Assignment 1: Nearest Neighbour Report

Introduction:

An algorithm that is able to search for the nearest neighbours to a given point is one that is of most importance in the real world due to its efficiency in solving many distance problems between certain locations and ensuring the closest point is being returned for the user's benefits. This algorithm is essential in situations that call for a proximity search such as online food delivery applications suggesting the closest restaurant of the specified type of food or the closest educational institution to the user's home address. To begin, this report discusses the setup of the experiment with the different methods used for calculating k - nearest neighbours, generating the test case files and generating the input and output files for when those test cases were run against each data structure. There is also a detailed discussion about the theoretical time complexity of those operations against the actual run time performances recorded from the tests.

Setup:

To start, this report will delve into how this experiment was set up. This experiment needed the utilization of varying sizes of data sets to be trialed upon the kd tree data structure as well as the naive nearest neighbour implementation of the algorithm.

To generate the different datasets for this assignment, a java class was run for data generation (DataGen) which generated various datasets based upon a subset of variables which included small dataset, medium dataset, large dataset as well as varying latitude and longitude values for each data point generated. For each of those variables, the small data set was set to a size of 150 data points, while the medium dataset was set to a size of 600 points and finally the large data set ws set to a size of 900 points. The longitude values of each data point were between 110 and 180 while the latitude value of the data points varied between 60 and 100. The category of each data point could also vary between 'education', 'hospital' and 'restaurant'.

Scenario 1 (k-nearest neighbour searches):

In Scenario 1, what was required is the algorithms' performance in searching for k-nearest neighbours on each dataset. As seen in our code, unfortunately we were not able to use data generated to assist our research and theories on each algorithm's complexity regardless if the algorithm was complete. This in result meant, we had to estimate the execution time based on knowledge of similar data structures and the time complexity of the algorithms we implemented. In the Naive implementation, a time complexity of O(n^2) was shown by the search algorithm being used which by this standard can be used to hypothesise that as the dataset becomes larger, the longer it will take to search through the dataset. The KD tree implementation however due to it's recursive nature, has a more efficient time complexity in the search method.

K - Value	**Naive (small)**	**Naive (medium)**	**Naive (large)**	**KD- tree (small)**	**KD- tree (medium)**	**KD- tree (large)**
k<10	Very fast	fast	average	slow	slow	fast
k<50	average	slow	slow	average	average	fast
k<100	slow	Very slow	Very slow	fast	fast	Very fast

Table 1: Estimation of nanoseconds it takes to execute K-nearest neighbour searches on different datasets

(ns = nanoseconds)

Very Slow: >10000000 ns
Slow: 8000000 ns < **x** < 10000000 ns
Average: 7000000 ns < **x** < 8000000 ns
Fast: 5000000 ns < **x** < 7000000 ns
Very Fast: <5000000 ns

As seen in Table 1 above, what these rough estimates gathered by resources and knowledge of the time complexities of each algorithm indicate, is that the lower the datasize, the more efficient the Naive algorithm is over the KD tree however the larger the datasize, the more efficient the KD tree algorithm is over the Naive algorithm.

Scenario 2 (Dynamic points set):

Add/delete

	Naive (small)	**Naive (medium)**	**Naive (large)**	**KD- tree (small)**	**KD- tree (medium)**	**KD- tree (large)**
Execution Time	Very fast	Very fast	Very fast	Very slow	Very slow	Very slow

Table 2: Estimation of nanoseconds it takes to execute add-delete operations on different datasets

Very Slow: 70000 < ms <14
Very Fast: