COMP20003 Algorithms and Data Structures
Assignment 2: Experimentation

Ming Hui Tan

December 4, 2020

Abstract

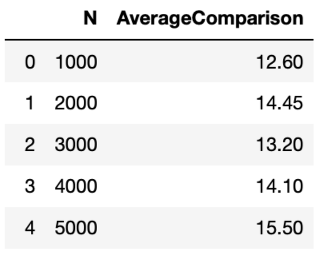
This report presents an investigation on the use of nearest neighbour search and radius search algorithms on KD Tree data structure (implemented in Assignment 2) to determine if the empirical evidence fits the theoretical complexity of the algorithms implemented. An experimental performance comparison of the nearest neighbour search algorithm between datasets of various forms (sorted or randomly arranged, with duplicates or without duplicates) is presented. Different kinds of keys (keys that exist or do not exist in the dataset, key that is out of the distribution of the dataset) were also experimented with to determine the optimal performance of the algorithm. In Stage 2, the radius search performance on randomly arranged dataset was examined with varying radius and the result is presented.

Introduction

To empirically test the complexity of the implemented algorithms, a large and diverse number of datasets were constructed and the number of comparisons for each dataset-key pair was recorded. There were various datasets ranging from 1000 rows to 100000 rows arranged in ascending, descending (in either x or y or both coordinates) as well as in random orders. There were also datasets with and without duplicates. The dataset-key pairs were run into the algorithm from within python using the following line:

```
os.system(f"./map1 data.csv output.txt < keyfile.txt *>> results.txt")
```

The results from each run of the program were collected from results.txt and averaged before inserted into a pandas DataFrame table like the one below for further analysis.



	N	AverageComparison
0	1000	12.60
1	2000	14.45
2	3000	13.20
3	4000	14.10
4	5000	15.50

Figure 1: Example of Record

Each row corresponds to a dataset of sample size **N** used for each run of the program. The **AverageComparison** column refers to the average number of comparison that each program takes to complete the search of a key in N-sized dataset. The average was taken to reduce noise from the random sampling of keys used as input.

Curve fitting and graphing were done using the numpy and matplotlib library in python while records reading and parsing were done using the regular expressions library in python.

1 Stage 1: Nearest Neighbour Search

1.1 Methods

At this stage, we are mainly concerned with the time complexity of the KD Tree algorithm in searching for the nearest neighbour. Therefore, a few experiments were constructed to determine how well the algorithm performs under certain constraints listed below:

1. Searching keys which exist and keys which do not exist in a randomly arranged dataset.
    - 100 randomly arranged datasets with sizes 1000 to 100,000 (with jumps of 1000) were created.
    - 20 keys were sampled from each dataset and another 20 keys were randomly generated within the range of coordinates in each dataset. The randomly generated keys were checked to ensure that the keys were not present in the dataset but were still within the distribution of the coordinates in the dataset.
    - Each dataset was first run with the 20 existing keys followed by the other 20 non-existing keys via the map1 program.
2. Searching existing keys in a dataset sorted by either x or y coordinates in ascending order.
    - 100 datasets with sizes 1000 to 100,000 (with jumps of 1000) consisted of random x and y coordinates (but still within the distribution in CLUE dataset) were created.
    - Each dataset was sorted first by x coordinate to run the map1 program with the 20 corresponding keys sampled from the dataset. Then, the same dataset was sorted by y coordinate to run the program again with the same keys.
3. Searching existing keys in a dataset sorted by both coordinates in either ascending or descending orders.
    - 50 datasets with sizes 200 to 10,000 (with jumps of 200) were created where each pair of coordinates in the dataset are of the same values. The 50 datasets were then sorted by the coordinates in ascending order and descending order.
    - 20 keys were sampled from each datasets once and the same keys were used to run the map1 program in both datasets of different coordinates orders.
4. Searching existing keys in a dataset with multiple duplicate coordinates.
    - 2 x 100 datasets with sizes 1000 to 100,000 (with jumps of 1000) were created. The first 100 datasets had no duplicate coordinates while the other 100 datasets had duplicate coordinates.
    - 20 keys were sampled from each dataset and the dataset-keys pairs were used to run the map1 program.
5. Searching a key positioned far from any points in the datasets.
    - 38 datasets with sizes 500 to 19000 (with jumps of 200) were created from the original randomly sorted CLUE dataset.
    - The coordinate (0, 0) was used as key to run the map1 program on each dataset.

After completing all searches in each dataset, the resulting number of comparison from each key were added and the average was plotted on a graph.

1.2 Data and Comparisons to Theory

1.2.1 Searching Existing and Non-Existing Keys in Randomly Arranged Datasets

The kinds of key appeared to matter for randomly arranged datasets, where keys existed in the dataset on average were found with fewer comparisons than those that did not exist in the dataset (see Figure 2). This is to be expected for a KD Tree data structure simply because any nearest neighbour search requires traversal to at least one leaf node of the tree. Besides, if the key is very far from its nearest neighbour, many leaf nodes must be checked. On the other hand, if a key exists in the dataset, the search for the key can terminate earlier without traversal until the leaf node, thus resulting in fewer key comparisons. Either way, these searches require at least $O(log\ n)$ comparisons in theory, and this is confirmed empirically by observing how well the log curves fit the data.

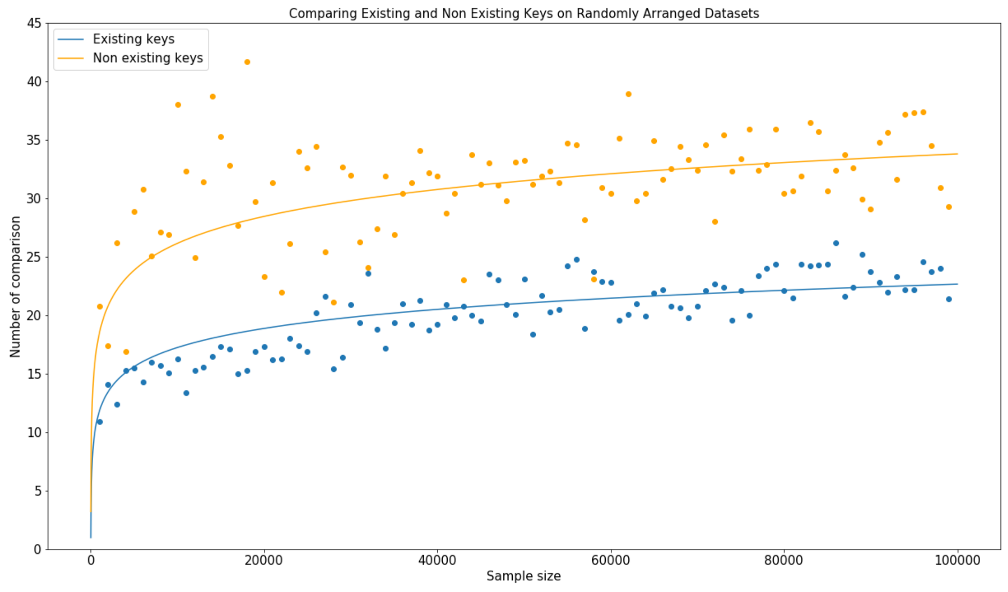


Figure 2: Result of comparison between existing keys and non existing keys search on randomly arranged dataset

1.2.2 Searching Existing Keys in a Dataset Sorted by Either Coordinate in Ascending Order

Sorting dataset based on one coordinate appeared to have degraded the search performance of a KD Tree. By comparing Figure 2 and Figure 3, we observe that datasets sorted by either coordinate take almost twice the number of comparison on average compared to randomly arranged datasets. This is because sorting one coordinate leads to a largely unbalanced KD Tree and increases the depth of the tree by a huge margin. Therefore, it takes more comparisons to reach the leaf node in a nearest neighbour search operation. However, the search remains $O(log\ n)$ operations since the characteristics of the tree are preserved despite the tree being unbalanced. On a side note, the types of coordinates being sorted appeared to have minimal effects on the number of comparisons due to the structure and characteristic of a KD Tree.

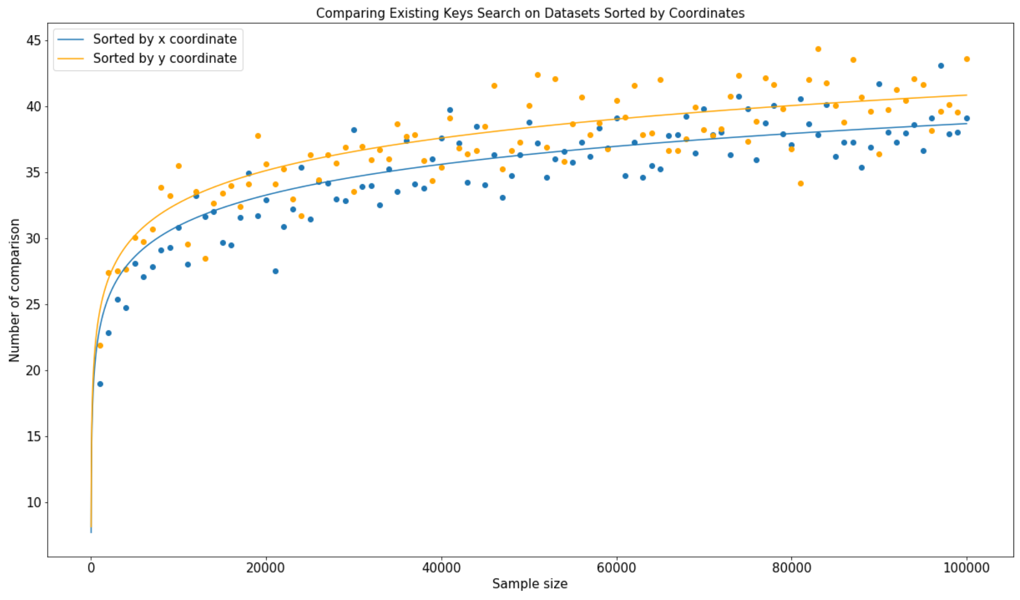


Figure 3: Result of comparison of existing key search between datasets sorted in either coordinates

1.2.3 Searching Existing Keys in a Dataset Sorted by Both Coordinates

Here we observe that when the datasets are sorted in the same order in both coordinates, the KD Tree turns into a stick, hence can be treated like a linked-list (worst case for a KD Tree). The slope of the line in Figure 3 is about 1/2 because on average it takes n/2 comparisons to find the position of a key in a sorted linked-list. Once again, the graph confirms the theory showing a search complexity of $O(n)$.

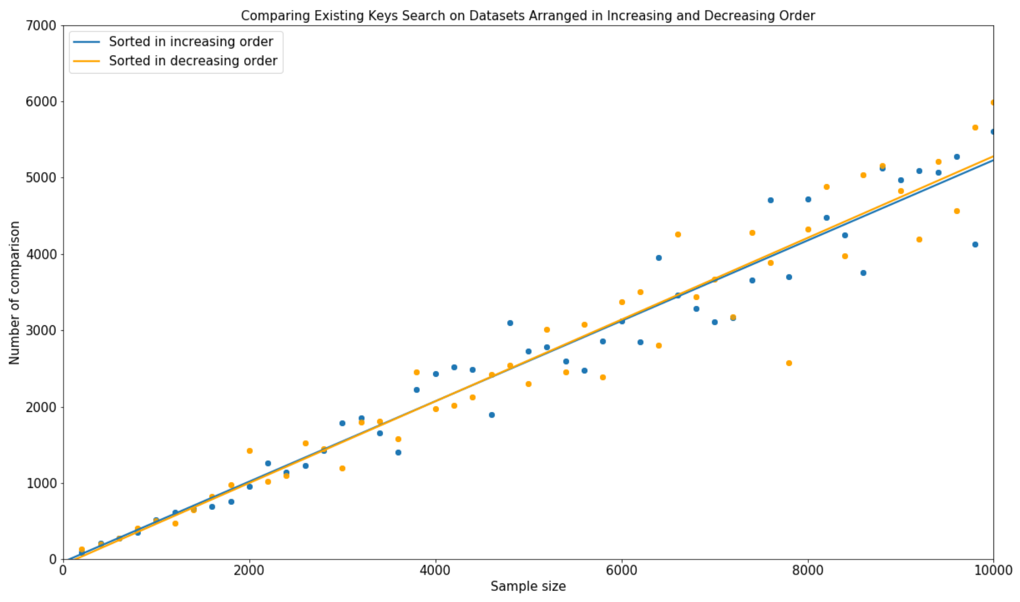


Figure 4: Result of comparison of existing key search between datasets sorted in both coordinates

1.2.4 Searching Existing Keys in Datasets With and Without Duplicate Coordinates

It appeared that whether or not duplicate coordinates existed in the datasets made little to no difference in terms of the number of comparisons. This is because all duplicates were chained in a linked-list and existed as only one node in the KD Tree. Therefore, it did not matter how long the linked-list was, no more key comparisons were made. In both cases, the number of comparisons remained as $O(log\ n)$ as expected (see Figure 5). However, for a fixed size of data, increasing the number of duplicates would yield a lower number of comparisons simply because the depth of the tree would be shorter while the linked-list chained to each node would be longer.

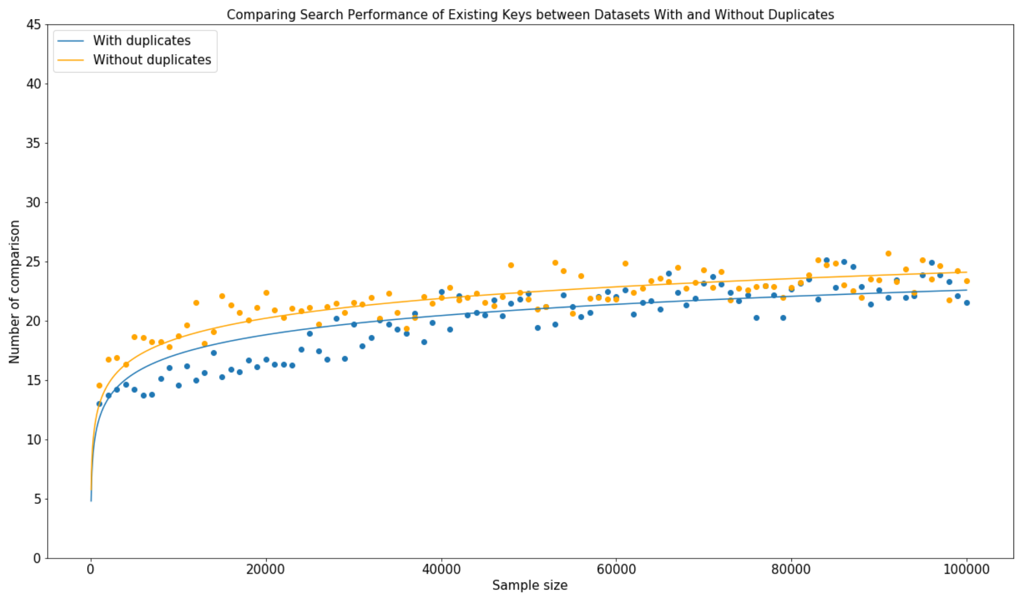


Figure 5: Result of comparison of existing key search between datasets sorted in both coordinates

1.2.5 Searching Key Positioned Far From Any Points in the Datasets.

Unlike in experiment 1 where the non-existing keys were still within the distribution of the coordinates in the dataset, here the search was aimed towards key positioned far from any points in the dataset. As expected, the number of comparisons is close to $O(n)$ (see Figure 5) despite the tree being relatively more balanced compared to the tree in experiment 3. This is due to the fact that the search for the key positioned far from any points in the dataset will need to involve comparisons with all nodes in the tree.

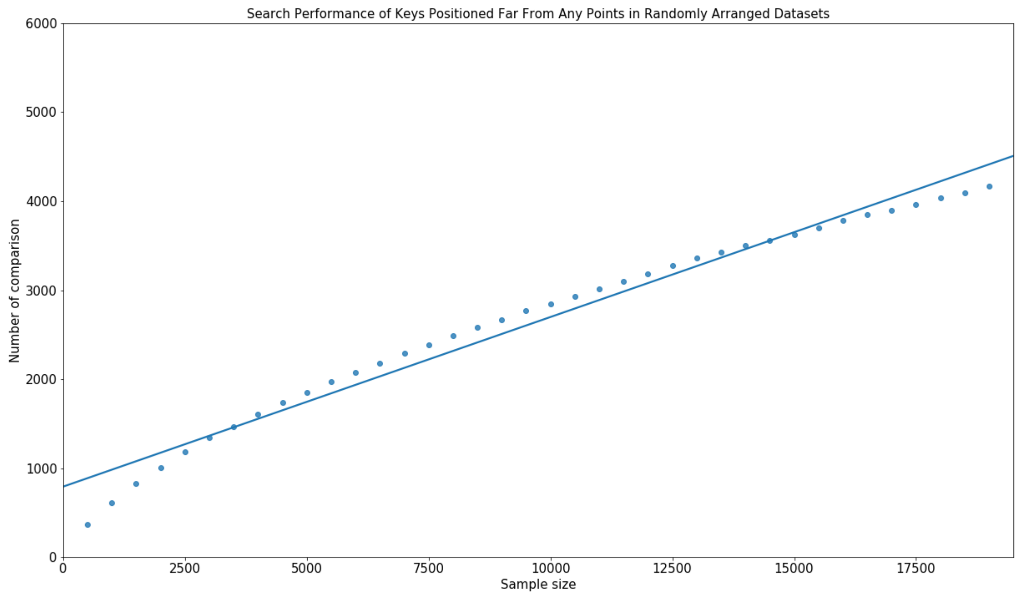


Figure 6: Result of search performance of key positioned far from points in the randomly arranged dataset

2 Stage 2: Radius Search

2.1 Methods

At this stage, we are mainly concerned with the time complexity of the algorithm in performing radius search. Thus, the following experiment was constructed to analyse this:

1. Radius Search Performance with Varying Radius in Randomly Arranged Datasets
    - 4 x 100 datasets with sizes 1000 to 100,000 (with jumps of 1000) were created. Each set of 100 datasets was paired with a radius as part of the key.
    - 20 keys were sampled from each dataset and the keys were paired with the dataset-radius pairs to run the map2 program.

After completing all searches in a dataset, the resulting number of comparisons from the 20 keys were averaged and plotted on a graph.

2.2 Data and Comparisons to Theory

2.2.1 Radius Search Performance with Varying Radius in Randomly Arranged Datasets

Here we observe that when the radius is small the number of comparison required is $O(log\ n)$ which is what we would expect for the best case of the radius search method implemented in map2 program (See Figure 7). This is because the radius is small enough that it approximates to the size of the point on the datasets, thus can be treated similar to an average nearest search performance in KD Tree. As the radius increases, however, the corresponding search performance begins to degrade to $O(n)$ number of comparisons due to the fact that the increasing query radius increases the number of points to be included in the search. In theory, if the radius is large enough that it covers the entire dataset (worst case), the number of comparison will be $O(n)$ which aligns with our findings.

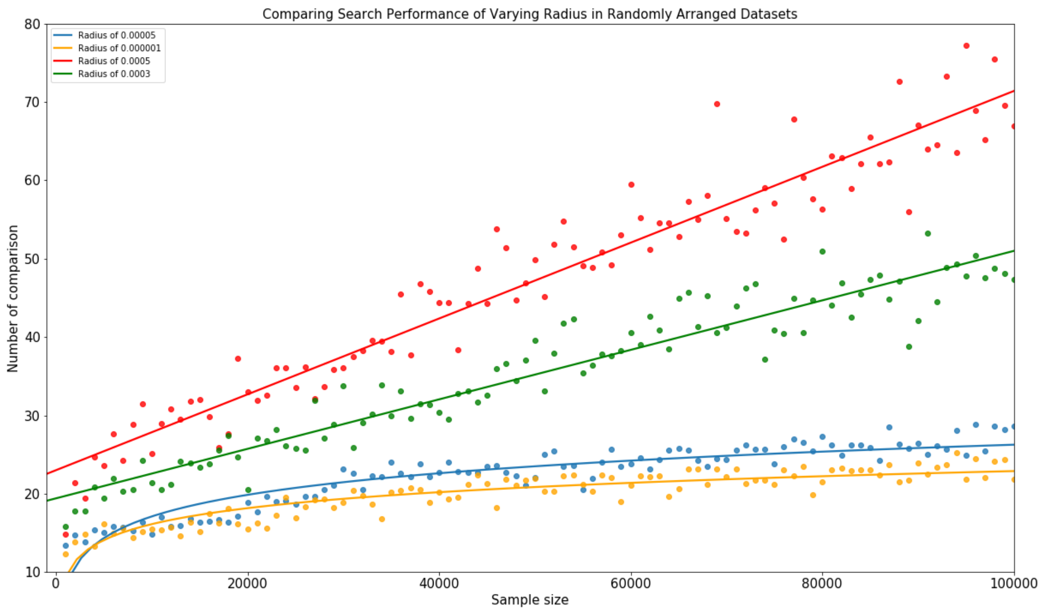


Figure 7: Result of radius search performance with varying radius in randomly arranged datasets

Limitations

While the empirical evidence closely aligns with the theory, there were certain limitations faced while conducting the experiments in terms of data sampling and assumptions made. Since most of the data were created by generating numbers within the distribution of the original dataset, it might not accurately represent a realistic dataset in a way that the original dataset had about 80% of duplicate data while the dataset generated had comparatively fewer duplicate data. Besides, the assumption that the KD Trees built using randomly arranged datasets were relatively more balanced than trees built using other datasets mentioned in this paper may not be a reliable and adequate assumption. This is because the algorithms used in both map1 and map2 programs to build the KD tree were not optimized. Finally, the traversal method used in both map1 and map2 programs is pre-order traversal which will yield different results compared to in-order and post-order traversal. Therefore, the result obtained cannot be generalized across similar algorithms with different traversal methods.

Conclusion

It was discovered that the theoretical complexities for nearest neighbour search and radius search using the KD Tree data structure were matched by empirical evidence as expected. The nearest neighbour searches on randomly arranged data were completed in $O(log\ n)$ on average while the searches on fully sorted data (on both x and y coordinates) were completed in $O(n)$. The number of key comparisons did not change between datasets that were sorted on both coordinates in ascending or descending order since both produced a stick. On data sorted by only either x or y coordinate, however, searches were completed in $O(log\ n)$ since the characteristic of the tree is preserved despite being unbalanced. The search performance was also not affected by duplicates in the dataset since duplicates are chained in a linked-list and were treated as a single node in the tree. On another note, keys that existed in the dataset were found more quickly than those that did not for the randomly arranged data. As for keys positioned far from any points in the datasets, it was found that $O(n)$ number of comparisons was required since the search had to involve all nodes in the tree. For Stage 2, the input radius had a huge impact on the complexity of radius search algorithm used in map2 program with $O(log\ n)$ being the best case where the radius is very small (approximates to a point on the dataset) and $O(n)$ for the worst case where the radius covers the entire datasets.