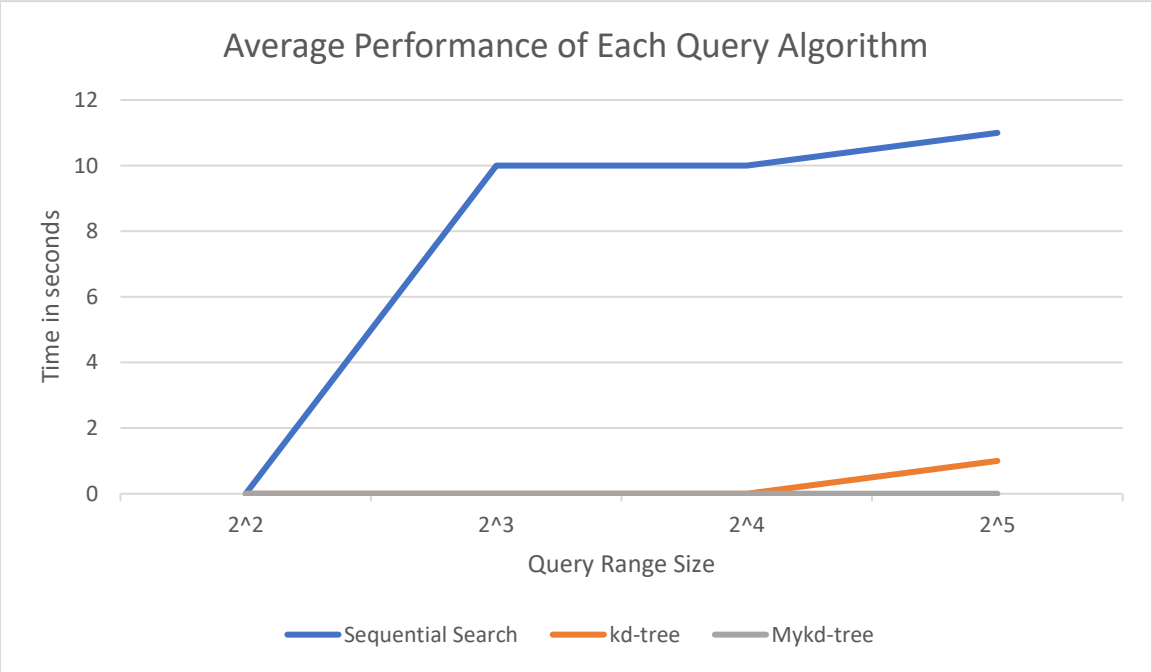
i) My variation on the kd-tree with the MYkd-tree algorithm is that every index block is size = 1, and every node in the tree contains just 1 point. Each node has an associated attribute, which alternates with every dimension in the data, and the data points are split with the associated attribute value less than a certain median value on the left child, and the associated attribute value greater than a certain median value on the right child. The attributes on every different level of the tree are different, with each level rotating through the attributes of all dimensions.

ii)



Each query algorithm was tested 10 times on each query range size and shown in the graph is the average running time for each algorithm. From the data, the sequential search suffers the most with increasing orders of query ranges, whereas the kd-trees both have phenomenal performance despite increasing query range sizes. It appears that the performance of the kd-trees are not affected by increasing query range sizes, most likely due to the tree structure and not having to search through every single data point. With the kd-tree, I collected the runtime data with an index block of 50, which averaged out to 1 second needed to find 2^5 queries. I lowered the size of the index block and noticed the average time decreasing, so for me the lower the index block size the better performance I achieved.