In [5]:
```python
#Write a program that finds the closest pair of points in a set of 2D points
import math

def calculate_distance(point1, point2):
    """
    Calculate the Euclidean distance between two points.
    """
    return math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2

def closest_pair_brute_force(points):
    """
    Find the closest pair of points using the brute force approach.
    """
    min_distance = float('inf')9
    closest_pair = (None, None)
    n = len(points)

    for i in range(n):
        for j in range(i + 1, n):
            distance = calculate_distance(points[i], points[j])
            if distance < min_distance:
                min_distance = distance
                closest_pair = (points[i], points[j])

    return closest_pair, min_distance

points = [(2, 3), (12, 30), (40, 50), (5, 1), (12, 10), (3, 4)]
closest_pair, min_distance = closest_pair_brute_force(points)

print(f"The closest pair of points is: {closest_pair}")
print(f"The distance between them is: {min_distance}")
```

```
The closest pair of points is: ((2, 3), (3, 4))
The distance between them is: 1.4142135623730951
```

In [18]:
```
import matplotlib.pyplot as plt

def is_left_turn(p, q, r):
    """Returns True if r is to the left of line pq, False if r is to the rig
    return (q[0] - p[0]) * (r[1] - p[1]) - (q[1] - p[1]) * (r[0] - p[0])

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

    hull = []

    # Iterate through each pair of points
    for i in range(n):
        for j in range(i + 1, n):
            left_turn = None
            for k in range(n):
                if k == i or k == j:
                    continue
                turn = is_left_turn(points[i], points[j], points[k])
                if turn != 0:
                    if left_turn is None:
                        left_turn = turn
                    elif left_turn * turn < 0:
                        break
            else:
                hull.append((points[i], points[j]))

    # Filter unique points
    unique_points = set()
    for edge in hull:
        unique_points.add(edge[0])
        unique_points.add(edge[1])

    return list(unique_points)

def plot_convex_hull(points, hull_points):
    plt.figure()
    plt.scatter(*zip(*points), label='Points')
    for i, point in enumerate(points):
        plt.annotate(f'P{i+1}', (point[0]+0.2, point[1]-0.2))

    hull_points.append(hull_points[0]) # to close the hull
    plt.plot(*zip(*hull_points), label='Convex Hull')
    plt.legend()
    plt.show()

points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5),
hull_points = convex_hull(points)
hull_points = sorted(hull_points)
```

In [8]:
```python
#Write a program to find the closest pair of points in a given set using the
import math

def euclidean_distance(p1, p2):
    return math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)

def closest_pair(points):
    n = len(points)
    if n < 2:
        return None, float('inf')

    min_distance = float('inf')
    closest_points = None

    for i in range(n):
        for j in range(i + 1, n):
            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

# Example usage
points = [(10, 0), (11, 5), (5, 3), (9, 3.5), (15, 3), (12.5, 7), (6, 6.5),
closest_points, min_distance = closest_pair(points)
print(f"The closest pair of points is: {closest_points}")
print(f"The minimum distance is: {min_distance:.4f}")
```

```
The closest pair of points is: ((9, 3.5), (7.5, 4.5))
The minimum distance is: 1.8028
```

In [10]:
```python
def dice_throw(num_sides, num_dice, target):
    dp = [[0] * (target + 1) for _ in range(num_dice + 1)]

    dp[0][0] = 1

    for i in range(1, num_dice + 1):
        for j in range(1, target + 1):
            dp[i][j] = 0
            for k in range(1, num_sides + 1):
                if j - k >= 0:
                    dp[i][j] += dp[i-1][j-k]

    return dp[num_dice][target]

num_sides = 6
num_dice = 3
target = 8
print(f"Number of ways to get a sum of {target} with {num_dice} dice each ha
```

```
Number of ways to get a sum of 8 with 3 dice each having 6 sides: 21
```

In [11]:
```python
from itertools import combinations

def knapsack_exhaustive(weights, values, capacity):
    num_items = len(weights)
    max_value = 0
    best_combination = []

    # Iterate through all possible subsets of items
    for i in range(1, num_items + 1):
        for subset in combinations(range(num_items), i):
            total_weight = sum(weights[j] for j in subset)
            total_value = sum(values[j] for j in subset)

            # Check if this subset is the best one found so far
            if total_weight <= capacity and total_value > max_value:
                max_value = total_value
                best_combination = subset

    return best_combination, max_value

# Example usage
weights = [2, 3, 4, 5]
values = [3, 4, 5, 6]
capacity = 5

best_combination, max_value = knapsack_exhaustive(weights, values, capacity)
print(f"The best combination of items is: {best_combination}")
print(f"The maximum value is: {max_value}")
```

```
The best combination of items is: (0, 1)
The maximum value is: 7
```

In [ ]:
```python

```