CSE 101 - Introduction to Programming

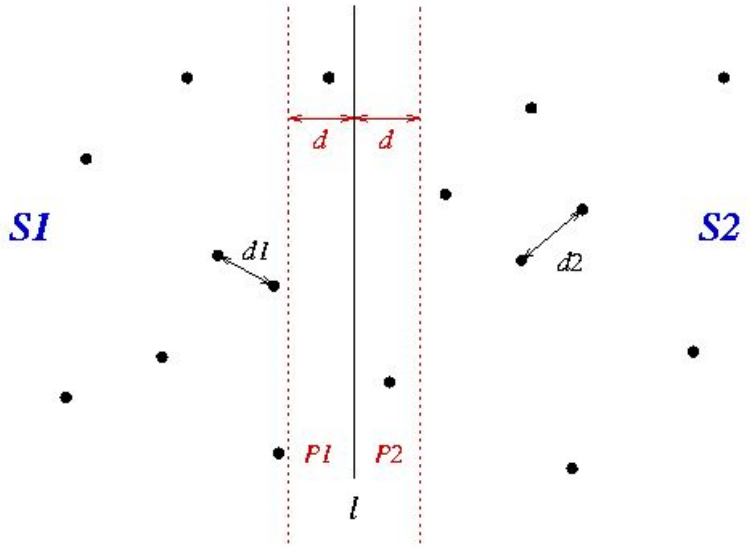
Assignment 3

Task:
Given a list of points, write a program to find the pair of closest points. If there are multiple such pairs, then find any one. Assume there are at least two given points.
A point p1 is closer to point p2 than point p3 if the euclidean distance between p1 and p2 is less than the euclidean distance between p1 and p3.

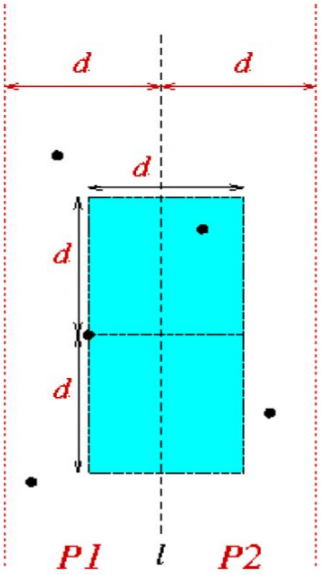
There can be multiple approaches to find such a closest pair of points. We'll look at two approaches.

Approach 1:
A naive brute-force way to find the closest pair of points is to compare each point with all other points. If the distance between two points is less than the minimum distance found so far, we update the minimum distance and continue comparisions.

Approach 2 [Source]**:**
A more efficient approach than the brute-force approach is based on divide and conquer algorithm. A divide-and-conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly [1].
Given a set of points `S` in the plane, we can partition the plane into two subsets `S1` and `S2` by a line `l`. Now, we can break our initial problem of finding the closest pair of points in `S` into two similar problems of finding the closest pair of points in `S1` and `S2`. We obtain `d1` (for points `p1_S1` and `p2_S1`) and `d2` (for points `p1_S2` and `p2_S2`) as the minimum distances in each subset. Let `d` be minimum of `d1` and `d2`.



There may exist a points `p1_S1` and `p1_S2` such that distance between these points is less than `d`. To find such a point, we focus at a strip around the line `l`. `p1_S1` and `p1_S2` can only exist in the region with distance `d` around `l`. However, we can narrow down the region of interest even more. For any point `p` in one of the sides of `l` in strip we have to only check those points within `d` of `p` in the direction of the other strip and those within `d` of `p` in the positive and negative y directions. It just so happens that because every point in this box is at least `d` apart, there can be at most six points within it [2].



We can simply sort the points in the strip by their y-coordinates and scan the points in order, checking each point against a maximum of 6 of its neighbors. If any pair of points is closer than **d** then we update our smallest distance.
We repeat this procedure everytime we break out set of points **S** into two subsets in our divide and conquer approach.

Overview of this efficient approach:

1. Divide the set into two equal sized parts by the line **l**, and recursively compute the minimal distance in each part.
2. Everytime we divide the points into two sets:
    a. Let **d** be the minimal of the two minimal distances.
    b. Eliminate points that lie farther than **d** apart from **l**.
    c. Sort the remaining points according to their y-coordinates.
    d. Scan the remaining points in the y order and compute the distances of each point to its five neighbors.
    e. If any of these distances is less than **d** then update **d**.

Some sample test cases with expected results:

- Points: `[(86, 65), (44, 33), (26, 38), (86, 67), (94, 42), (63, 69)]`
  Closet Pair: `[(86, 65), (86, 67)]`
  Distance: `2.0` units

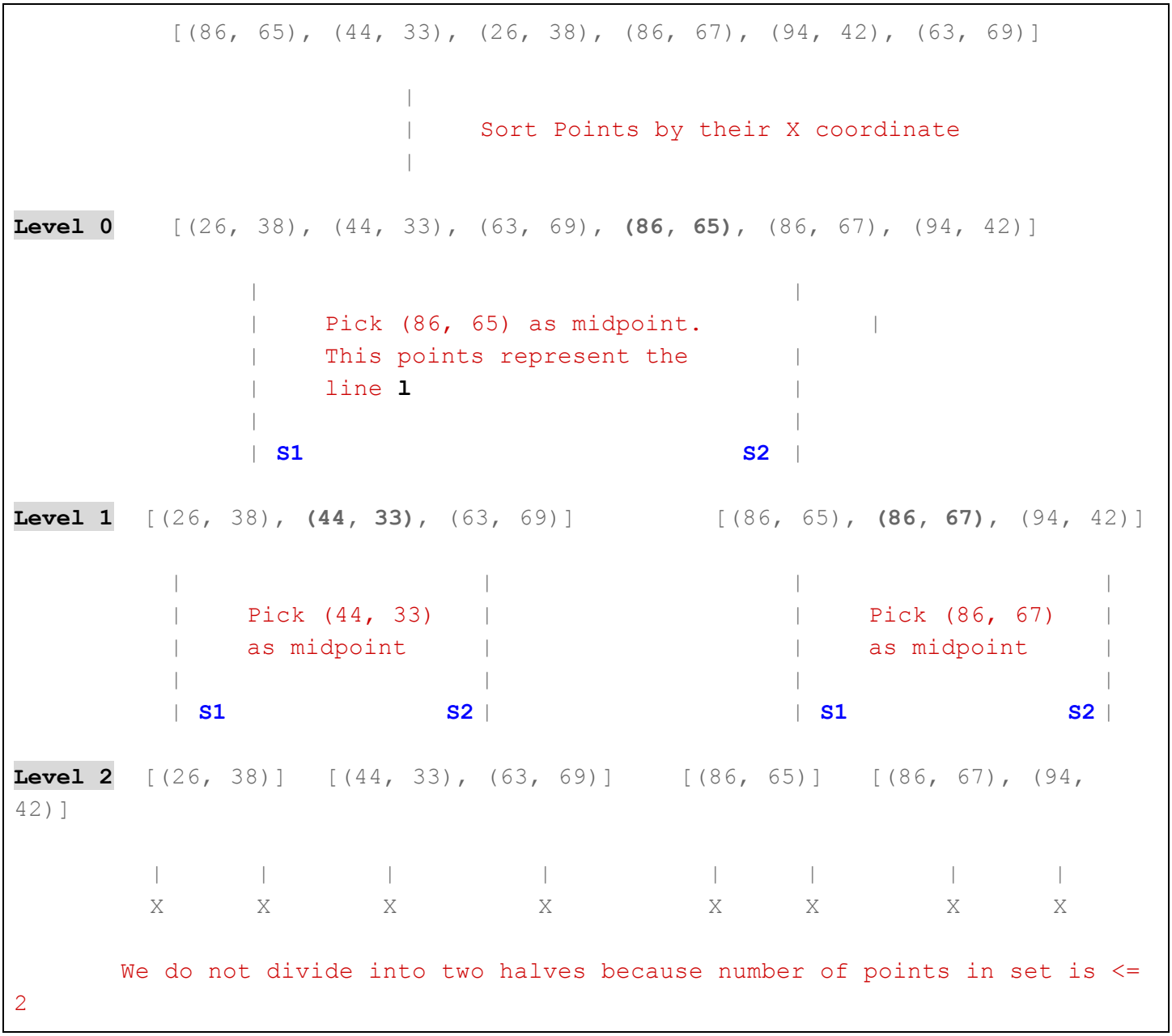
- Points: `[(42, 27), (51, 87), (69, 86), (69, 20), (63, 74), (67, 51), (26, 53), (6, 13), (98, 63), (7, 30)]`
  Closet Pair: `[(69, 86), (63, 74)]`
  Distance: `13.4` units

- Points: `[(5, 5), (5, 10), (5, 4)]`
  Closest Pair: `[(5, 5), (5, 4)]`
  Distance: `1.0` units

Let's consider the following example:
We are given points `[(86, 65), (44, 33), (26, 38), (86, 67), (94, 42), (63, 69)]` and we have to find the closest pair of points.

First we sort the given points according to their x coordinate. We pick the midpoint from the points in consideration and imagine line **l** such that it divides the points into two parts: points to the left of midpoint (**S1**) and the rest of the points (**S2**).

At level 0, we pick `(86, 65)` as midpoint and define **S1** as `[(26, 38), (44, 33), (63, 69)]` and **S2** as `[(86, 65), (86, 67), (94, 42)]`. We keep dividing the points into two sets **S1** and **S2** untill we cannot do it any further.



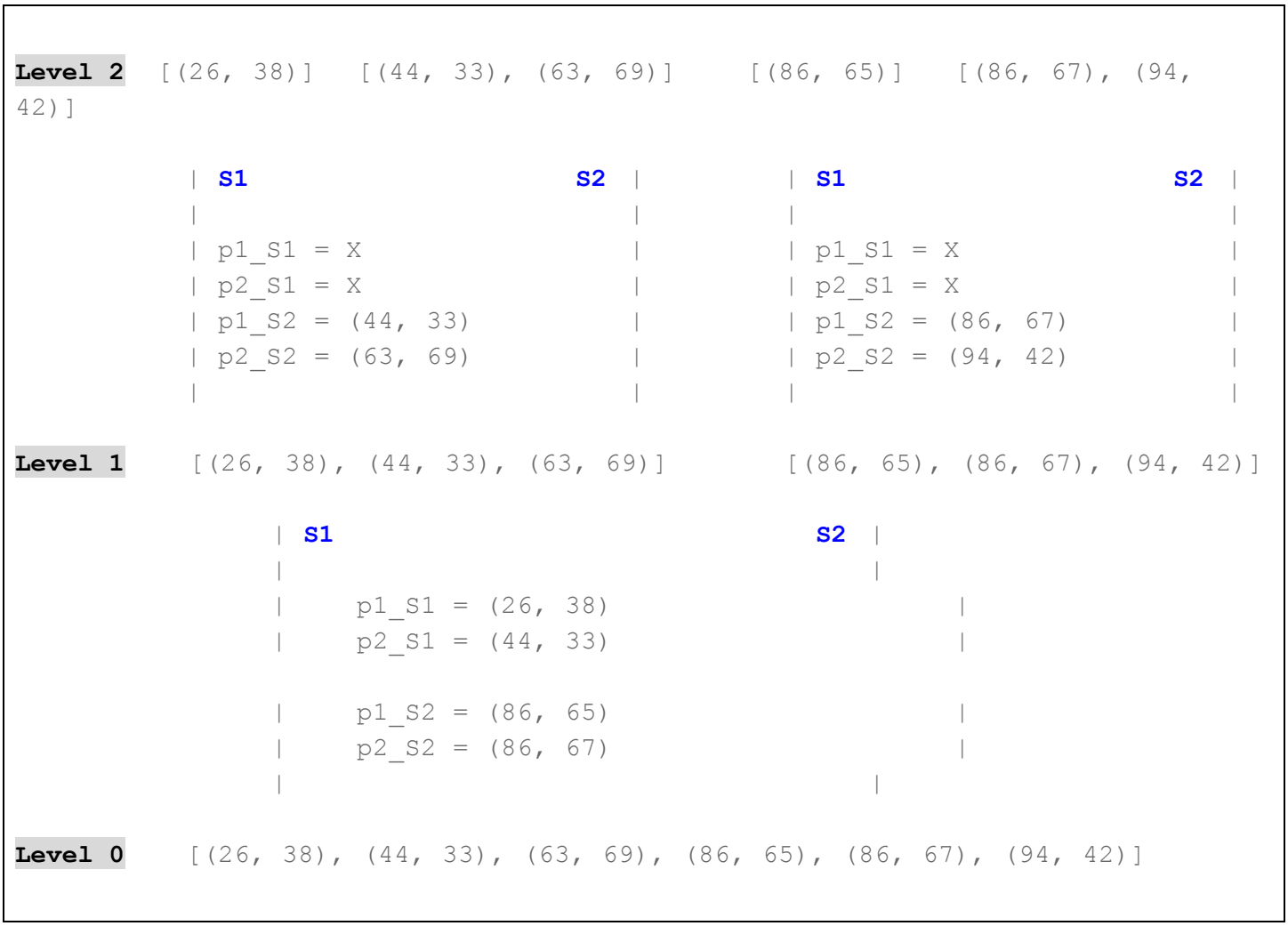
After partitioning the set of points into two subsets at each level, we look at how **d1** and **d2** are determined. On level 2, consider the set **S1** = `[(26, 38)]` and **S2** = `[(44, 33), (63, 69)]`. **d1** cannot be

computed as there is only one point in **S1** and **d2** is distance between **p1_S2** (44, 33) and **p2_S2** (63, 69), the only two points in **S2**.
For these sets **d** = min(**d1**, **d2**) = **d2**. However, when we form the strip region, we find that the distance between (26, 38) and (44, 33) is less than **d**. We update the value of **d** and the closest pair of points in combined region **S1** and **S2** is [(26, 38), (44, 33)].

Similary, for **S1** = [(86, 65)] and **S2** = [(86, 67), (94, 42)], **d1** cannot be computed and **d2** is distance between **p1_S2** (86, 67) and **p2_S2** (94, 42). After observing the strip region, we find the closest pair of points to be [(86, 65), (86, 67)].

At level 1, **S1** = [(26, 38), (44, 33), (63, 69)] and **S2** = [(86, 65), (86, 67), (94, 42)]. **d1** is distance between [p1_S1, p2_S1] = [(26, 38), (44, 33)] and **d2** is distance between [p1_S2, p2_S2] = [(86, 65), (86, 67)], the closest pair in sets **S1** and **S2** at this level respectively.

While going from level 1 to level 0, we have **d1** = 18.68 units and **d2** = 2 units. Therefore, **d** for this level becomes min(**d1**, **d2**) = 2 units.
There will be no points in the strip because there are no points **d1_S1** and **d1_S2** such that distance between them is less than **d**. Therefore, the closest pair of points for given points is [(86, 65), (86, 67)] with 2 units distance between them.



You are provided with a template code. Complete the functions present in the template code to implement Approach 1 and Approach 2 for finding closest pair of points.