# **CS-375 Assignment 2 Report:**

# **Closest Pair of Points (2D and 3D Implementation)**

**Student Name**: Alex Eskenazi

**Student ID**: B00944943

**Date**: September 29, 2025

## **Implemented**

* 2D/3D Brute force - finds the pair of points that is the closest comparing all pairs of points.
* 2D/3D Divide and Conquer - uses D&C to find the pair of points that is the closest.
* 3D - Optional Extra: if non-zero values are specified for Z coords then it works with 3D.

## **(1) Algorithm Description**

### **Brute Force Algorithm**

The brute force way works by checking every single pair of points to find which two are closest together. It goes through all the points and compares each one with every other point.

**Steps:**

1. Initialize minimum distance to a large value
2. For each point i from 0 to n-1:
   * For each point j from i+1 to n-1:
     + Calculate distance between points i and j
     + If distance < current minimum, update minimum and store indices
3. Return minimum distance and corresponding point indices

**Time Complexity:** O(n²) because we have to check every pair

**Space Complexity:** O(1) since we just store a few variables

### **Divide and Conquer Algorithm**

The divide and conquer method splits all the points in half, finds the closest pair in each half, then checks if there's a closer pair that crosses the middle.

**Steps:**

1. Sort points by x-coordinate and y-coordinate
2. Divide points into left and right halves at median x-coordinate
3. Recursively find closest pairs in left and right halves
4. Find minimum distance d from the two halves
5. Check strip of width 2d around dividing line for closer pairs
   * The strip contains points that are close to the dividing line
   * Only need to check points within distance d of the middle line
   * For each point in the strip, only check points that are within distance d in the y-direction
   * This is the tricky part because you might find a pair that's closer than what you found in the left/right halves
6. Return the overall minimum distance

**Time Complexity:** O(n log n) which is much better than n²

**Space Complexity:** O(n) because we need arrays for sorting

## **(2) Major Codes**

### **Distance Calculation Function**

****long double dist(const Point& a, const Point& b) {

long long dx = a.x - b.x;

long long dy = a.y - b.y;

long long dz = a.z - b.z;

return sqrt((long double)dx \* dx +

(long double)dy \* dy +

(long double)dz \* dz);

}

### **Brute Force Implementation**

****double bruteForceClosest(const vector<Point>& pts, int& bestI, int& bestJ) {

long double bestDist = LARGE\_NUMBER;

bestI = -1; bestJ = -1;

int n = (int)pts.size();

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

long double d = dist(pts[i], pts[j]);

if (d < bestDist) {

bestDist = d;

bestI = i;

bestJ = j;

}

}

}

return bestDist;

}

### **Divide and Conquer Implementation**

****double divideAndConquerClosest(const vector<Point>& pts, int& bestI, int& bestJ) {

vector<Point> Px = pts;

vector<Point> Py = pts;

std::sort(Px.begin(), Px.end(), compareX);

std::sort(Py.begin(), Py.end(), compareY);

return closestPairRec(Px, Py, bestI, bestJ);

}

## **(3) Running Results**

### **Test Case 1: Points Going Down Diagonally (n=10000)**

**Input:** Points like (10000,10000,10000), (9999,9999,9999), etc. going down

**Results:**

* Brute Force: distance = 1.73205, Time: ~330ms
* Divide & Conquer: distance = 1.73205, Time: ~20ms
* **Speed up:** about 16 times faster!

### **Test Case 2: Squared Coordinates (n=10000)**

**Input:** Points like (0,0,0), (1,1,1), (4,4,2), (9,9,3) where x and y are i²

**Results:**

* Brute Force: distance = 1.73205, Time: ~310ms
* Divide & Conquer: distance = 1.73205, Time: ~20ms
* **Speed up:** about 15 times faster!

### **Test Case 3: Circle/Spiral Pattern (n=20000)**

**Results:**

* Brute Force: distance = 1.41421, Time: ~1230ms
* Divide & Conquer: distance = 1.41421, Time: ~42ms
* **Speed up:** about 30 times faster!

### **Test Case 4: Mixed Positive/Negative Points (n=20000)**

**Results:**

* Brute Force: distance = 1.41421, Time: ~1206ms
* Divide & Conquer: distance = 1.41421, Time: ~42ms
* **Speed up:** about 29 times faster!

### **Performance Analysis**

The divide and conquer algorithm is much faster than brute force! You can really see the difference when there are more points. With 10000 points, divide and conquer is about 15 times faster, and with 20000 points it's like 30 times faster. This makes sense because brute force has to check way more pairs of points.

## **(4) Report of Any Bugs**

No known issues.

### **Current Status**

* Everything compiles and runs without errors
* Both algorithms give the same answers.
* The 3D version works and gets different distances than 2D
* All the test cases work and create the right input files
* Performance timing shows divide and conquer is way faster

The hardest part was definitely understanding how the divide and conquer algorithm works, especially the strip part. But once I got it working the performance difference is really impressive!