

Limitations of kd-tree:

Suppose we are just doing for 2-D or 2 features, now if query point is at boundary of recatangles, so we have to look for 4 adjoining regions(doing the whole process again after back track), that means for 2-D we have to look for 22.

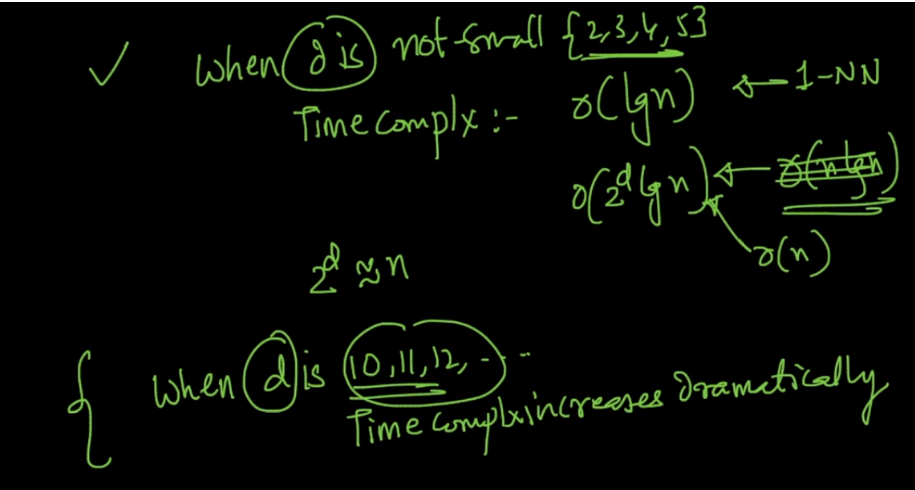
similarly if our point is at boundary in cube we have to look 8(23) adjoining regions.(number of regions is not equal to number of faces)

So if we have d dimensions so we’ve to look 2d regions.

So for finding 1 NN our time complexity will not be O(log n), it will be O(2d \*log n),

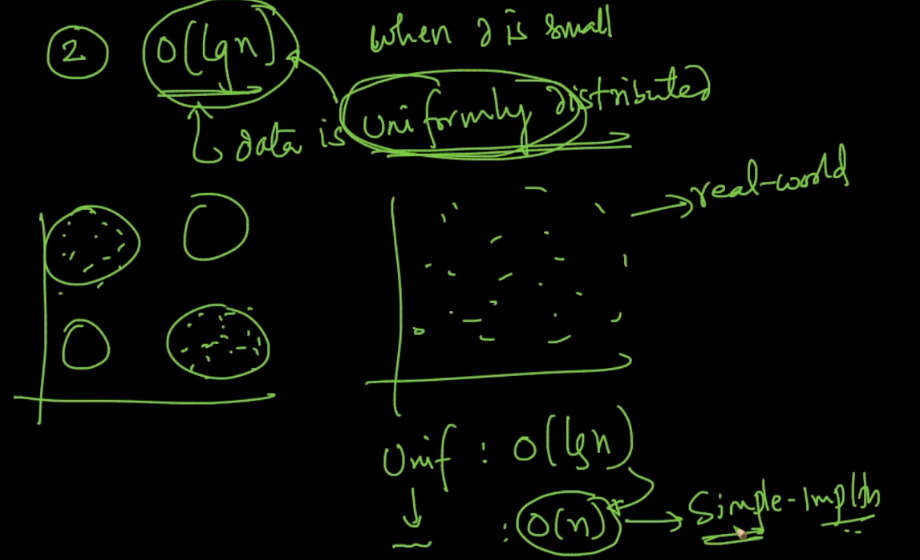
Where 2d is approximately equal to n, so it would be O(n\*logn) or o(n)

So we can’t use kd-tree whenever d is not small(1 – 5), because a small increase in dimension would dramatically increase the time.



Again Even if our dimensions are small, then if our data is not uniformly distributed then time complexity will start moving towards O(n) from O(log n).

And in real word most of the data are not uniformly distributed.



KD-Tree should not be used for larger dimensional data. It has to be used on data with low or moderate number of dimensions. As the dimensionality increases, we might get a poor performance in terms of efficiency of nearest neighbor search using KD-trees. This is because we won’t be able to prune many partitions as in high dimension, the radius of our nearest neighbour would intersect many different partitions, which would force us to look for a better nearest neighbour among the points stored in them. So we won’t be able to prune many of these partitions, and what ends up happening is that we would have to search many partitions which completely defeats the purpose of using KD-trees.