Report for Algorithms & Analysis Assignment 1

Student : Ningthouba Meitei Nahakpam. S3733745

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:

Firstly, I have generated my data using my own implementation class called "DataGenerator" which is in the generation folder. The first agrs[0] will be the dataset output file name. But we need to manually modify the sample size of the dataset in DataGenerator class before generated.

In addition to this, datasets called "generated100Sample.txt" is for scenario k nearest neighbour search containing 100 points, for 1000 points it called "generated1000Sample.txt", and so on. Further, for the k nearest neighbour search test files are called "ownTest100_5k.in" for the 100 points and 5 neighbours, "ownTest1000_15k" for the 1000 points and 15 neighbours, and so on. Also, for the second scenario, the files are called "100DataS2.txt" which contains 100 initial points, "1000DataS2.txt" for the 1000 initial points, and so on. For the test, the file is called "10DataS2_AddDelete.in" which contains 10 points to add and delete. These are all located in generation folder.

Second, I have decided to generate three data samples. One of the data samples contains 100 points, the other one contains 1000 points, and the other last one contains 10000 points.

As the number of points increases our algorithm (i.e. naive or KD tree) will take more time for each operation (i.e add, search, delete). Also, The performance between naive and KD Tree will be distinguishable for operations.

Third, I have generated the data to be the first one will be the category of Restaurant, Hospital, and Education. Why? When searching for k neighbours will give more time as the points are not in order. What I mean by in order means when building the point would be (e.g Category of the first point is Restaurant, the second one is also the same category of the first, and so on).

Lastly, for the timing, I used the System.nanotime() method as this will give more accurate timing than the time command, and System.nanotime() will not evaluate the run time of my algorithms, not the program. But, I have to modify the source code to measure time using System.nanotime().

Further, I have performed three tests with different k value for each generated data set.

1) For the data set which contains 100 points (see appendix 1):
   i) **Naive approach**:
      a) The average of three tests when the k is 5 = **0.002535979** sec.
      b) The average of three tests when the k is 15 = **0.002447368** sec.
      c) The average of three tests when the k is 25 = **0.002690763** sec.
   ii) **KD Tree approach**:

a) The average of three tests when the k is 5 = **0.00083** sec
b) The average of three tests when the k is 15 = **0.00089** sec.
c) The average of three tests when the k is 25 = **0.00093** sec.

2) For the data set which contains 1000 points (see appendix 2):
   i) **Naive approach**:
      a) The average of three tests when the k is 5 = **0.053798158** sec.
      b) The average of three tests when the k is 15 = **0.05356729** sec.
      c) The average of three tests when the k is 25 = **0.052957225** sec.
   ii) **KD Tree approach**:
      a) The average of three tests when the k is 5 = **0.00083** sec.
      b) The average of three tests when the k is 15 = **0.00089** sec.
      c)The average of three tests when the k is 25 = **0.00095** sec.

3) For the data set which contains 10000 points (see appendix 3):
   i) **Naive approach**:
      a) The average of three tests when the k is 5 = **9.006098458** sec.
      b) The average of three tests when the k is 15 = **9.00915376** sec.
      c) The average of three tests when the k is 25 = **9.115328922** sec.

   ii) KD Tree approach:
      a) The average of three tests when the k is 5 = **0.001023333** sec.
      b) The average of three tests when the k is 5 = **0.001258771** sec.
      c) The average of three tests when the k is 5 = **0.001311535** sec.

Evaluation:

Scenario 1 (k-nearest neighbour searches)

I found that when the numbers of nearest neighbours increase (i.e. 5, 15, 25), k, the naive, brute force performance degrades (see Figure 1). We hypothesise the reason for this is that as k increases, it takes longer to check each point against the current k-nearest neighbour. Compare this to kd-tree performance see (Figure 2), kd-tree takes very little time due to kd-tree does not check each point in the dataset, but it checks only when a belongs to the search point radius area. See (figure 3) for performance comparison.

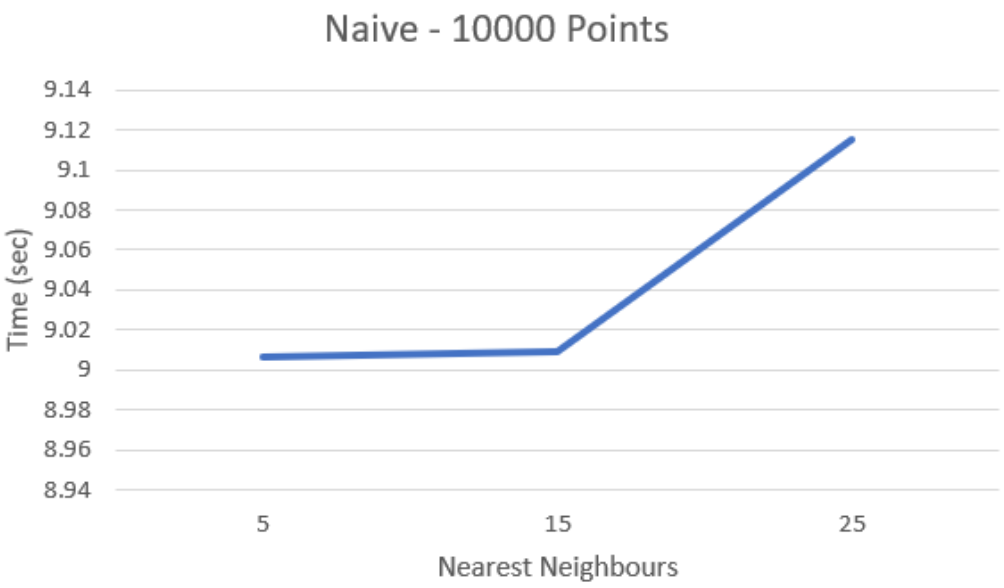


Figure - 1

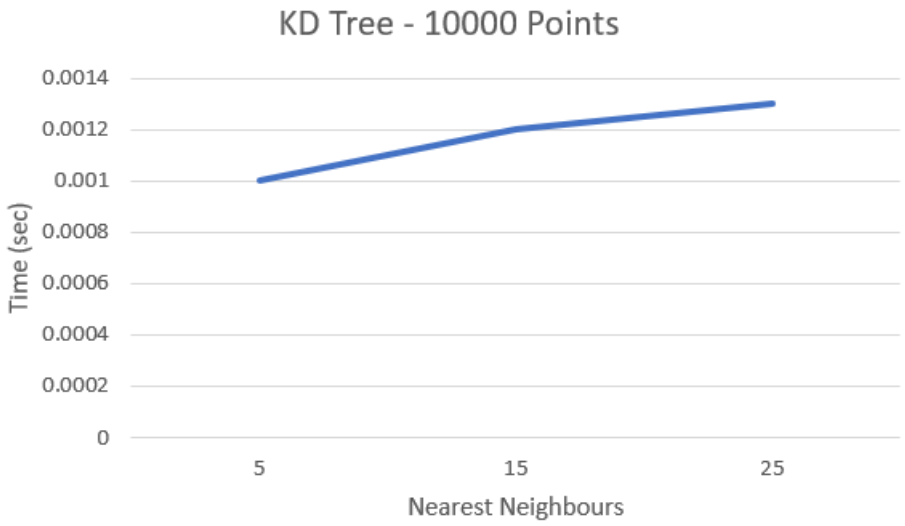


Figure - 2

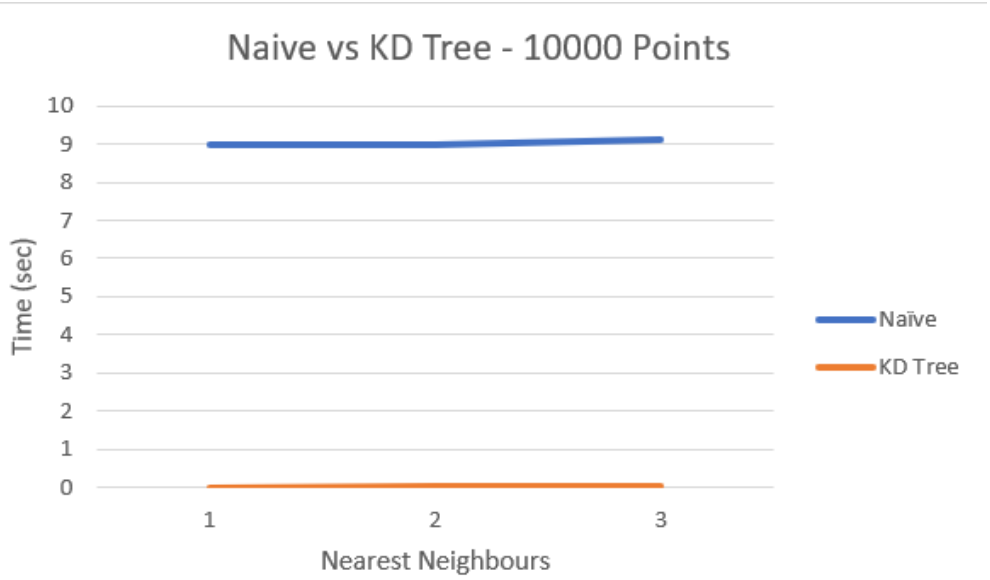


Figure – 3

As we can see that KD tree performance is much better than Naive. See (Figure 3)

Scenario 2 (Dynamic points set)

As I performed more adds and an equivalent number of deletions (i.e 10) to the kd-tree, I found that the run time is 0.000916523 sec for the initial data set of 1000 points, 0.000785839 sec for the initial data set of 10000 points.
In contrast, in the naive approach, the run time is 0.002274675 sec for the initial data set of 100 points, 0.055871884 sec for the initial data set of 1000 points, and 8.794378251 sec for the initial data set of 10000 points.
Therefore, for this scenario, the kdtree approach is much faster than the naive approach. But for the smaller initial data set, the performance between kdtree and naive approach is negligible. Whereas for the bigger initial data the kdtree approach is faster.

Recommendation:

For different scenarios, which data structures do you recommend to use?

1) For the scenario, when the data size is smaller, I recommend both approaches, that is, Naive or KD Tree.
2) For the scenario, when the data size is larger, I recommend the KD Tree approach.

Appendix:

1)

	Scenario - 1				
	Naive			**Kd Tree**	
Data Size	K	Time (in sec)	Data Size	K	Time (in sec)
100	5	0.002480153	100	5	0.00084
100	5	0.002434423	100	5	0.00082
100	5	0.00269336	100	5	0.00084
	Average	**0.002535979**		Average	**0.000833333**
	Naive			**Kd Tree**	
Data Size	K	Time (in sec)	Data Size	K	Time (in sec)
100	15	0.002474196	100	15	0.00093
100	15	0.002399244	100	15	0.00089
100	15	0.002468663	100	15	0.00085
	Average	**0.002447368**		Average	**0.00089**
	Naive			**Kd Tree**	
Data Size	K	Time (in sec)	Data Size	K	Time (in sec)
100	25	0.002647755	100	25	0.00099
100	25	0.002711546	100	25	0.00087
100	25	0.002712989	100	25	0.00093
	Average	**0.002690763**		Average	**0.00093**

2)

Data Size	K	Time (in sec)	Data Size	K	Time (in sec)
1000	5	0.053041714	1000	5	0.00082
1000	5	0.055050699	1000	5	0.00089
1000	5	0.05330206	1000	5	0.00078
	Average	**0.053798158**		Average	**0.00083**
	Naive			**Kd Tree**	
Data Size	K	Time (in sec)	Data Size	K	Time (in sec)
1000	15	0.053222315	1000	15	0.00093
1000	15	0.053502492	1000	15	0.00088
1000	15	0.053977064	1000	15	0.00087
	Average	**0.05356729**		Average	**0.000893333**
	Naive			**Kd Tree**	
Data Size	K	Time (in sec)	Data Size	K	Time (in sec)
1000	25	0.054379785	1000	25	0.00087
1000	25	0.050601326	1000	25	0.00088
1000	25	0.053890565	1000	25	0.0011
	Average	**0.052957225**		Average	**0.00095**

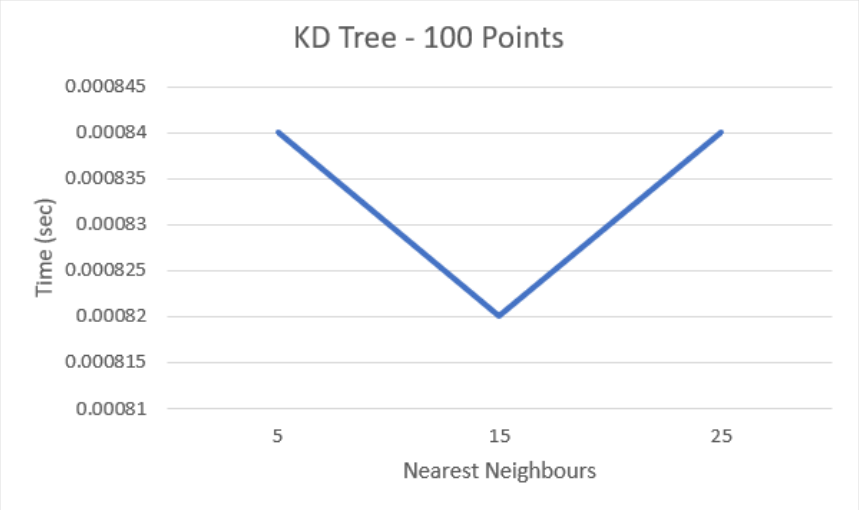
3)

	Naive				Kd Tree	
Data Size	K	Time (in sec)		Data Size	K	Time (in sec)
10000	5	8.942847852		10000	5	0.00086
10000	5	9.157135507		10000	5	0.00116
10000	5	8.918312015		10000	5	0.00105
	Average	**9.006098458**			Average	**0.001023333**
	Naive				**Kd Tree**	
Data Size	K	Time (in sec)		Data Size	K	Time (in sec)
10000	15	8.693858252		10000	15	0.00104984
10000	15	9.251693035		10000	15	0.001404843
10000	15	9.081909994		10000	15	0.001321631
	Average	**9.00915376**			Average	**0.001258771**
	Naive				**Kd Tree**	
Data Size	K	Time (in sec)		Data Size	K	Time (in sec)
10000	25	9.048707681		10000	25	0.001254834
10000	25	9.077538461		10000	25	0.001342897
10000	25	9.219740624		10000	25	0.001336874
	Average	**9.115328922**			Average	**0.001311535**

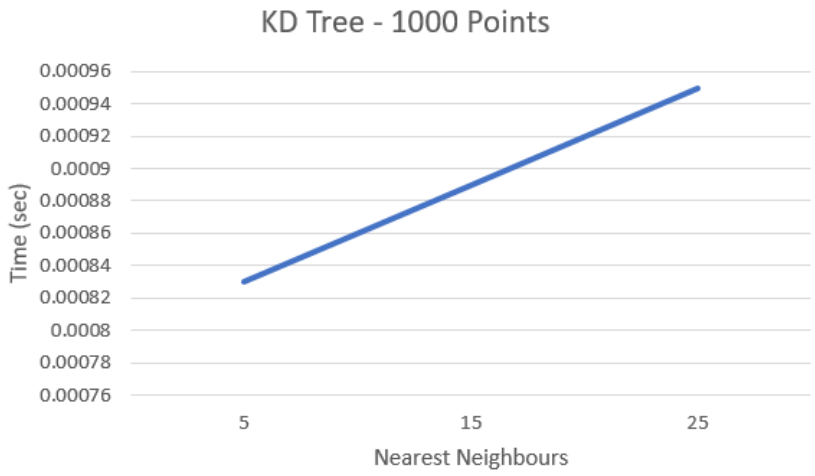
4)

	Scenario - 2				
	Naive			Naive	
Data Size	Add/Delete	Time (in sec)	Data Size	Add/Delete	Time (in sec)
100	10	0.002274675	100	10	NA
1000	10	0.055871884	1000	10	0.000916523
10000	10	8.794378251	10000	10	0.000785839

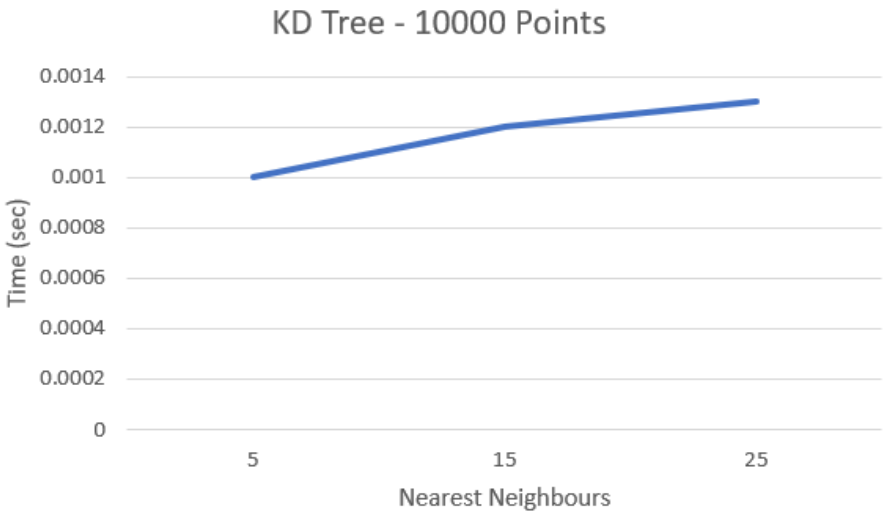
5)



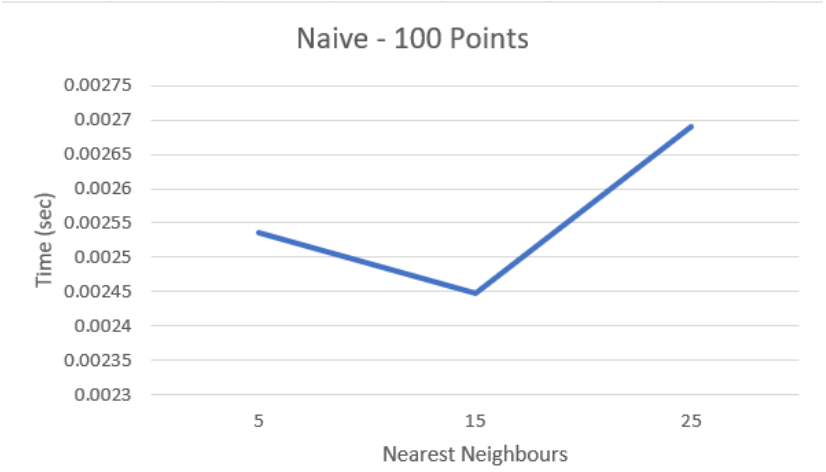
6)


KD Tree - 1000 Points
0.00096
0.00094
0.00092
0.0009
0.00088
0.00086
0.00084
0.00082
0.0008
0.00078
0.00076
Time (sec)
5
15
25
Nearest Neighbours

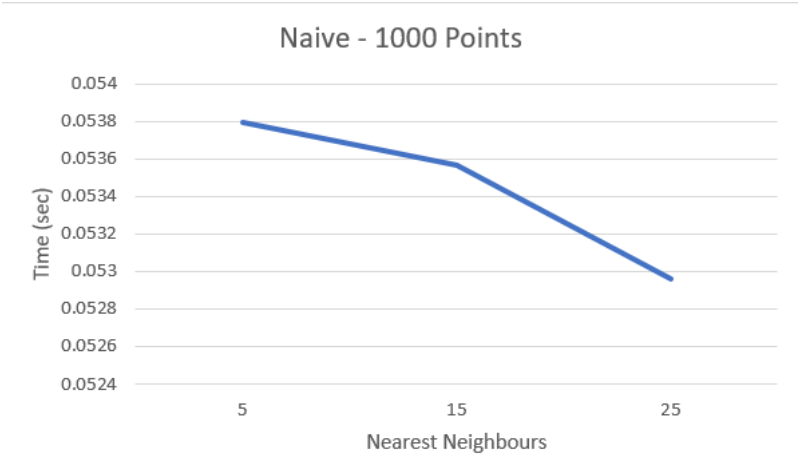
7)


KD Tree - 10000 Points
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
Time (sec)
5
15
25
Nearest Neighbours

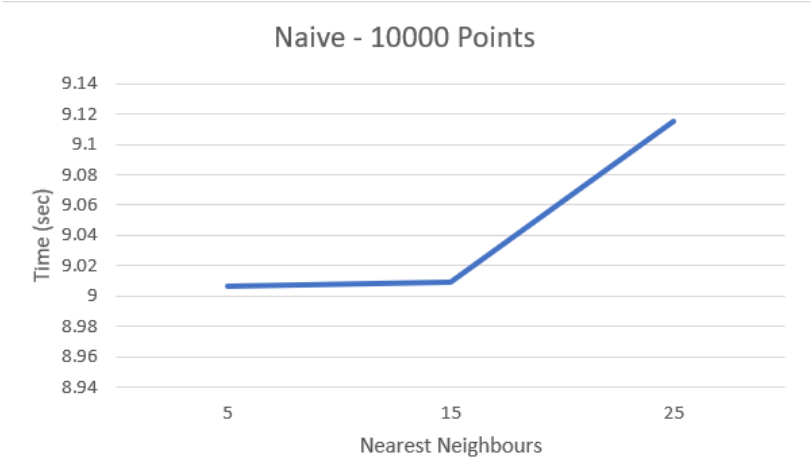
8)


Naive - 100 Points
0.00275
0.0027
0.00265
0.0026
0.00255
0.0025
0.00245
0.0024
0.00235
0.0023
Time (sec)
5
15
25
Nearest Neighbours

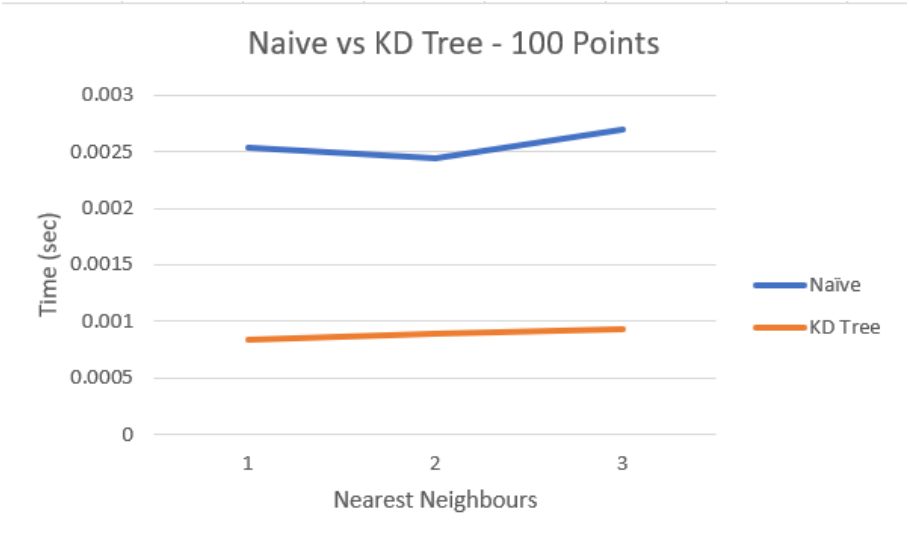
9)


Naive - 1000 Points
0.054
0.0538
0.0536
0.0534
0.0532
0.053
0.0528
0.0526
0.0524
Time (sec)
5
15
25
Nearest Neighbours

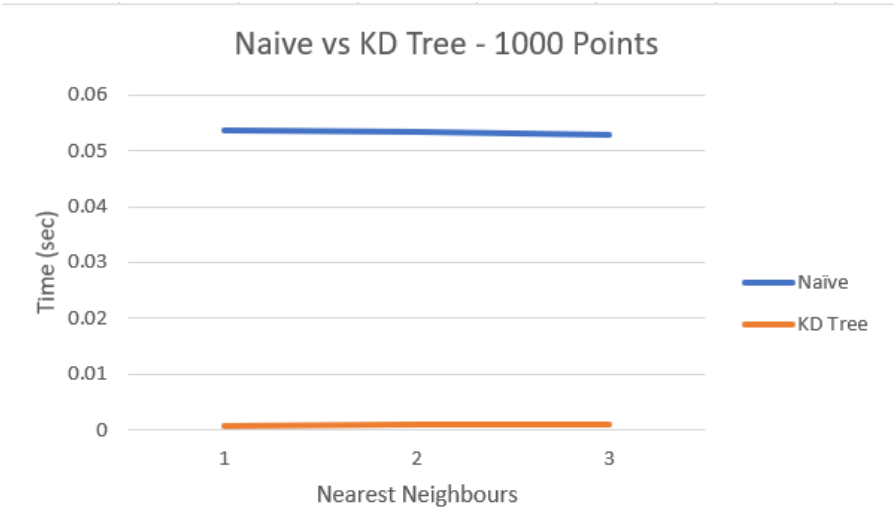
10)


Naive - 10000 Points
9.14
9.12
9.1
9.08
9.06
9.04
9.02
9
8.98
8.96
8.94
Time (sec)
5
15
25
Nearest Neighbours

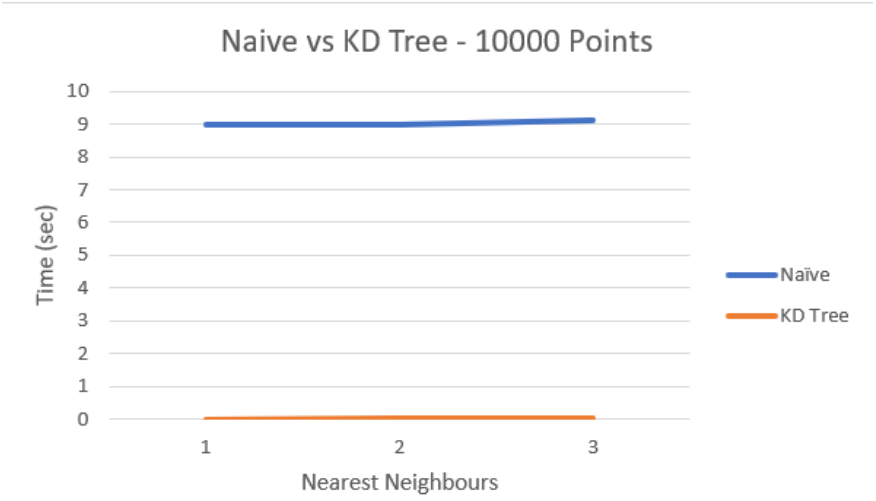
11)


Naive vs KD Tree - 100 Points
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
Time (sec)
1
2
3
Nearest Neighbours
Naïve
KD Tree

12)


Naive vs KD Tree - 1000 Points
0.06
0.05
0.04
0.03
0.02
0.01
0
Time (sec)
1
2
3
Nearest Neighbours
Naïve
KD Tree

13)


Naive vs KD Tree - 10000 Points
10
9
8
7
6
5
4
3
2
1
0
Time (sec)
1
2
3
Nearest Neighbours
Naïve
KD Tree