CS130B Project1 Report

Shunji Zhan

a)

```
void closestPair(n, X, Y) {
    minDistance = 999999999;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            distance = getDistance(X[i], Y[i], X[j], Y[j]);
            if (distance < minDistance) {
                minDistance = distance;
                x0 = X[i];
                y0 = Y[i];
                x1 = X[j];
                y1 = Y[j];
            }
        }
    }

    // print the results of x0, y0, x1, y1
}
```

The brute force algorithm went through all pairs of points that we have, and calculate distance of each pairs. Since there are n(n+1)/2 pairs, the comparison operation is n(n+1)/2, so the complexity is O(n(n+1)/2) = O(n^2)

b)

```
/*
    helper function: brute force method similar to question a)
    the returned result format:
    result[0] : x0
    result[1] : y0
    result[2] : x1
    result[3] : y1
    result[4] : minDistance
*/
vector bruteForce(n, X, Y) {}
```

```
/*
    the returned result format:
    result[0] : x0
    result[1] : y0
    result[2] : x1
    result[3] : y1
    result[4] : minDistance
*/
vector closestPair(n, X, Y) {
    if (n < 5) {
        // if n is small, use brute force directly.
        return bruteForce(n, X, Y);
    } else {
        array leftX, leftY;
        array rightX, rightY;
        half = getAverage(X);     // compute the average x coordiate

        // separate points into left and right parts
        for (int i = 0; i < n; i++) {
            if (X[i] < half) {
                leftX.push(X[i]);
                leftY.push(Y[i]);
            } else {
                rightX.push(X[i]);
                rightY.push(Y[i]);
            }
        }

        // divide and conqure left and right part
        vector leftResult = closestPair(leftX.size(), leftX, leftY);
        vector rightResult = closestPair(rightX.size(), rightX, rightY);
        leftMin = leftResult[4];
        rightMin = rightResult[4];
        minDistance = min(leftMin, rightMin)

        // deal with points in the strip
        leftBound = half - minDistance；
        rightBound = hald + minDistance；

        for all pairs p1 and p2 {
            if (p1 and p2 both in the strip) {
                if (abs(p2.y - p1.y) < minDistance) {
                    dist = getDistance(p2, p1);
                    if (dist < minDistance) {
```

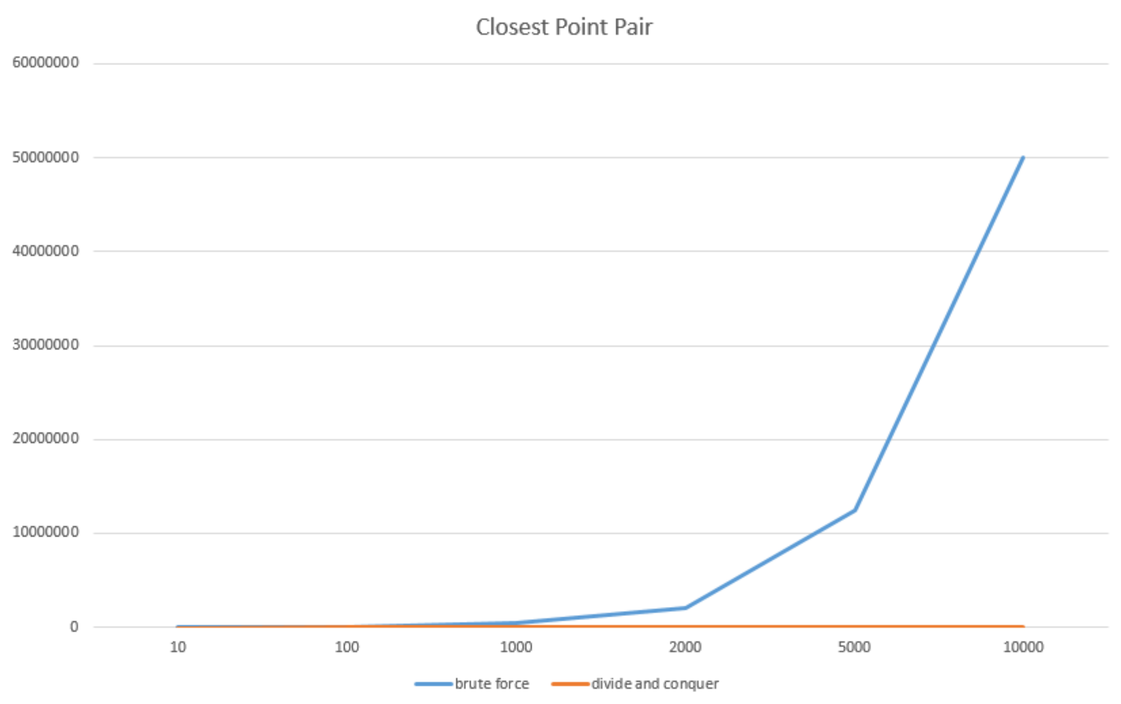
```
                                        minDistance = dist;
                                        x0 = p1.x;
                                        y0 = p1.y;
                                        x1 = p2.x;
                                        y1 = p2.y;
                                    }
                                }
                            }
                        }

                        // decide which result to return
                        vector result;
                        if (minDistance < leftMin && minDistance < rightMin) {
                            // result was in the middle strip
                            result.push(x0, y0, x1, y1, minDistance);
                            return result;
                        } else if (leftMin < rightMin) {
                            return leftResult;
                        } else {
                            return rightResult;
                        }

                    }
                }
```

This algorithm uses divide and conquer technics. If the points number are less than 5, use brute force directly. If there are many points, first separates them into two parts, according to whether their x value is greater than the average x, this will separate roughly to half-half. Then recursively call on these two halves, determine the closest pairs and minDistance so far. The only exception can be that there exist pairs in the middle strip that has even smaller distance, so we test all pairs in the middle strip [half - averageX, half + averageX] to determine if there is any closer points. A little trick was that we only calculate distance of p1 and p2 if there y value differs less than minDistance, this can reduce distance calculation a lot.

// each call consist of two half sized recursive calls, and when calculate the points in the strip, on average there are N^0.5 points in the strip, and there are around N/2 pairs of points in the strip. We can also reduce the distance calculation more by comparing y coordinates first, as mentioned above. So to deal with points in the strip, we can safely argue it is O(n). Thus for the whole function call T(N) = O(N) + 2 * T(N/2) = O(NlogN).


Closest Point Pair
60000000
50000000
40000000
30000000
20000000
10000000
0
10
100
1000
2000
5000
10000
brute force
divide and conquer