### **Required Analysis**

By:Drew Sadler, Daniel Mao

In addition to your code, you must create a written document addressing several aspects of your implementation and its efficiency.

* Provide a brief overview of your implementation, discussing any unusual design decisions, and disclosing any known problems with the correctness of your code.

Given array of n points

We start by sorting the x values from left to right, Then sorting the y values from top to bottom

Now that the list is sorted we call the recursive function solve with parameters const X sorted vector, sorted vector Y,start and stop which represents the indexes that the program is allowed to access into the const X vector.

If there are only two points left in the division of sides, then the base case applies, which is that those two points are considered the minimum.

But if the size is greater, then we separate the size of the x data in half, creating a left and right divide.

Then finding the best distance by taking the minimum of the distances of the left and right side

After, we will collect the points that are inside the strip and we will compare them with each other with another for-loop. This might sound inefficient but there should not be more than 4 points with the mini-square of the strip so it does not take a long time.

By going through all of the data points we collected in a loop, at the end of it we should be able to determine the minimum distance between the points of the left and right side inside the median/middle strip

* Give a more specific explanation of the data structures that you use for representing collections of points, and how this information is passed from one recursive level to another.

Passing to the solve function our x data, y data, and our plane coordinates. Allowing us to do recursion of comparisons to the middle vs the coordinates of the plane to then divide into 2 evenly split sides of their own data.

We pass a const vector with X sorted and a normal vector with Y sorted. When we know the indexes that indicate the right and left part, we create a new Y vector to insert the points of each side with the Y points already sorted.

Once we have our sorted left and right sides of data, we can recursively plug in our points and data, so we are able to find the distance squared of our points when we hit our base case.

After, we get the minimum distance from the left side and right side and get also the minimum of both in order to determine the length of the strip.

When we have the length of the strip we start to insert the points within the strip that we have already sorted by Y in another vector which will not contain a lot of points so it will not take too much memory.

We compute the minimum distance within our strip and compare it with delta in order to take our answer.

* Just as it is common for a merge-sort implementation to eventually revert to insertion sort for small enough data sets, you can optimize your closest-pair implementation by relying on the existing brute-force implementation once N falls below some threshold. Please explain what value you have chosen for such a threshold and what experimental data you gathered to support that decision.

From N 2 to 3 you see the first significant decrease due to efficiency in runtime. Going from 0.000744 ms to 0.0006 ms, And afterwards times exponentially grow as you can see in N 4 at 0.000637 ms to N 5 at 0.000853 ms. Giving us 3 as our base case of N, because it gives us our most efficient runtime and use of the brute force algorithm.

* Provide a table such as the following showing observed running times of your program for various choices of N. Report on both the brute-force and the divide-and-conquer solution, yet omitting values once running times get beyond 20 seconds or so.

| **N** | **brute-force** | **divide-and-conquer** |
| --- | --- | --- |
| 1,000 | 2.17583 ms | 3.1303 ms |
| 2,000 | 8.41198 ms | 8.83416 ms |
| 4,000 | 21.9418 ms | 34.8432 ms |
| 8,000 | 89.1201 ms | 24.1471 ms |
| 16,000 | 306.351 ms | 78.9079 ms |
| 32,000 | 1369.06 ms | 151.322 ms |
| 64,000 | 4921.08 ms | 560.94 ms |
| 128,000 | 19823.3 ms | 1676.34 ms |
| 256,000 | 82667.7 ms | 6820.58 ms |
| 512,000 | 327351 ms | 57413 ms |
| 1,000,000 | 1.95318e+06 ms | 104847 ms |
| 2,000,000 | Omitting because it takes more than 20s | Omitting because it takes more than 20s |
| 4,000,000 |  |  |
| 8,000,000 |  |  |
| 16,000,000 |  |  |
| 32,000,000 |  |  |

* For both brute-force and divide-and-conquer, provide a brief analysis of how the theoretical asymptotic performance of the algorithm is (or is not) reflected in your observed data.

Brute force O(n^2), as it uses a double for-loop to check all the points. As it shows its an exponential function, as its time increases exponentially with the input size.

Our algorithm has a linear time 𝑂(𝑛lg𝑛) as we are doing the sorting before calling the recursive function and we are working with a constant vector and its indexes.

We can see in the table that the divide and conquer algorithm is more efficient than the brute force algorithm when the number of points increases. We can start seeing the effectiveness when we have 8000 points where the divide and conquer algorithm takes way less time to run than brute force and we can see a trend that the higher the number of points the bigger the difference is between the running times of divide and conquer and brute force which follows our claim that one has O(n^2) and the other 𝑂(𝑛lg𝑛).