Distance in Motion Planning

Algorithms and Data Structures 2 – Motion Planning and its applications

University of Applied Sciences Stuttgart

Dr. Daniel Schneider

Recap

Algorithm 3: sPRM($\{(c^i_{init}, c^i_{goal})\}, r, n$)

```
E ← ∅, V ← ∅                                                          1
foreach ((c^i_init, c^i_goal) ∈ {(c^i_init, c^i_goal)}) do            2
    V ← V ∪ c^i_init ∪ c^i_goal                                       3
for j ← 0 to n do                                                     4
    V ← V ∪ CFreeSample()                                             5
foreach v ∈ V do                                                      6
    U ← Neighbors(v, V, r)                                            7
    foreach u ∈ U do                                                  8
        if (edgeIsValid(u, v)) then                                   9
            E ← E ∪ (u, v)                                            10
foreach ((c^i_init, c^i_goal) ∈ {(c^i_init, c^i_goal)}) do            11
    if connected(c^i_init, c^i_goal, V, E) then                       12
        σ_i = shortestPath(c^i_init, c^i_goal, V, E)                  13
    else                                                              14
        σ_i ← ∅                                                       15
return {σ_i}                                                          16
```

- In most algorithms there is the need to compute configurations that are “close” to a given configuration.

→What does close mean?

→When are configurations neighbours?

→How can you compute neighbours efficiently?

How is the distance defined?

Definition 2.6 (Metric) *A metric is a function* $\rho : X \times X \to \mathbb{R}$ *such that* $\forall a, b, c \in X$ *:*

1. **[Non–negativity]** $\rho(a, b) \geq 0$.
2. **[Reflexivity]** $\rho(a, b) = 0$ *if and only if* $a = b$.
3. **[Symmetry]** $\rho(a, b) = \rho(b, a)$.
4. **[Triangle inequality]** $\rho(a, b) + \rho(b, c) \geq \rho(a, c)$.

Exercise

- Proof that this is a metric or show a contra example:

$$\rho(a,b) = \sqrt{|a-b|} \quad ; a,b \in \mathbb{R}$$

Note: || is a metric

Solution

- Proof that this is a metric or show a contra example:

$$\rho(a,b) = \sqrt{|a-b|} \quad ; a,b \in \mathbb{R}$$

1) $\rho(a,b) = \sqrt{|a-b|} \geq 0, \quad as\ |a-b| \geq \pm 0$

2) $\rho(a,b) = 0 = \sqrt{|a-b|} \rightarrow |a-b| = \pm 0 \rightarrow |a-b| = 0 \rightarrow a = b$

3) $\rho(a,b) = \sqrt{|a-b|} = \sqrt{|b-a|} = \rho(b,a)$

Solution

- Proof that this is a metric or show a contra example:

$$\rho(a,b) = \sqrt{|a-b|} \quad ; a,b \in \mathbb{R}$$

$$4)\ \rho(a,c) = \sqrt{|a-c|} = \sqrt{|a-b+b-c|} \overset{\text{Triangle inequality of } ||}{\leq} \sqrt{|a-b| + |b-c|} \overset{(1)}{\leq} \sqrt{|a-b|} + \sqrt{|b-c|}$$

(1) $\sqrt{a+b} \leq \sqrt{a+2\sqrt{ab}+b} = \sqrt{(\sqrt{a}+\sqrt{b})^2} = \sqrt{a}+\sqrt{b}$

Common metrics

Definition 2.7 (L_p **Metric and Manhattan Metric**) *The* L_p *metric* $\rho : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ *for any* $p \geq 1$ *and* $x, x' \in \mathbb{R}^n$ *is defined as follows.*

$$\rho(x, x') = (\sum_{i=1}^{n} |x_i - x'_i|^p)^{\frac{1}{p}}. \tag{2.9}$$

For $p = 1$ *denote the Manhattan Metric as*

$$\rho(x, x') = \sum_{i=1}^{n} |x_i - x'_i|. \tag{2.10}$$

- These are two broadly used metric in motion planning.
- This is the multi-dimensional version of the metric in the exercise.
- With $p = 2$ we get the well-known Euclidean Metric.
- With $\mathrm{p} = 1$ we get the Manhattan Metric/ Chess-Board metric

Exercise

For the following points

x=(1/1)

y=(3/1)

z=(2/4)

compute the distance according to the $L_p$metric for $p = 1, p = 2,\ p = 3$ for each possible pair with using the characteristics 3 of the metric.

Solution

As a metric is symmetric only 3 distances may and have to be computed.

$$p = 1$$
$$\rho(x, y) = (|1 - 3|^1 + |1 - 1|^1)^{1/1} = 2$$
$$\rho(y, z) = (|3 - 2|^1 + |1 - 4|^1)^{1/1} = 4$$
$$\rho(z, x) = (|2 - 1|^1 + |4 - 1|^1)^{1/1} = 4$$

Solution

As a metric is symmetric only 3 distances may and have to be computed.

$$p = 2$$

$$\rho(x, y) = (|1 - 3|^2 + |1 - 1|^2)^{\frac{1}{2}} = \sqrt{2^2 + 0} = 2$$

$$\rho(y, z) = (|3 - 2|^2 + |1 - 4|^2)^{1/2} = \sqrt{(1^2 + 3^2)} = 3{,}162$$

$$\rho(z, x) = (|2 - 1|^2 + |4 - 1|^2)^{1/2} = \sqrt{1^2 + 3^2} = 3{,}162$$

Solution

As a metric is symmetric only 3 distances may and have to be computed.

$$p = 3$$

$$\rho(x, y) = (|1 - 3|^3 + |1 - 1|^3)^{\frac{1}{3}} = \sqrt[3]{2^3 + 0} = 2$$

$$\rho(y, z) = (|3 - 2|^3 + |1 - 4|^3)^{1/3} = \sqrt[3]{1^3 + 3^3} = 3{,}036$$

$$\rho(z, x) = (|2 - 1|^3 + |4 - 1|^3)^{1/3} = \sqrt[3]{1^3 + 3^3} = 3{,}036$$

But what about rotations?

Recap:

Rotations can be expressed by a matrix.

2D:

$$\begin{matrix} x' \\ y' \end{matrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} * \begin{matrix} x \\ y \end{matrix}$$

But what about rotations?

Recap:

Rotations can be expressed by a matrix.

3D:

$$
\begin{aligned}
R &= R_z(\psi) \cdot R_y(\vartheta) \cdot R_x(\varphi) \\
&= \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} cos(\vartheta) & 0 & sin(\vartheta) \\ 0 & 1 & 0 \\ -sin(\vartheta) & 0 & cos(\vartheta) \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\varphi) & -sin(\varphi) \\ 0 & sin(\varphi) & cos(\varphi) \end{pmatrix} \\
&= \begin{pmatrix} cos(\vartheta) \cdot cos(\psi) & -cos(\varphi) \cdot sin(\psi) + sin(\varphi) \cdot sin(\vartheta) \cdot cos(\psi) \\ cos(\vartheta) \cdot sin(\psi) & cos(\varphi) \cdot cos(\psi) + sin(\varphi) \cdot sin(\vartheta) \cdot sin(\psi) \\ -sin(\vartheta) & sin(\varphi) \cdot cos(\vartheta) \end{pmatrix} \\
&\quad \begin{matrix} sin(\varphi) \cdot sin(\psi) + cos(\varphi) \cdot sin(\vartheta) \cdot cos(\psi) \\ -sin(\varphi) \cdot cos(\psi) + cos(\varphi) \cdot sin(\vartheta) \cdot sin(\psi) \\ cos(\varphi) \cdot cos(\vartheta) \end{matrix} \Bigg) \qquad (2.13)
\end{aligned}
$$

But what about rotations?

Recap:

These matrices have some special properties:

Definition 2.9 (Rotation Matrix) *$R \in \mathbb{R}^{3\times3}$ is called a rotation matrix, if it fulfils the following properties*

- $R^T = R^{-1}$, *R is orthogonal*
- $det(R) = 1$

The group of Rotation matrices is called $SO(2)$ and $SO(3)$

How to measure rotational distance?

Definition 1 *Let the rotation matrices* $\mathrm{R}_A, \mathrm{R}_B \in \mathrm{SO}(3)$ *represent two rotations in Euclidean 3-space. The operation* $\mathrm{d}_\angle : \mathrm{SO}(3) \times \mathrm{SO}(3) \to [0, \pi]$ *defines the metric*

$$\mathrm{d}_\angle(\mathrm{R}_A, \mathrm{R}_B) := \angle\left(\mathrm{R}_B^{-1}\mathrm{R}_A\right), \tag{1}$$

where the angle operator $\angle(\cdot)$ *yields the rotation angle of the given rotation matrix after decomposition into rotation axis and angle.*

Source:
On the Relation between two Rotation Metrics, Thomas Ruland
https://arxiv.org/pdf/1512.04219.pdf

How to get the angle of a rotation matrix?

$$angle = \text{acos}(\frac{tr(A) - 1}{2})$$

A configuration,

which represents a translation and a rotation can be written as $c = \left(\vec{x}, R(\theta)\right)$ and in 3D:

Proposition 2.8 (Metric on the $SE(3)$) *Let X be $SO(3)$ with a metric ρ_x and Y be $\mathbb{R}^3$ with a metric ρ_y. A metric ρ_z for the $SE(3)$ can be constructed for the Cartesian Product $Z = X \times Y$ by the metric ρ_z as*

$$\rho_z(z_1, z_2) := c_1\rho_x(R_1, R_2) + c_2\rho_y(\overrightarrow{v_1}, \overrightarrow{v_2}) \tag{2.12}$$

in which $c_1 > 0$ and $c_2 > 0$ are any positive real constants, and $R_1, R_2 \in X$, $\overrightarrow{v_1}, \overrightarrow{v_2} \in Y$ and $z_1, z_2 \in Z$. Each $z_1, z_2 \in Z$ is represented as $z_1 = \begin{pmatrix} R_1 & \overrightarrow{v_1} \\ 0 & 1 \end{pmatrix}$ and $z_2 = \begin{pmatrix} R_2 & \overrightarrow{v_2} \\ 0 & 1 \end{pmatrix}$.

Exercise

- Given two configurations

$$c_1 = \left([0,1,0], \begin{matrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{matrix} \right)$$

$$c_2 = \left([2,1,2], \begin{matrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{matrix} \right)$$

Compute the distance of these two configurations ($c_1 = 0{,}5$ and $c_2 = 0{,}5$). For use the Euclidean distance for the vectors and the distance from Definition 1 for the rotations.

Solution

- Given two configurations

Distance Vector: $\sqrt{2}$

Distance Rotation: $\mathrm{acos}(\frac{0-1}{2})$

Distance Config: $0{,}5 * \sqrt{2} + 0{,}5 * \mathrm{acos}(-0{,}5) =$

Some notes

- In practice, especially in 3D, rotation matrices are rarely used. Why?
  - Numerical issues. If you rotate a point a few times (e.g. 45° multiped times) with a rotation matrix you will not end up where you have started.
  - Computation time. Each matrix has 3x3 entries that have to be computed.
- In practice rotations are not expressed by rotations matrices but by quaternions.
  - Quaternions are higher dimensional complex numbers.
  - These Quaternions can express rotations and there are metrics that can give the distance between quaternions as well.
  - Compute a rotations with a quaternion is faster (less computational effort)
  - Less numerical issues.

Brute force algorithm

How to compute the closest neighbour to a given config.

List:

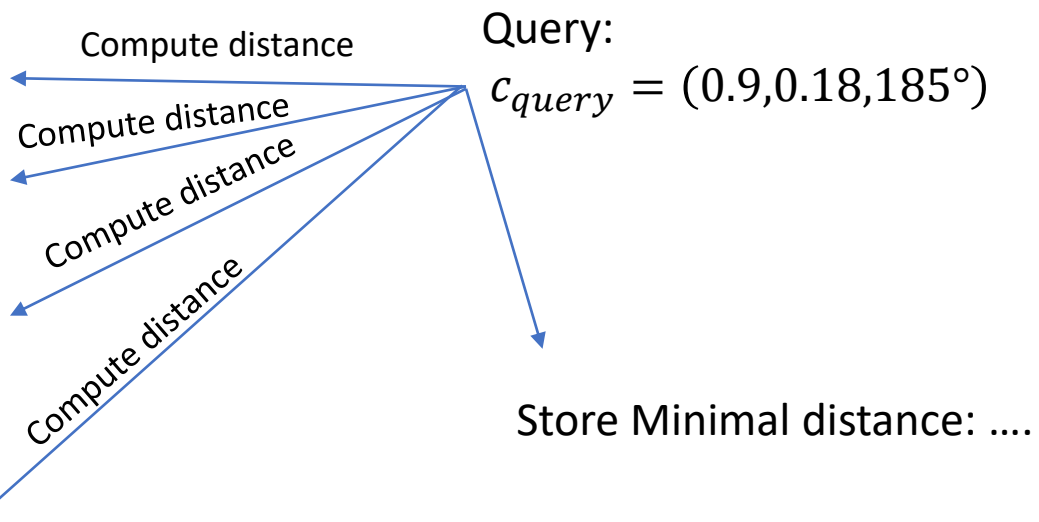
$c_1 = (0.1,0.2,45°)$

$c_2 = (0.4,0.1,10°)$

$c_3 = (0.9,0.2,190°)$

...

$c_n = (0.5,0.3,35°)$



Brute force algorithm

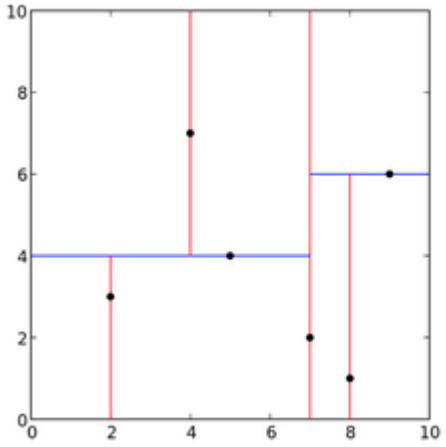
How to compute the closest neighbour to a given config:

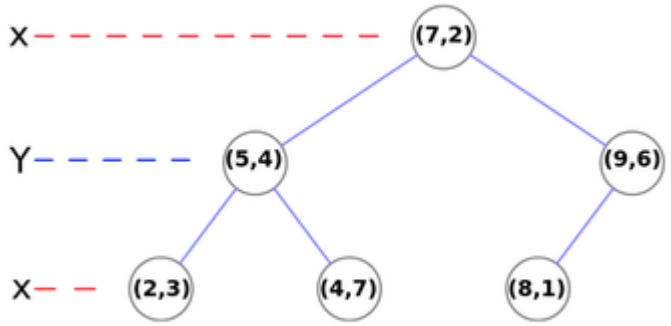
Notes:

- It is the most straight forward approach (→ Used in the practical work)
- This needs n distance computations. →It is getting slower with each added configuration sample in configuration space.
- For simple problems the approach the approach is fine but there are more sophisticated approaches.

The “fastest” way to compute neighbours

The kd-Tree:



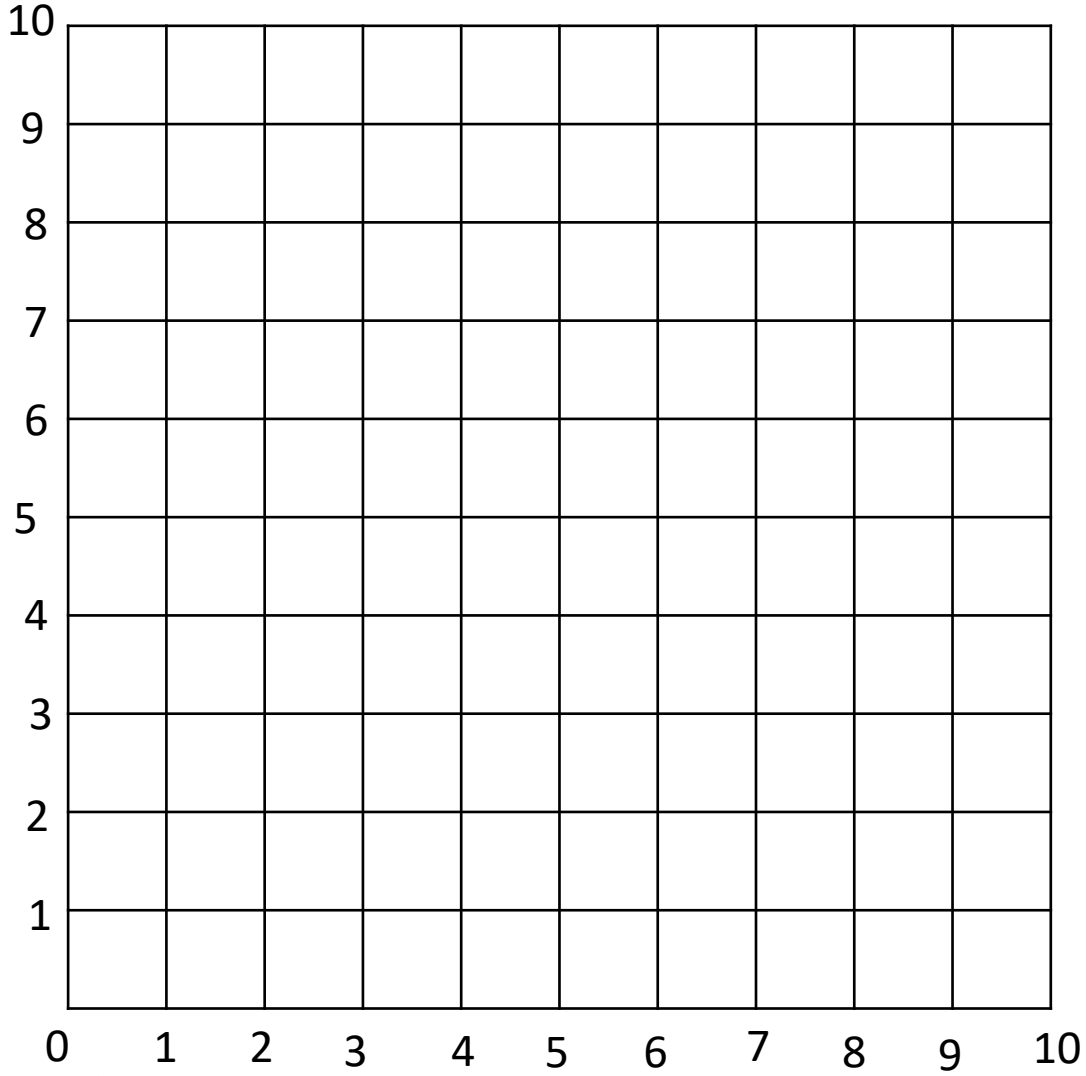


Source:
Steven Schmatz, BSE/MSE Computer Science, University of Michigan College of Engineering (2019)

Idea:

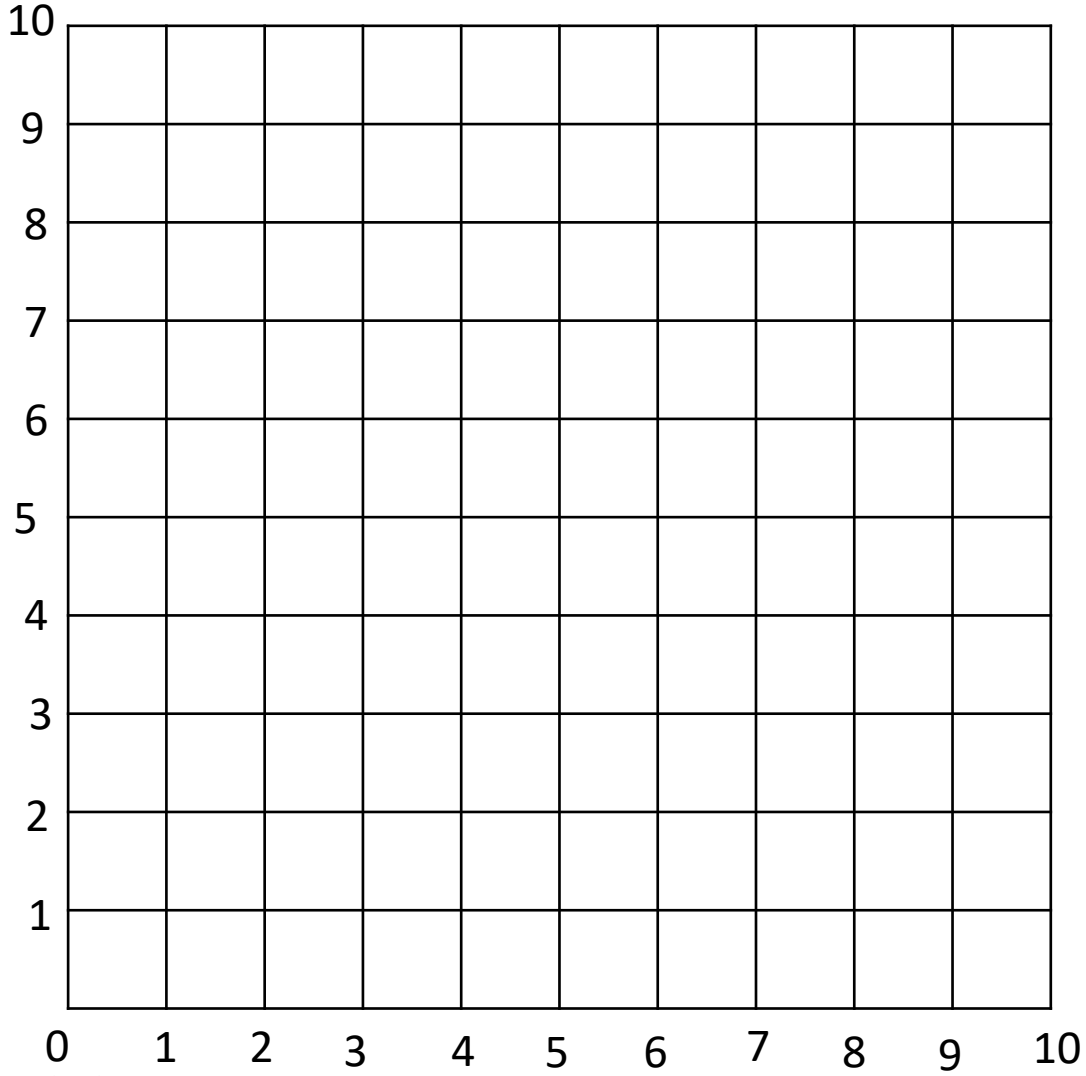
1. Use algorithmic geometry.
2. Divide the space into subparts and build up a tree.
3. On query → parts of the tree are skipped.
4. Not all n configuarations are checked.

The kd-Tree – Build the data structure



Input:
(8,4)
(3,2)
(1,5)
(0,8)
(7,4)
(5,5)
(7,1)

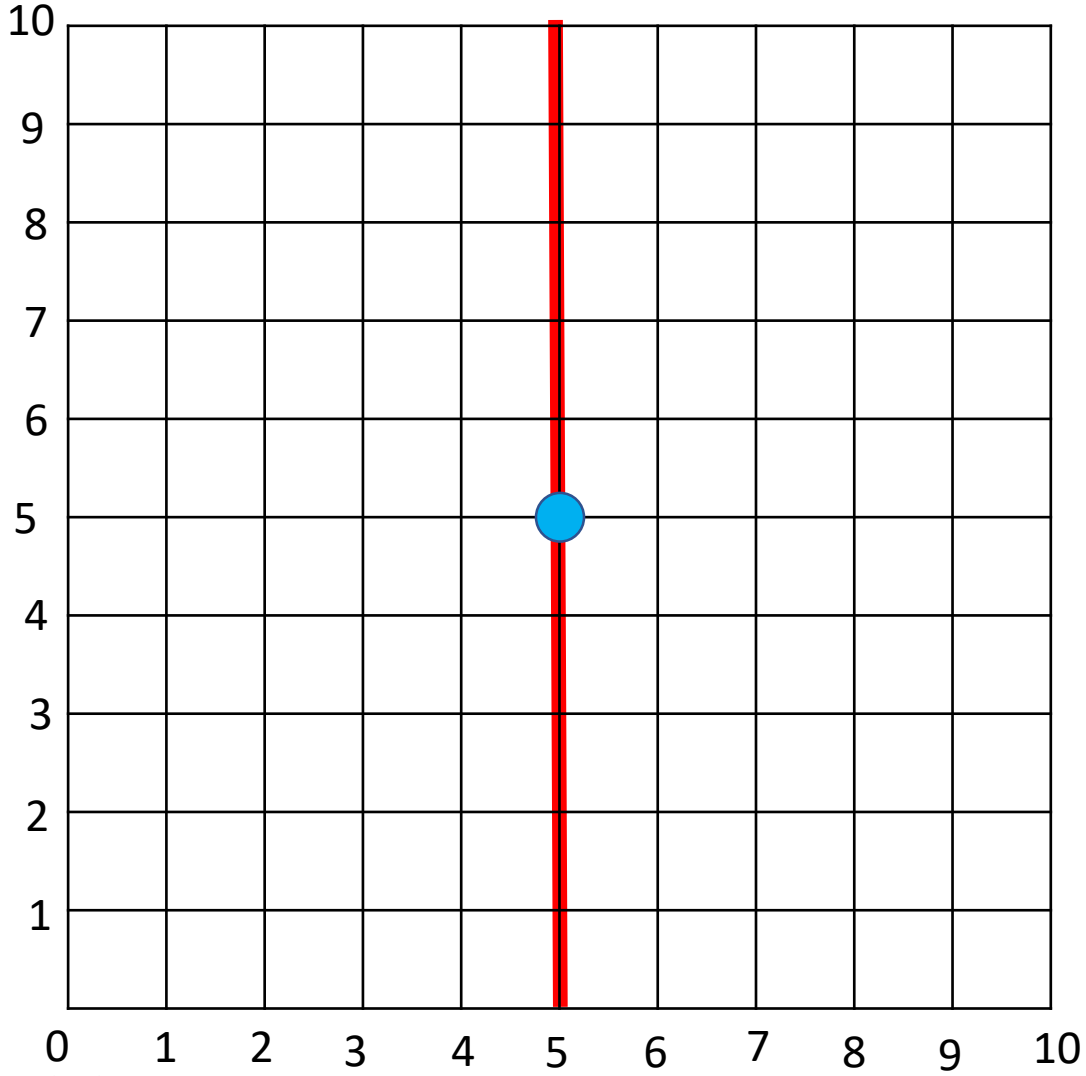
The kd-Tree – Build the data structure



Input:
(0,8)
(1,5)
(3,2)
(5,5)
(7,4)
(7,1)
(8,4)

Sort and find median in x

The kd-Tree – Build the data structure

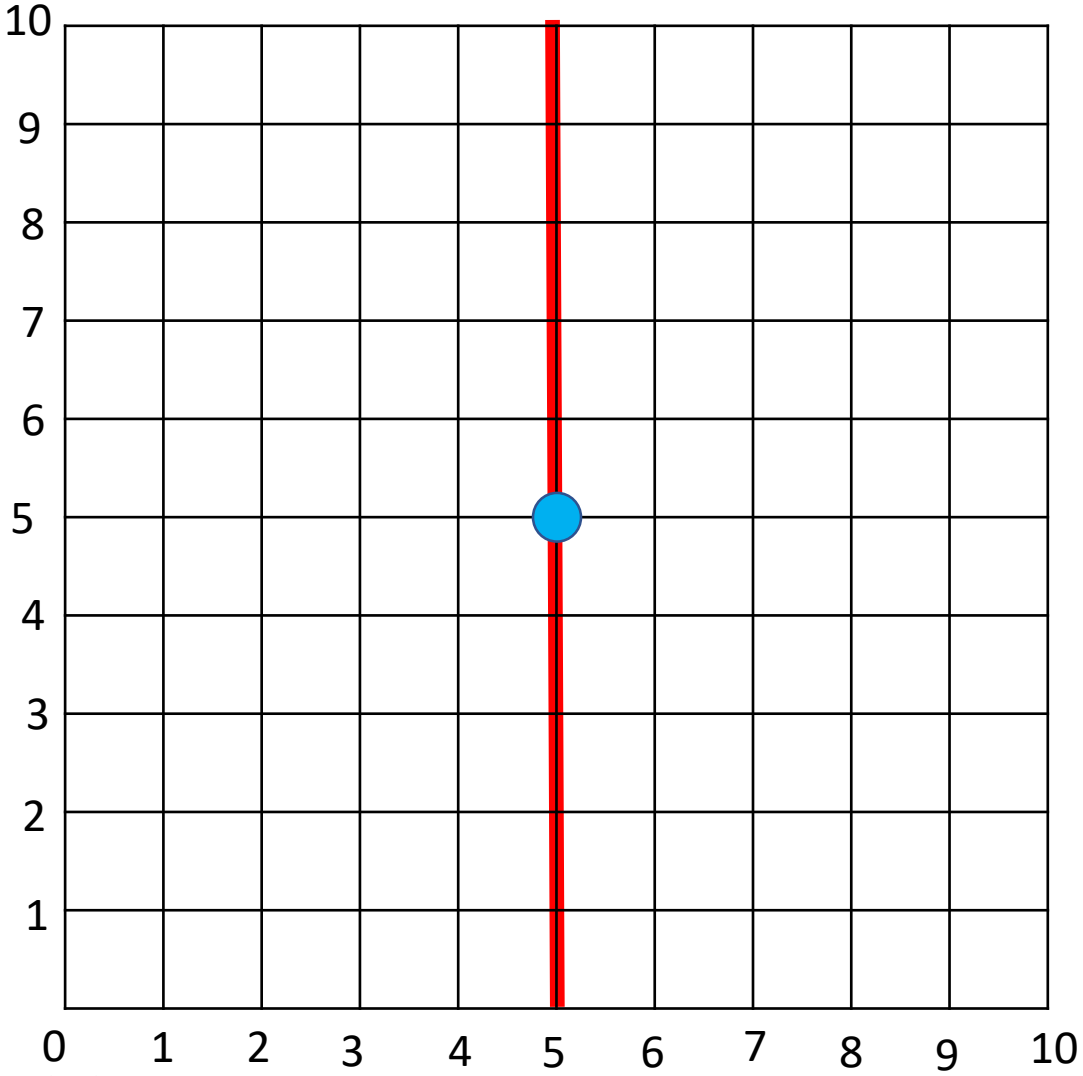


(5,5)

Split in x

Input:
(0,8)
(1,5)
(3,2)
(5,5)
(7,4)
(7,1)
(8,4)

The kd-Tree – Build the data structure



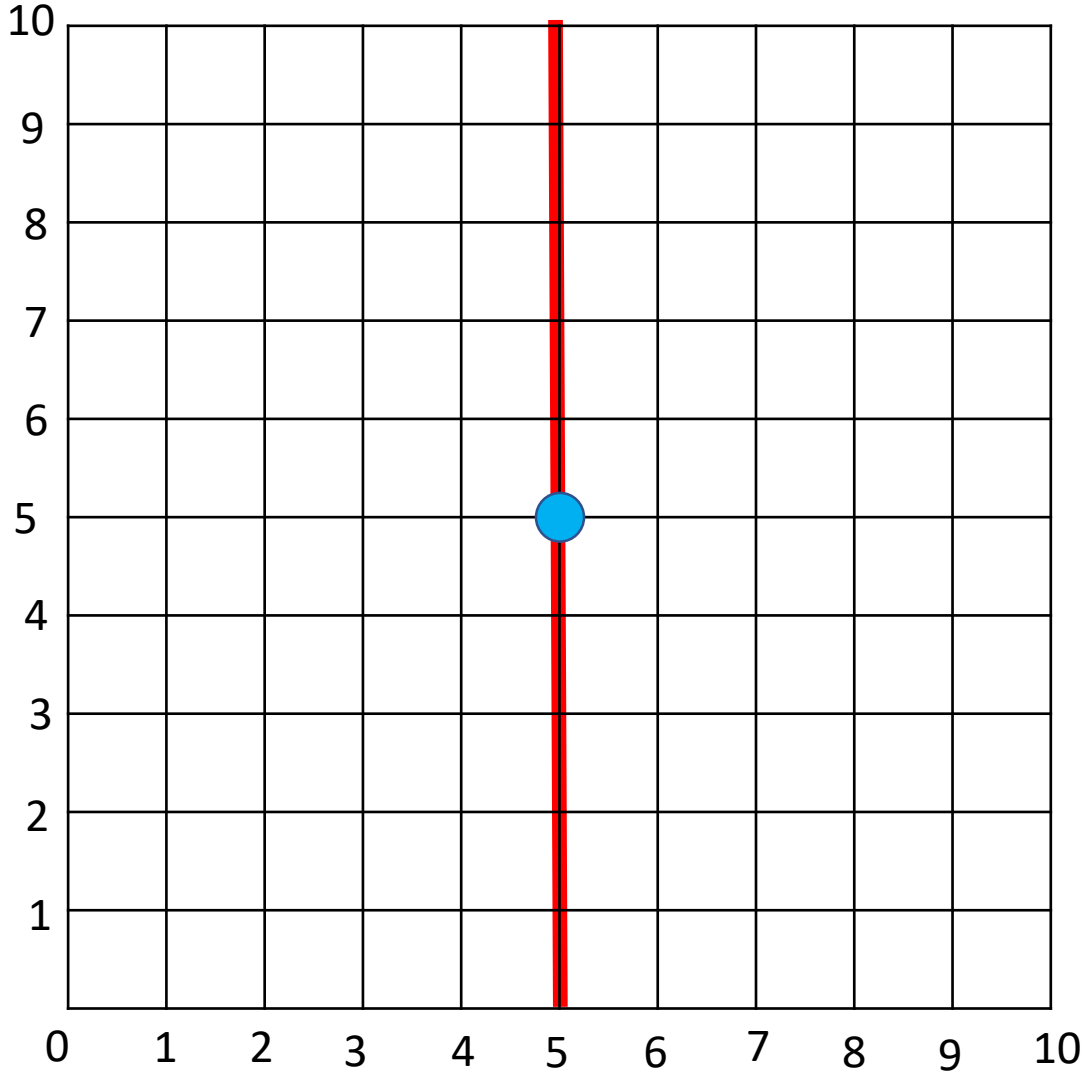
Input:
(3,2)
(1,5)
(0,8)
(5,5)
(7,1)
(7,4)
(8,4)

(5,5)

Sort both halves again this time in y.
Some Notes:

- If values in y equal take x as measure
- You can alternate the dimensions or you can pick them randomly

The kd-Tree – Build the data structure

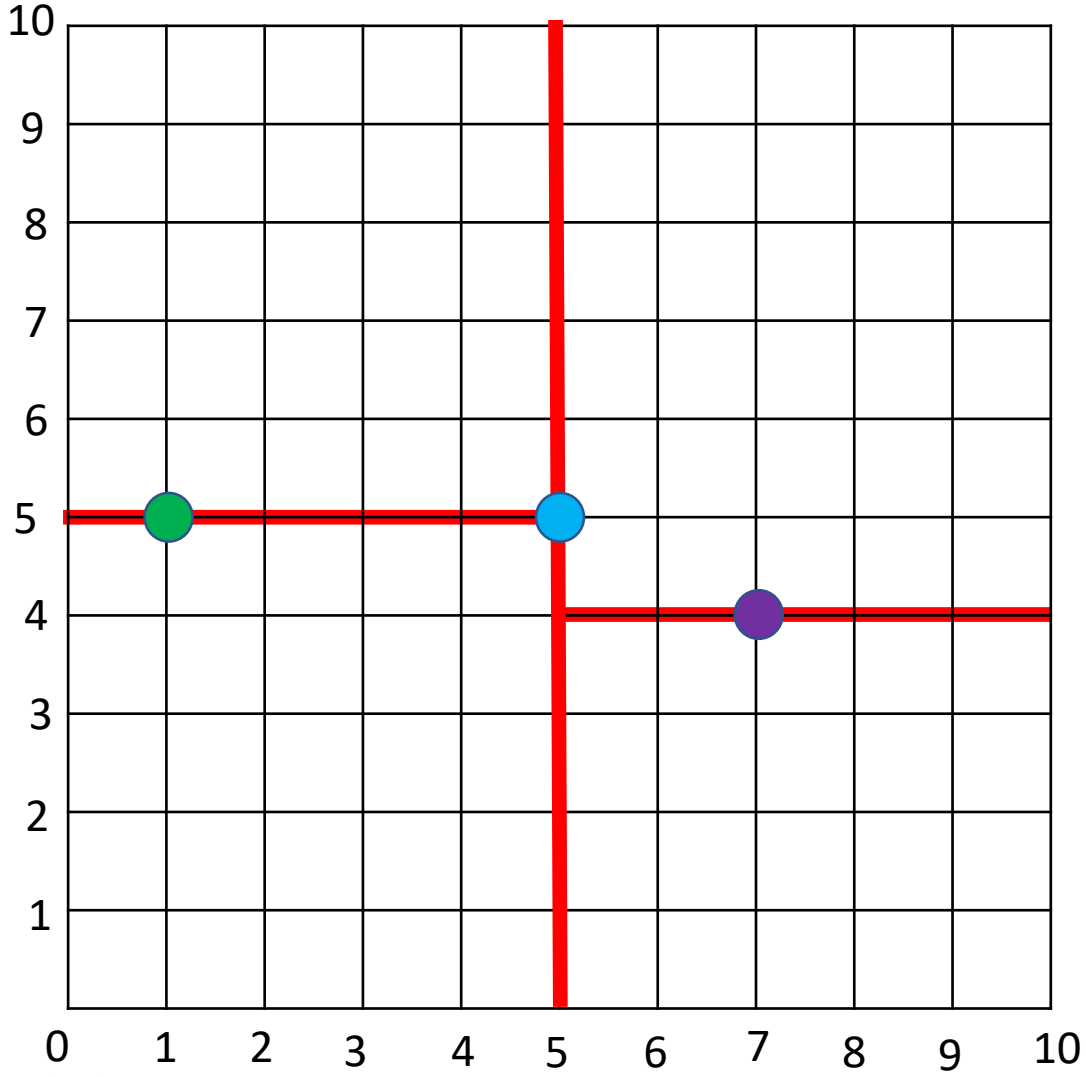


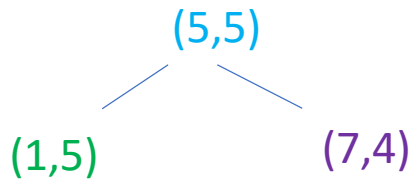
(5,5)

Input:
(3,2)
(1,5)
(0,8)
(5,5)
(7,1)
(7,4)
(8,4)

Find the median again.

The kd-Tree – Build the data structure

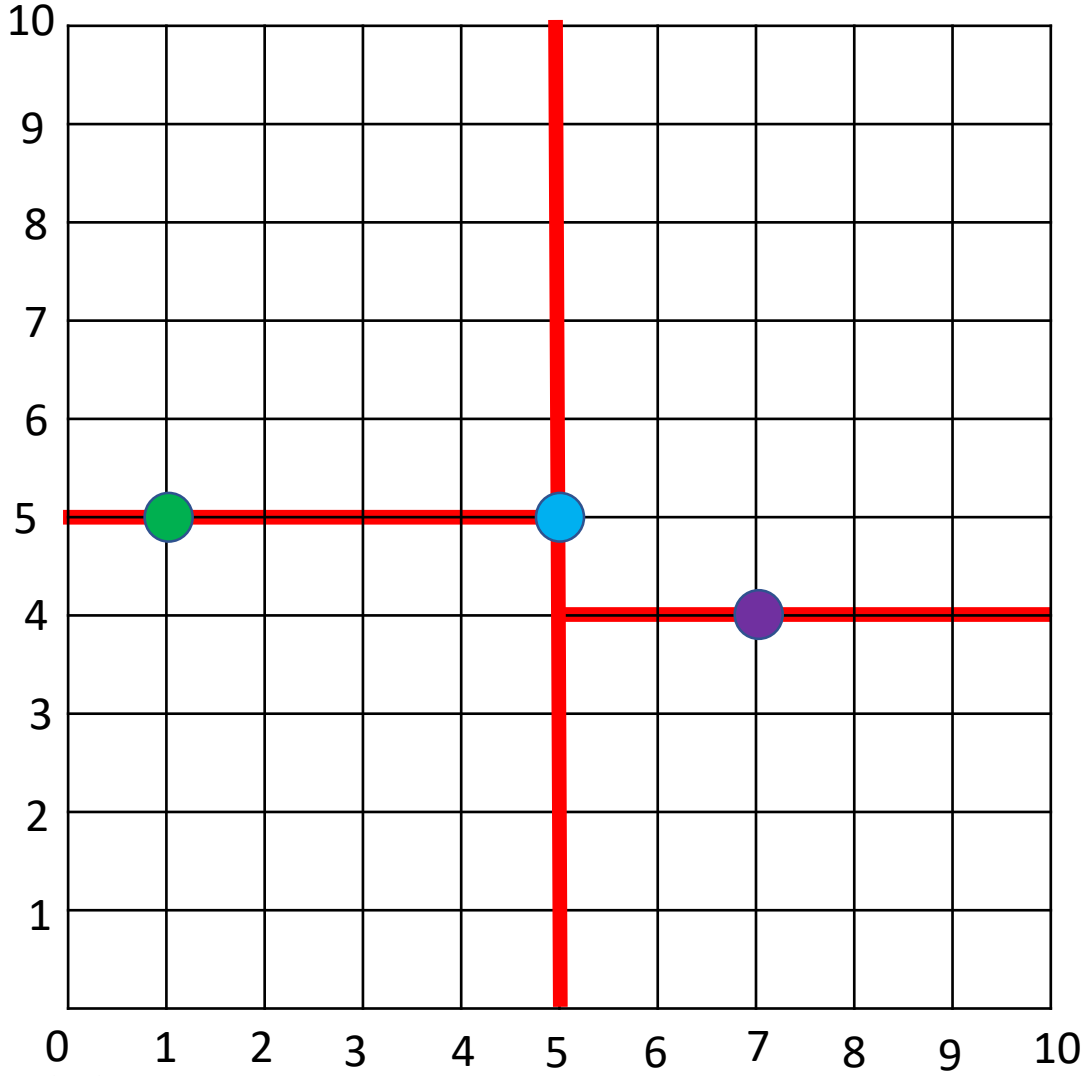


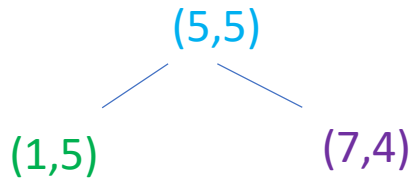


Split in y.

Input:
(3,2)
(1,5)
(0,8)
(5,5)
(7,1)
(7,4)
(8,4)

The kd-Tree – Build the data structure



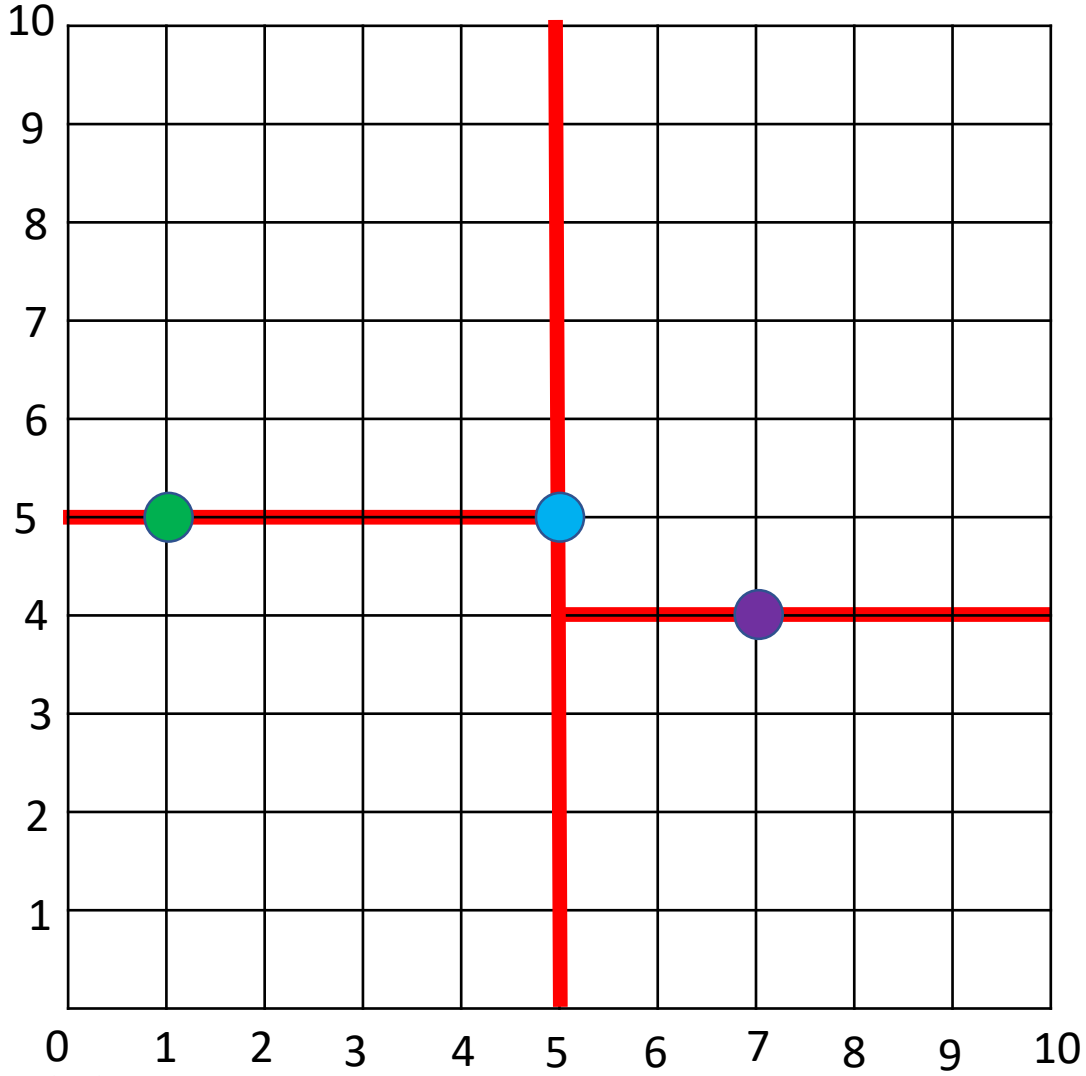


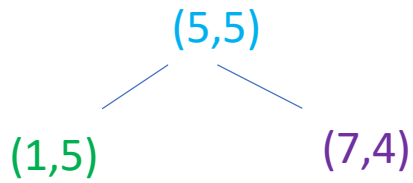
Input:
(3,2)
(1,5)
(0,8)
(5,5)
(7,1)
(7,4)
(8,4)

Sort in x again. → nothing happens as there is only 1 entry left.

The kd-Tree – Build the data structure

Input:
(3,2)
(1,5)
(0,8)
(5,5)
(7,1)
(7,4)
(8,4)

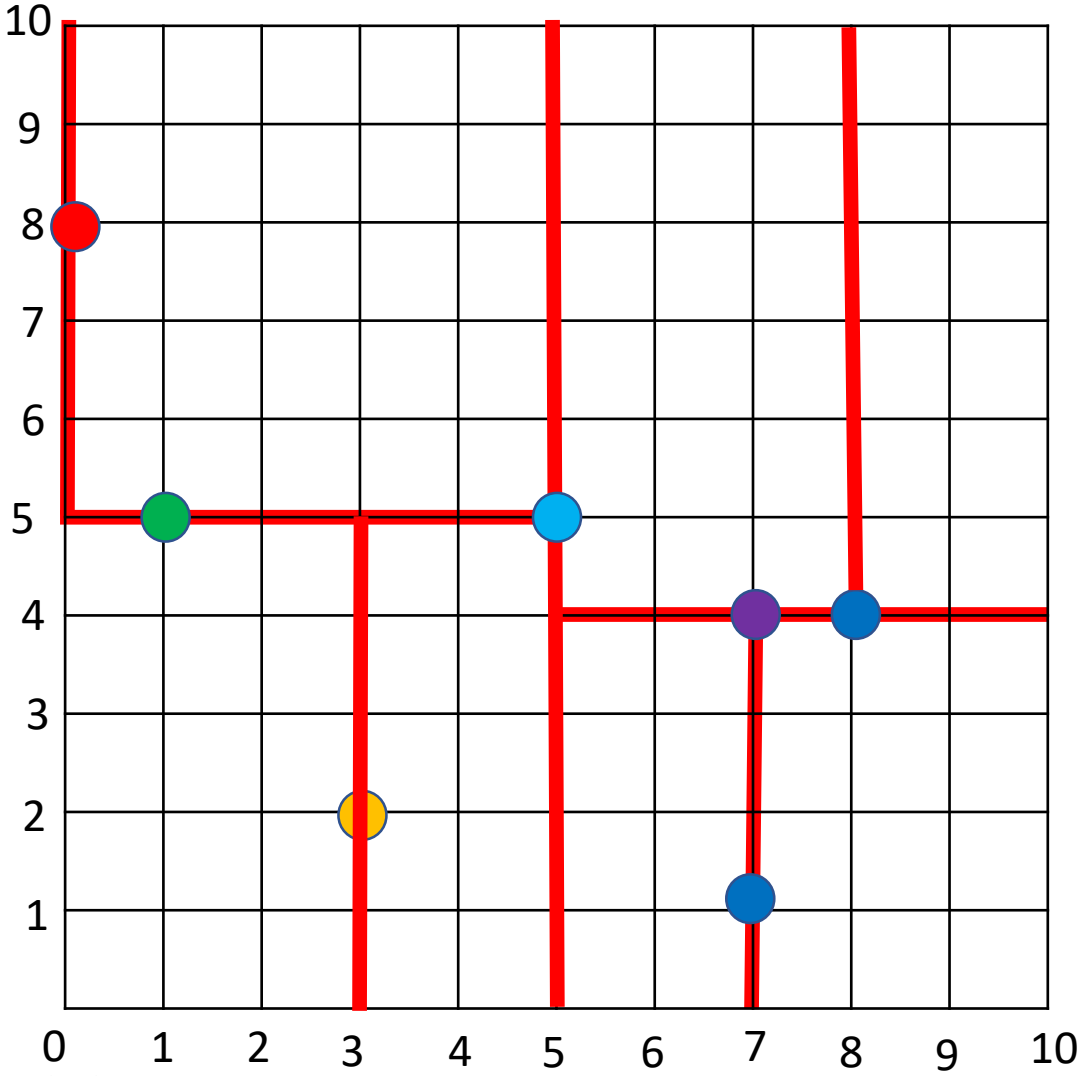


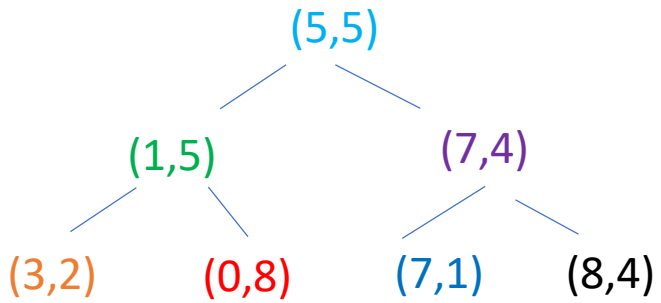


Get median → same, as only one left → rdy

The kd-Tree – Build the data structure

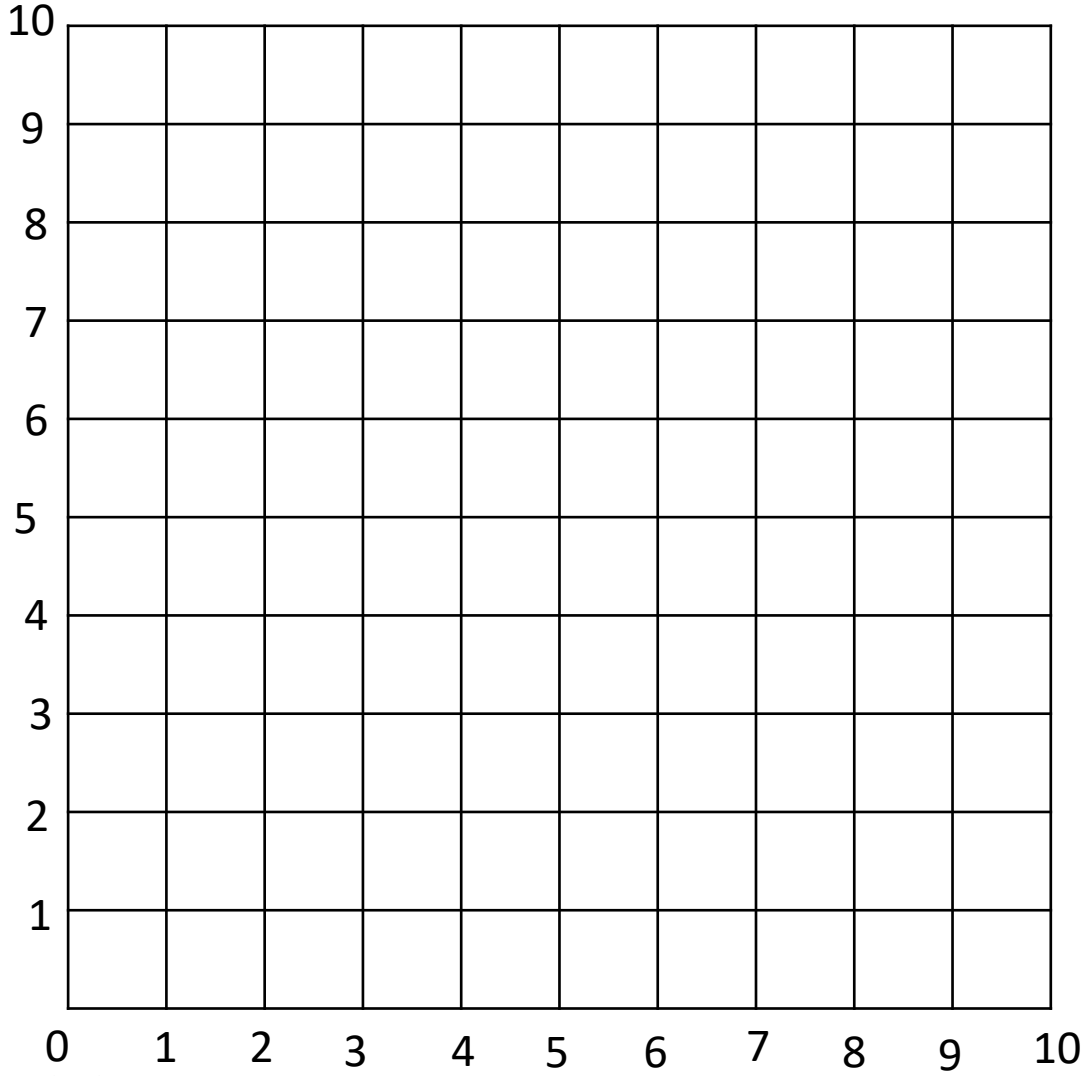
Input:
(3,2)
(1,5)
(0,8)
(5,5)
(7,1)
(7,4)
(8,4)





Add them with splits in x

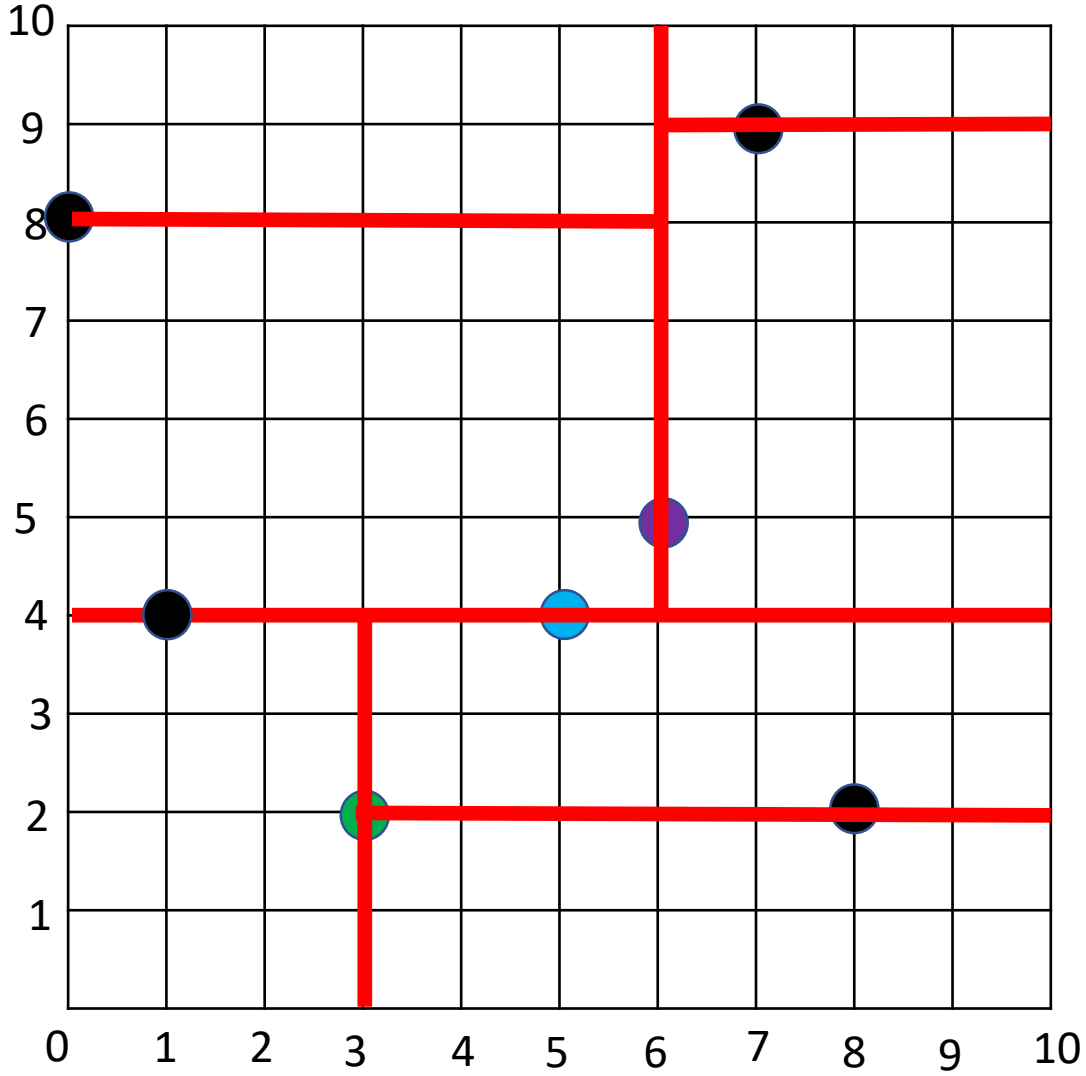
The kd-Tree – Exercise

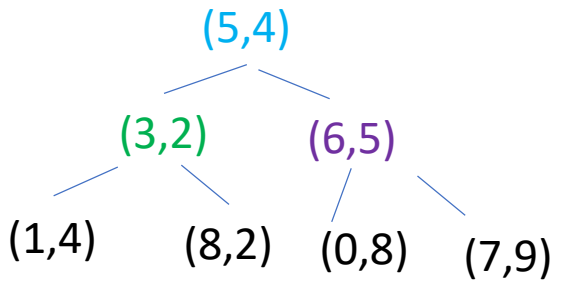


Write down the kd-Tree for this input as tree and in the grid.
Use **alternating splitting**, starting with **y.**

Input:
(1,4)
(3,2)
(8,2)
(0,8)
(5,4)
(6,5)
(7,9)

The kd-Tree – Solution



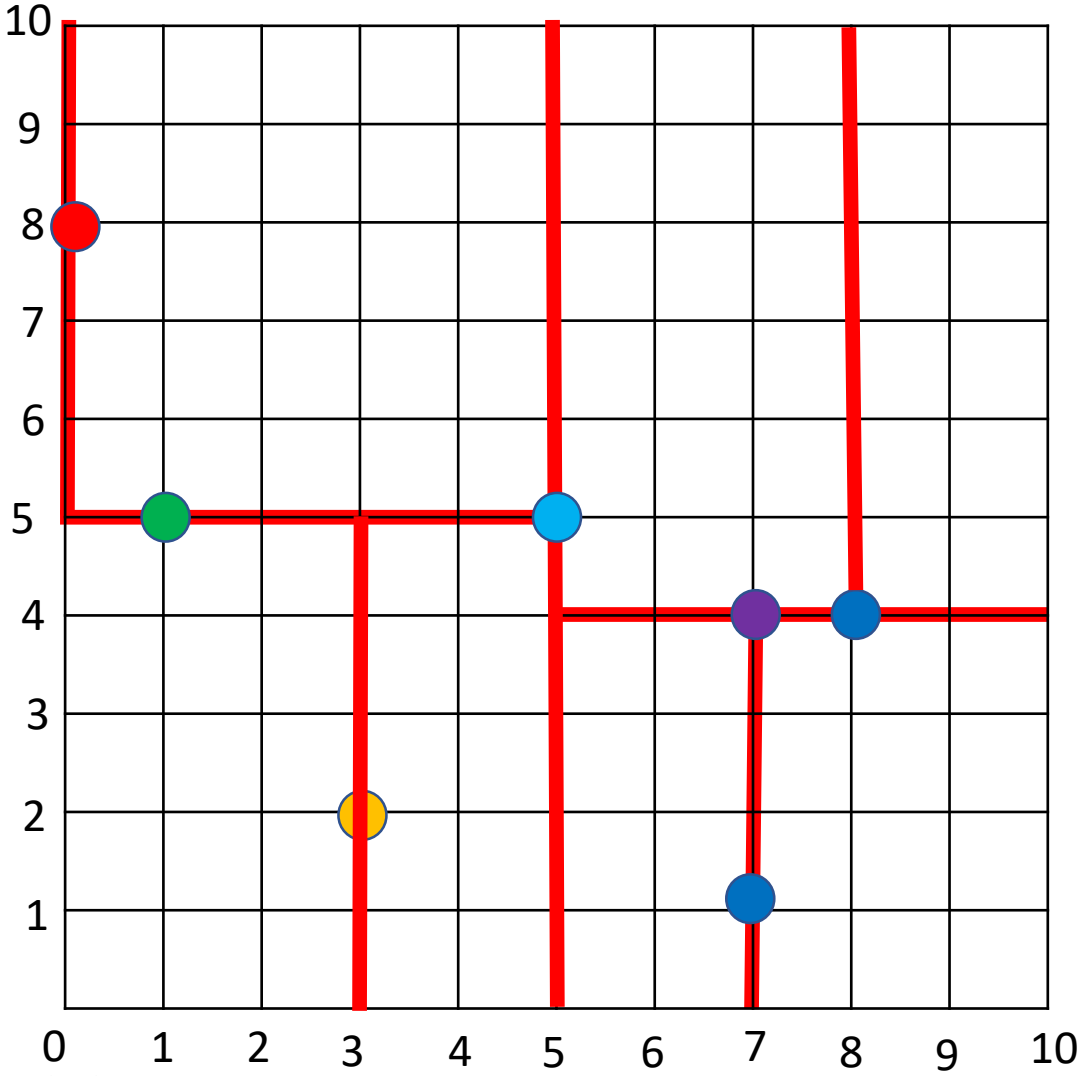


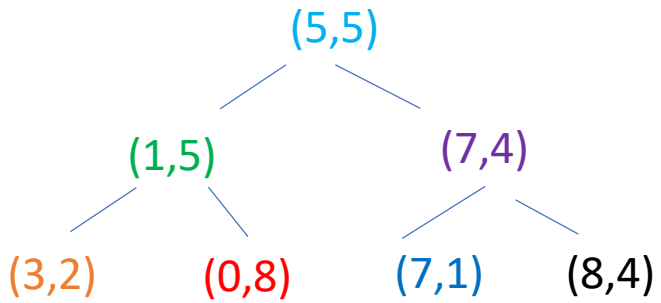
Write down the kd-Tree for this input as tree and in the grid.
Use **alternating splitting**, starting with **y.**

Input:

(1,4)	(3,2)	(1,4)	(1,4)
(3,2)	(8,2)	(3,2)	(3,2)
(8,2)	(1,4)	(8,2)	(8,2)
(0,8)	(5,4)	(5,4)	(5,4)
(5,4)	(6,5)	(0,8)	(0,8)
(6,5)	(0,8)	(6,5)	(6,5)
(7,9)	(7,9)	(7,9)	(7,9)

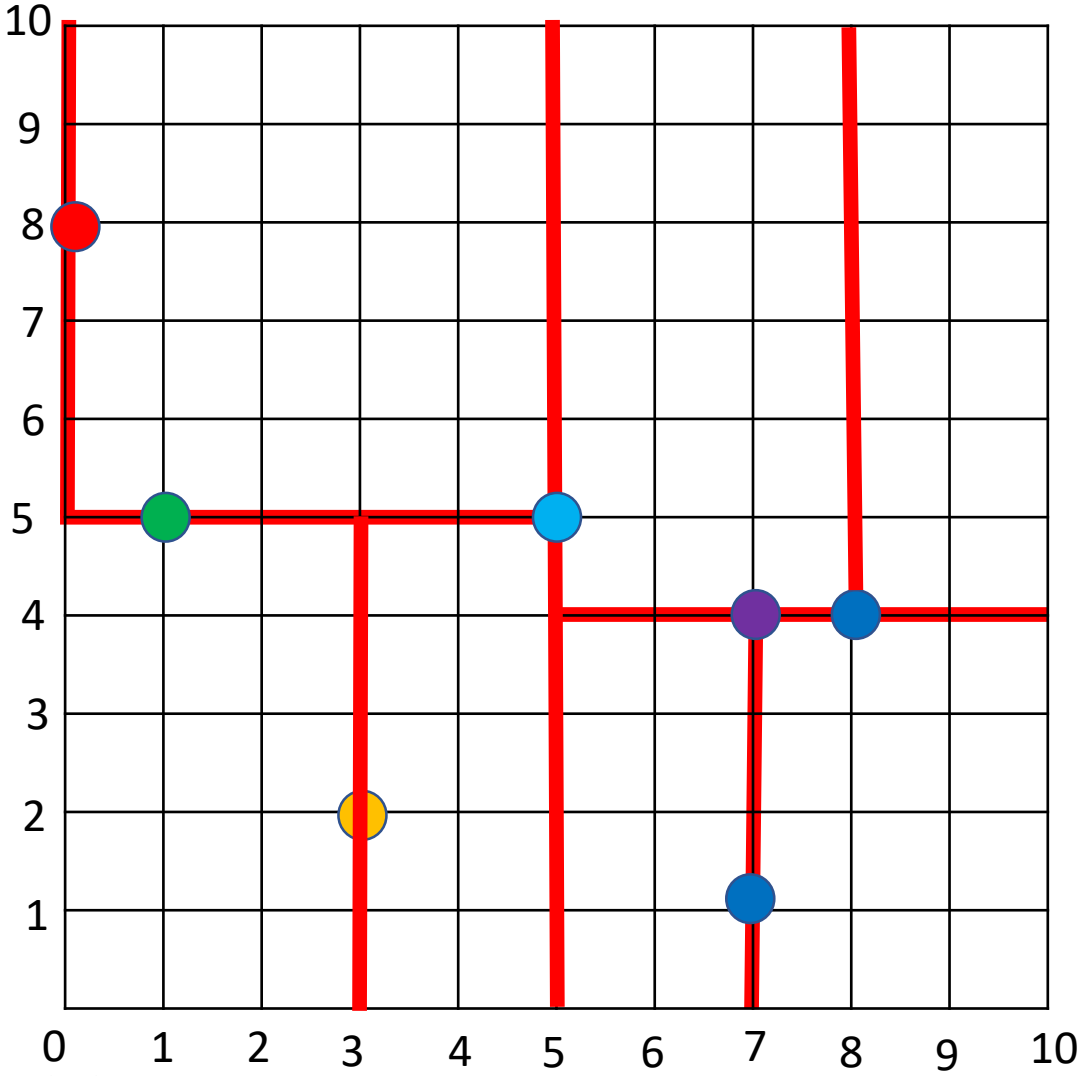
The kd-Tree – How to find nearest Neighbour?

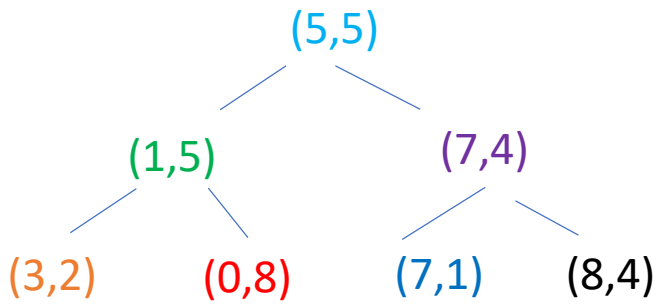




Input: (6/2)

The kd-Tree – How to find nearest Neighbour?





Split in x

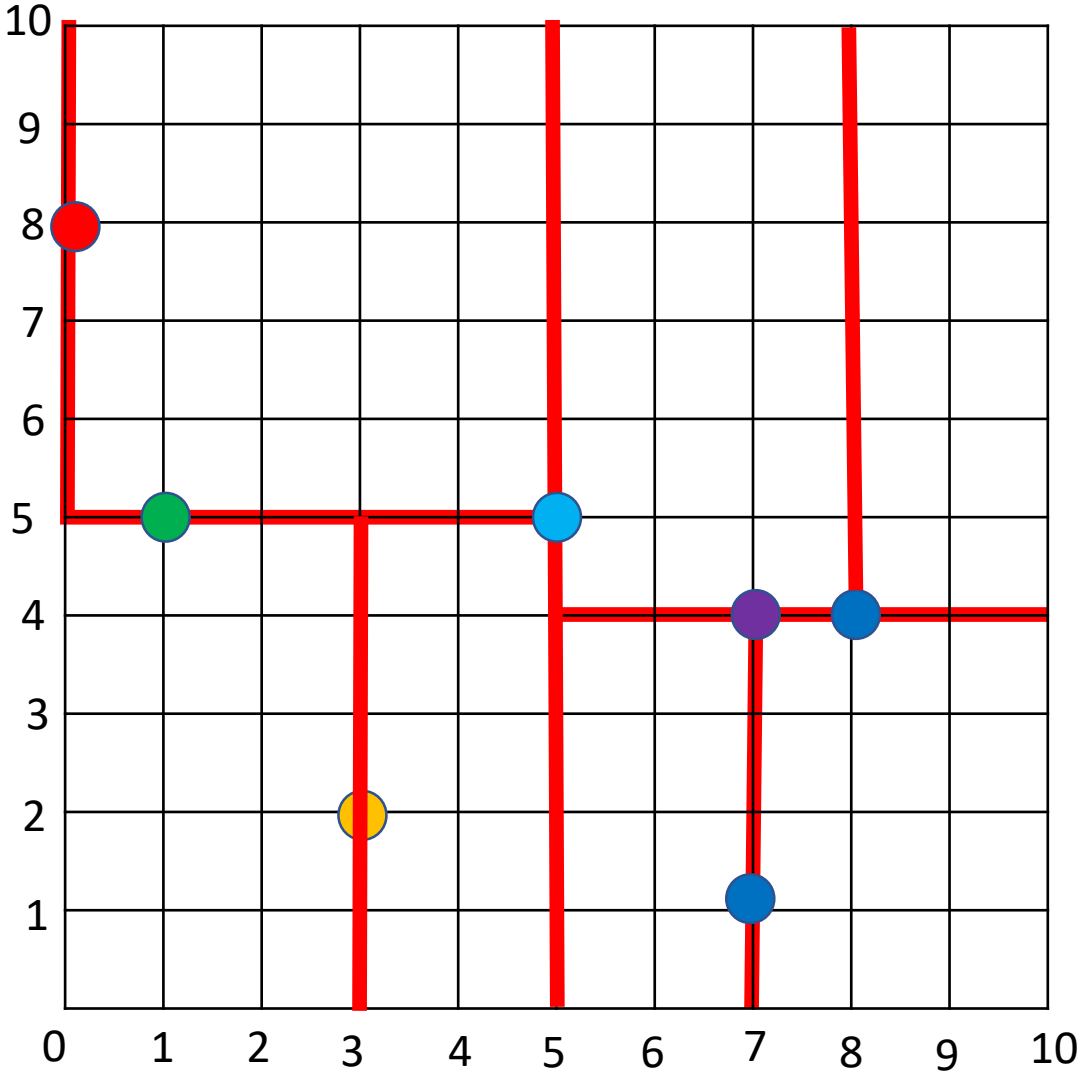
Split in y

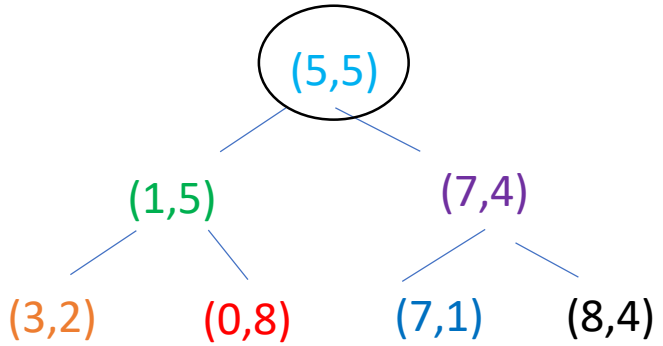
Split in x

Remember!

Input: (6/2)

The kd-Tree – How to find nearest Neighbour?





Split in x

Split in y

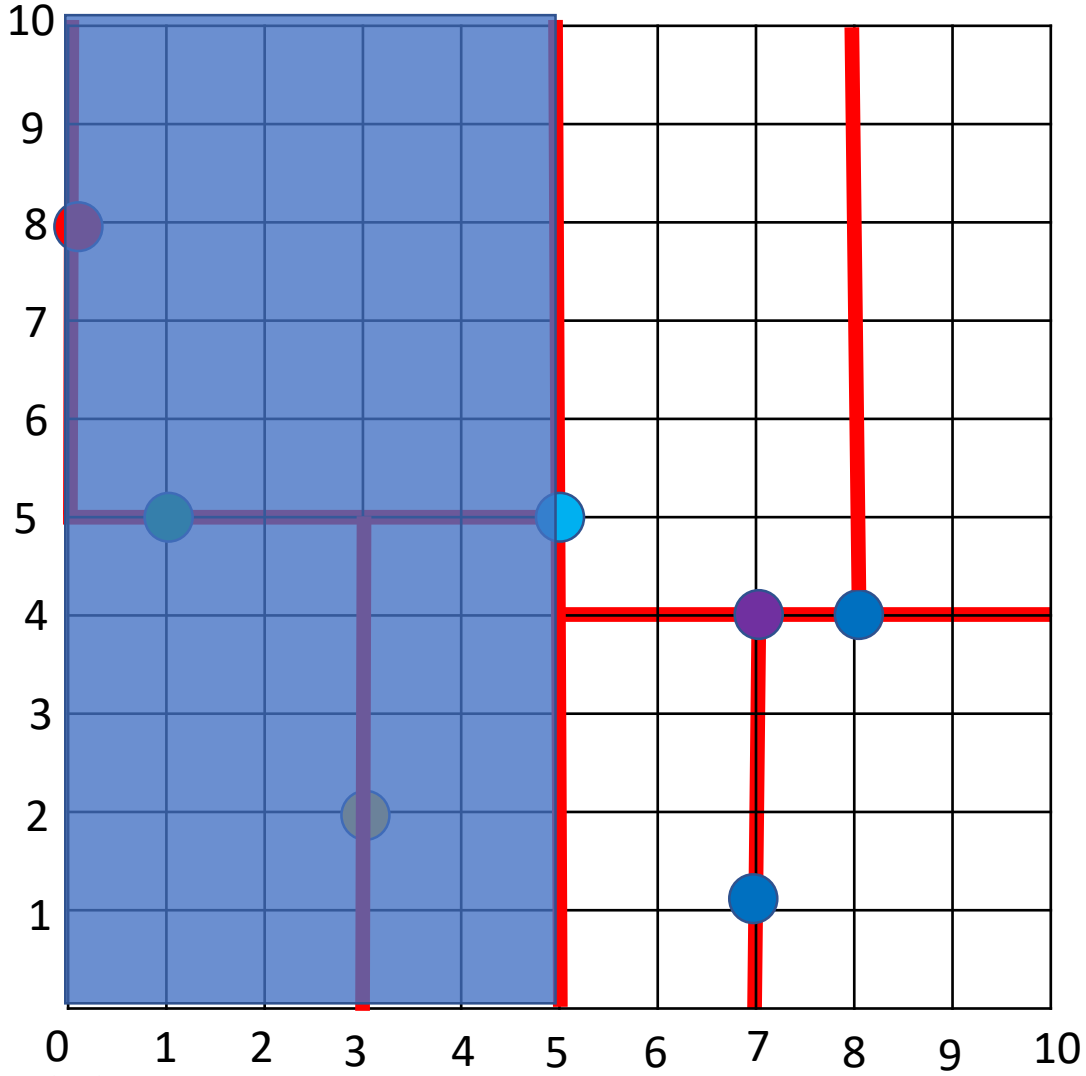
Split in x

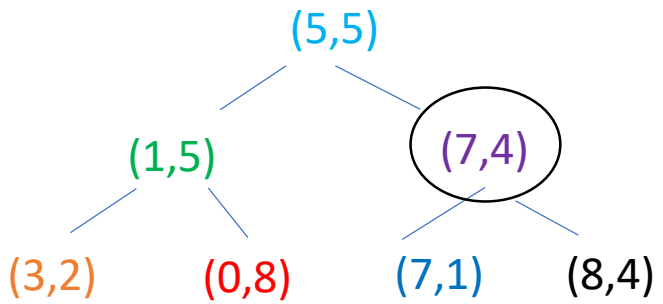
Remember!

6 is bigger than 5 (in x!)

Input: (6/2)

The kd-Tree – How to find nearest Neighbour?





Split in x

Split in y

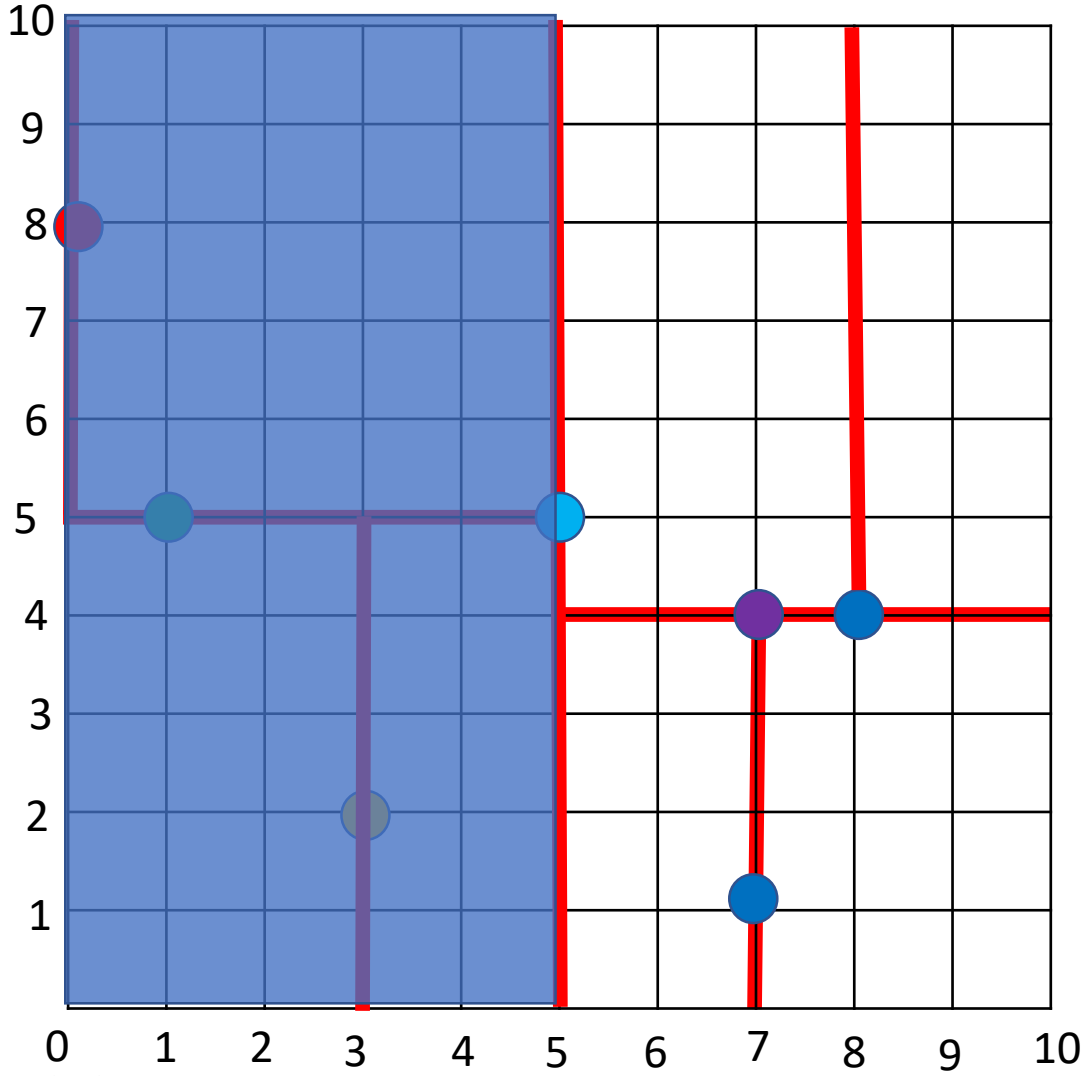
Split in x

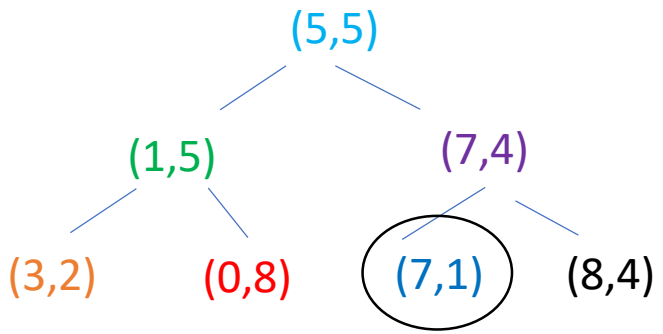
Remember!

Left regions left out (for now!)

Input: (6/2)

The kd-Tree – How to find nearest Neighbour?





Split in x

Split in y

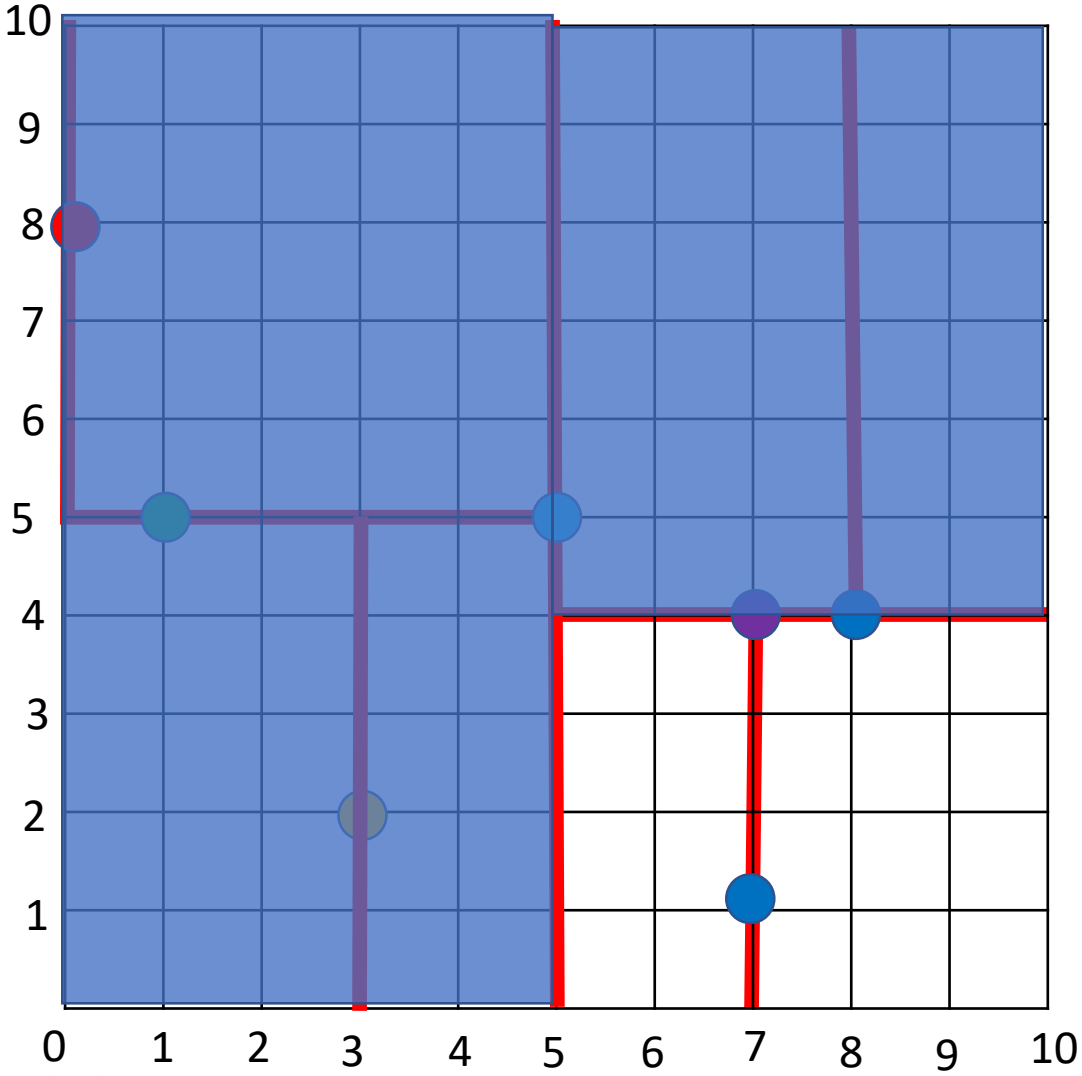
Split in x

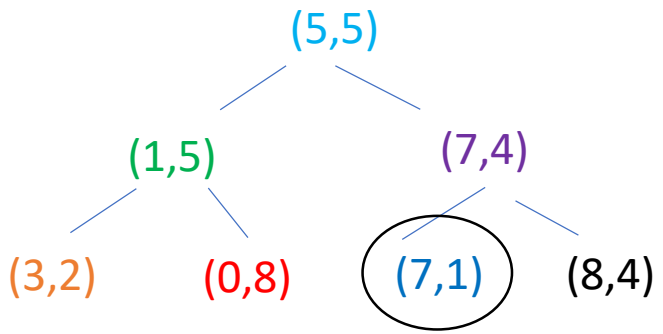
Remember!

Check (7,4) → 2 is lower than 4

Input: (6/2)

The kd-Tree – How to find nearest Neighbour?





Split in x

Split in y

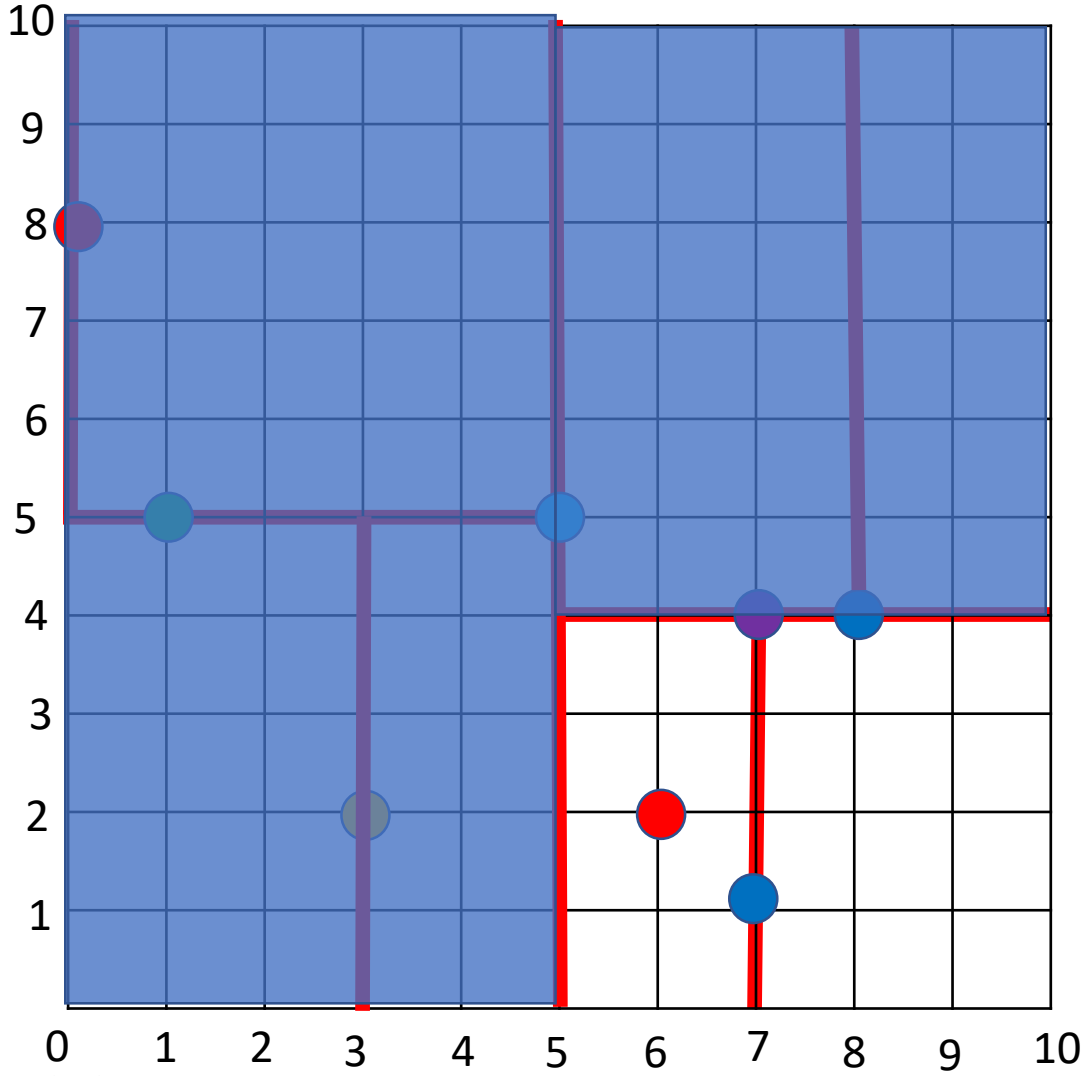
Split in x

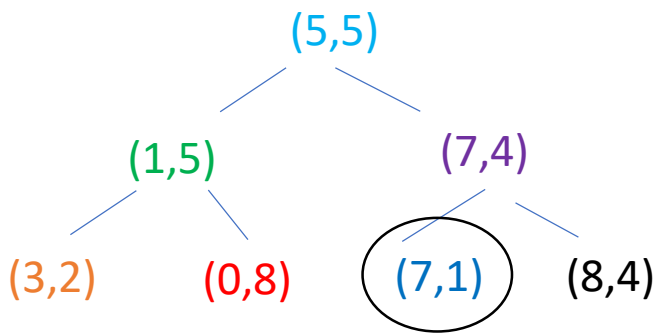
Remember!

Check (7,4) → 2 is lower than 4

Input: (6/2)

The kd-Tree – How to find nearest Neighbour?





Split in x

Split in y

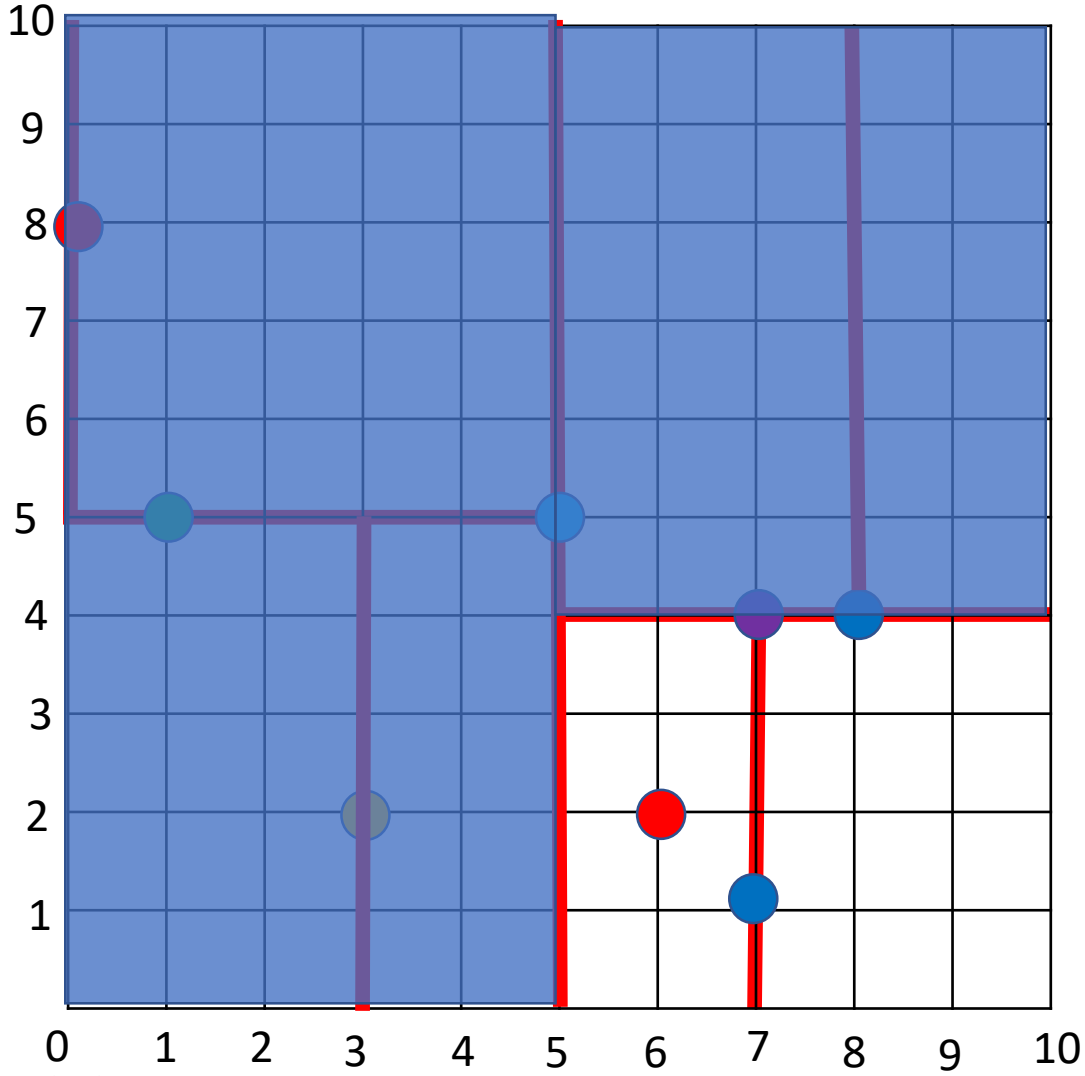
Split in x

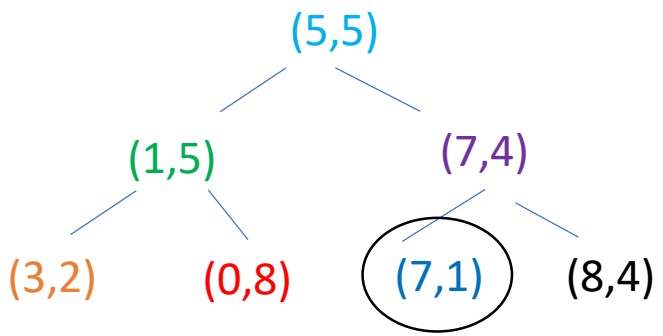
Remember!

Distance is $\sqrt{2}$

Input: (6/2)

The kd-Tree – How to find nearest Neighbour?





Split in x

Split in y

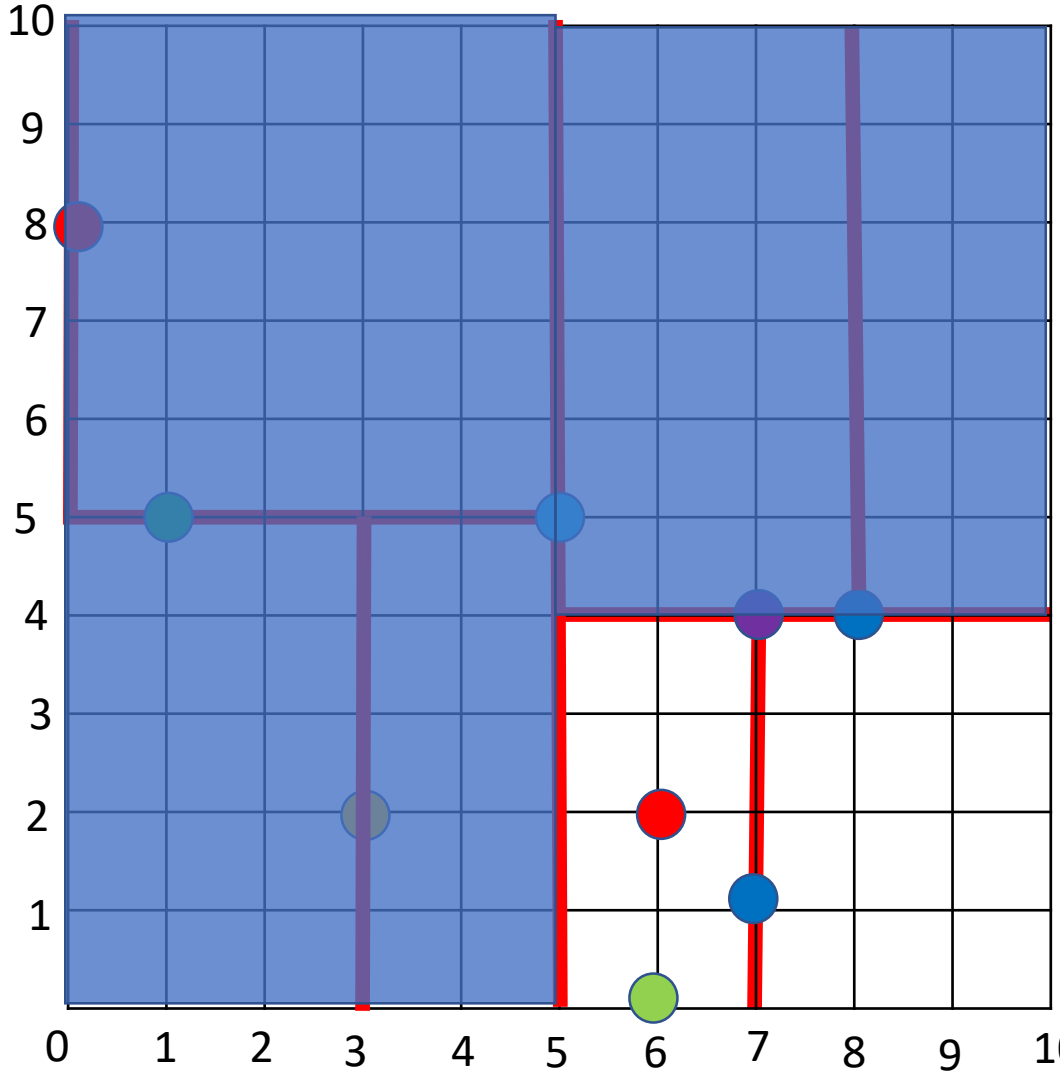
Split in x

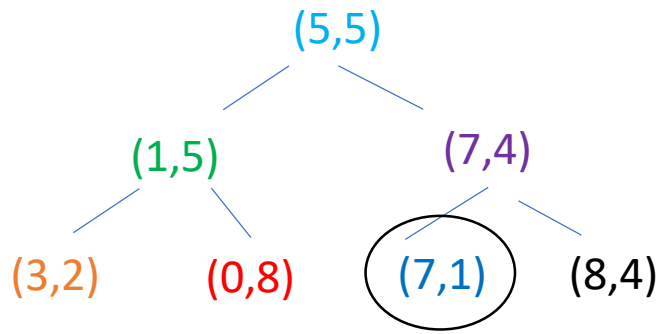
Remember!

Distance is $\sqrt{2}$

Input: (6/2)

The kd-Tree – Some Notes



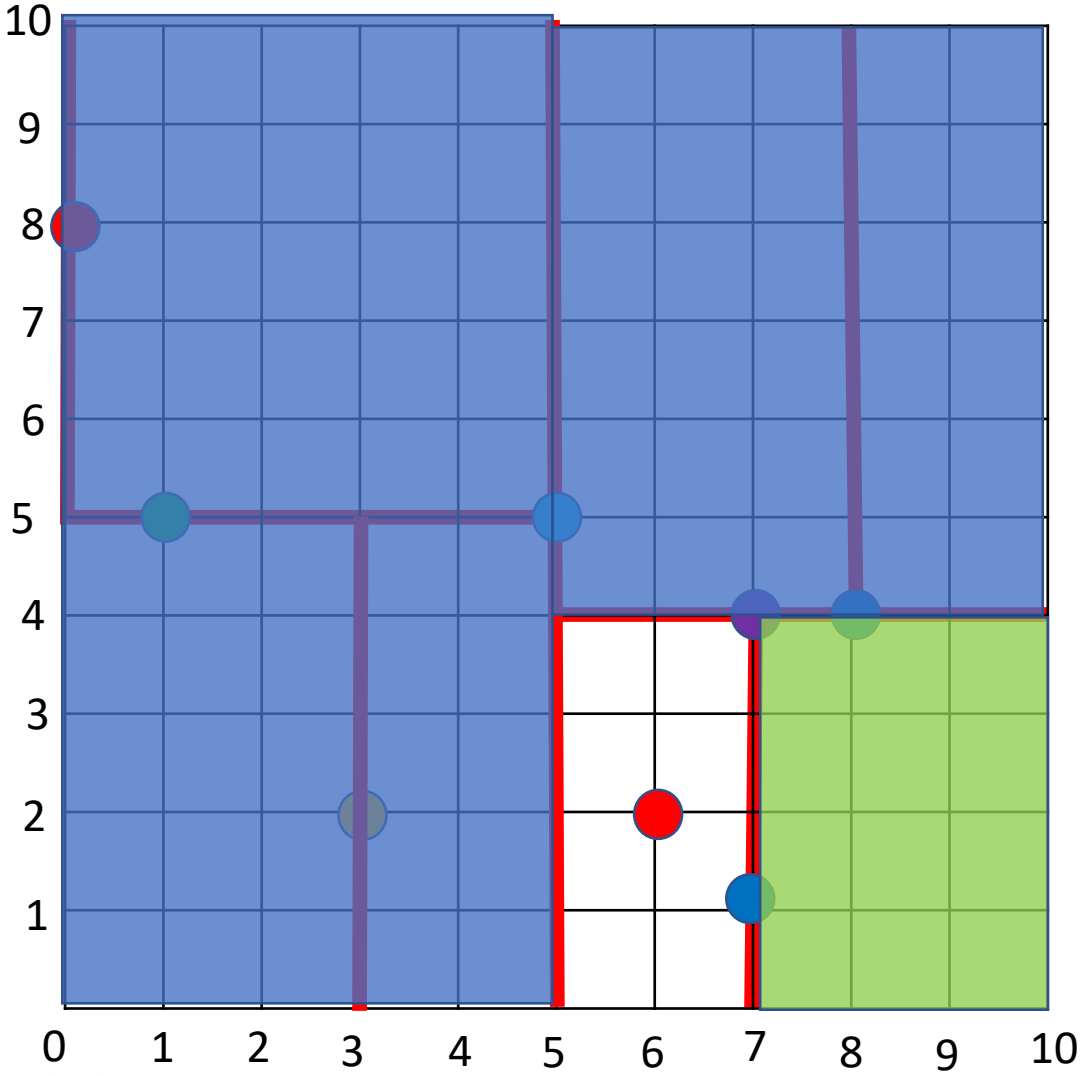


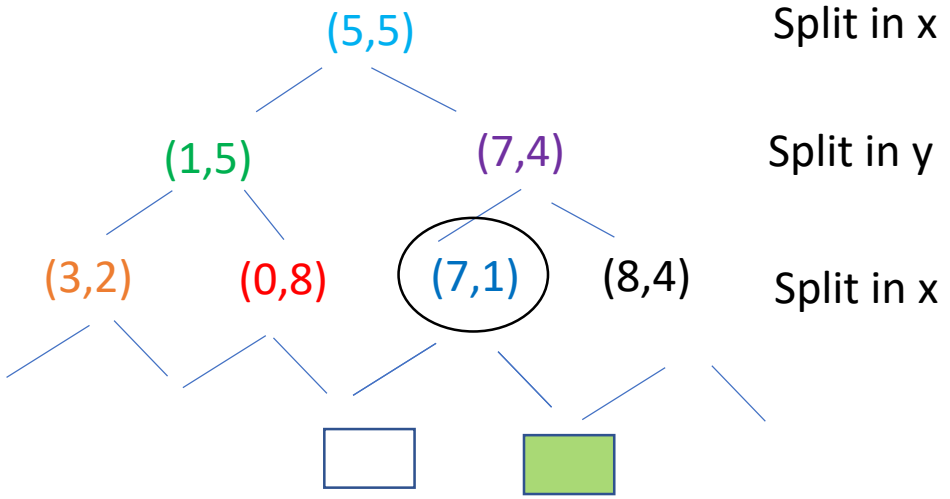
Notes:

- The point (7,1) is just a candidate on the for the nearest neighbour.
- You can also call it an approximation of the nearest neighbour.
- In motion planning we are in most of the time not interested in exact neighbours.
  We need to find a “good one” fast.
- Therefore in MP the kd-Tree is often stopped here.

Input: (6/2)

The kd-Tree – Some Notes



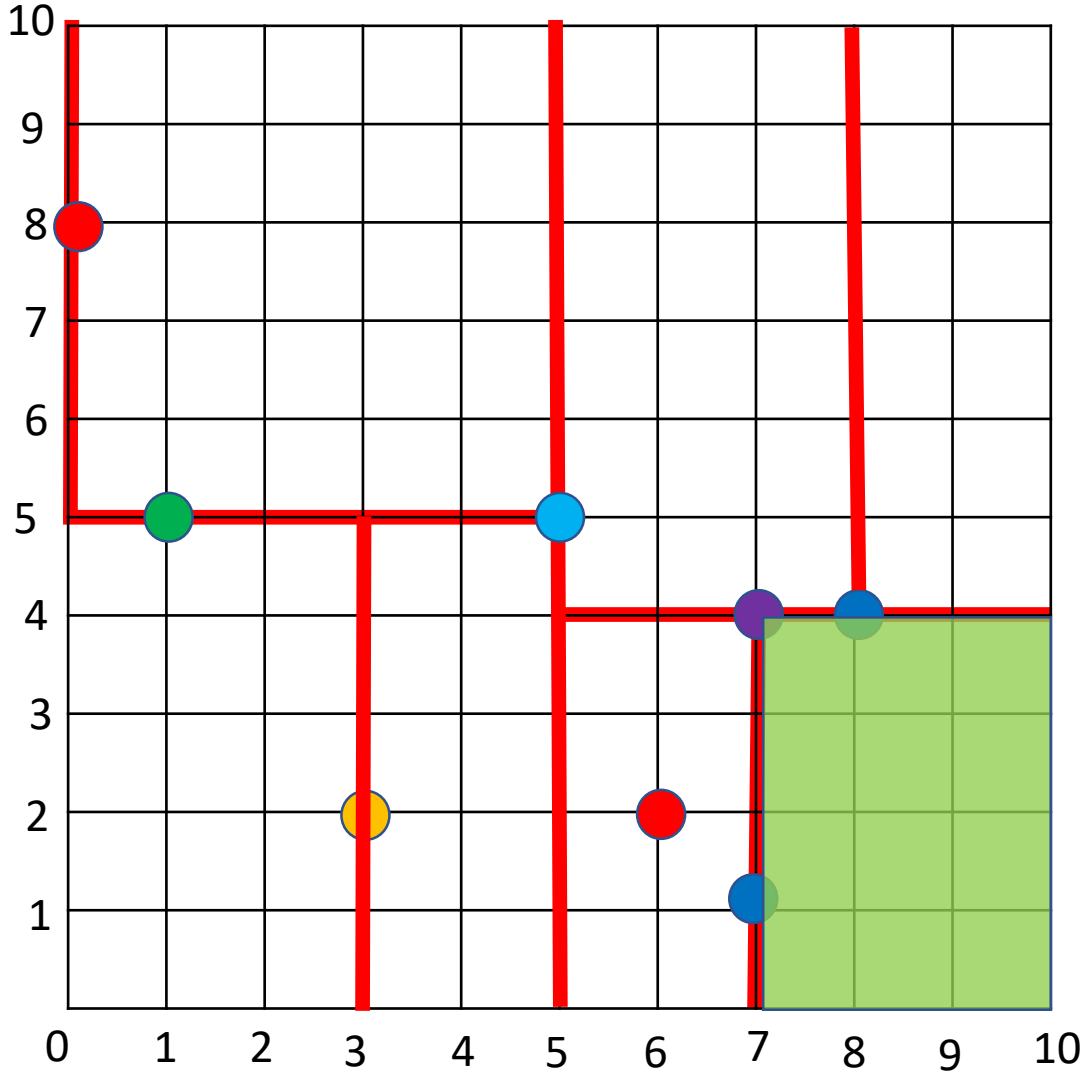


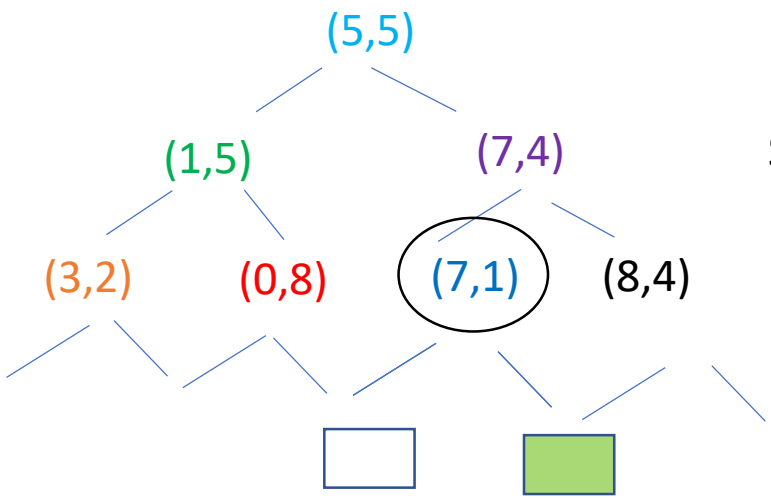
Notes:

- Note also, that below (7,1) there are regions.
- Same for the others

Input: (6/2)

The kd-Tree – Recursion

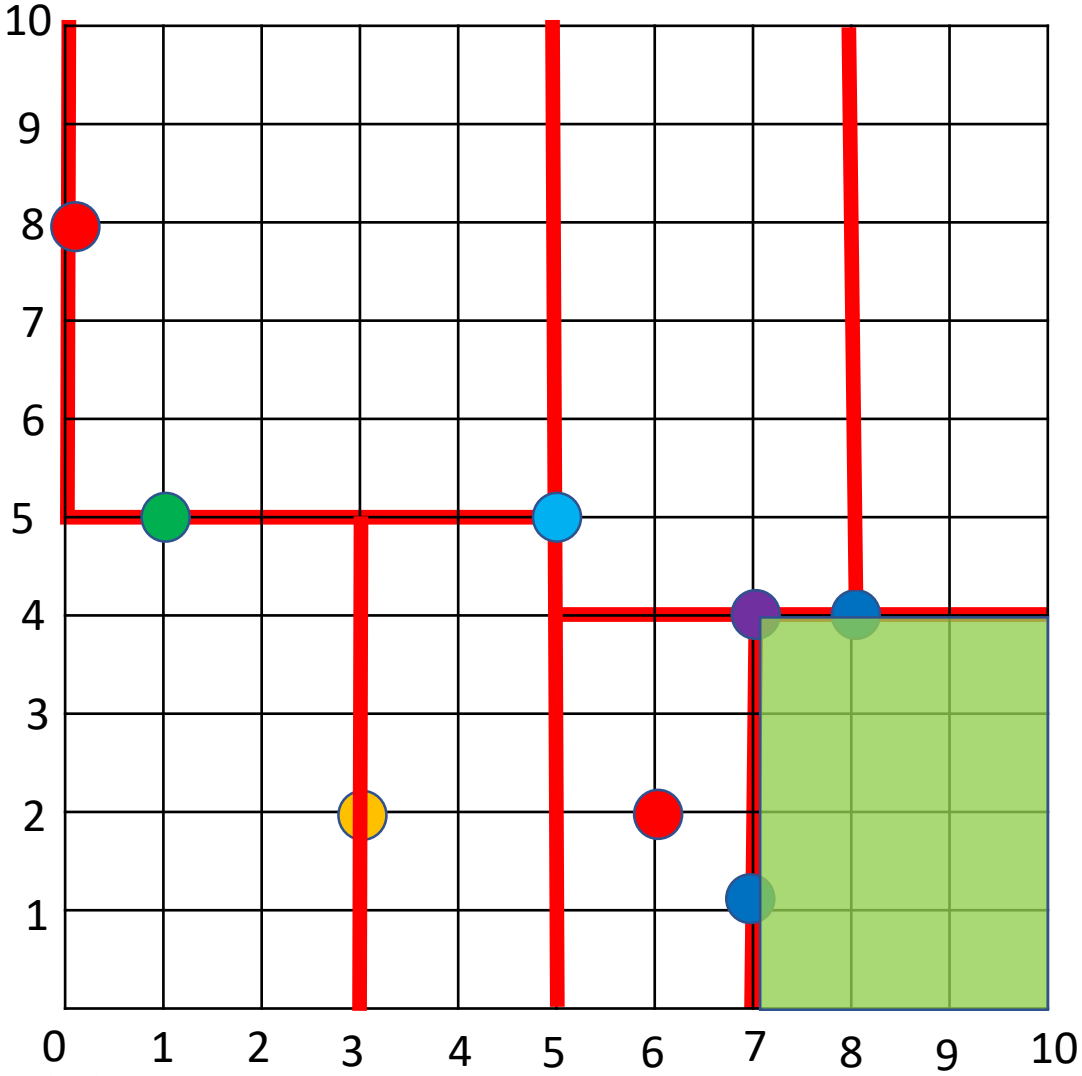


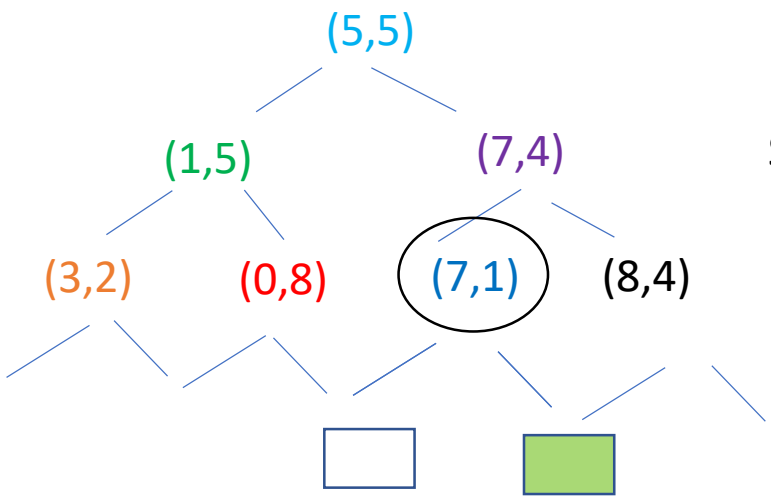


To find the exact NN you have to travel now the tree upwards and check, with the computed distance if you can exclude the other parts of the tree for sure

Input: (6/2)

The kd-Tree – Recursion





Can in this region be a point with less thjan $\sqrt{2}$?

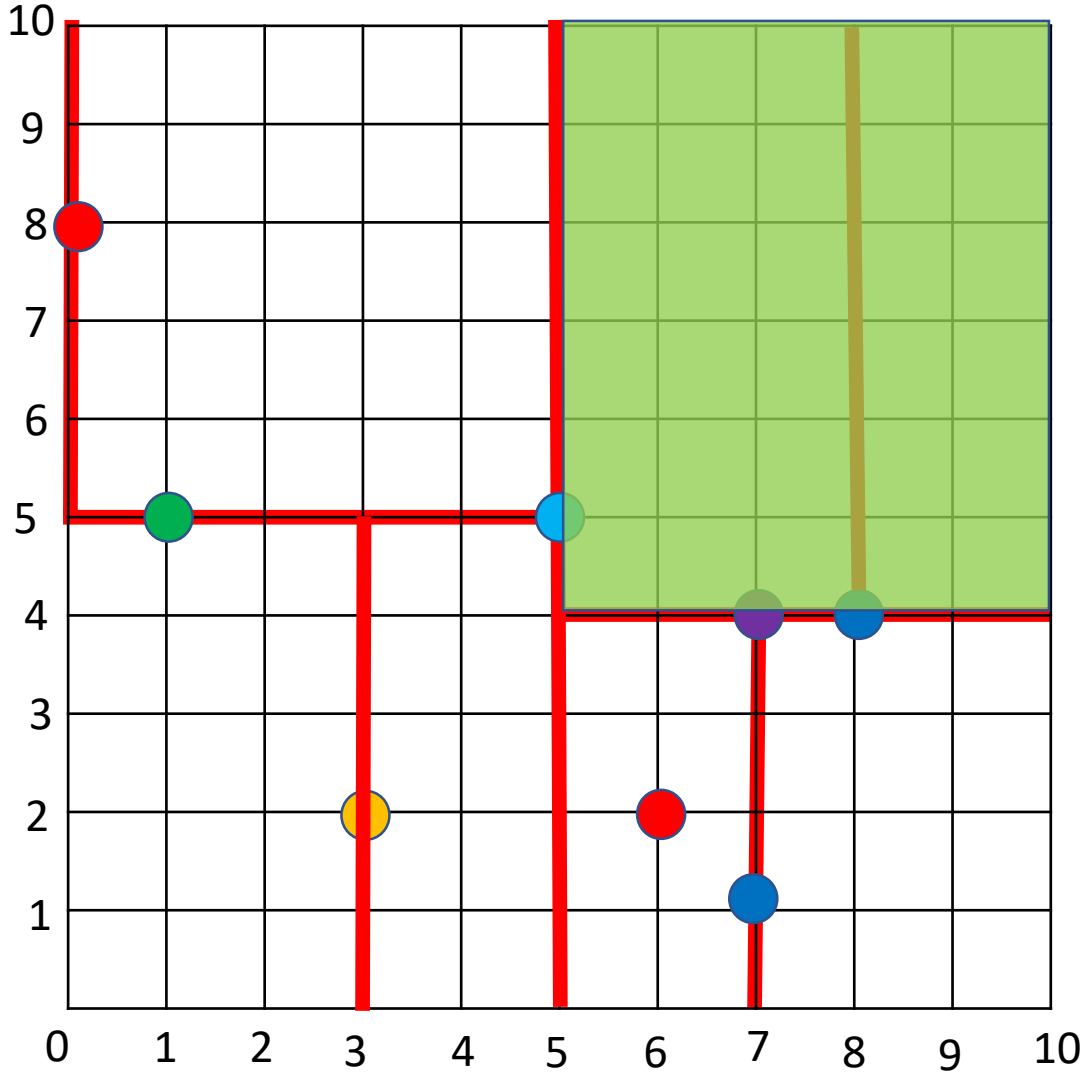
→ Yes

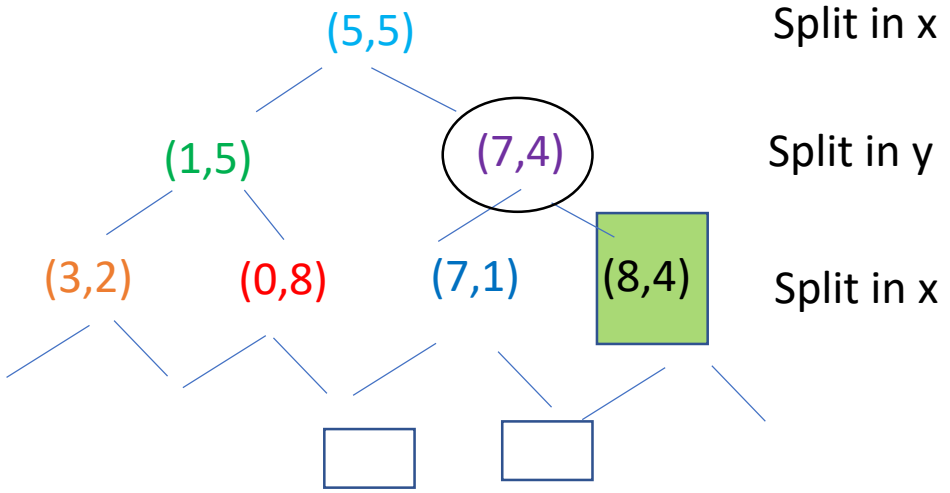
→ Check this region.

→ Not point in there → got ahead

Input: (6/2)

The kd-Tree – Recursion





Can in this region be a point with less than $\sqrt{2}$?

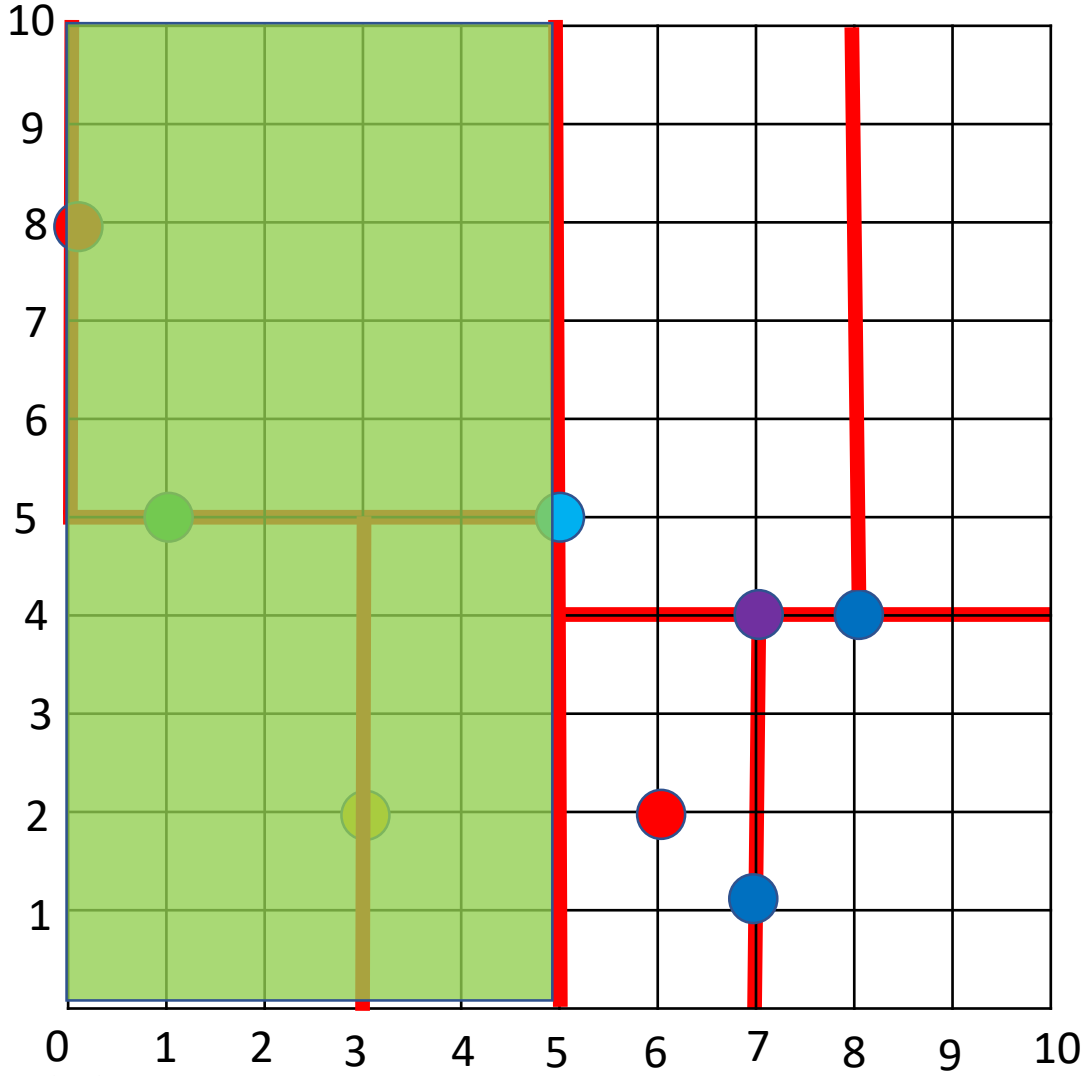
→ No

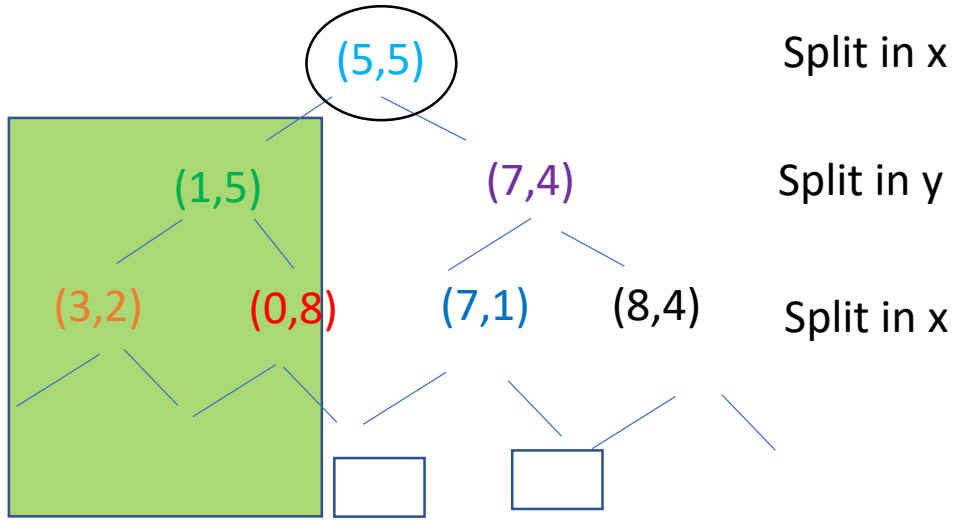
→ Why? (7,4) was a split in y and distance in y is 2 (this is> $\sqrt{2}$)

→ The region under node (8,4) can be discarded for sure.

Input: (6/2)

The kd-Tree – Recursion





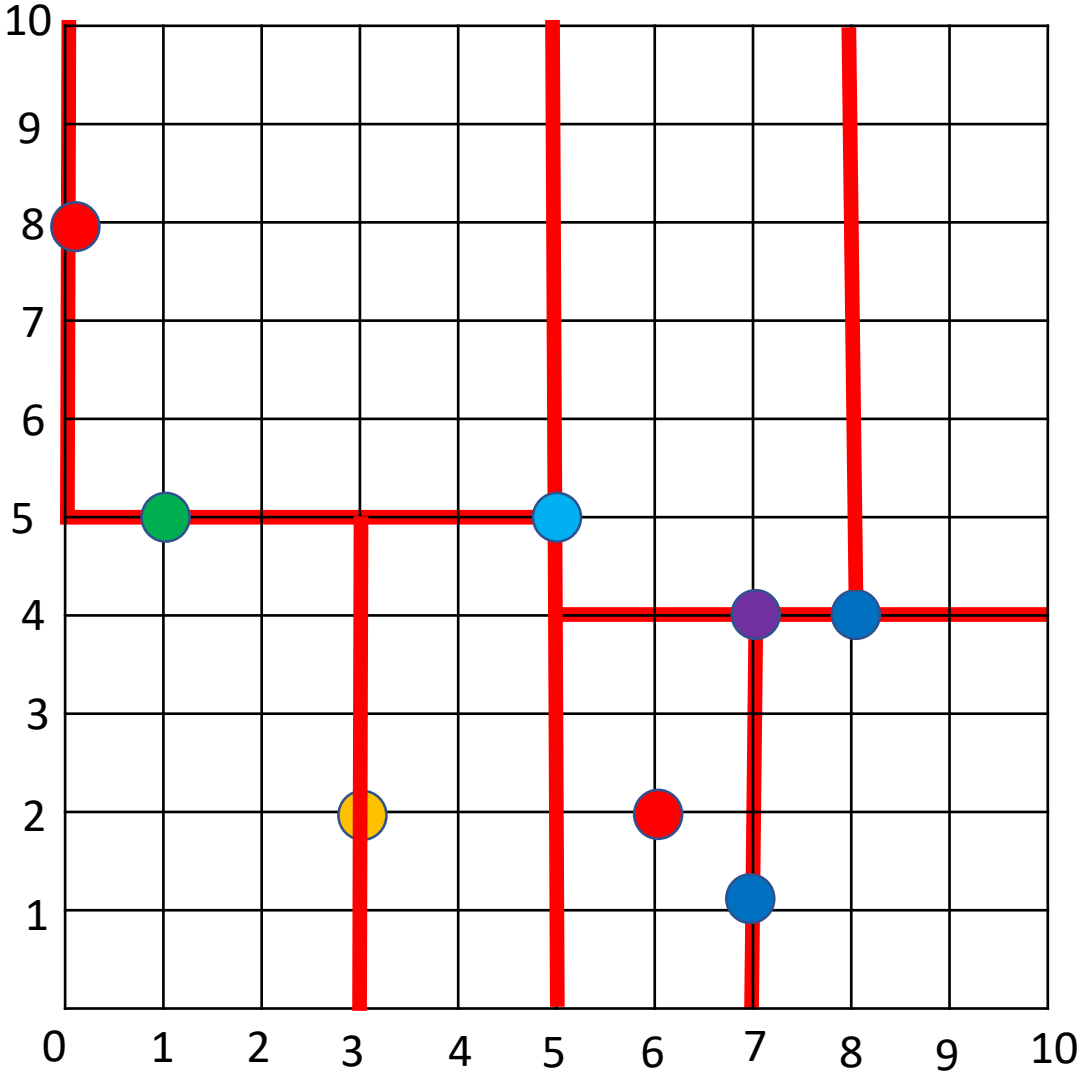
Can in this region be a point with less than $\sqrt{2}$?

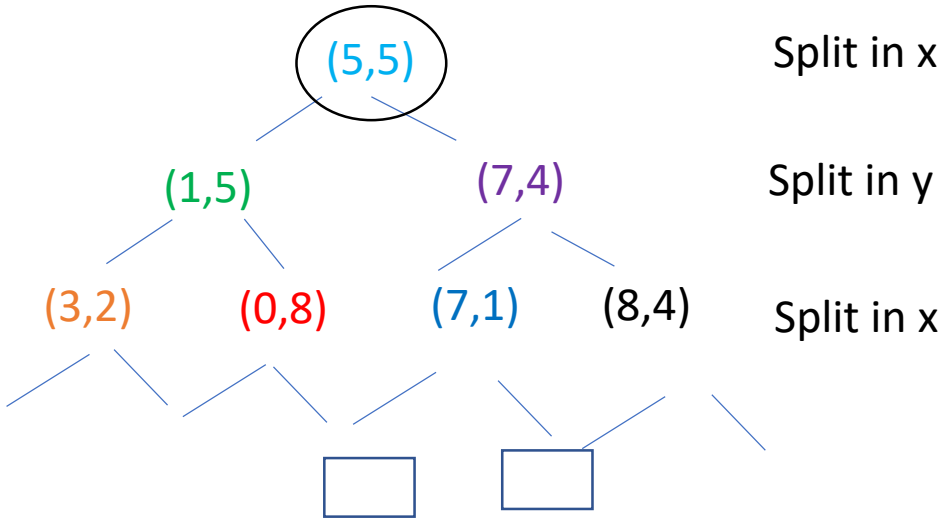
→ Yes

→ Why? (5,5) was a split in x and distance in x is 1 (this is< $\sqrt{2}$)

Input: (6/2)

The kd-Tree – Recursion





- This goes on until all regions are excluded.
- If a new closest points are found, the distance is updated and the recursion continues.

Input: (6/2)

An algorithm often used in practice: LSH

Local sensitive Hashing:

Idea:

1. The LSH approach is (similar to the motion planners) randomized.
2. The algorithm can the used to find the nearest of k nearest neighbour to a given point.
3. But it does not compute the exact neighbour. It does only:
   With a very high chance the LSH approach returns a point
   that is the nearest neighbour or a point that is a very close approximation.

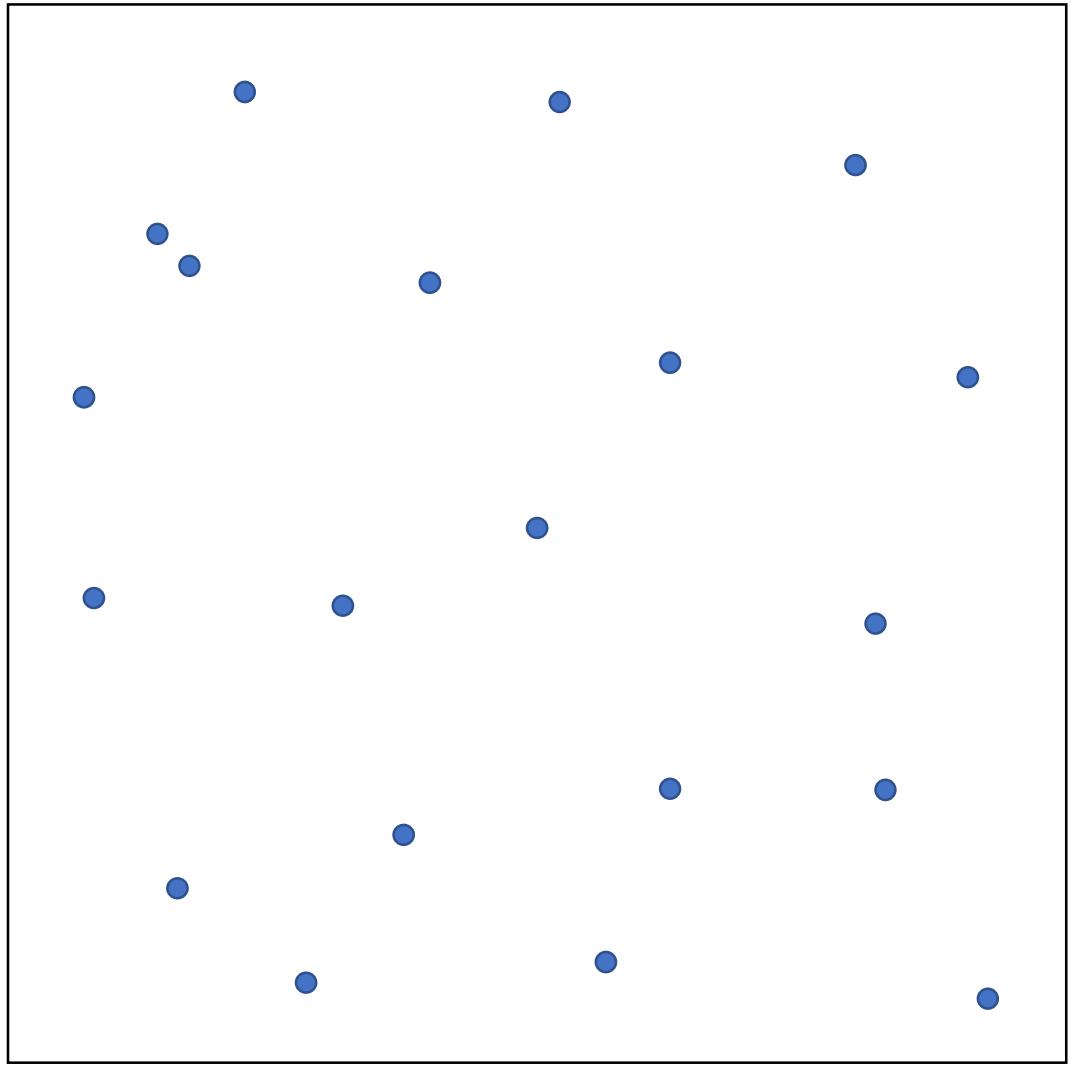
An algorithm often used in practice: LSH

Local sensitive Hashing:

Why use LSH?

1. By allowing a minor mismatch of the nearest neighbour definition, the algorithm can compute neighbours much faster than a kd-Tree or a brute Force algorithm.
2. In motion planning the algorithms are based of a random data structure that covers the configuration space. If we only consider approximation of the nearest neighbours
   the motion planner still cover the configuration space.
3. The LSH approach can be easily parallelized and is therefore used in nowadays multi core scenarios.

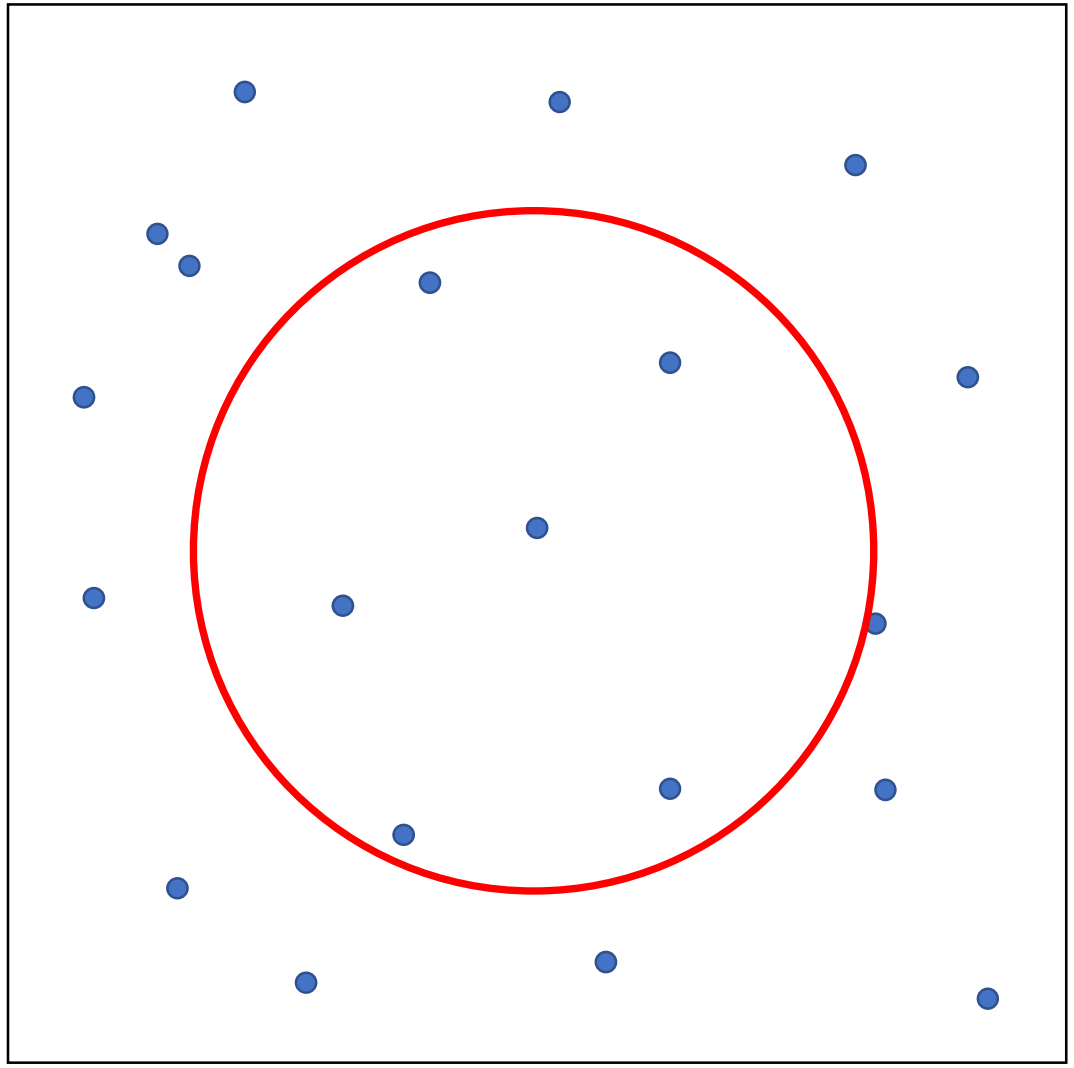
A simplified version of LSH



Notes:

- Given is space, for example a bounded 2d space with configuration.

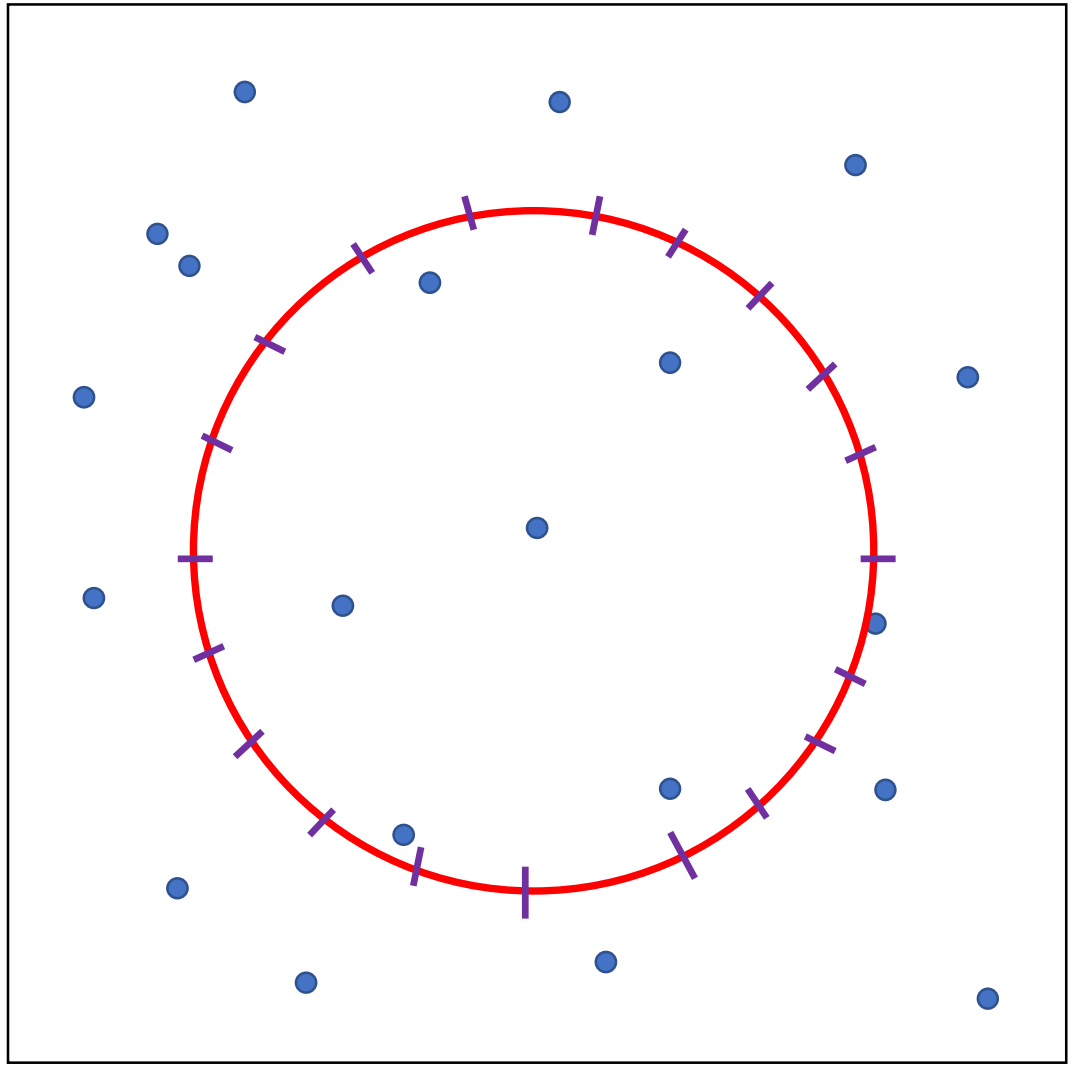
A simplified version of LSH



Notes:

- Given is space, for example a bounded 2d space with configuration.
- Define subspace that can be mapped to a hash set.
- In this example we will use a circle.

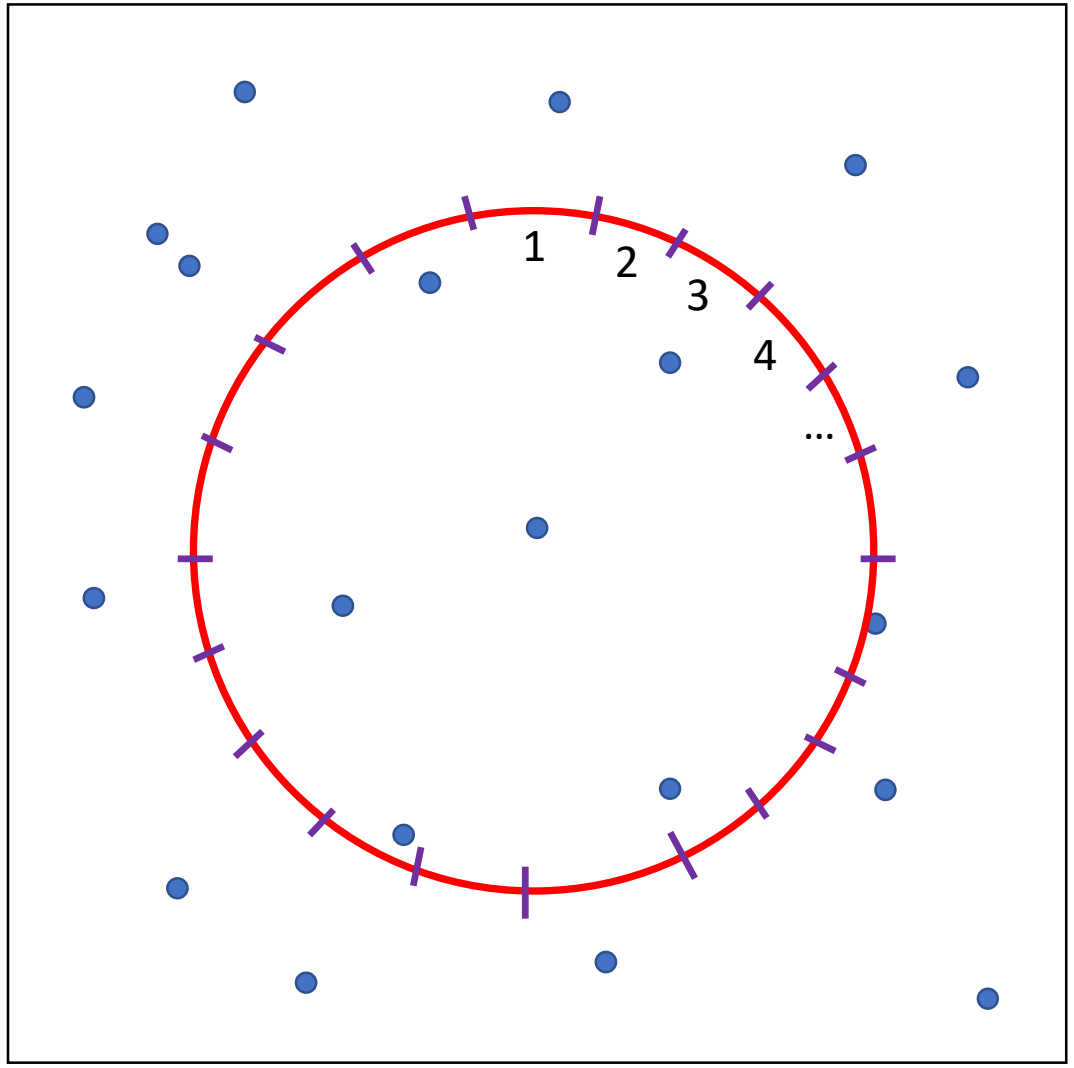
A simplified version of LSH



Notes:

- Given is space, for example a bounded 2d space with configuration.
- Define subspace that can be mapped to a hash set.
- In this example we will use a circle.
- Subdivide this circle into a array.

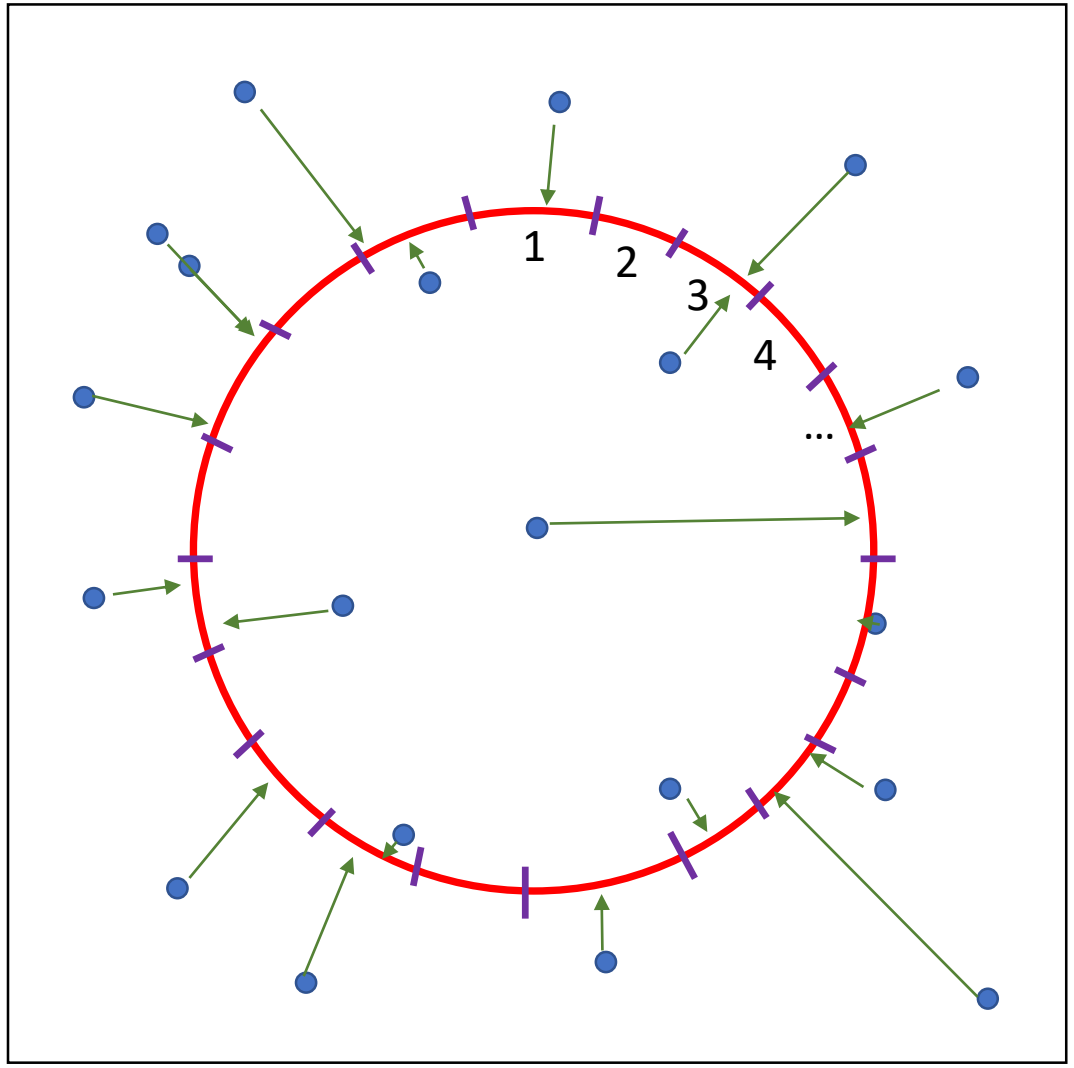
A simplified version of LSH



Notes:

- Given is space, for example a bounded 2d space with configuration.
- Define subspace that can be mapped to a hash set.
- In this example we will use a circle.
- Subdivide this circle into a array.
- Number the array.

A simplified version of LSH

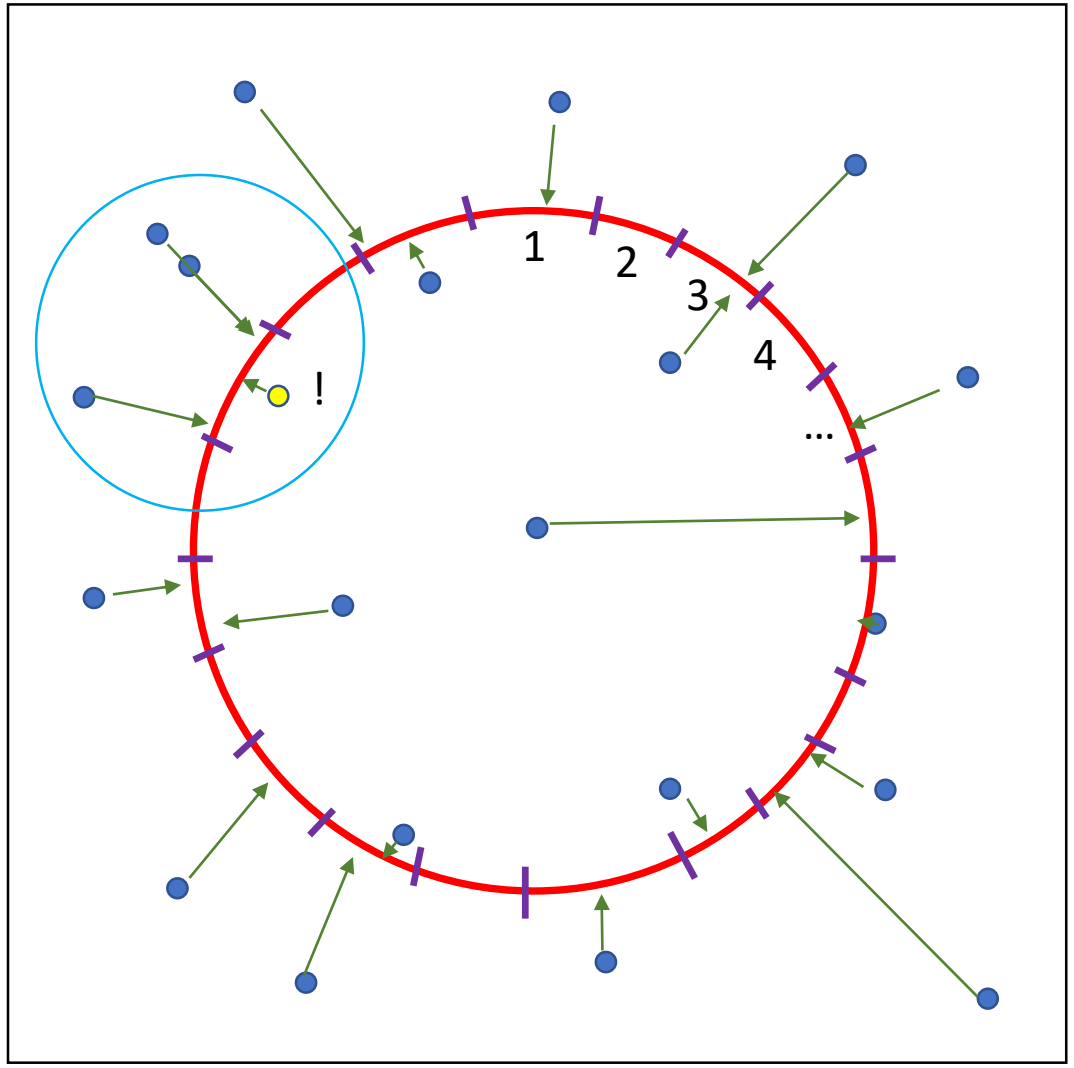


Notes:

- Given is space, for example a bounded 2d space with configuration.
- Define subspace that can be mapped to a hash set.
- In this example we will use a circle.
- Subdivide this circle into a array.
- Number the array.
- Map each point to a array cell.

Idea of LSH: Points that are close to each other are mapped to the same array.

A simplified version of LSH



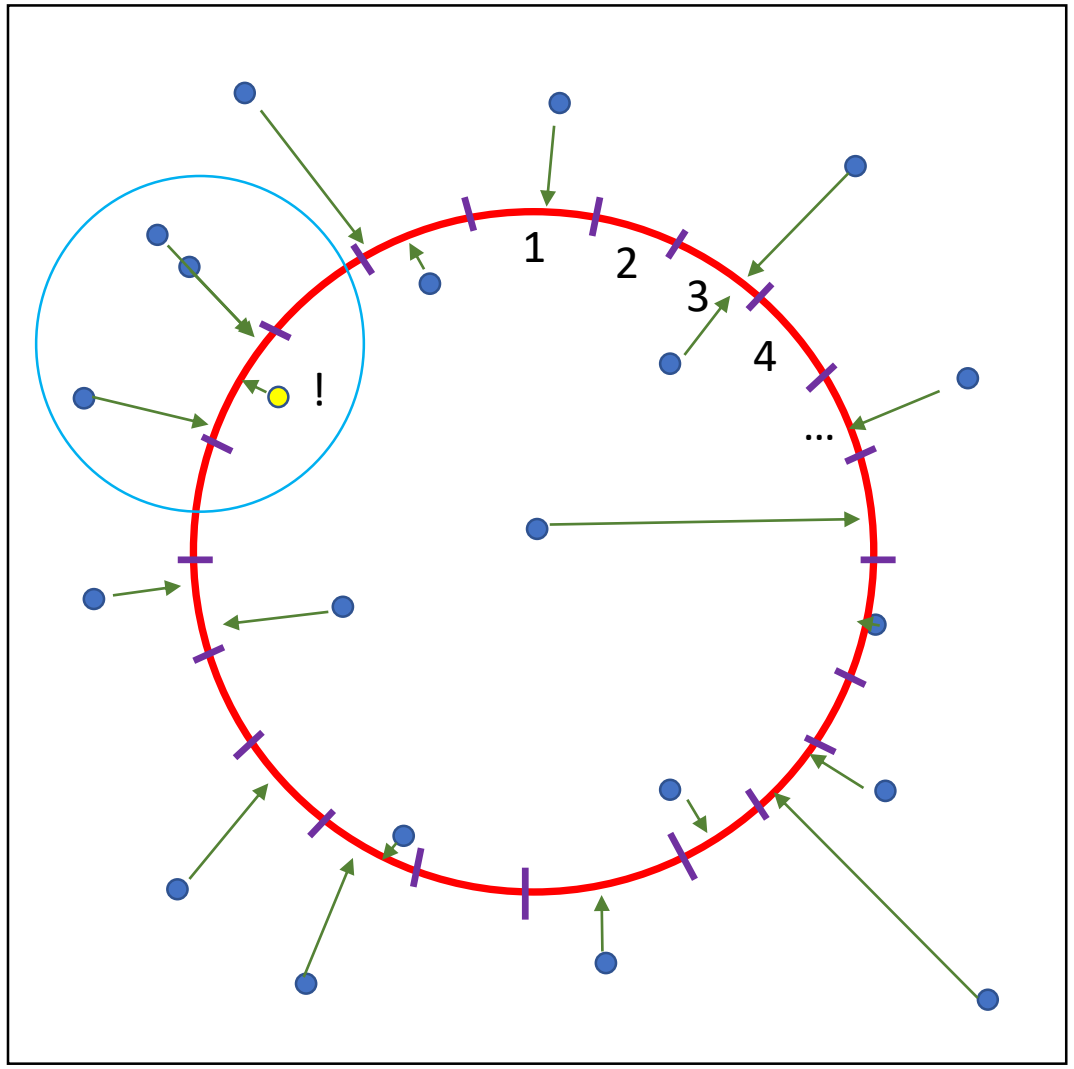
Notes:

- Given is space, for example a bounded 2d space with configuration.
- Define subspace that can be mapped to a hash set.
- In this example we will use a circle.
- Subdivide this circle into a array.
- Number the array.
- Map each point to a array cell.

Idea of LSH: Points that are close to each other are mapped to the same array.

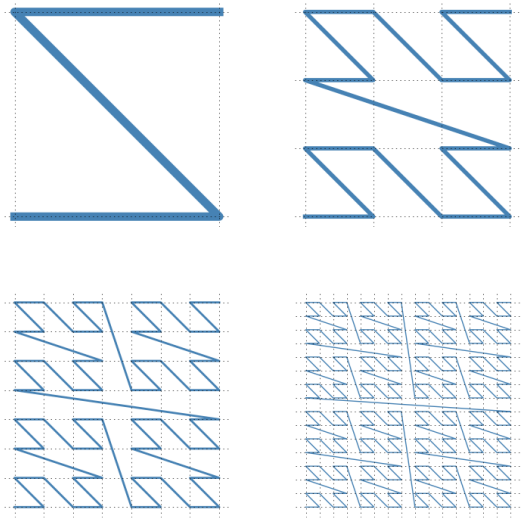
- On query, compute the array of the queried point and compute the distance to all points that are referenced to the same array. → Return the closest one.

Exercise

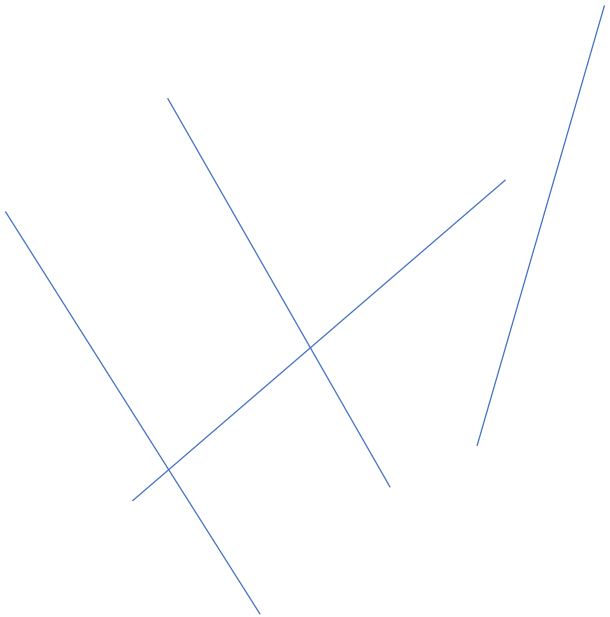


1. Is the circle a good idea? If no, which areas will cause problems.
2. Name alternatives to the circle.
3. Can you think of a method how to extend this idea?

Some examples



Using the z-curve



Use multiple hyperplanes→ most commonly used

Some notes on LSH

- The LSH approach is often used when the dimension of the search space is very high.
- As you can query each the hash function independently and the major task is to compute the distance in all bucket points → the algorithm can be parallelized easily. This algorithm is even GPU friendly.
- In lower dimensions 2-3 the kd-Tree is often preferred. Although this is way harder to parallelize than the