

# The First Project Report

Samira Vaez Barenji

Student ID Number: 9921384

In this project, I have solved the famous problem of finding out the closest pair of points in three-dimensional space using Divide and Conquer algorithm. The Brute force solution for this problem is  $O(n^2)$ , which computes the distance between each pair and return the smallest. I reduce this time complexity to  $O(n \cdot (\log n)^2)$  using the Divide and Conquer strategy. The procedure of the algorithm is as follows:

The input is a list of  $n$  points and the output is the closest pair of points in the list and the corresponding distance.

First, the input array is sorted according to x coordinates and then the following steps are done:

- 1) Find the middle point in the sorted array and take  $n/2$  number of the input list as middle point.
- 2) Divide the given list in two halves. The first subarray contains points from index 0 to  $n/2$ . The second subarray contains points from index  $n/2+1$  to  $n-1$ .
- 3) Recursively find the smallest distances in both subarrays. We find the minimum of these two distances and call it  $d$ .
- 4) From the above 3 steps, we have an upper bound  $d$  of minimum distance. Now we need to consider the pairs such that one point in pair is from the left half and the other is from the right half. We consider the vertical line passing through the element with index  $n/2$  and find all points whose x coordinate is closer than  $d$  to the middle vertical line. Then, we build a list *strip* of all such points.
- 5) Sort the list *strip* according to y coordinates. This step is  $O(n \log n)$ . It can be optimized to  $O(n)$  by recursively sorting and merging.
- 6) Find the smallest distance in strip list. This is actually a  $O(n)$  step. It is proved geometrically that for every point in the *strip*, we need to check a

few fixed numbers of points after it. (Note that the *strip* is sorted according to Y coordinate)

- 7) Finally, we return the minimum of  $d$  and distance calculated in the above step (step 6).

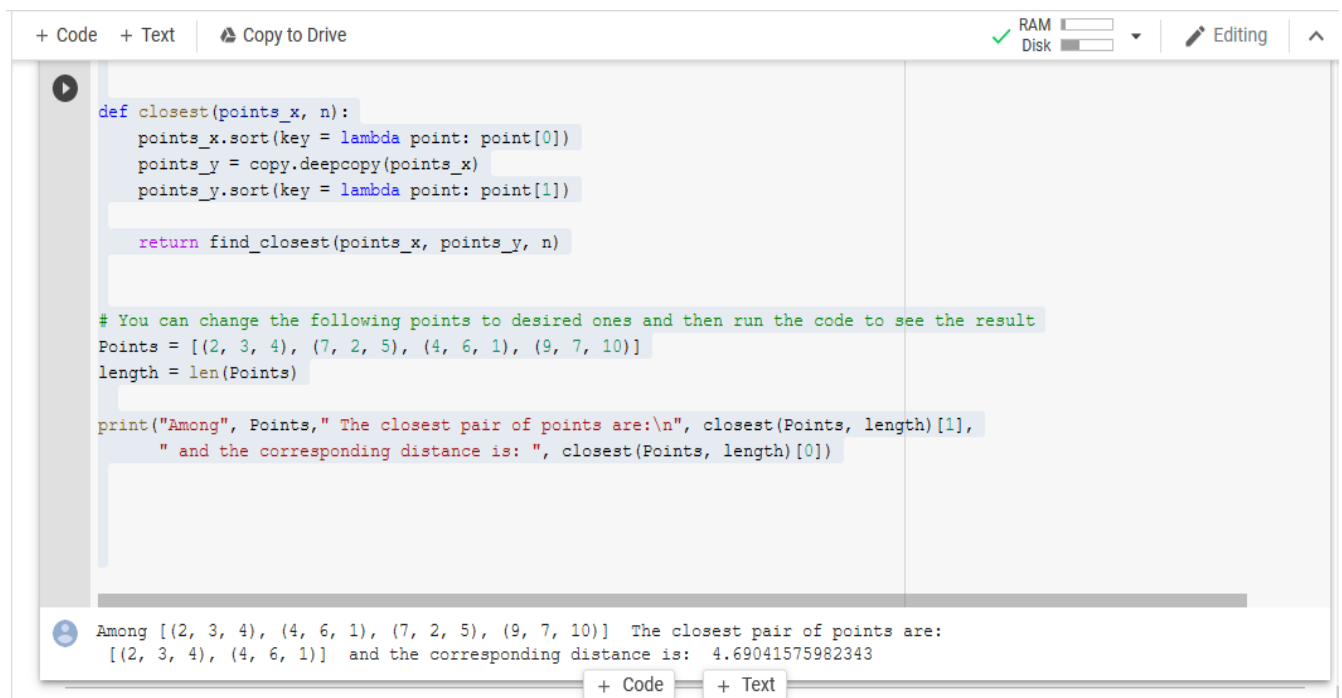
We call the time complexity of above algorithm  $T(n)$ . We suppose that the complexity of the sorting algorithm is  $O(n \log n)$ . The implemented algorithm divides all points in two sets and recursively calls for two sets. After dividing, it finds the *strip* in  $O(n)$  time, sorts the strip in  $O(n \log n)$  time and finally finds the closest points in strip in  $O(n)$  time. So,  $T(n)$  can be expressed as follows:

$$T(n) = 2T(n/2) + O(n) + O(n \log n) + O(n)$$

$$T(n) = 2T(n/2) + O(n \log n)$$

$$T(n) = T(n * \log n * \log n)$$

The code uses quick sort which can be  $O(n^2)$  in the worst case. A  $O(n \log n)$  sorting algorithm like merge sort or heap sort can be used to have the upper bound as  $O(n(\log n)^2)$ . The following picture shows the output of this algorithm for some random points.



```
+ Code + Text Copy to Drive RAM Disk Editing ^

def closest(points_x, n):
    points_x.sort(key = lambda point: point[0])
    points_y = copy.deepcopy(points_x)
    points_y.sort(key = lambda point: point[1])

    return find_closest(points_x, points_y, n)

# You can change the following points to desired ones and then run the code to see the result
Points = [(2, 3, 4), (7, 2, 5), (4, 6, 1), (9, 7, 10)]
length = len(Points)

print("Among", Points, "The closest pair of points are:\n", closest(Points, length)[1],
      " and the corresponding distance is: ", closest(Points, length)[0])

Among [(2, 3, 4), (4, 6, 1), (7, 2, 5), (9, 7, 10)] The closest pair of points are:
[(2, 3, 4), (4, 6, 1)] and the corresponding distance is: 4.69041575982343

+ Code + Text
```