Robert Owens (rao7utn)
Trading (Advanced HW Divide and Conquer)

This problem requires finding the smallest possible distance between any two stars from a given collection of stars. These stars are represented as points in real three space ($\mathbb{R}^3$). It is assumed that the distribution of stars across the z-axis is tight to simplify this algorithm; this assumption is valid for this problem because Professor Florian told us to make it.

My approach was to first, read in the coordinates for the stars and store them in an array of a custom object data type: Point. This custom object Point has fields of doubles x, y, z which are the x, y and z coordinates of a star. This Point class also implements a static method to find the euclidian distance between 2 points. Once this array of Points is filled the array is sorted and passed to the main algorithm called 'gracefulForce' in my Trading.java file.

'GracefulForce' is a divide and conquer algorithm that is defined recursively with a base case of being passed an array of Points that is less than 100 Points in length, the base case will compute the minimum distance in this subset with a naïve brute force solution called 'bruteForce' which simply computes the distance from a point to all other points for every point of this subset (subset has cardinality less than 100). For the recursive case, 'gracefulForce' first divides the given array in half then calls 'gracefulFoce' reclusively on the left half of the input array and on the right half of the input array. This is the divide step of the divide and conquer.

Now assuming the subproblem on the left and right have been solved, take the smaller of those two solutions and call it delta. Find the x value of the split by finding the x value of the Point object that is the median of the array (this is trivial since the array is sorted in non-descending order by x coordinates). Next, create a new list of all points whose x coordinates lay between: $x\_split - \delta < p.x < x\_split + \delta$. This can be done by looping through all Points in the array that was passed into 'gracefulForce' and add the Points to the new list that satisfy the condition above.

Now finally, to check the strip pass the list of Points that exist in the strip and the delta value to a function 'walkThoughStrip'. This function will check to ensure no Points within the strip have a smaller distance between them than the given delta (the current smallest known distance). Checking the strip involves first, sorting the list of Points in the strip in non-descending order by the y value of the points. Then starting from the first Point in the sorted list (this will be the trailing point), check every Point ahead of it (this will be the leading point) while the difference in the y coordinate is less than delta. If the difference is beyond delta then break, and move the trailing Point up 1 position. Compute the distance between any two Points that do not exist more than delta apart in the y direction. If a distance between two Points is less than delta, change delta to equal the new smaller distance. At the end of this function return delta; which in the case of the initially recursive call, on the way up the stack is the correct answer for the smallest distance between any two Stars in the set. The correct answer is handled by a function called 'convertOutput' which formats the answer to the styling required for gradescope.

The runtime for this algorithm is worst-case: $\theta(n^2)$ and average-case: $\theta(n \cdot log^2 n)$. The algorithm preforms $\theta(n)$ work to set-up an array from the read in data. Then it preforms $\theta(n \cdot$

$logn$) to sort the array. In 'gracefulForce' dividing the array into left and right subarrays and then recursively calling 'gracefulForce' will require $\theta(logn)$ work. In the base-case where the problem is solved directly using a brute-force algorithm, is $\theta(1)$ work since the brute force must perform less than 100^2 operations. To create a list of all points within the strip requires $\theta(n)$ work since you must loop through the whole given array and check every point to see if it belongs. Then sorting the list of Points in the stripe is another $\theta(n \cdot logn)$ work which occurs in every recursive call. Finally, 'walkThroughStrip' can execute worst-case in $\theta(n^2)$ but will most often execute in $\theta(n)$. This is because in a situation where all points exist within the strip and exist closer together as y increases, the 'walkThroughStrip' method would preform $\theta(n^2)$ work. This is a very uncommon case and so it is more likely that 'walkThroughStrip' would execute in $\theta(n)$ time because each point would only need to check some small number of points ahead of it before exceeding delta. Hence the recurrence statement for this algorithm in the average case is: $T(n) = 2T(\frac{n}{2}) \cdot log(n)$ [The log(n) is to account for the additional sort of the strip points that occurs in 'gracefulForce']. Which becomes $\theta(n \cdot log^2 n)$ by the master theorem case 2.

No pseudocode is required, if the grader wishes they can view the Java code that is submitted to gradescope along with this pdf.