$\text{Distance} = \frac{|(y_2 - y_1)\cdot x_0 - (x_2 - x_1)\cdot y_0 + x_2\cdot y_1 - y_2\cdot x_1|}{\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}}$$

For the line segment from $(0,0)$ to $(4,0)$:

- The distances for the points above the line:

1. For (1, 1):

$$\text{Distance} = \frac{|(0-0)\cdot 1 - (4-0)\cdot 1 + 4\cdot 0 - 0\cdot 0|}{\sqrt{(0-0)^2 + (4-0)^2}} = \frac{4}{4} = 1$$

2. For (3, 1):

$$\text{Distance} = \frac{|(0-0)\cdot 3 - (4-0)\cdot 1 + 4\cdot 0 - 0\cdot 0|}{\sqrt{(0-0)^2 + (4-0)^2}} = \frac{4}{4} = 1$$

3. For (1, 3):

$$\text{Distance} = \frac{|(0-0)\cdot 1 - (4-0)\cdot 3 + 4\cdot 0 - 0\cdot 0|}{\sqrt{(0-0)^2 + (4-0)^2}} = \frac{12}{4} = 3$$

4. For (3, 3):

$$\text{Distance} = \frac{|(0-0)\cdot 3 - (4-0)\cdot 3 + 4\cdot 0 - 0\cdot 0|}{\sqrt{(0-0)^2 + (4-0)^2}} = \frac{12}{4} = 3$$

- **Farthest points**: (1, 3) and (3, 3) both have a distance of 3.

Step 5: Update the Hull

5. Add (1, 3) and (3, 3) to the hull:
   - Hull = { (0, 0), (4, 0), (1, 3), (3, 3) }

Step 6: Recursively Find Hull Points

Now, for each of the new line segments:

1. From (0, 0) to (1, 3)
2. From (1, 3) to (4, 0)
3. From (3, 3) to (4, 0)

Line Segment (0, 0) to (1, 3)

6. **Points Above**: Only the point (2, 2) is relevant.
   - Calculate the distance from (2, 2) to the line defined by (0, 0) and (1, 3):

$$\text{Distance} = \frac{|(3-0)\cdot 2 - (1-0)\cdot 2 + 1\cdot 0 - 3\cdot 0|}{\sqrt{(3-0)^2 + (1-0)^2}} = \frac{4}{\sqrt{10}} \approx 1.264$$

   - This point is included.

7. Add (2, 2) to the hull:
   - Hull = { (0, 0), (4, 0), (1, 3), (3, 3), (2, 2) }

Line Segment (1, 3) to (4, 0)

8. Check for points above this line. There are no additional points contributing to the hull.

Line Segment (3, 3) to (4, 0)

9. Check again, no additional points.

Step 7: Final Hull Construction

The final convex hull from the given points is:

$$\{(0,0),(1,3),(4,0),(3,3)\}$$

Summary of Quick Hull Steps

1. **Find the extreme points**: (min and max x-coordinates).
2. **Divide the remaining points** into those above and below the line defined by these extremes.
3. **Find the farthest point** from the line for the convex hull.
4. **Repeat** the process recursively for new segments until all points are processed.

Conclusion

The Quick Hull algorithm efficiently constructs the convex hull by dividing points into manageable subsets, identifying extremes, and applying geometric distance calculations. This step-by-step approach clearly illustrates the process of constructing a convex hull through the Quick Hull elimination method.

Quick elimination.
• Choose a quadrilateral Q or rectangle R with 4 points as corners.
• Any point inside cannot be on hull
4 ccw tests for quadrilateral
4 comparisons for rectangle

The Quick Elimination technique you're describing involves determining whether a point is inside or outside a quadrilateral or rectangle defined by its corners. Points that are inside do not contribute to the convex hull. Below, I'll explain how to implement this using counter-clockwise (ccw) tests for a quadrilateral and simple comparisons for a rectangle.

1. Quick Elimination for a Quadrilateral

Steps to Check if a Point is Inside a Quadrilateral

1. **Define the Quadrilateral**: Let Q be defined by four points A, B, C, D in counter-clockwise order.
   - $A(x_1, y_1)$
   - $B(x_2, y_2)$
   - $C(x_3, y_3)$
   - $D(x_4, y_4)$
2. **Counter-Clockwise Test**: Use the cross product to determine if a point $P(px, py)$ is on the left side (or counter-clockwise) of each edge of the quadrilateral.

   The cross product can be calculated using the formula:

$$\text{cross}(A,B,P) = (B.x - A.x)(P.y - A.y) - (B.y - A.y)(P.x - A.x)$$

If the result is greater than 0, point P is on the left (ccw); if less than 0, it's on the right (cw); if 0, it's collinear.

3. **Perform the Tests**: Check the point against all four edges of the quadrilateral:
   - **Edge AB**: Check $\text{cross}(A,B,P)$
   - **Edge BC**: Check $\text{cross}(B,C,P)$
   - **Edge CD**: Check $\text{cross}(C,D,P)$
   - **Edge DA**: Check $\text{cross}(D,A,P)$

   If all results are non-negative or all are non-positive, then point P is inside the quadrilateral. If not, it's outside.

Example for Quadrilateral

Let’s say we have the quadrilateral defined by points:

- $A(0,0)$
- $B(4,0)$
- $C(4,3)$
- $D(0,3)$

And we want to check if point $P(2,1)$ is inside.

Calculations:

1. **Check against edge AB**:

$$\text{cross}(A,B,P) = (4-0)(1-0) - (0-0)(2-0) = 4 > 0 \quad \text{(inside)}$$

2. **Check against edge BC**:

$$\text{cross}(B,C,P) = (4-4)(1-0) - (3-0)(2-4) = 6 > 0 \quad \text{(inside)}$$

3. **Check against edge CD**:

$$\text{cross}(C,D,P) = (0-4)(1-3) - (3-4)(2-4) = 8 > 0 \quad \text{(inside)}$$

4. **Check against edge DA**:

$$\text{cross}(D,A,P) = (0-0)(1-3) - (0-3)(2-0) = 6 > 0 \quad \text{(inside)}$$

Since all tests show that point P is on the left side of each edge, $P(2,1)$ is inside the quadrilateral.

2. Quick Elimination for a Rectangle

Steps to Check if a Point is Inside a Rectangle

1. **Define the Rectangle**: Let R be defined by two opposite corners:

- $A(x_1, y_1)$ (bottom-left)
- $C(x_3, y_3)$ (top-right)

2. **Compare Coordinates**: For a point $P(px, py)$ to be inside the rectangle, it must satisfy:

$$x_1 \leq px \leq x_3 \quad \text{and} \quad y_1 \leq py \leq y_3$$

3. **Perform Comparisons**: Check the following conditions:
   - Is $P.x$ greater than or equal to $A.x$?
   - Is $P.x$ less than or equal to $C.x$?
   - Is $P.y$ greater than or equal to $A.y$?
   - Is $P.y$ less than or equal to $C.y$?

Example for Rectangle

Let’s say we have the rectangle defined by points:

- $A(1, 1)$
- $C(3, 3)$

And we want to check if point $P(2, 2)$ is inside.

Calculations:

1. Check the conditions:
   - $1 \leq 2 \leq 3$ (True)
   - $1 \leq 2 \leq 3$ (True)

Since both conditions are satisfied, point $P(2, 2)$ is inside the rectangle.

Conclusion

Using Quick Elimination, you can efficiently determine whether a point is inside a quadrilateral or rectangle, helping to simplify the process of constructing a convex hull. Points found to be inside can be ignored when creating the convex hull.

Quick elimination.
• Choose a quadrilateral Q or rectangle R with 4 points as corners.
• Any point inside cannot be on hull
4 ccw tests for quadrilateral
4 comparisons for rectangle

Three-phase algorithm
• Pass through all points to compute R.
• Eliminate points inside R.
• Find convex hull of remaining points.

The Quick Elimination method you described is an efficient way to compute the convex hull of a set of points by first identifying and eliminating points that lie within a specified quadrilateral or rectangle. This approach utilizes a three-phase algorithm. Here's a detailed explanation of how to implement this method:

Quick Elimination Algorithm

Phase 1: Compute the Bounding Rectangle or Quadrilateral (R)

1. **Define the Quadrilateral or Rectangle**:
   - For a rectangle R, determine the four corners, typically the minimum and maximum coordinates of the set of points.
   - For a quadrilateral Q, you may select any four points as the corners.

2. **Compute the Rectangle (R)**:

   - Given a set of points, determine the minimum and maximum x and y coordinates:
     - $x_{min} = \min\{x_i | (x_i, y_i) \in P\}$
     - $x_{max} = \max\{x_i | (x_i, y_i) \in P\}$
     - $y_{min} = \min\{y_i | (x_i, y_i) \in P\}$
     - $y_{max} = \max\{y_i | (x_i, y_i) \in P\}$

   - Define the rectangle R by the corners:
     - Bottom-left: (x_{min}, y_{min})
     - Top-right: (x_{max}, y_{max})

Phase 2: Eliminate Points Inside the Rectangle (R)

1. **Check Points Against Rectangle (R)**:
   - For each point $P(px, py)$:
     - Check if it lies inside the rectangle:

$$x_{min} \leq px \leq x_{max} \quad \text{and} \quad y_{min} \leq py \leq y_{max}$$

   - If both conditions are satisfied, discard the point (it is inside the rectangle).

Example of Phase 1 and 2

Given a set of points $P = \{(1,2), (2,3), (3,1), (4,4), (0,0), (5,5)\}$:

1. **Compute R**:

   - $x_{min} = 0$
   - $x_{max} = 5$
   - $y_{min} = 0$
   - $y_{max} = 5$
   - Rectangle R corners:

    - Bottom-left: $(0, 0)$
    - Top-right: $(5, 5)$

2. **Eliminate Points Inside R**:

    - Check each point:
        - $(1, 2)$: Inside
        - $(2, 3)$: Inside
        - $(3, 1)$: Inside
        - $(4, 4)$: Inside
        - $(0, 0)$: On the edge (considered on the hull)
        - $(5, 5)$: On the edge (considered on the hull)

    - Remaining points for convex hull: **none**. Only $(0, 0)$ and $(5, 5)$ are on the hull.

Phase 3: Find Convex Hull of Remaining Points

Since we only have points on the edge of the rectangle R:

- The convex hull of the remaining points is the points that are at the corners of R or part of the hull.

Convex Hull Output

From the example, the convex hull consists of:

- $(0, 0)$
- $(5, 5)$

Three-Phase Algorithm Summary

1. **Compute Rectangle (R)**:
    - Find the minimum and maximum x and y coordinates to form rectangle R.
2. **Eliminate Points**:
    - Remove points inside the rectangle based on comparison checks.
3. **Find Convex Hull**:
    - Compute the convex hull using only the points that are outside of R.

Quadrilateral (Q) Implementation

If instead, a quadrilateral Q is defined by points A, B, C, D, you would perform ccw tests for each edge as described in the previous explanation:

1. Perform 4 ccw tests against each point.
2. Eliminate points based on the ccw tests.

Conclusion

The Quick Elimination algorithm allows for efficient computation of the convex hull by first narrowing down the candidate points through the use of bounding geometries (rectangle or quadrilateral). This method significantly reduces the complexity involved in convex hull algorithms, especially for larger datasets.

what if the point is lie on the outside of the rectangle

If a point lies outside the rectangle (or quadrilateral), it means that it has the potential to be part of the convex hull, depending on its position relative to the other points. Here's how you handle points that lie outside the rectangle during the Quick Elimination process:

Handling Points Outside the Rectangle

1. **Definition of Outside**:
    - A point $P(px, py)$ is considered outside the rectangle R defined by corners (x_{min}, y_{min}) and (x_{max}, y_{max}) if:

$$px < x_{min} \quad \text{or} \quad px > x_{max} \quad \text{or} \quad py < y_{min} \quad \text{or} \quad py > y_{max}$$

    - In other words, the point does not satisfy the conditions for being inside the rectangle.

2. **Retention of Points Outside**:
    - Points that are found to be outside the rectangle are retained for further processing, as they may contribute to the convex hull.

Phase 3: Finding the Convex Hull

After you have eliminated the points inside the rectangle, you proceed to compute the convex hull using the remaining points, which now includes:

- Points that are exactly on the edges of the rectangle (which may be included in the hull).
- Points that are completely outside the rectangle.

Steps to Find the Convex Hull of Remaining Points

1. **Use Convex Hull Algorithms**:
    - You can apply algorithms like **Graham's scan**, **Jarvis's march (Gift wrapping)**, or **QuickHull** to compute the convex hull of the remaining points.
2. **Construct the Convex Hull**:
    - The algorithm will iterate through the remaining points, determining the outermost boundary that contains all of them.

Example

Let's say we have the following set of points, including some outside the rectangle:

Given points $P = \{(1, 2), (2, 3), (3, 1), (4, 4), (0, 0), (5, 5), (-1, -1), (6, 6)\}$:

1. **Compute the Rectangle R**:
   - From the previous example, R corners would be:
     - Bottom-left: $(0, 0)$
     - Top-right: $(6, 6)$

2. **Eliminate Points Inside R**:
   - Check points:
     - $(1, 2)$: Inside
     - $(2, 3)$: Inside
     - $(3, 1)$: Inside
     - $(4, 4)$: Inside
     - $(0, 0)$: On the edge (considered on the hull)
     - $(5, 5)$: On the edge (considered on the hull)
     - $(-1, -1)$: Outside (retained)
     - $(6, 6)$: On the edge (considered on the hull)
   - Remaining points for convex hull: $(-1, -1)$, $(0, 0)$, $(5, 5)$, $(6, 6)$.

3. **Compute Convex Hull**:
   - Use an algorithm to determine the convex hull:
     - The convex hull in this case would likely include the points $(-1, -1)$, $(0, 0)$, $(6, 6)$, and $(5, 5)$ as these define the outer boundary.

Conclusion

- **Points Outside the Rectangle**: Retain these points for convex hull calculations since they contribute to the outer boundary.
- **Points Inside the Rectangle**: Discard these points as they cannot be part of the convex hull.
- **Convex Hull Calculation**: Finally, apply an appropriate algorithm to find the convex hull from the remaining points, ensuring that you correctly include points on the edges and those identified as outside the rectangle.