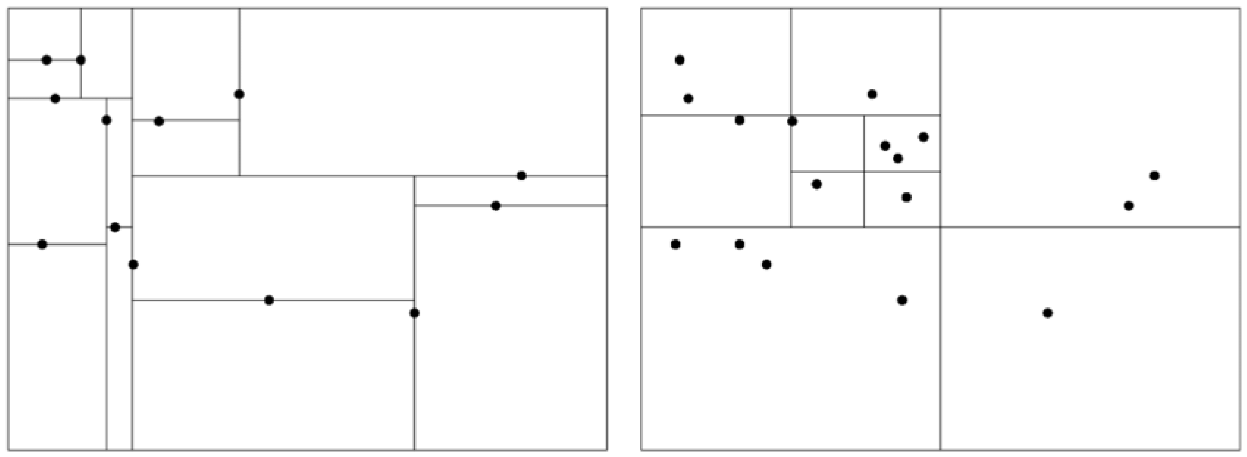
Quadtree vs kd-tree

Quadtrees and kd-trees are both a type of Space Partitioning Trees. Both kd-trees and quadtrees are used in many applications such as computer graphics, statistics, data compression, pattern recognition, and database systems to store objects in low or high-dimensional spaces.
The figure below shows a kd-tree (left) and of a quadtree (right).



Comparison of Kd-trees & Quadtrees:

a. Nearest Neighbor Search:

The nearest neighbor problem is defined as follows: A set S of n elements in k dimensions is given, and we are searching for the closest element in S to a query object q. The idea is to arrange the objects into a tree structure so that the time for answering queries depends on the height of the tree.

Kd-trees in nearest neighbor problem: In the nearest neighbor problem, the most-used space partitioning trees are kd-trees and it is a linear time approach. The kd-tree approach performs very well for a small number (less than 100) of objects. The kd-tree should only be used for less than 20 dimensions since it performs very poorly in higher-dimensional spaces.

To find the nearest neighbor using kd-trees:

- Search from the root node downwards and at each level compare which of the two children nodes have the lesser Euclidean distance with the point for which the nearest neighbor has to be found.
- Choose the node with the lesser Euclidian distance to traverse next.
- Keep going downwards while maintaining the minimum Euclidean distance found till now and output the node which is the closest when a leaf node is reached.

Quadtrees in nearest neighbor problem:

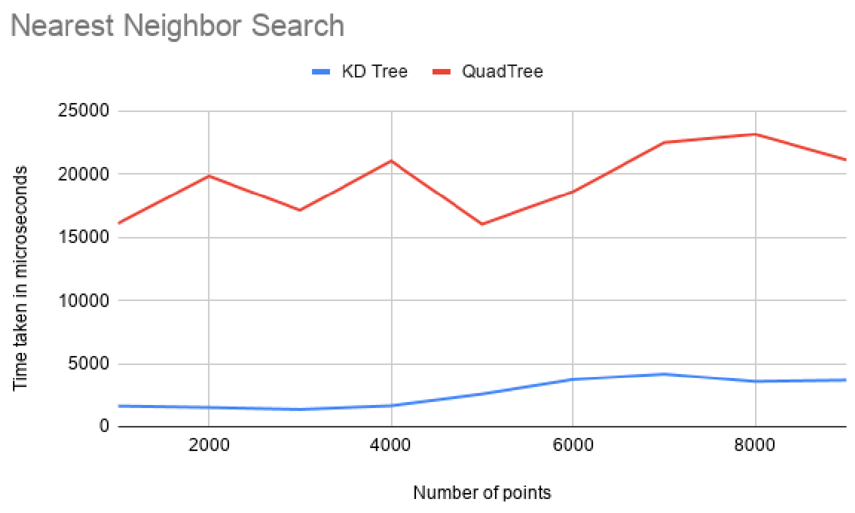
To find the nearest neighbor using quadtrees:

To search for the nearest neighbor Q we choose a point to search and a 2-dimensional window W around this point in which we expect to find its nearest neighbor. Then, the following steps are performed:

```
● Put the root on a stack
● Repeat
    Pop the next node T from the stack
    For each child C of T
      If C intersects with W around Q, add C to the stack
      If C is a leaf, examine point(s) inside C and record the minimum Euclidean
  distance to Q.
● Return Q
```

In finding the nearest neighbor, kd-trees perform better with an average time complexity of O(log(N)) and a worst case O(N).

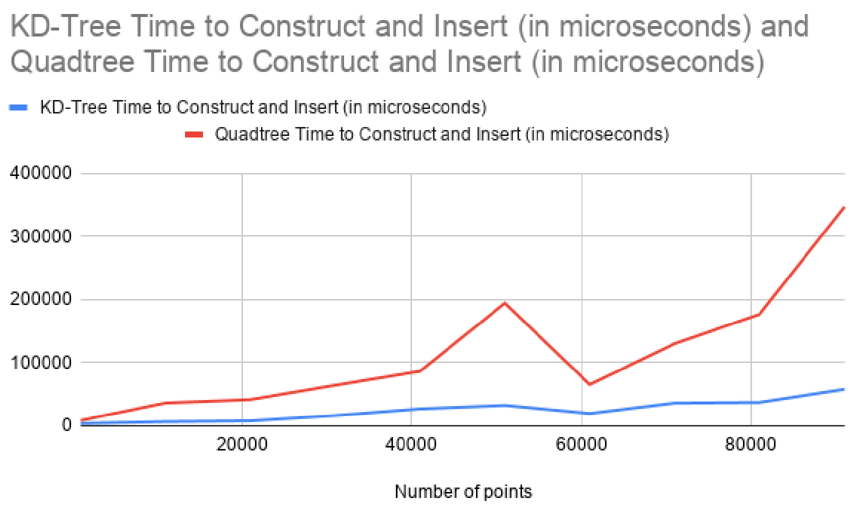
The figure below shows kd-trees vs quadtrees in the nearest neighbor problem:



b. Insertion based on number of data points:

In practice, the construction and insertion time based on the number of data points for quadtrees and kd-trees is between O(N) and O(NlogN) because some data points result in a skewed insertion. In theory, the worst case construction time for a quadtree and kd-tree is O(nLogn) but the average time comes out to be O(N).

The figure below shows kd-trees vs quadtrees for insertion based on number of data points:



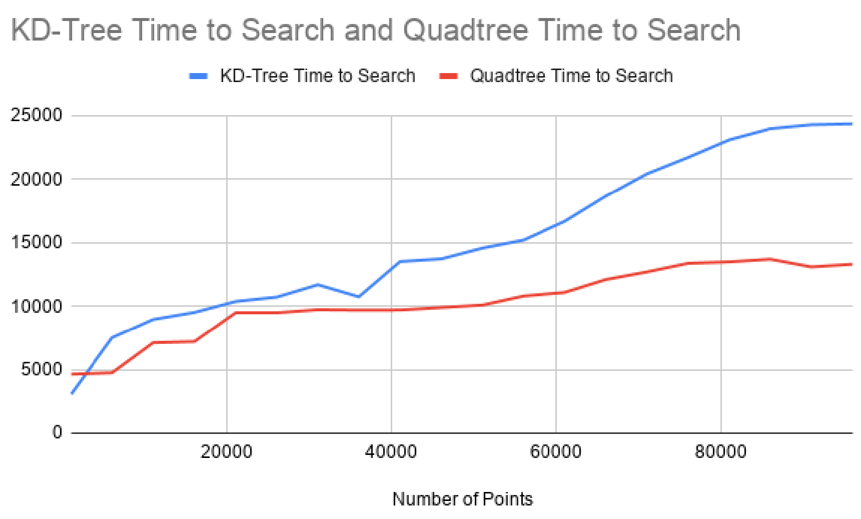
Insertion into Quadtrees based on number of data points: To insert an element in a Quadtree, the average case time complexity is O(Logd(N)), where N is the size of distance. Here, d=4 since the branching factor is 4.

Insertion into kd-trees based on number of data points: The insertion time complexity for kd-trees is O(Logd(N)) with d=2, since the branching factor is 2.

Insertion based on the number of points into quadtrees requires a traversal of greater depth per insertion than kd-trees. Thus, in practice, construction of quadtrees takes more time than kd-trees.

c. Search based on number of data points:

The figure below shows kd-trees vs quadtrees for search based on number of data points:



Quadtrees are much faster than kd-trees in searching as data size increases. Kd-trees take log2(N) time and quadtrees take log4(N) on average to search a point.