COMP20003
Algorithms and Data Structures

Assignment #2 - Experimentation

Goh Xing Yang
1001969
September 17, 2020

1 Introduction

1.1 K-D Tree Data Structure

The K-D Tree is a data structure that can be visualised as a multi-dimensional binary search tree (BST). The focus of this report will be on a 2 dimensional K-D Tree, referring to the x and y coordinates of Melbourne business establishments data taken from the Victorian government. The creation of a K-D tree is similar to that of a BST, however, each depth of the K-D tree cycles between the different dimensions for comparisons and duplicates (identical x and y coordinates) are appended to a linked list structure.

1.2 K-D Tree Search Algorithms

The report aims to identify and analyze the complexity of the KD-tree nearest neighbour and radius search algorithms through empirical testing, varying the size of data sets and ordering of the input data—sortx, random and median to perform complexity analysis. The nearest neighbour search algorithm uses recursion to traverse down the tree, calculating the distance to the query point at each node and updating the closest distance to the query point if the distance is closer than the previous best. Next is the dimensional distance comparison of the current node to the query point, visiting both children if the node's dimensional distance difference to the query point is within the radius of the closest distance—since the closest node could be in either of the branches. If this distance is greater than the closest distance, the tree is traversed down the branch which approaches the query point for that dimension. This recursion occurs until the leaf, where it begins the return process. This return process requires the node with the closest distance to be unwinded back to the initial function call, therefore, each recursive call has to compare the distance between it's current node and the traversed children nodes, returning the node that is closest to the query point. The radius search uses the same concept however, instead of comparing distances of nodes, it simply prints the data if the node is within the radius of the query point. This report will focus on empirically calculating the efficiency of these search algorithms by performing tests on varying sizes of data sets using different insertion orders to create the K-D tree—median, random and sorted by x coordinates. Using the number of key comparisons done on each test, the efficiency of the algorithm can be observed.

2 Analysis

2.1 Empirical Procedure

To empirically analyse the efficiency of the search algorithms, a variety of data sets must be produced. This has been done in the UNIX environment in the following way. First, the data from the 3 provided csv files (median, sortx and random) were extracted as samples of n = 100, 200, 300, 1000, 3000, 6000, 9000, 12000, 15000, 18000. Query files were then produced by taking 100 random lines from the entire data file to use as coordinate inputs to the program. The same method is used to create the radius search query file with a fixed radius search of 0.0005. Lastly, the query file is passed into the program for each data sample and the average number of searches for each n is calculated. These averages are plotted against one another for initial comparison of the data sets.

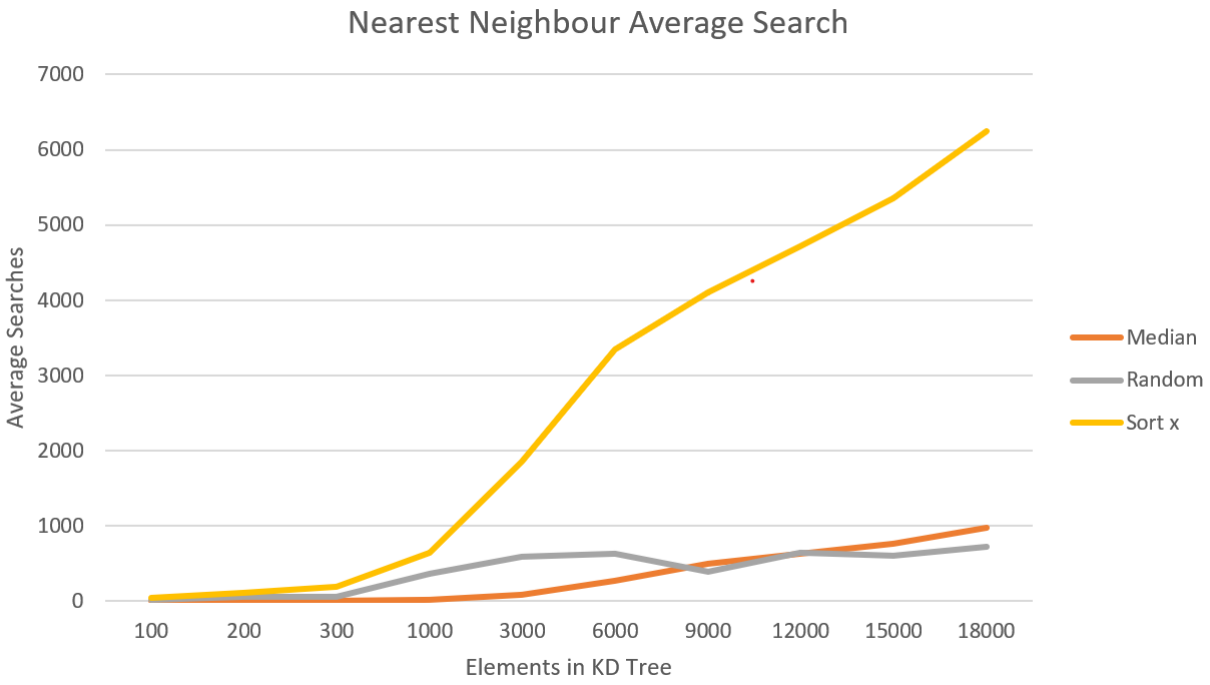

Figure 1: Nearest neighbour average searches for different data sets

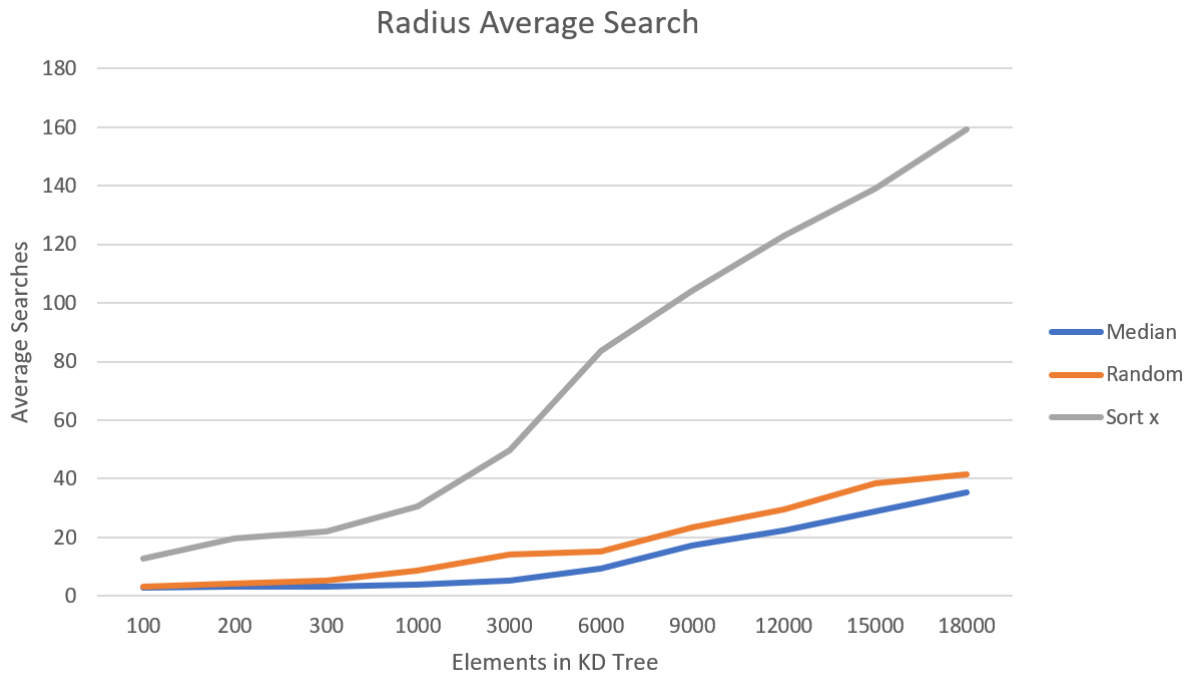

Figure 2: Radius search average searches for different data sets

2.2 Observations

Figure 1 and 4 plots these average searches and some initial patterns are observed. For both the nearest neighbour and radius search, the number of searches for the sort x data grows

at a much faster rate than the median and random data sets. Also, the average searches for the nearest neighbour is much greater than the radius average search. The similar growth and patterns between the different searches can be easily explained due to the creation of the K-D tree and the similarities in the search algorithms. In the sorted x data (ascending), the K-D tree is highly unbalanced since every time the x dimension is compared, the K-D tree will add the node as a right child. The only difference between the algorithms is the constant comparisons and changes to the nearest distance from the nearest neighbour search, compared to the fixed radius search where this distance is fixed. This leads to the large difference in magnitude from the two searches, where due to the constant narrowing of the closest distance, the recursive algorithm has to traverse down both children of a given node at a much higher frequency. The radius search only requires to traverse down both children of the node once it approaches the specified radius distance, which in this case is 0.0005. Therefore, it means that a higher radius search input will increase the average number of searches required.

2.3 Theory

For complexity analysis, the worst case for both searches is $\mathcal{O}(n)$, when both children are visited at every node or the data is entered into the K-D tree as a linked list. This can occur when the coordinates are sorted in a way rotating between x and y (sorted in x for odd instances and sorted in y for even instances). The best case is $\Omega(\log n)$, which occurs for the nearest neighbour search when the K-D Tree is perfectly balanced and the nearest neighbour to the query point is the root node and all subsequent points does not fall within this radius. This causes the search to traverse down the K-D tree visiting one child at a time until reaching the leaf, which is theoretically a time complexity of $\log_2 n$. The best case for the radius search has the same time complexity as the neighbour neighbour search, occurring when only a single point at the leaf node is in the given radius. In practical applications, the worst case of $\mathcal{O}(n)$ is highly unlikely to occur with a random data set, where the average search complexity lies somewhere between the two.

The master's theorem can be used to prove this as the algorithms are recursive, however, some assumptions have to be made. For the worst case performance, assume that the data is split in half, with both children being visited each time until the leaf node (a=2, b=2, d=0).

$$T(n) = 2 * T(\frac{n}{2}) + n^0$$

$$d < \log_2 2$$
$$T(n) \in \Theta(n^{log_2 2})$$
$$T(n) \in \Theta(n)$$

Best case performance occurs when data is split in half, traversing down only a single child until the leaf node (a=1, b=2, d=0).

$$T(n) = 1 * T(\frac{n}{2}) + n^0$$

$$d = \log_2 1$$
$$T(n) \in \Theta(n^0 \log n)$$
$$T(n) \in \Theta(\log n)$$

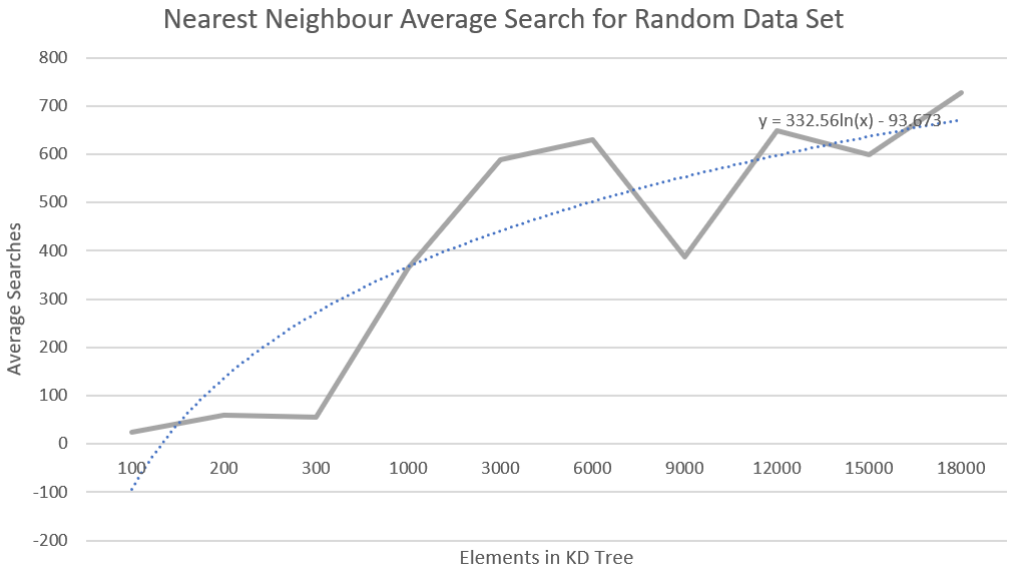

Figure 3: Logarithmic trend-line for nearest neighbour search with random data sets

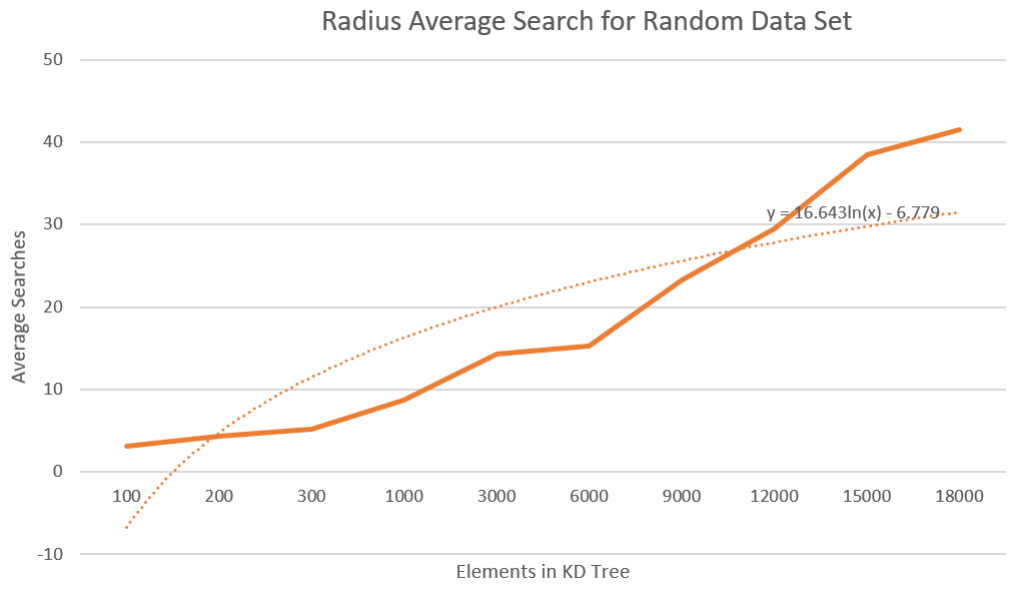

Figure 4: Logarithmic trend-line for radius search with random data sets

Figure 3 and 4 fits a logarithmic trend-line through random data sets and conforms the theoretical calculations for the complexity, falling somewhere between $\mathcal{O}(n)$ and $\Omega(\log n)$. The average complexity adheres closer to $\mathcal{O}(\log n)$, which is expected due to the stringent criteria required to achieve the worst case of $\mathcal{O}(n)$. The coefficient of the log terms in the trend-line is likely associated with the dimension of the K-D Tree, however, further analysis on multiple K-D tree dimensions is required to empirically analyse the effects.

3 Conclusion

The theoretical complexity of the nearest neighbour and radius searches matches the empirical analysis, with a random data set expecting $\mathcal{O}(\log n)$ behaviour for both searches. This analysis was only performed on a 2 dimensional K-D tree, where the average complexity for random data sets likely to increase considerably for higher dimensions. The data set which was sorted ascending in x coordinates had searches that grew much quicker than the random and median data set, which was expected since the tree is much more unbalanced due to the node always being placed to the right child when comparing in the x dimension. The radius search required significantly less searches than the nearest neighbour search as it traverses down both child nodes less since the value is fixed from the start and not constantly updating, although this is dependant on the size of the input radius.