Cynthia Joseph (938358)
COMP20003 - Algorithms and Data Structures Assignment 2 – Experimentation – 18/09/20

Introduction:

The purpose of this report is to identify the trends between the average number of comparisons needed to complete a query search and the number of elements provided within a sorted and randomized 2d tree. The two searches involve a nearest neighbour search, which will return the closest CLUE business from the query point, and a radius search, which will return all businesses, if any, that lies within a radius from the query point. The data from the experimentation will be used to construct charts to illustrate the relationship between the constant number of key queries against varying data subsets (N) within the tree. In theory, a 2d tree will have an average search complexity of O(log N), with its worst case being Θ(N). Therefore, an upwards trend is predicted for the stages for both random and sorted datasets, random datasets having an average complexity of O(log N) with a worse case complexity of Θ(N), and sorted datasets having a complexity of O(N).

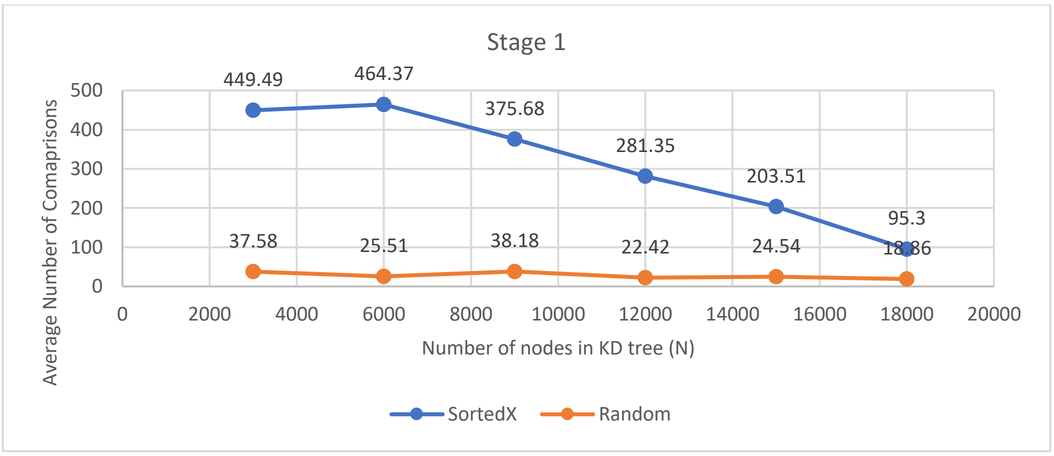
Methodology:

1. Create test queries for both maps, each consisting of 100 random keys.
    a. Stage 1 keys: xcoordinate ycoordinate
    b. Stage 2 keys: xcoordinate ycoordinate radius
2. Create sorted and random subsets between 300 and 18000 records from the full dataset, with increments of 300.
3. Test each dataset with the 100 random key queries for each map
    a. Output Stage 1: xcoordinate ycoordinate → comparisons
    b. Output Stage 2: xcoordinate ycoordinate radius → comparisons
4. Record the average number of comparisons it takes to complete the search for each dataset and map.
5. Create graphs to illustrate the trend observed from the experimentation.

Results from experimentation:

Number of KD-tree Input (N)	Average Comparisons for SortedX	Average Comparisons for Random
3000	449.49	37.58
6000	464.37	25.51
9000	375.68	38.18
12000	281.35	22.42
15000	203.51	24.54
18000	95.3	18.86

<u>Stage 1</u>
(figure 1)



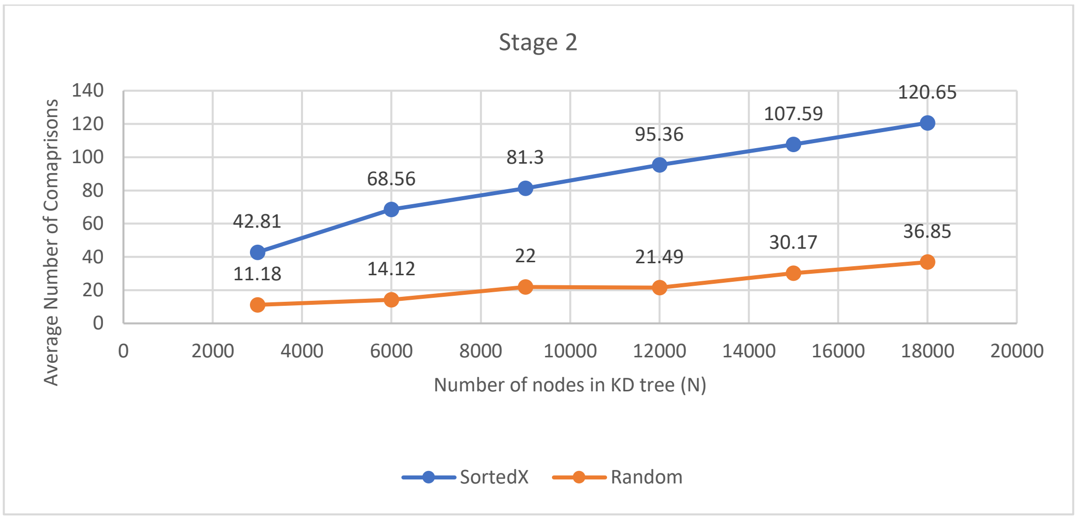
(figure 2)

The results from the experiment showed that a randomized 2d-tree produced less affected average comparisons compared to that of the sorted 2d-tree when varying the number of elements in the tree. However, both results created a negative trend, showing that the average number of comparisons per search key was indirectly proportional to the number of nodes in the 2d tree (figure 2). The sorted 2d tree had a steeper decline, from an average of 449.5 comparisons to an average of 95.3 comparisons, whereas the randomized 2d tree varied slightly from an average of 37.58 comparisons to an average of 18.63 comparisons when given a tree with nodes ranging from 18000 to 3000 respectively (figure 1).

<u>Stage 2</u>

Number of KD-tree Input (N)	Average Comparisons for SortedX	Average Comparisons for Random
3000	42.81	11.18
6000	68.56	14.12
9000	81.3	22
12000	95.36	21.49
15000	107.59	30.17
18000	120.65	36.85

(figure 3)



(figure 4)

The results from the experiment showed that there was an upwards trend when comparing the size of the 2d tree and the number of nodes needed to complete the search. Using a sorted tree showed that increasing its number of elements caused a rise towards the average number comparisons for each query. This is demonstrated by the steep increase in average comparisons (from 42.81 to 120.65) when introducing more input nodes in the 2d tree. All average comparisons from the randomized tree were lower than that of the sorted tree and had smaller increases in average complexity, with the exception of N = 12000 where it dipped slightly. (figure 4).

Data and Comparisons to Theory:

<u>Stage 1</u>

In theory, searching through a 2d tree would have a worse case complexity of O(N), as the search would traverse through the entire tree and visit all the nodes. This case would occur if the entire tree were sorted, creating a stick. In our sorted data however, all the data were sorted in the x dimension, meaning that every x dimension of the 2d tree will only produce right nodes. Although this will not create a stick, it will increase the depth of the tree and thus increase its complexity. However, the experimentation results did not coincide with the initial prediction of an upwards trend for both randomized and sorted datasets. This may be due to the inaccuracies of the search, causing it to return inexact nearest neighbours. With the logic implemented in the code, more elements in the tree may have allowed for more pruning and elimination of branches where the best node would not be, thus causing smaller average comparisons per search key. There was no back tracking implemented in the search, so it may not have produced the exact "nearest" but will be good for large datasets since it will have a faster time complexity.

<u>Stage 2</u>

The experimentation results from Map 2 produced an upwards trend as predicted. As mentioned previously, the sorted tree will not be a stick, but its x dimensional nodes will only produce right children, giving it a complexity of approximately O(N) since the average comparisons will grow as more elements are inserted into the tree. As for the randomized tree, it is observed to have an average search complexity of O(log N) as its branches are more balanced. This idea could be supported by visualizing more elements being introduced into the tree and making the map more populated, causing more ‘possible’ branches to lie within the radius provided, thus increasing the average comparisons as presented by the upwards trend seen in figure 4. The worst-case complexity for this search would be Θ(N), which would occur if the radius given encompasses the entire tree.

Conclusion and Recommendations:

Overall, it is found that the average complexity of the nearest neighbour search in Stage 1 grows indirectly proportional to the number of elements within the tree. This may be due to the inaccuracy of the search applied, pruning the branches that it predicts to not contain the nearest neighbour, thus returning an inexact result. However, the search may be more practical for larger datasets as it will cut down its time complexity by a significant amount. Stage 2 results showed that traversing through a randomized tree will give an average search complexity of O(logN), whilst a sorted tree will give an approximate average complexity of O(N). Perhaps future experiments will involve an exact nearest neighbour search, where backtracking is implemented, in order to obtain an upwards trend between the average comparisons and number of tree elements.