REPORT

EFFICIENT BALL-TREE CONSTRUCTION

Team Members :

Devanshi Gupta (MT21024)

Niharika Poddar (MT21057)

Parul Jain (MT21064)

Suraj Singh (MT21099)

INTRODUCTION

Balltree is a simple geometric space partitioning data structure used to partition data points in n-dimensional space.

Balltree is a complete binary tree where each node represents an n-dimensional hypersphere called “ball”. Each internal node represents the minor ball containing the balls of its children and partitions the data points into two disjoint sets. The hypersphere(balls) may intersect each other but points are assigned to a particular ball depending on distance from the centers of balls.

Ball trees have a broad range of applications in geometric learning tasks, nearest neighbor retrieval, intersection and constraint queries, and probability maximization.

We have studied five different algorithms used to develop these data structures.

The structure of the ball tree is as follows - balls represent the coordinates, that is, center and radius length. Balltree is a complete binary tree, with nodes representing the balls, leaf nodes storing any information and the internal nodes are to guide the leaf nodes.

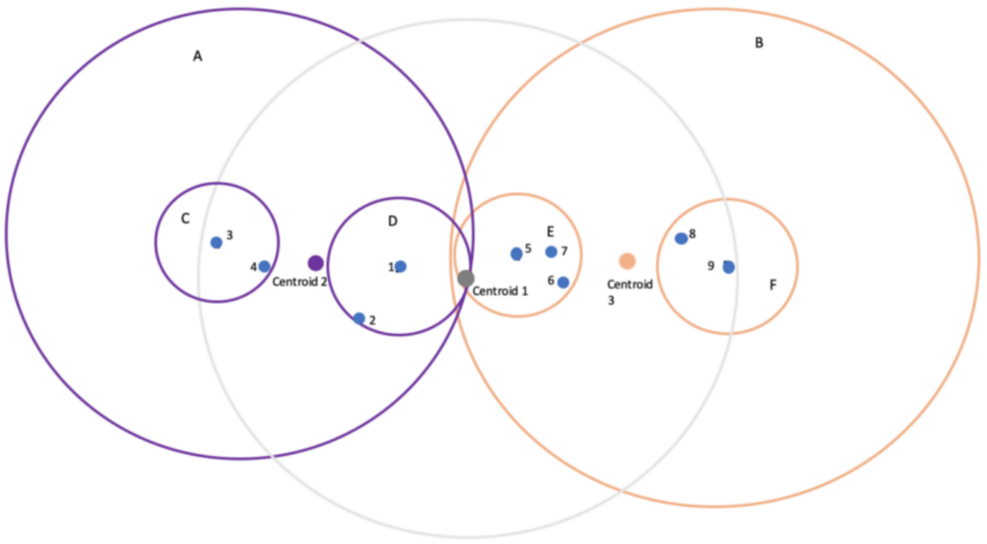
Among all the different algorithms, the bottom-up approach produces the best trees but with the longest construction time.

BALL TREES AND BALLS

In the n-dimensional Euclidean space, region under a hyper-sphere is referred to as a ball.

Balls are represented by point values specifying coordinates of their center and length of the radius.

A complete binary tree where a ball is connected to each node such that the ball of an interior node is the smallest which has a ball of its children is called a BALLTREE. Unlike node regions in kd trees, siblings in balltrees can intersect. The criteria for a good ball tree structure depends on - type of query and on the nature of data it stores.



Visualization of the Ball Tree Algorithm.

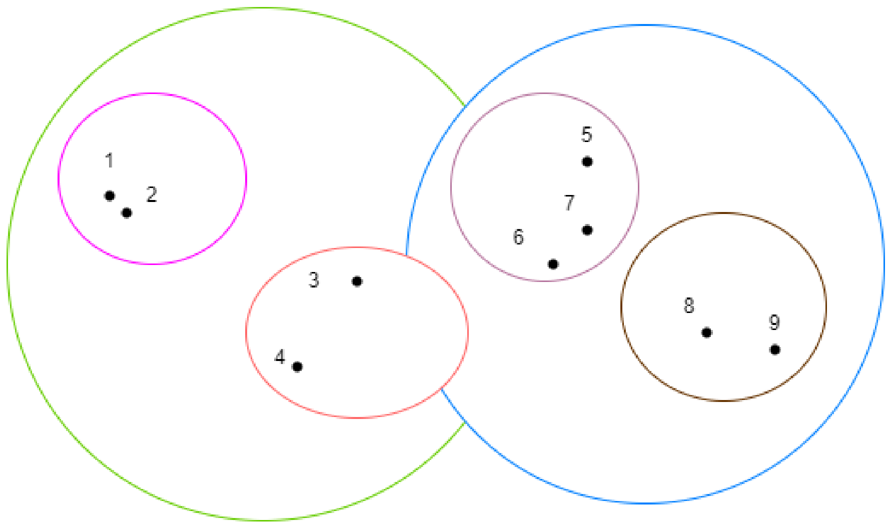


Figure 2: Ball Tree Decomposition example

BALL TREE ALGORITHMS

There are five algorithms:

- ★ K-d Construction Algorithm
- ★ Top Down Construction Algorithm
- ★ On-line Insertion Algorithm
- ★ Cheaper On-line Algorithm
- ★ Bottom Up Construction Algorithm

KD Construction Algorithm

The KD Tree Construction Algorithm is one of the Nearest Neighbor Algorithms. Data points are split at each node into two parts. KD Tree algorithm is also a binary tree algorithm. The splitting for the same is done mostly using the median.

For low-dimensional data, the KD Tree Algorithm might be the best solution. As seen above, the node divisions of the KD Tree are axis-aligned and cannot take a different shape. So the distribution might not be correctly mapped, leading to poor performance.
For a high-dimensional space, the Ball Tree Algorithm might be the best solution. Its performance depends on the amount of training data, the dimensionality, and the structure of the data. Having many data points that are noisy can also lead to a bad performance due to no clear structure. It is an offline top down construction algorithm.
Splitting criteria is based on the median, and is compared with the center coordinate of a given dimension.

Time complexities

- ➢ Finding median O(N)
- ➢ LogN height of the tree
- ➢ Total TC O(N logN)

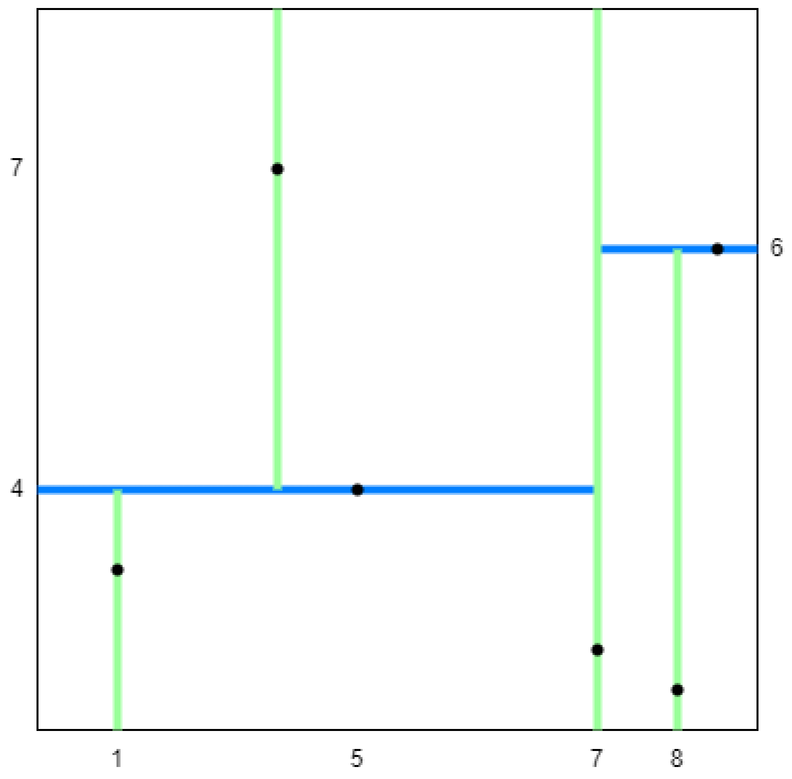
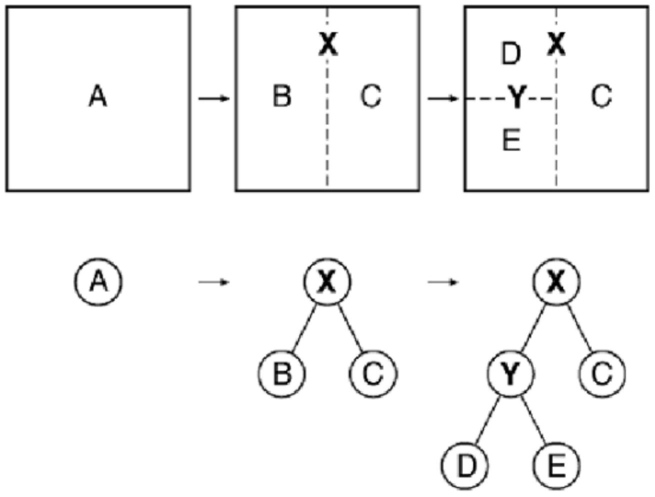
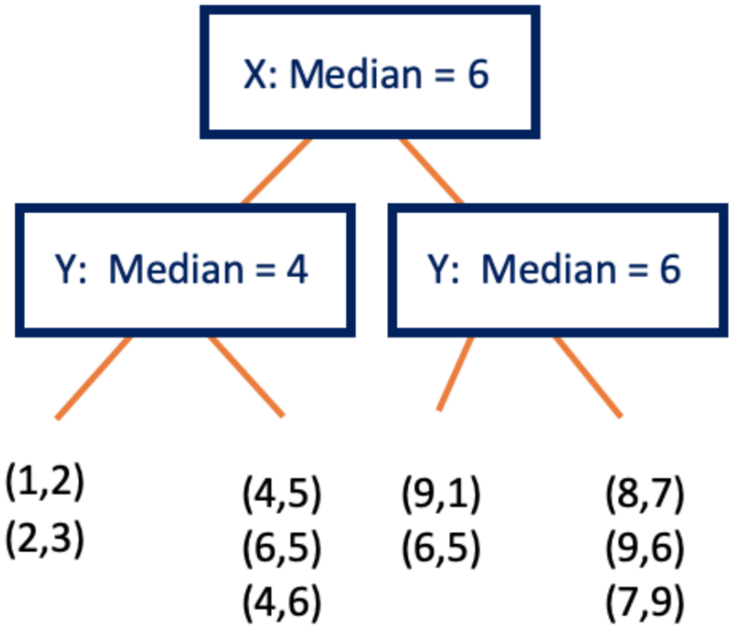


Figure 1: K-D Tree decomposition example





The KD tree.

Top-Down Construction Algorithm

At each stage, we choose the split dimension and the splitting value along that dimension to minimize the total volume of the two bounding balls of the two sets of balls. To find this optimal dimension and split value, we sort the balls along each dimension and construct a cost array that gives the cost at each split location.

While the exact volume of the bounding ball of a set of balls depends on the order in which they are inserted, this approach gives a good approximation to the actual parent ball volume.

Time complexity

Finding split location :NlogN .

- Height of tree: logN,
- Total TC: N(logN)2

NOTE:

- ❖ The Kd tree does not minimize the volume.
- ❖ Top-down uses explicit volume minimization heuristic to minimize The total volume of two bounding balls.
- ❖ Explicit volume minimization is used to improve performance by
    - Choose dimension to cut (split dim at each stage)
    - Value at which to cut (splitting value)

Steps to find the best dimension and split location

- ➢ Sort balls in each dimension
- ➢ Cost array (cst) : cost of each split location (volume)
- ➢ Pass from left to right and expand the test ball and insert vol in the array
- ➢ Pass from right to left and the bounding ball's volume is added to the array.
- ➢ TC: O(N log N)
- ➢ TC of whole algorithm: O(N (log N)^2)

On-line Insertion Algorithm

This algorithm is based on tree construction which we call a Ball Tree. A new node is added to an existing balltree in such a way that it tries to minimize the overall increase in the volume of the balltree.
By adding a new node along with the addition of a volume of the new leaf node two more additions also happen: the parent's volume and volume expansion of the ancestors above the new leaf node.
As new nodes keep on adding, the volume expansion of the ancestors increases. A priority queue is maintained ordered by the ancestor volume expansion.
Termination point: When the smallest ancestor expansion becomes greater than the expansion by the best node in the tree, it gets terminated.
It keeps the track of best insertion points.
New balls larger than the existing balls are kept at the top.
Balls that are far from existing balls are also kept near the top of the tree.
In such a way Ball tree is constructed by recursively adding the new balls to the tree structure.

Cheaper Online Algorithm

This algorithm has no priority queue. The cheaper of the two child nodes at any point is explored further. The search is terminated when the ancestor expansion exceeds the best total expansion.

Bottom-Up Construction Algorithm

This one is similar to the Huffman algorithm.

Firstly we find two balls with the smallest bounding ball, then make them siblings and insert the parent ball back.

For Brute force implementation :

Time complexity : N passes x O(N^2) = O(N^3).

Observations

- A node keeps track of the other node such that vol. Minimized. Node with a minimum stored cost + its mate = Best pair to join.
- Balls with same mate -> mate paired elsewhere -> cost increase.
- A ball tree is ideal to determine the Best mate.
- Thus, we can say we have improved the bottom-up algorithm.

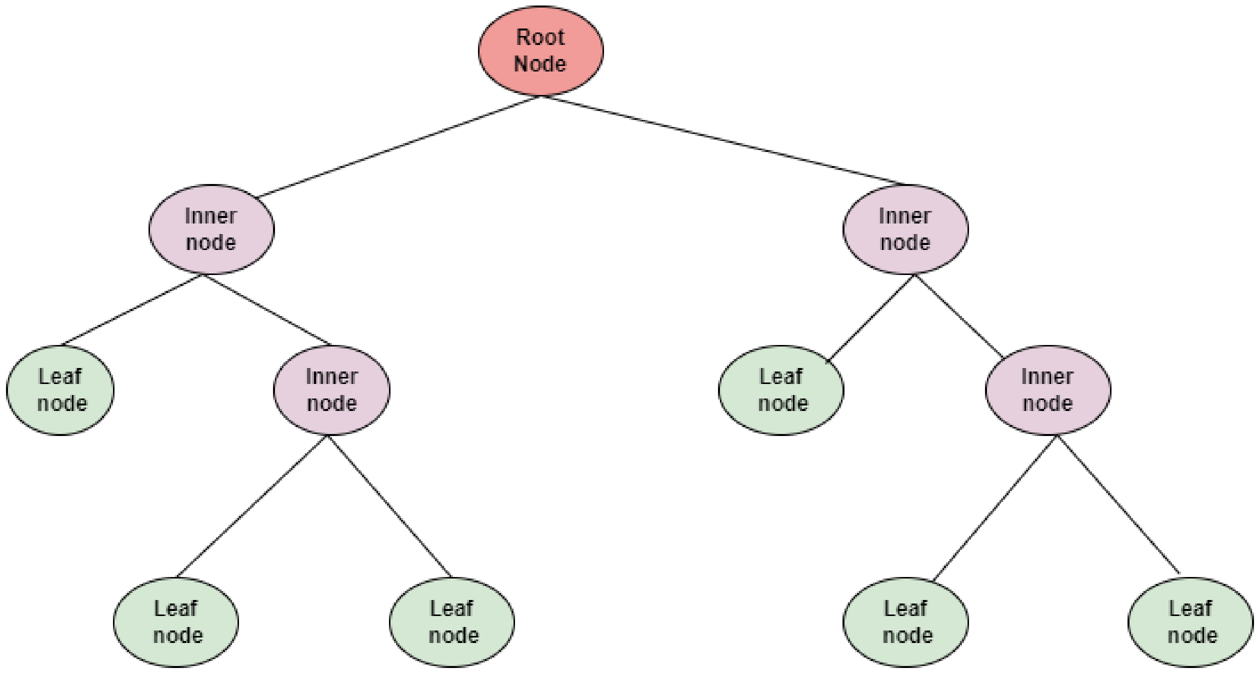
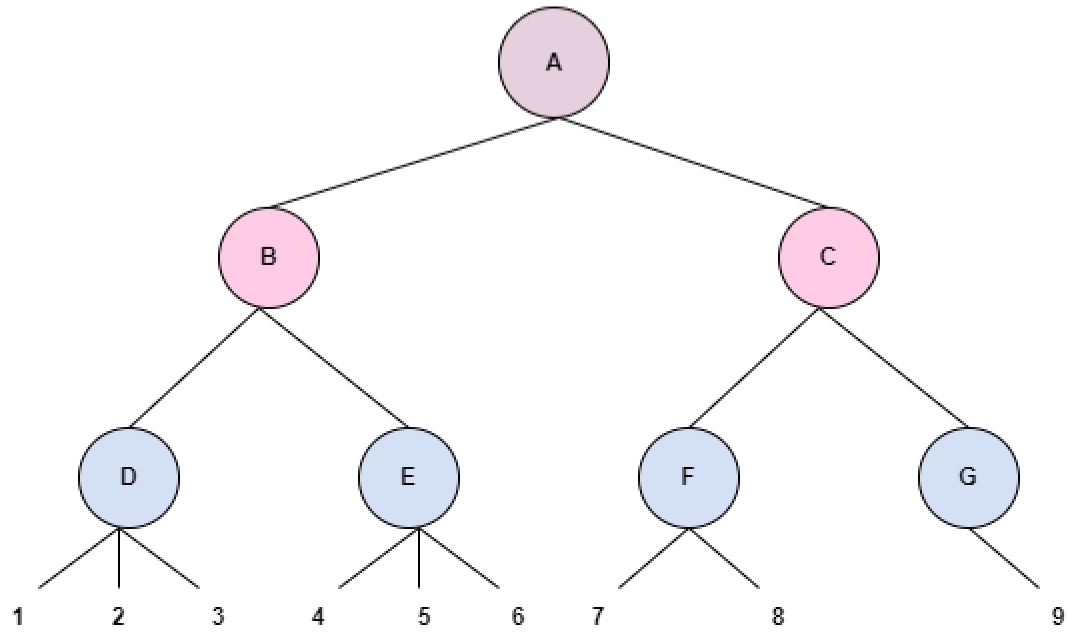


Figure 3 : The Structure of Tree



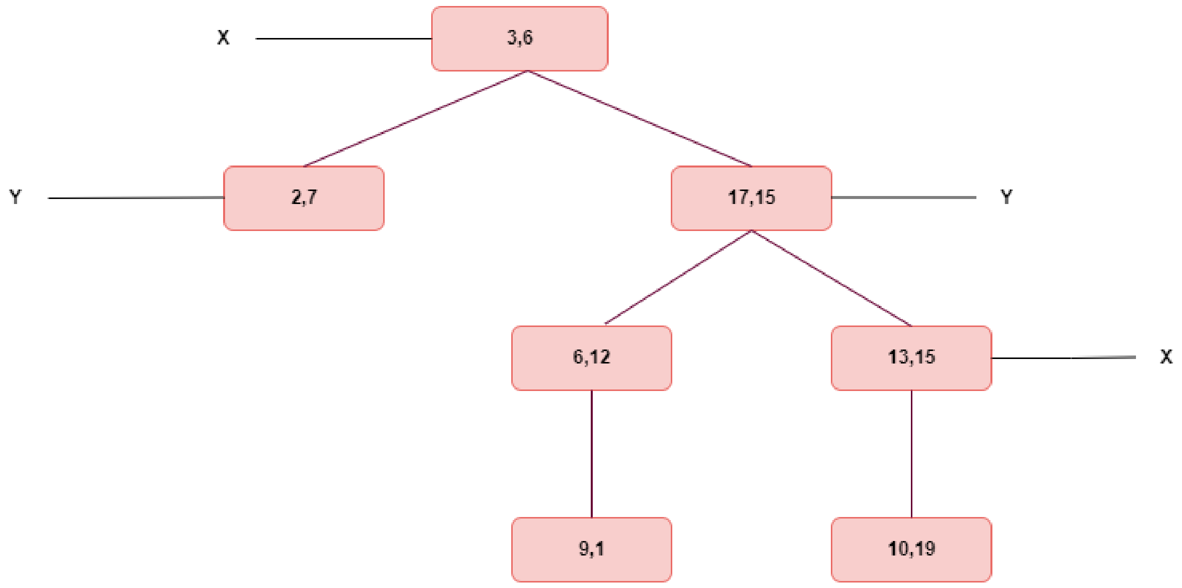
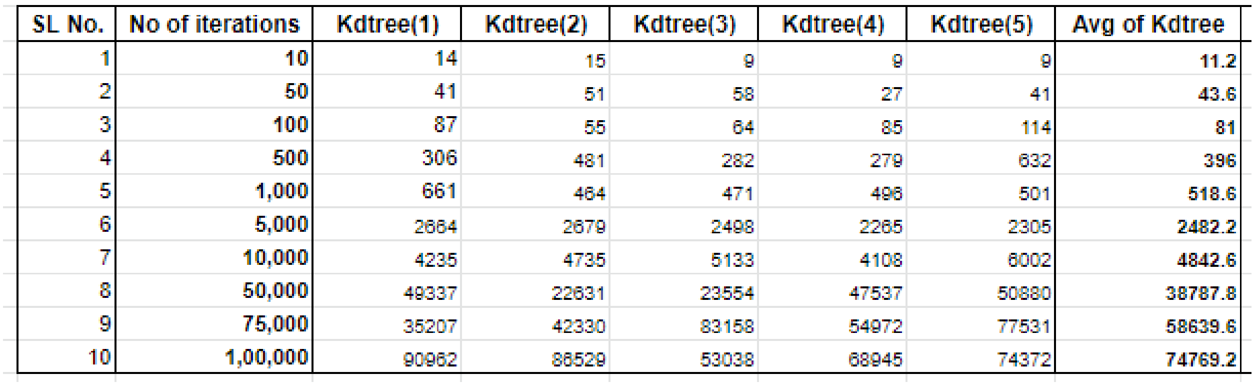


Figure 4 : Example of K-D Tree

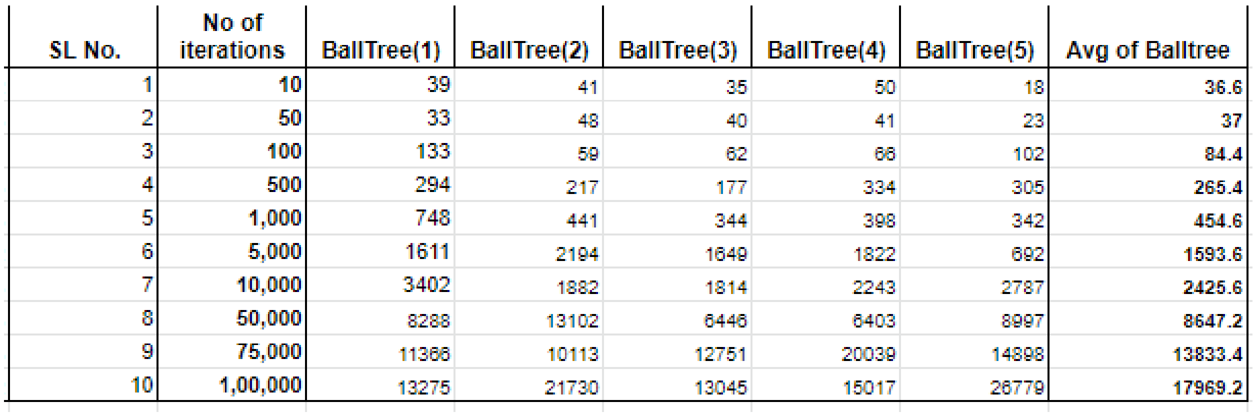
Comparisons and Results

Time taken at different no. of Iterations for KD Tree and Ball Tree

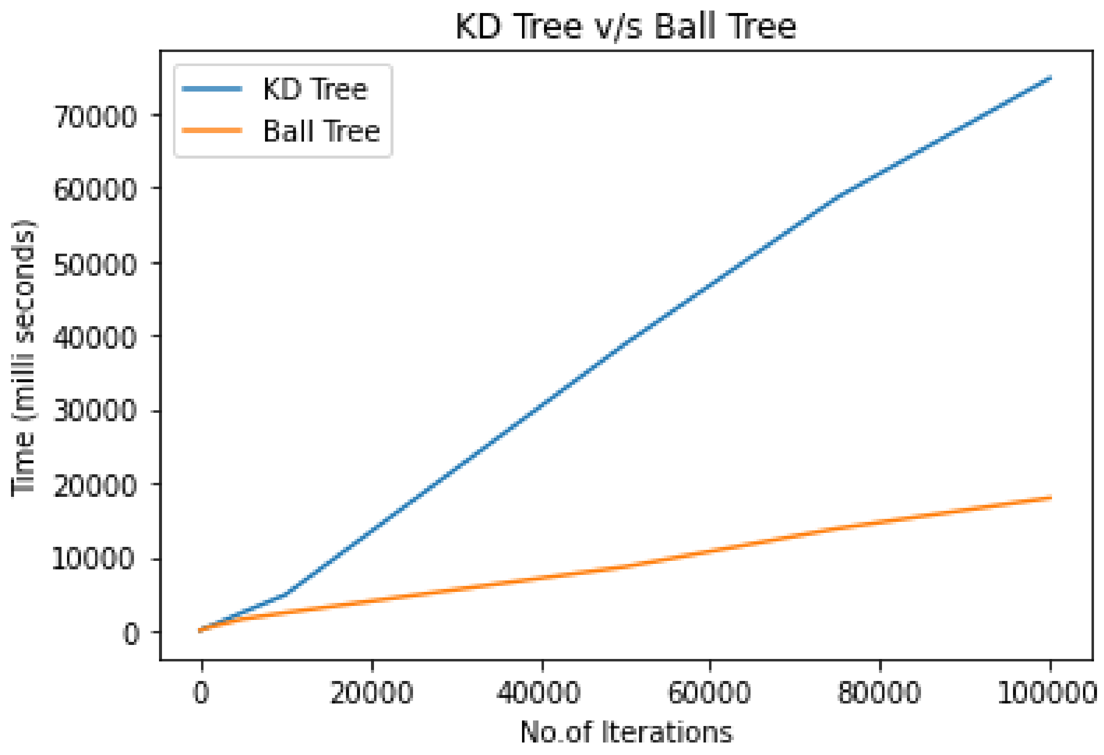
We have observed the performance of KD tree and Ball Tree for different numbers of Iterations several times and observed that the Ball tree has better performance than KD tree.



SL No.	No of iterations	Kdtree(1)	Kdtree(2)	Kdtree(3)	Kdtree(4)	Kdtree(5)	Avg of Kdtree
1	10	14	15	9	9	9	11.2
2	50	41	51	58	27	41	43.6
3	100	87	55	64	85	114	81
4	500	306	481	282	279	632	396
5	1,000	661	464	471	496	501	518.6
6	5,000	2664	2679	2498	2265	2305	2482.2
7	10,000	4235	4735	5133	4108	6002	4842.6
8	50,000	49337	22631	23554	47537	50880	38787.8
9	75,000	35207	42330	83158	54972	77531	58639.6
10	1,00,000	90962	86529	53038	68945	74372	74769.2



SL No.	No of iterations	BallTree(1)	BallTree(2)	BallTree(3)	BallTree(4)	BallTree(5)	Avg of Balltree
1	10	39	41	35	50	18	36.6
2	50	33	48	40	41	23	37
3	100	133	59	62	66	102	84.4
4	500	294	217	177	334	305	265.4
5	1,000	748	441	344	398	342	454.6
6	5,000	1611	2194	1649	1822	692	1593.6
7	10,000	3402	1882	1814	2243	2787	2425.6
8	50,000	8288	13102	6446	6403	8997	8647.2
9	75,000	11366	10113	12751	20039	14898	13833.4
10	1,00,000	13275	21730	13045	15017	26779	17969.2



Conclusion :

Above graph and table shows a comparison between KDtree and BallTree performance. It is clearly visible that Ball tree which uses a top down construction algorithm works better than KDtree. Time required increases drastically with an increase in the number of iterations.