3D closest pair algorithm

Preprocess
Sort the points individually by x, y and z dimensions. We get three ordered arrays: points_x[n], points_y[n], and points_z[n]

Divide

1. Separate the points into two set according to x dimension. The separate plane should separate the set evenly, and because we already sorted the points by x dimension, it is Θ(1) cost step
2. Make two recursive calls to our separated two set of points, get the closest pair distance ε_L, and right distance ε_R.

Conquer

1. And let ε be the closest pair distance of the input points set, set ε = min(ε_L, ε_R)
2. Just like 2D case, we do not need to consider points outside of the space between two planes centered by and parallel to the separation plane, and the distance between both planes to separation plane is ε. Let p is the points inside the two bound.
3. Then we could use points_y as order reference to get the posible closest pair. In 2D case we could compare p with its 8 neighbors in points_y, but in 3D case it should be more, but still O(1) neighbors. The proof of it is another topic, we could just use a constant C to denote the max number of neighors we need to examine. We enumerate every p in points_y, update ε if we find a closer pair. And the same process happens in s z dimension. This process cost Θ(n) time.

Time complexity

The recursive model is
$T(n) = T(n/2) + n$
Then $T(n) = \theta(n\log n)$ according to Master theorem.