Nikitha Jagadish

Project 2 Analysis

When N is equal to the size of the input (measured by the number of coordinates)

The running time of Brute as a function of N is N+(N)\*(N-1)\*(N-2)\*(N-3) which is **~N^4**.

Brute Proof:

This algorithm uses four nested for loops. Starting from the outermost for loop which iterates N times, each proceeding iterates one less time than the loop above it. Therefore, I multiply the time complexities of each for loop and get ~N^4.

The running time of Fast as a function of N is N+Nlog(N)+N\*(Nlog(N)+(N-1)\*N)+(N-1)+N = N^3 + N^2(log(N)) - N^2 + Nlog(N) + 3N -1 which is **~(N^3).**

Fast Proof:

The first N represents the while loop in the constructor that loads N coordinates into an array. These coordinates are added to Arrays.sort(), which is called after we have exited out of the while loop. Arrays.sort() has a worst-case time complexity of N\*log(N). After, we through a new while loop N times. Inside the while loop, we call another Arrays.sort(), which, again, will be having a worst-time complexity of N\*logN. Also inside the while loop, there is a for loop that gets iterated N-1 times. Inside the for loop, there is a while loop that, in the worst case scenario, will have a time complexity of N. After we exit the for loop and the while loop, we have another while loop that has a worst-case time complexity of (N-1). Lastly, we have a for loop that in the worst-case scenario, will be iterated through N times.

Plot:

|  |  |  |
| --- | --- | --- |
| **N** | **Brute-Force Time** | **Fast Time** |
| 10 | 0.021 | 0.022 |
| 20 | 0.021 | 0.023 |
| 50 | 0.024 | 0.03 |
| 75 | 0.029 | 0.036 |
| 100 | 0.032 | 0.044 |
| 150 | 0.057 | 0.077 |
| 200 | 0.077 | 0.123 |
| 300 | 0.173 | 0.327 |
| 400 | 0.344 | 0.465 |
| 500 | 1.015 | 0.815 |
| 1000 | 6.742 | 2.636 |
| 2000 | 88.336 | 11.731 |

Graph:

Using the graph, the runtime estimates for N=1,000,000 are:

Brute Runtime = 0.0313e^(0.0044(1,000,000)

Fast Runtime = 0.048e^(0.0032(1,000,000))