```
# Closest Pair

import sys
import os
import math
import argparse

# find the distance between two points
def fake_euclid(x, y):
    """
    Find a value given two points that's a multiple of the distance between the
    two points. Good for comparing distances where the actual value doesn't
    matter.
    """
    return (x[0] - y[0])**2 + (x[1] - y[1])**2

def closest_pair(points):
    """
    Find the closest pair of points given a list of points.
    In: [(x,y), (x,y), ...], where (x,y) are the points.
    Out: (i,j), where i and j are the index of the points.
    """
    x_ind = 0
    y_ind = 1

    # a recursive function to do the work
    def closest_pair_recurse(by_x, by_y):
        """
        Points sorted by x and y, and the span of the points on the x-axis)
        """
        # end cases
        if len(by_x) == 1:
            return None
        elif len(by_x) == 2:
            return (by_x[0], by_x[1])

        # divide
            # find a midpoint by looking at the middle x value
        mid = int(len(by_x) / 2)
        mid_point = by_x[mid]

            # find all the sorted point indexes for each side
        left_by_x = by_x[:mid]
        left_by_y = filter(lambda i: points[i][x_ind] < points[mid_point][x_ind], by_y)

        right_by_x = by_x[mid:]
        right_by_y = filter(lambda i: points[i][x_ind] >= points[mid_point][x_ind], by_y)

        # conquer
        l_pair = closest_pair_recurse(left_by_x, left_by_y)
        r_pair = closest_pair_recurse(right_by_x, right_by_y)

        # combine
            # find which side has the smaller distance pair
        try:
            l_dist = fake_euclid(points[l_pair[0]], points[l_pair[1]])
        except TypeError:
            l_dist = float("inf") # if one point, then infinite distance
        try:
            r_dist = fake_euclid(points[r_pair[0]], points[r_pair[1]])
        except TypeError:
            r_dist = float("inf")

        if l_dist < r_dist:
            dist = l_dist
            closest_pair = l_pair
        else:
            dist = r_dist
            closest_pair = r_pair

            # find the strip in the middle within the distance
        y_strip = filter(lambda i: abs(points[left_by_x[-1]][x_ind] - points[i][x_ind])
                         < dist, by_y)

            # Loop through all the points in the strip and compare
        for key, val in enumerate(y_strip):
            # loop through the next 15 elements
            for i in xrange(key+1, key+1+15):
                try:
                    d = fake_euclid(points[val], points[y_strip[i]])
                    if d < dist:
                        dist = d
                        closest_pair = (val, y_strip[i])
                except IndexError:
                    pass

        return closest_pair

    # sort by x and y, but only store the indices
    by_x = range(len(points))
    by_x.sort(key=lambda x: points[x][x_ind])
    by_y = range(len(points))
    by_y.sort(key=lambda x: points[x][y_ind])

    # return the correct values
    c = closest_pair_recurse(by_x, by_y)

    # map back to the point x,y values
    return tuple(points[i] for i in c)

def slow_closest_pair(points):
    """
    Finds the closest pair using a brute force algorithm.  Slower than the above
    algorithm, but it's simpler so it's handy for testing the above algorithm.
    """
    dist = float('inf')
    closest_pair = None
    for x in points:
        for y in points:
            if x != y:
                d = fake_euclid(x, y)
                if d < dist:
                    dist = d
                    closest_pair =(x, y)
    return closest_pair

def parse_points_file(file_name):
    f = open(file_name)
    if f:
        points = []
        for l in f:
            try:
                points.append(tuple(map(int, l.split(','))))
            except:
                pass
        return points
    else:
        return []
```

```python
def plot(points, closest=None):
    import matplotlib.pyplot as plt

    # setup the figure
    fig = plt.figure()
    ax = fig.add_subplot(111)

    # plot the points
    x, y = zip(*points)
    ax.plot(x, y, 'ro')

    # plot the closest pair
    if closest is not None:
        x, y = zip(*closest)
        ax.plot(x, y, 'go-')

    # set additional settings and show plot
    ax.grid()
    ax.set_xlim(-5,10)
    ax.set_ylim(-5,10)
    ax.set_title('Closest Pair')
    plt.show()

def main():
    # parse command line arguments
    parser = argparse.ArgumentParser(
        description="Find the closest pair of points given many points.")
    parser.add_argument('--graph', dest='graph', # nargs='?',
                        const=True, default=False, action='store_const',
                        help='Graph the points and closest pair.')
    parser.add_argument('filename', metavar='FILE', type=str,
                        help='The file to parse the points out of. Expects a '+\
                             'file of lines with comma seperated x/y values.')
    args = parser.parse_args()

    # open the file and run the algorithm
    points = parse_points_file(args.filename)

    if points:
        #print "Working on points: " + repr(points)
        closest_points = closest_pair(points)

        # really_closest_points = slow_closest_pair(points)
        # if fake_euclid(*closest_points) != fake_euclid(*really_closest_points):
        #     raise Exception("There is a problem with the closest pair algorithm")

        print "Closest: " + repr(closest_points)

        if args.graph:
            plot(points, closest_points)
    else:
        print "Can't parse points file."

if __name__ == "__main__":
    main()
```