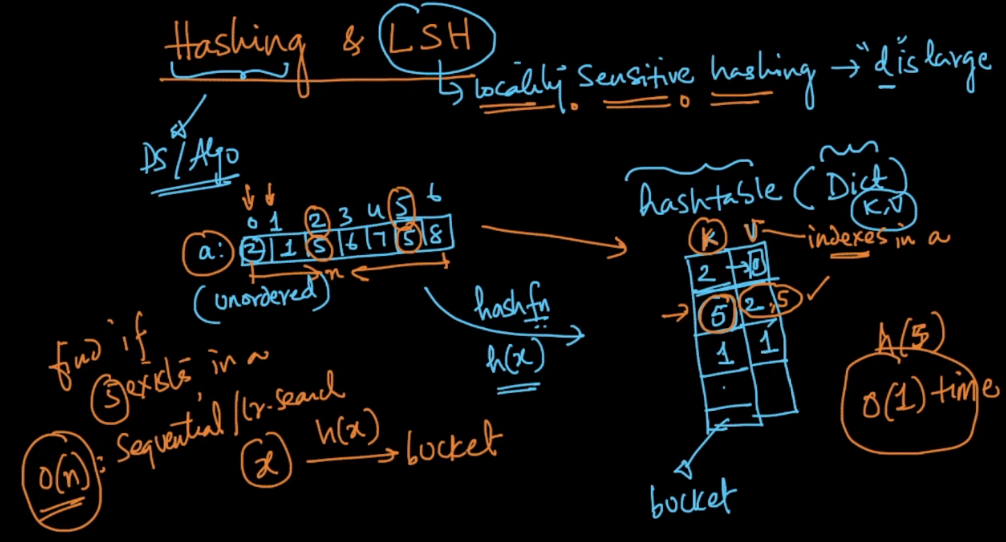
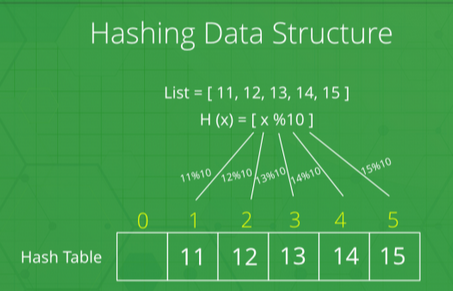
Below 2 image explains hashing.



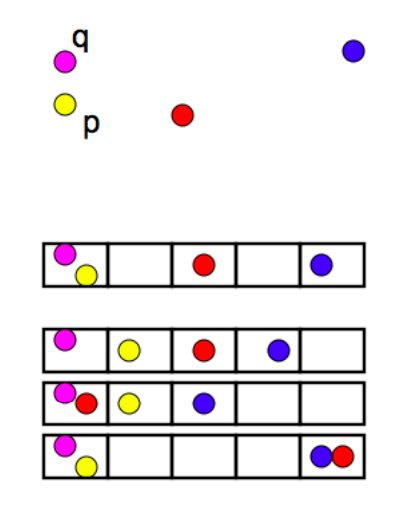
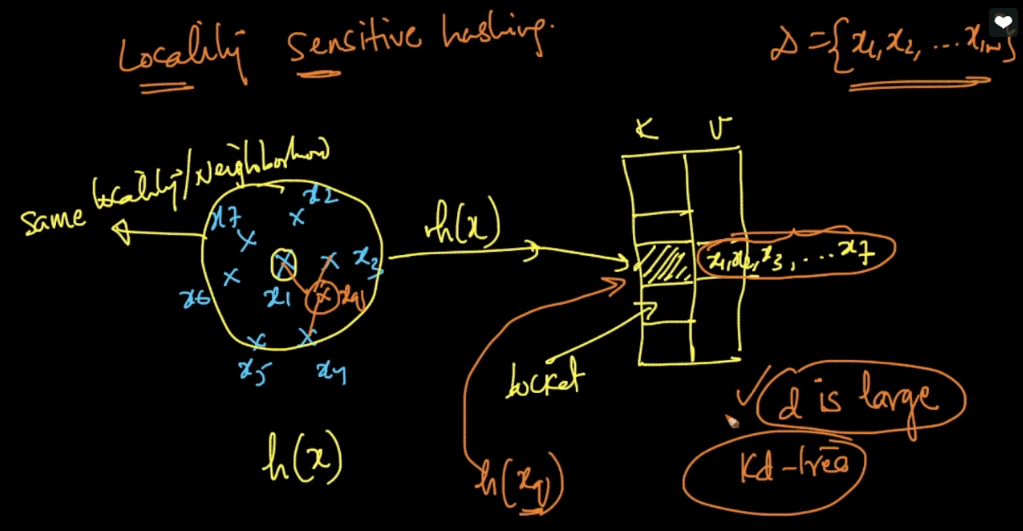


pto

LSH (Locality Sensitive Hashing):

**LSH** refers to a family of functions (known as LSH families) to hash data points into buckets so that data points *near each other are located in the same buckets with high probability*, while data points *far from each other are likely to be in different buckets*. This makes it easier to identify observations with various degrees of similarity.

In LSH, h(x) will compute the key for each dat point x and that key will be your entry in hast table and the value in each hash entry contains a list of all of the points that are close to each other i.e, Hash function returns a unique value in the hash table based on that unique value we will get nearest neighbors

So now whenever a new query point comes, we’ll apply it to hash function which gives us the index or bucket in which all the similar element exists, and then we get only the similar points to that query point, And only from this similar points we’ll find K nearest neighbors.

**Comments:**

* How do we perform k-NN with CV using LSH

You simply break the data into train, CV and test. Build the hashtable using only train data and measure the performance metric on CV-dataset for various values of K. The K that results in best performance on CV-dataset would be the hyper-param. This has got nothing to do with the method of finding NNs. Note that LSH and KD-Tree only help us find NNs faster. That's it.

* suppose we have lot of elements in a single bucket value then searching for the nearest one will lead to O(n) time complexity which will be the worst of worst cases so in that case do we need to find a better hash function which could create such hash table where any of the bucket does not hold much of values rather values are distributed widely among different buckets and if that is true can you please throw some light on how to find the best hash function for the same or sir we will drop the idea of using this concept ?

The hash function depends on the planes that are generated so we need to check with different planes so that generated planes not form big clusters. so for this, we need to generate random planes and check how those planes are dividing our data.