Sample Structure of Report for Algorithms & Analysis Assignment 1

Student 1: Yi Yang S3798354

Student 2: Youzhe Liang S3813425

We certify that this is all our group’s original work. If we took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in our submission. We will show that we agree to this honour code by typing ``Yes": Yes.

Experimental Setup

Data generation:

To compile, from the directory, execute: > javac *.java >cd generation

To run, from the directory where DataGeneration.java is, execute:

> java DataGeneration.java [MAX of range] [MIN of range] [cat] [amount of data]

After running DataGeneration.java, data.txt will be automatically saved in the current directory. For the first scenario, only need to set the same MAX, MIN and Category, and change the Amount to generate different size of data sets. For the second scenario, ensure the same Category and the same Amount, and control the dispersion by reducing or expanding the range of coordinates as much as possible

Parameters to test on:

Input size, Data dispersion, Time consuming.

Reason:

By comparing the time consumption of the two algorithms in the case of different data input sizes and different data dispersion, we can summarize and analyse the influence of different parameters on the two algorithms, to choose a more suitable algorithm for the current scenario.

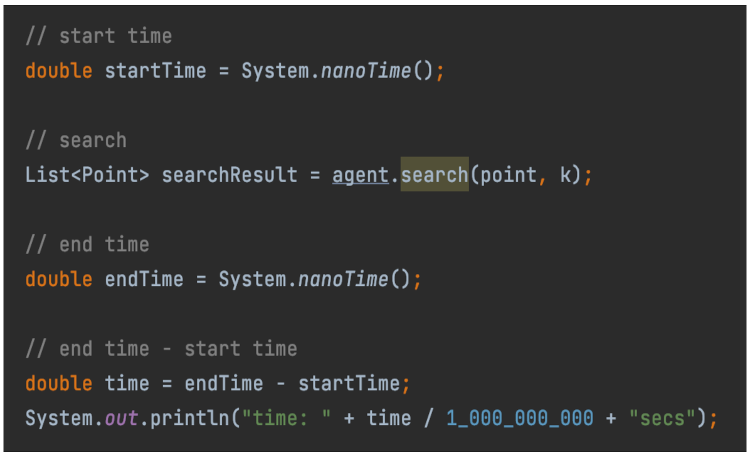
Generation of scenarios

Through consulting and researching the literature, we found that the efficiency of the KDTree KNN algorithm is not only affected by the size of the data input, but also has certain requirements for the data dispersion. So, we decided to generate two scenarios based on the above reasons:

1. (The K value is the same) The data input size gradually rises from a small amount to a large amount and analyse algorithm efficiency through time consuming.

2. Input two different degrees of dispersion data to these two algorithms (the K value is the same, the data is the same), and analyse algorithm efficiency through time consuming.

Timing: We are using java.lang.System.nanoTime() to get execution time. The example is for searching. (Figure 1). Adding and deleting used the same way.



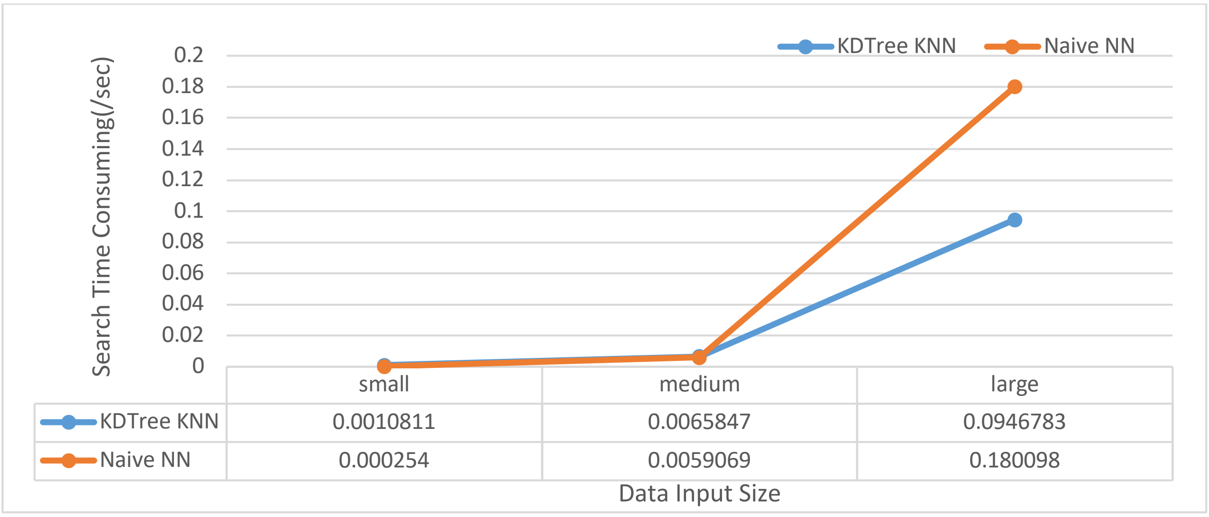
```
// start time
double startTime = System.nanoTime();

// search
List<Point> searchResult = agent.search(point, k);

// end time
double endTime = System.nanoTime();

// end time - start time
double time = endTime - startTime;
System.out.println("time: " + time / 1_000_000_000 + "secs");
```

Figure 1

Evaluation

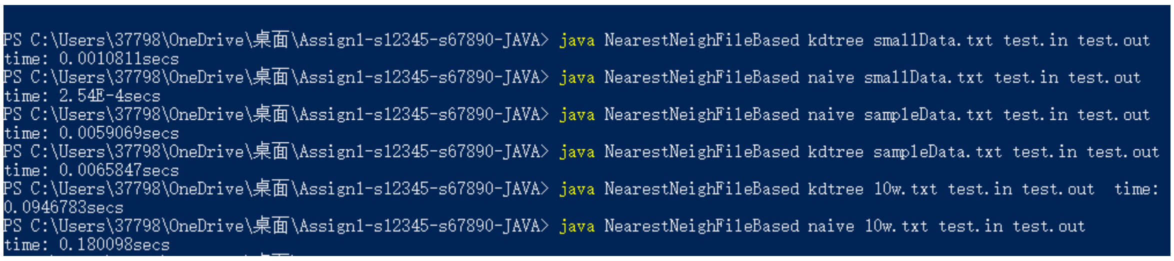
Scenario 1 (k-nearest neighbour search, addition and delete of different data input size)

We found that by the data input size increasing from small to large amount, the naive, brute force performance degraded (Figure 2 and Figure 3). We hypothesise the reason for this is that as input size increases, it takes longer to build index, as well as addition (Figure 4 and Figure 5), it will take more time to traverse. So as well the delete. We use Insertion sort to implement NaiveNN, the average case time complexity of Insertion sort is $O(n^2)$ and of KDTree is $O(n \log n)$, so when input size is not big enough, Naïve NN always have a better performance on search function.



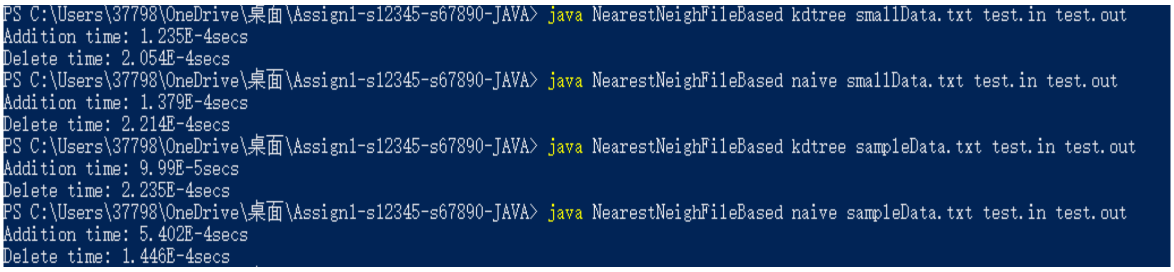
	small	medium	large
KDTree KNN	0.0010811	0.0065847	0.0946783
Naive NN	0.000254	0.0059069	0.180098

Figure 2



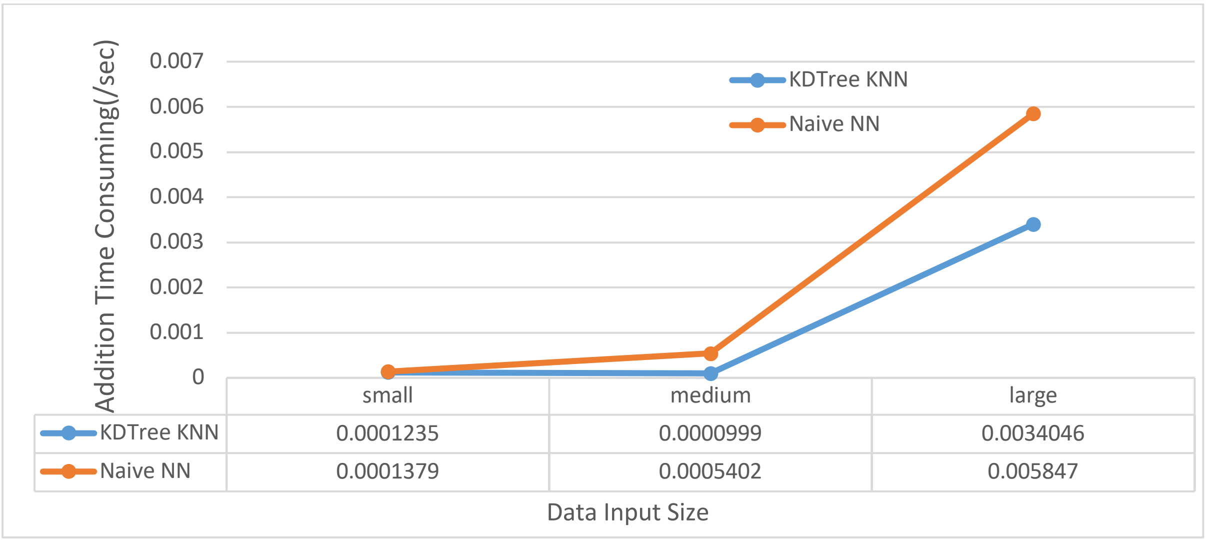
```
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased kdtree smallData.txt test.in test.out
time: 0.0010811secs
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased naive smallData.txt test.in test.out
time: 2.54E-4secs
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased naive sampleData.txt test.in test.out
time: 0.0059069secs
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased kdtree sampleData.txt test.in test.out
time: 0.0065847secs
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased kdtree 10w.txt test.in test.out  time:
0.0946783secs
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased naive 10w.txt test.in test.out
time: 0.180098secs
```

Figure 3



```
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased kdtree smallData.txt test.in test.out
Addition time: 1.235E-4secs
Delete time: 2.054E-4secs
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased naive smallData.txt test.in test.out
Addition time: 1.379E-4secs
Delete time: 2.214E-4secs
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased kdtree sampleData.txt test.in test.out
Addition time: 9.99E-5secs
Delete time: 2.235E-4secs
PS C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA> java NearestNeighFileBased naive sampleData.txt test.in test.out
Addition time: 5.402E-4secs
Delete time: 1.446E-4secs
```

Figure 4



	small	medium	large
KDTree KNN	0.0001235	0.0000999	0.0034046
Naive NN	0.0001379	0.0005402	0.005847

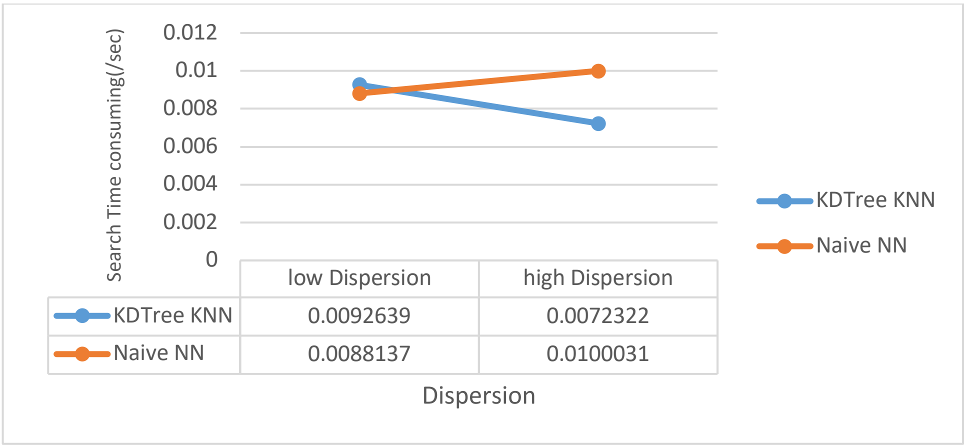
Figure 5

Scenario 2 (**k-nearest neighbour in different data dispersion**)

In this scenario, we found that by the input data dispersion increasing from low to high, KDTree has more efficiency. But on search function, when the input data has a low dispersion, NavieNN performs better. (see Figure 6 to Figure 9).

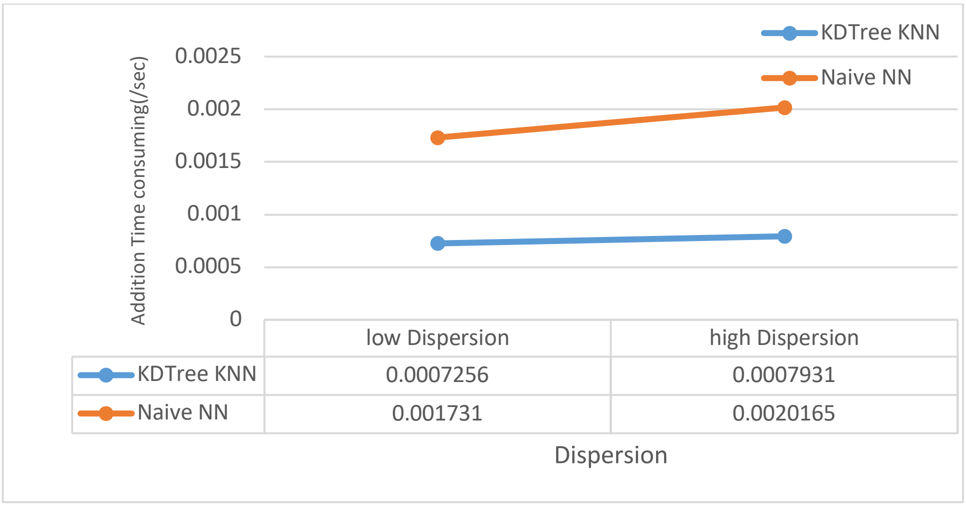
We use Insertion sort to implement NaiveNN could be the reason, because the best-case input is an array that is already sorted which has a linear running time (i.e., O(n)), and the simplest worst case input is an array sorted in reverse order, this gives insertion sort a quadratic running time (i.e., O(n^2)).

According to the line chart, we can also conclude that the efficiency of KDTree is greatly affected by the dispersion. The efficiency is lower when the dispersion is low.



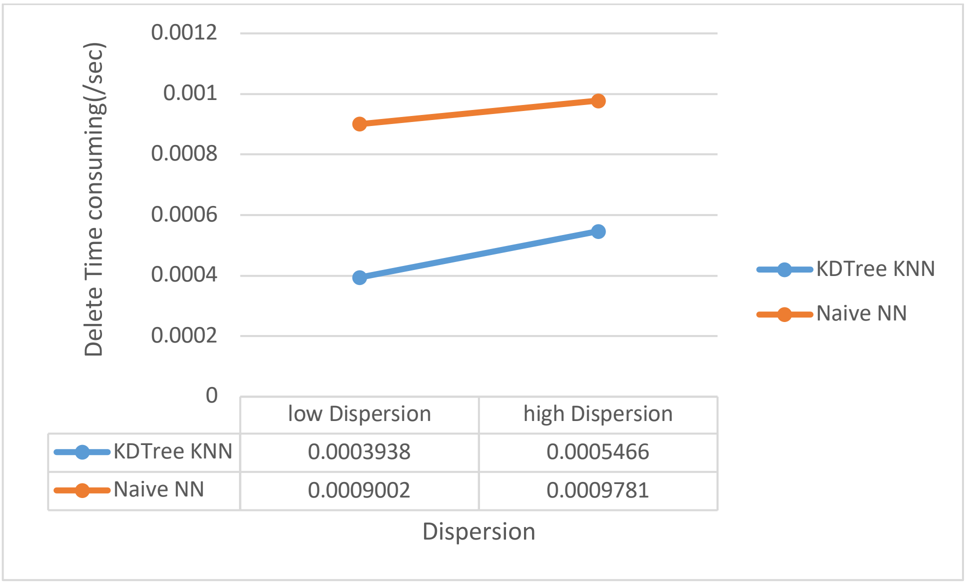
	low Dispersion	high Dispersion
KDTree KNN	0.0092639	0.0072322
Naive NN	0.0088137	0.0100031

Figure 6



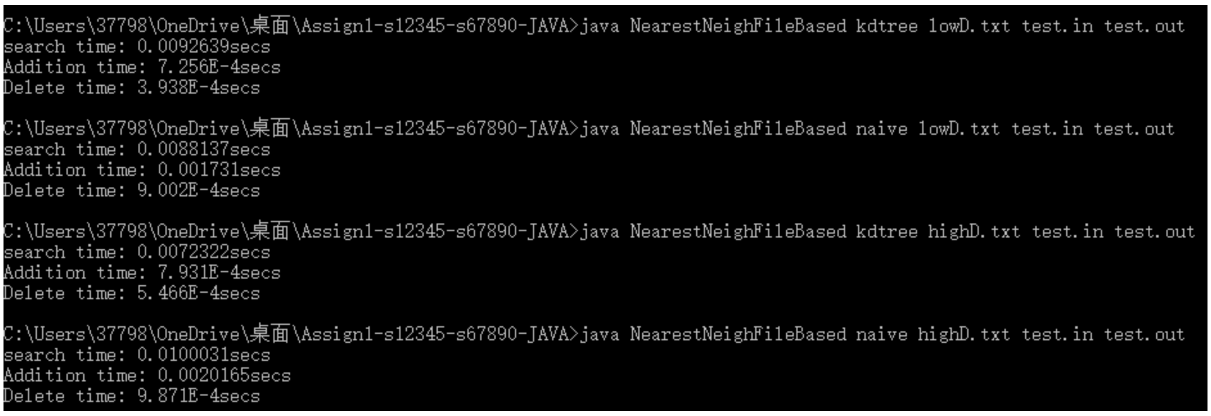
	low Dispersion	high Dispersion
KDTree KNN	0.0007256	0.0007931
Naive NN	0.001731	0.0020165

Figure 7



	low Dispersion	high Dispersion
KDTree KNN	0.0003938	0.0005466
Naive NN	0.0009002	0.0009781

Figure 8



```
C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA>java NearestNeighFileBased kdtree lowD.txt test.in test.out
search time: 0.0092639secs
Addition time: 7.256E-4secs
Delete time: 3.938E-4secs

C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA>java NearestNeighFileBased naive lowD.txt test.in test.out
search time: 0.0088137secs
Addition time: 0.001731secs
Delete time: 9.002E-4secs

C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA>java NearestNeighFileBased kdtree highD.txt test.in test.out
search time: 0.0072322secs
Addition time: 7.931E-4secs
Delete time: 5.466E-4secs

C:\Users\37798\OneDrive\桌面\Assign1-s12345-s67890-JAVA>java NearestNeighFileBased naive highD.txt test.in test.out
search time: 0.0100031secs
Addition time: 0.0020165secs
Delete time: 9.871E-4secs
```

Figure 9

Recommendation

According to the above conclusion, we can know:

1. When the data input size is small, we should choose Naive NN to get better data results and usage efficiency.
2. When the data input size is large, we should choose KDTree KNN to get better data results and usage efficiency.
3. When the data input size is small and the dispersion is low, we should choose NaiveNN to get better data results and use efficiency.
4. When the amount of data input is large and the degree of dispersion is low, we should compare both algorithms to obtain better data results and use efficiency.
5. When the amount of data input is small and the degree of dispersion is high, we should choose NaiveNN to get better data results and use efficiency. Because Insertion sort always has a better performance when data input size is small.
6. When the amount of data input is large and the degree of dispersion is high, we should choose KDTree KNN to obtain better data results and use efficiency.