Drew Goldman
Trading - Description

The Trading solution requires a means by which to visualize the problem in three dimensions. The objective of the problem was to find the minimum three-dimensional distance, between two stars, among up to 100,000 stars in the Milky Way galaxy for one or more test cases. I approached this problem by recursively narrowing down a continuous interval of planes (a three dimensional region bounded on the left and right by variables *l* and *r*, respectively, along the x-axis) within which the solution could be found. I accomplished this by **first** sorting the vector of coordinates by monotonically increasing x-coordinate Θ(nlog(n)). **Then**, the method *routes*() recursively, via Θ(log(n)) time, split the interval of the range within which the solution may be until *l* - *r* <= 1000, **at which point** the algorithm would check every possible combination of points via a brute force $\Theta(n^2)$ method to find the smallest distance only within those intervals, *delta*. However, since the base case was only 1,000 elements, the brute force method can be simplified to Θ(1) because those approximate $1{,}000^2$ iterations were all but a large **constant**. Thus, the time to find *delta* is Θ(nlog(n)) because sorting asymptotically grows faster than recursion. **Once** *delta* was found, the algorithm inserts every coordinate triplet within the range [-delta,+delta] along the x-axis to a new vector of coordinates, *xD*. **Finally**, the algorithm calls the method *trading()* to narrow down the set of possible solutions. The method *trading()* sorts the new vector of coordinates, *xD*, by both the y-direction and and z-direction Θ(nlog(n)). The algorithm will **subsequently** utilize brute force to find all coordinates along the y and z axes that are less than *delta* away from each other to return the **smallest** distance. Since the list is already sorted, the second inner-loop will terminate after a constant number of comparisons, once a distance is found that exceeds delta, hence the search is Θ(n). Therefore, the overall time complexity of this algorithm is **Θ(nlogn)**, which can be represented by the recurrence *T(n) = 2T(n/2) + O(nlogn) + O(n)* *Note the use of sequential terms that indicate run time complexities that add together rather than multiply together.

No pseudocode is required, as the code is available via Gradescope.