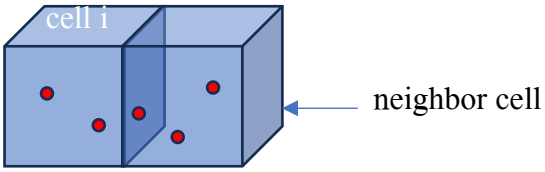
Possible approaches:

1. **The "brute force"** approach, with a double nested loop on the point set, is not considered feasible.
    - Time Complexity: **O(N^2)**. For each of the N points, we need to compare it with all other N-1 points to calculate the distance, resulting in N*(N-1) comparisons.
    - Space Complexity: **O(1)**. No extra data structures are required other than the input array of points and the result list of pairs.

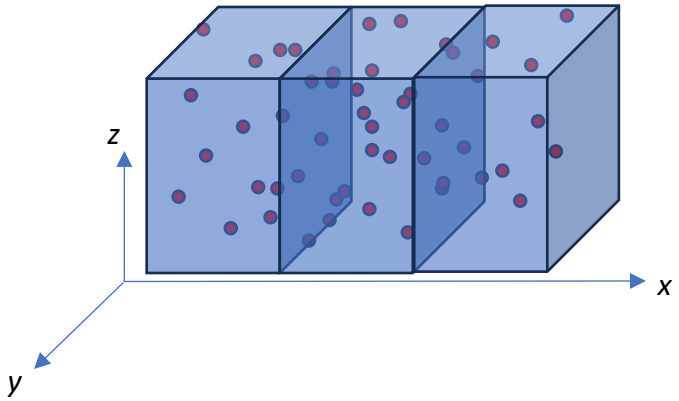
2. **Spatial Hashing (Grid-based Approach).** One can think of subdividing the space in cubes and creating a hash table that assigns points to cells. Then, one could search in parallel in each cell the pair of points located at a distance smaller than a given threshold. The neighbors' cells must be explored as well, since some points might be located at a distance that is very close to the cell considered. This is shown in the scheme below. "Cell i" is the current cell.



    - Time Complexity: **O(N).** Constructing the spatial hash map takes linear time, as we only need to consider each point once and assign it to the appropriate cell. For each point, we only need to check the neighboring cells, and the number of cells we need to check is limited by the distance threshold.
    - Space Complexity: **O(N).** The spatial hash map requires the space to store the cells and the points assigned to each cell.

2. **KD-Tree**. Another option is to use KD-tree data structure that allows for an efficient search of the nearest neighbor. This is what I am going to implement.
    - Time Complexity: **O(N Log N).** Constructing the spatial hash map takes linear time, as we only need to consider each point once and assign it to the appropriate cell. For each point, we only need to check the neighboring cells, and the number of cells we need to check is limited by the distance threshold.
    - Space Complexity: **O(N).** The KD-Tree data structure requires space to store the points and internal nodes of the tree.

In any case the number of points (10^7) is too large. Therefore, I am going to split the set of points in chunks along the x-direction. Then I analyze “chunk by chunk”. The image below represents 3 chunks of points enclosed by blue boxes.



Finally, I consider each interface between the chunks (see figure below for the interface definition). I define a small box across each of these interfaces. Within each box, I will find all the pairs of points at a distance smaller than the assigned distance. Then I will filter the result to remove the repeated pairs of points. The filter allows to keep only the pairs involving points located on both sides of the interface. The figure below shows the interfaces between the original chunks and the “volume of points” (blue color) considered for the new pair search.