KNN:

Algorithm:

for every point (N)

→ find the distance (d) to every other point (N)

→ sort all those distances ($N \log N - N^2$)

→ take the points corres-ponding to the K-smallest (1) and do majority voting

total TC = $O(N^3)$

→ suffers from curse of Dimensionality

→ Distance based, no training required.

→ No learnable parameters.

→ Hyperparameter = K

```python
#Implementation of GridSearch
grid = {'n_neighbors':np.arange(1,235),
        'p':np.arange(1,3),
        'weights':['uniform','distance'],
        'algorithm':['auto','ball_tree','kd_tree','brute']
       }
knn = KNeighborsClassifier()
knn_cv = GridSearchCV(knn,grid,cv=3)
knn_cv.fit(X_train,Y_train)

print("Hyperparameters:",knn_cv.best_params_)
print("Train Score:",knn_cv.best_score_)
result_train["GridSearch-Best-Train"] = knn_cv.best_score_
```

```
Hyperparameters: {'algorithm': 'auto', 'n_neighbors': 9, 'p': 1, 'weights': 'distance'}
Train Score: 0.9472667596142674
```

p

Different distance metrics are available which use for the tree (like KDTree or Ball Tree). One of them is Minkowski metric. This p parameter is power parameter for Minkowski metric.

- If p = 1, this is equivalent to using **manhattan_distance(l_1)**
- If p = 2, this is equivalent to using **euclidean_distance(l_2)**
- If p > 2, It is **minkowski distance(l_p)**.

To who wonder what these functions are;

- EuclideanDistance ⇒ sqrt(sum((x - y)^2))
- ManhattanDistance ⇒ sum(|x - y|)
- MinkowskiDistance ⇒ sum(|x - y|^p)^(1/p)

KD-Tree:

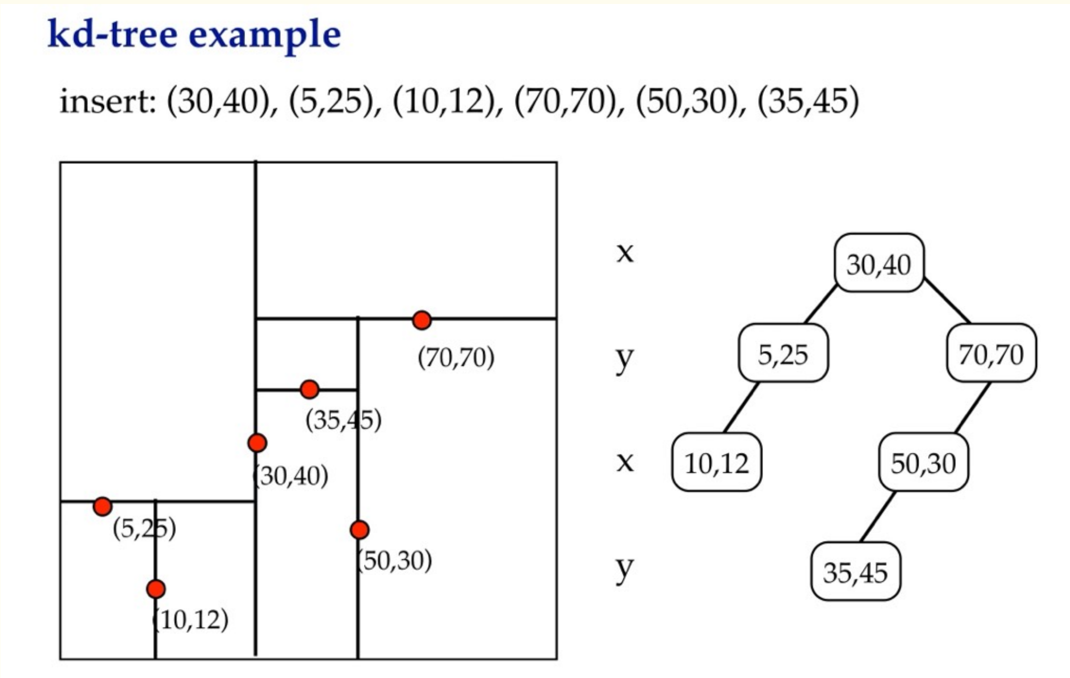
→ K-Dimension Tree is a Space partitioning data structure for organizing points in K-D space.

→ similar to BST.

→ KD tree are capable of guarantee a depth = $\log_2(n)$

n = # points in Dataset

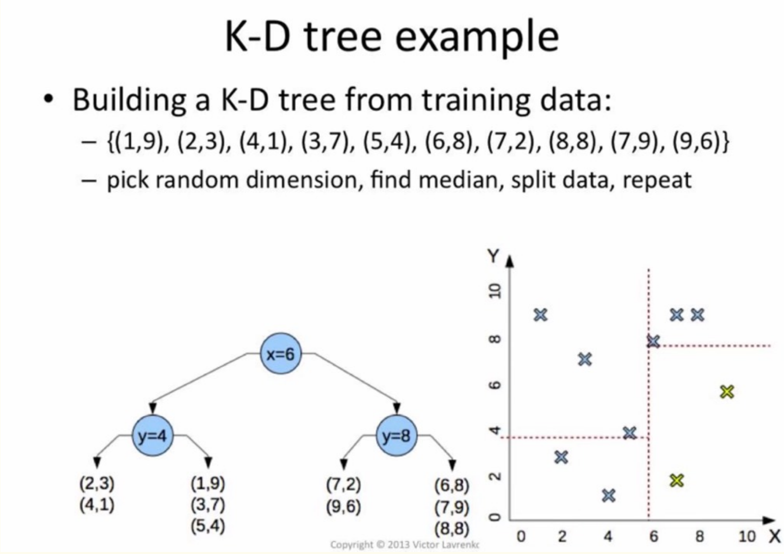
Approach-1:

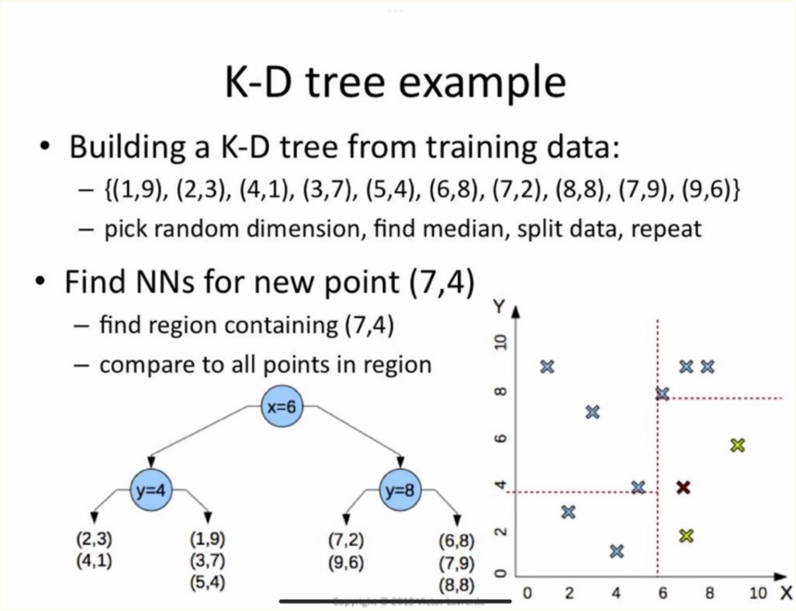


→ may need balancing → HEAP

→ divide space into hypercuboids in each split.

Approach-2:





→ median method produces balanced Binary Tree.

→ $\boxed{\text{TC}}$:

- height $= O(\log n)$
- find median every time $= O(n)$
- search $= O(\log n)$

$\therefore$ total $= O(n \log^2 n)$