Programming assignment 2 report

CSI2110
Fall Term 2022

By
Alp Tatar (300241739)

<u>Experiment 1</u>

1)

The KD Tree Algorithm
```
number of neighbors= 5
[(-5.415942549526783,0.7715622302147948,-0.3968421613600826),
(-5.420458778974271,0.7891803562243134,-0.3973486218703048),
(-5.429850154613408,0.8075670478362598,-0.3982168226988382),
(-5.43030556398262,0.8246710769927127,-0.3984338736632657),
(-5.432677820578597,0.8420909833742529,-0.3987956432309413)]
```

Linear Search Algorithm
```
number of neighbors= 5
[(-5.415942549526783,0.7715622302147948,-0.3968421613600826),
(-5.420458778974271,0.7891803562243134,-0.3973486218703048),
(-5.429850154613408,0.8075670478362598,-0.3982168226988382),
(-5.43030556398262,0.8246710769927127,-0.3984338736632657),
(-5.432677820578597,0.8420909833742529,-0.3987956432309413)]
```

These results match

2)
The KD Tree Algorithm
```
number of neighbors= 2
[(-12.992860583393504,5.051138148093654,0.7622934861842156),
(-12.976373725118926,5.090611379773172,0.7622388885867976)]
```

Linear Search Algorithm
```
number of neighbors= 2
[(-12.992860583393504,5.051138148093654,0.7622934861842156),
(-12.976373725118926,5.090611379773172,0.7622388885867976)]
```

These results match

3)
The KD Tree Algorithm
```
number of neighbors= 1
[(-36.10818686248445,14.241618397722052,4.293473761897471)]
```

These results match
Linear Search Algorithm
```
number of neighbors= 1
```

```
[(-36.10818686248445,14.241618397722052,4.293473761897471)]
```

These results match

4)

The KD Tree Algorithm

```
number of neighbors= 17
[(3.120500565609992,0.0626817334571532,0.4610553779823442),
(3.1325477563686954,0.0533098761028178,0.4628090065256689),
(3.135663991597881,0.0437404978846847,0.4632473987090112),
(3.124626229458665,0.071407284130788,0.4308557147843563),
(3.101060042170619,0.0613504651643957,0.4275782041593502),
(3.102224266499409,0.051852810620631 8,0.4277147594345744),
(3.127132326652067,0.0426729987558792,0.4311288249443483),
(3.1074370070938127,0.0328693350571258,0.4283975616247083),
(3.110494894440637,0.0233580135464159,0.4288072530996791),
(3.1491863135474705,0.0139866281830143,0.4341331975609065),
(3.124448583809515,0.0042909825235033,0.4307191323087426),
(3.106996584743287,0.0607965822112821,0.3982019007320806),
(3.1121272286089696,0.0513459811348819,0.3988373764321324),
(3.123179921500925,0.041944387155322,0.4002354917039907),
(3.1431306038656457,0.032567946655484,0.4027774803812885),
(3.1233786019830654,0.0227801237087123,0.4002354923207727),
(3.1274271451816897,0.0036195964459265,0.4007438901352219)]
```

Linear Search Algorithm

```
number of neighbors= 17
[(3.120500565609992,0.0626817334571532,0.4610553779823442),
(3.1325477563686954,0.0533098761028178,0.4628090065256689),
(3.135663991597881,0.0437404978846847,0.4632473987090112),
(3.124626229458665,0.071407284130788,0.4308557147843563),
(3.101060042170619,0.0613504651643957,0.4275782041593502),
(3.102224266499409,0.0518528106206318,0.4277147594345744),
(3.127132326652067,0.0426729987558792,0.4311288249443483),
(3.1074370070938127,0.0328693350571258,0.4283975616247083),
(3.110494894440637,0.0233580135464159,0.4288072530996791),
(3.1491863135474705,0.0139866281830143,0.4341331975609065),
(3.124448583809515,0.0042909825235033,0.4307191323087426),
(3.106996584743287,0.0607965822112821,0.3982019007320806),
(3.1121272286089696,0.0513459811348819,0.3988373764321324),
(3.123179921500925,0.041944387155322,0.4002354917039907),
(3.1431306038656457,0.032567946655484,0.4027774803812885),
(3.1233786019830654,0.0227801237087123,0.4002354923207727),
(3.1274271451816897,0.0036195964459265,0.4007438901352219)]
```

These results match

5)
The KD Tree Algorithm
number of neighbors= 2
[(11.597053489523276,3.032865894391464,1.8696242228185609),
(11.580473933555549,2.9906018684790574,1.8654633424019456)]

Linear Search Algorithm
number of neighbors= 2
[(11.597053489523276,3.032865894391464,1.8696242228185609),
(11.580473933555549,2.9906018684790574,1.8654633424019456)]

These results match
6)
The KD Tree Algorithm
number of neighbors= 2
[(14.159820885717384,4.680702456874969,-0.1337915844837233),
(14.180766680737111,4.639415392714257,-0.1338543779659772)]

Linear Search Algorithm
number of neighbors= 2
[(14.159820885717384,4.680702456874969,-0.1337915844837233),
(14.180766680737111,4.639415392714257,-0.1338543779659772)]

These results match

Discussion
To actually find these points, the given Exp1 was used to solve them. It was run for every single point and pasted into the respective text files in the exp1 directory. For this part all the data was collected by compiling all the programs for each point and then using control f in google docs to compare each of the outputs, they were all matched meaning experiment 1 was a success.
.

Experiment 2 Note* (nanoseconds were converted to ms)

Finding the 10th point
The KD Tree Algorithm
Total time for finding neighbors: 101.67489099999993ms

Linear Search Algorithm
Total time for finding neighbors: 855.556540000001ms

Finding the 20th point
The KD Tree Algorithm
Total time for finding neighbors: 50.800478000000005ms

Linear Search Algorithm
Total time for finding neighbors: 469.28282499999983ms

Finding the 30th point
The KD Tree Algorithm
Total time for finding neighbors: 35.81470799999996ms

Linear Search Algorithm
Total time for finding neighbors: 284.174794ms

Finding the 29630th point
The KD Tree Algorithm
Total time for finding neighbors: 1.095002ms

Linear Search Algorithm
Total time for finding neighbors: 7.982901ms

Discussion
As one can see the KD tree algorithm always out performed the linear search algorithm as expected. It was however a very significant difference for most, I did expect a significant difference due to the less amount of general searches and the time complexity of each algorithm. This was implemented with a simple class taking command line arguments and in turn, simply calculating the end time of the rangequeary.

<u>**Experiment 3**</u>
The KD Tree Algorithm
Point_cloud_1.csv: 1402 Milliseconds
Point_cloud_2.csv:3941 Milliseconds
Point_cloud_3.csv:1980 Milliseconds

Linear Search Algorithm
Point_cloud_1.csv: 29633 Milliseconds
Point_cloud_2.csv:14180 Milliseconds
Point_cloud_3.csv:45141 Milliseconds

Discussion

As clearly demonstrated the overall KD algorithm stands over the linear search algorithm even including the tree construction the runtime still remained superior to the linear search algorithm. As one can see there are significant differences between them, on average around 10 000 milliseconds different. This was done by calculating the times for each point cloud, one with the assignment 1 solutions for the linear search. This algorithm was adjusted with the KD-trees then run again to see the respective outputs.