Valid Square

Given the coordinates of four points in 2D space, return whether the four points could construct a square.

The coordinate (x,y) of a point is represented by an integer array with two integers.

Example:

```
Input: p1 = [0,0], p2 = [1,1], p3 = [1,0], p4 = [0,1]
Output: True
```

Note:

1. All the input integers are in the range [-10000, 10000].
2. A valid square has four equal sides with positive length and four equal angles (90-degree angles).
3. Input points have no order.

Solution 1

If we calculate all distances between 4 points, 4 smaller distances should be equal (sides), and 2 larger distances should be equal too (diagonals). As an optimization, we can compare squares of the distances, so we do not have to deal with the square root and precision loss.

Therefore, our set will only contain 2 unique distances in case of square (beware of the zero distance though).

```
int d(vector<int>& p1, vector<int>& p2) {
    return (p1[0] - p2[0]) * (p1[0] - p2[0]) + (p1[1] - p2[1]) * (p1[1] - p2[1]);
}
bool validSquare(vector<int>& p1, vector<int>& p2, vector<int>& p3, vector<int>& p4)
{
    unordered_set<int> s({ d(p1, p2), d(p1, p3), d(p1, p4), d(p2, p3), d(p2, p4), d(
p3, p4) });
    return !s.count(0) && s.size() == 2;
}
```

written by votrubac original link here

Solution 2

Just find the square of lenghts, and validate that

1. There are only two equal longest lenghts.
2. The non longest lengths are all equal.

```
public boolean validSquare(int[] p1, int[] p2, int[] p3, int[] p4) {
    long[] lengths = {length(p1, p2), length(p2, p3), length(p3, p4),
            length(p4, p1), length(p1, p3),length(p2, p4)}; // all 6 sides

    long max = 0, nonMax = 0;
    for(long len : lengths) {
        max = Math.max(max, len);
    }
    int count = 0;
    for(int i = 0; i < lengths.length; i++) {
        if(lengths[i] == max) count++;
        else nonMax = lengths[i]; // non diagonal side.
    }
    if(count != 2) return false; // diagonals lenghts have to be same.

    for(long len : lengths) {
        if(len != max && len != nonMax) return false; // sides have to be same lengt
h
    }
    return true;
}
private long length(int[] p1, int[] p2) {
    return (long)Math.pow(p1[0]-p2[0],2) + (long)Math.pow(p1[1]-p2[1], 2);
}
```

written by jaqenhgar original link here

Solution 3

Number of unique distances should be 2. (4 for sides, and 2 for diagonals)

```
class Solution(object):
    def validSquare(self, p1, p2, p3, p4):
        points = [p1, p2, p3, p4]

        dists = collections.Counter()
        for i in range(len(points)):
            for j in range(i+1, len(points)):
                dists[self.getDistance(points[i], points[j])] += 1

        return len(dists.values())==2 and 4 in dists.values() and 2 in dists.values
()

    def getDistance(self, p1, p2):
        return (p1[0] - p2[0])**2 + (p1[1] - p2[1])**2
```

written by aditya74 original link here