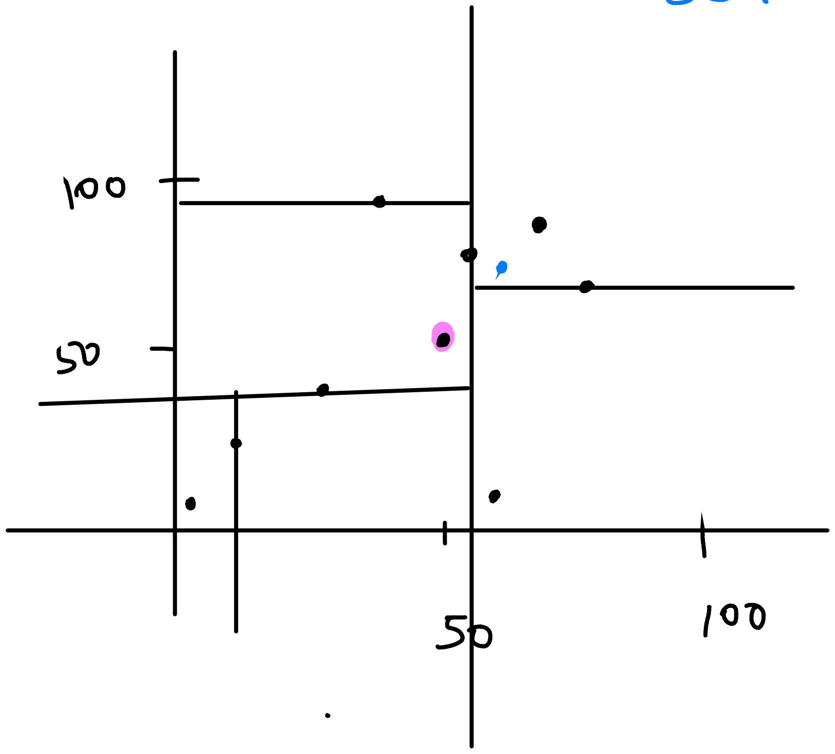
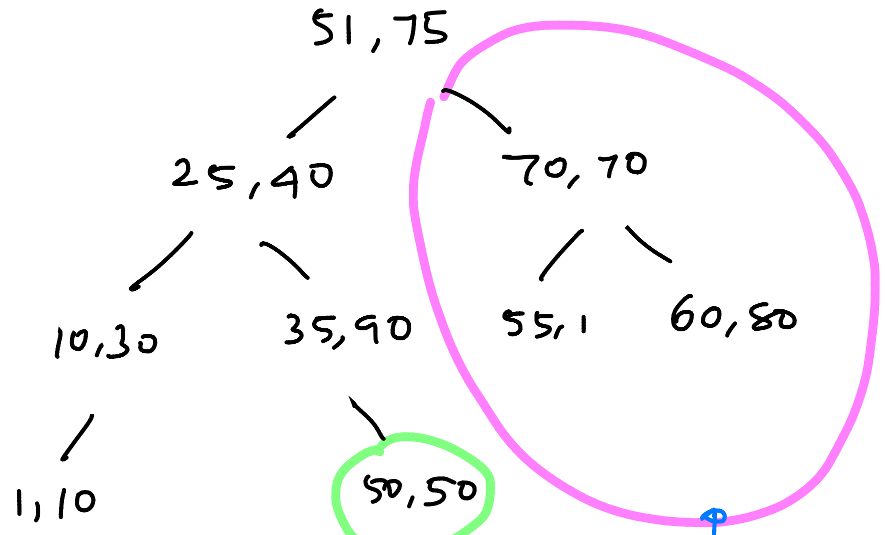
```
delete ( x, node t, cut-dim)
                 root node and its cut-dimension

if x = t.data
    if t.right != NULL
        t.data = FINDMIN ( t.right, cut-dim, (cut-dim+1) % total-dim)
        delete ( t.data , t.right , (cut-dim+1) % total-dim)

    else if t.left != NULL
        t.data = FINDMIN ( t.left, cut-dim, (cut-dim+1) % total-dim)
        t.right = t.left
        t.left = NULL
        delete ( t.data, t.right , (cut-dim+1) % total-dim)
else if x[cut-dim] < t.data[cut-dim]
        delete ( x , t.left, (cut-dim +1) % total-dim)

    else
        delete ( x , t.right, (cut-dim +1) % total-dim)
```

Search the data in the tree nearest to (52, 52)





Actual nearest point

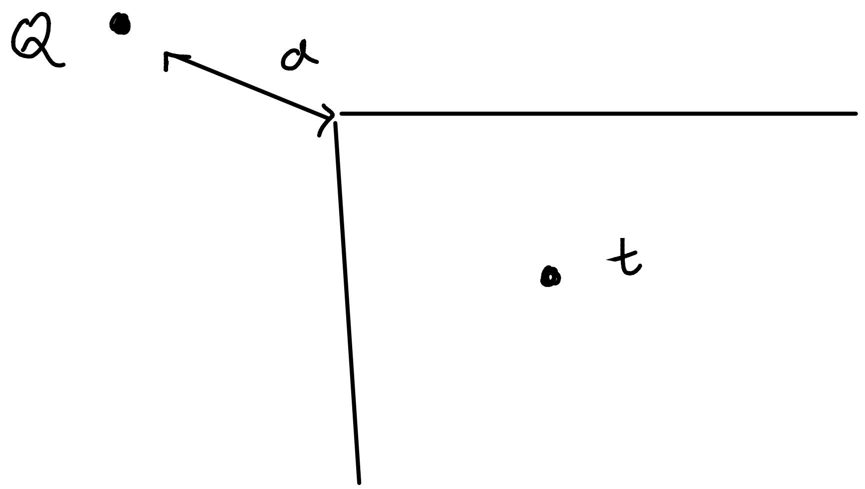
Since 52 > 51 Query point will get routed to right sub-tree

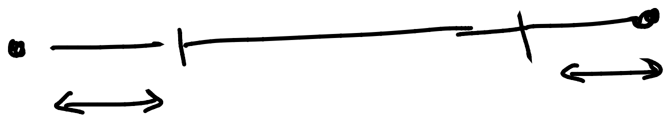
Query : (52, 52)

Why does this happen : 51 is nearer to 52 than 50 but other coordinates will make a difference

Idea: Maintain

- closest point found till now C
- Bounding box for each sub-tree





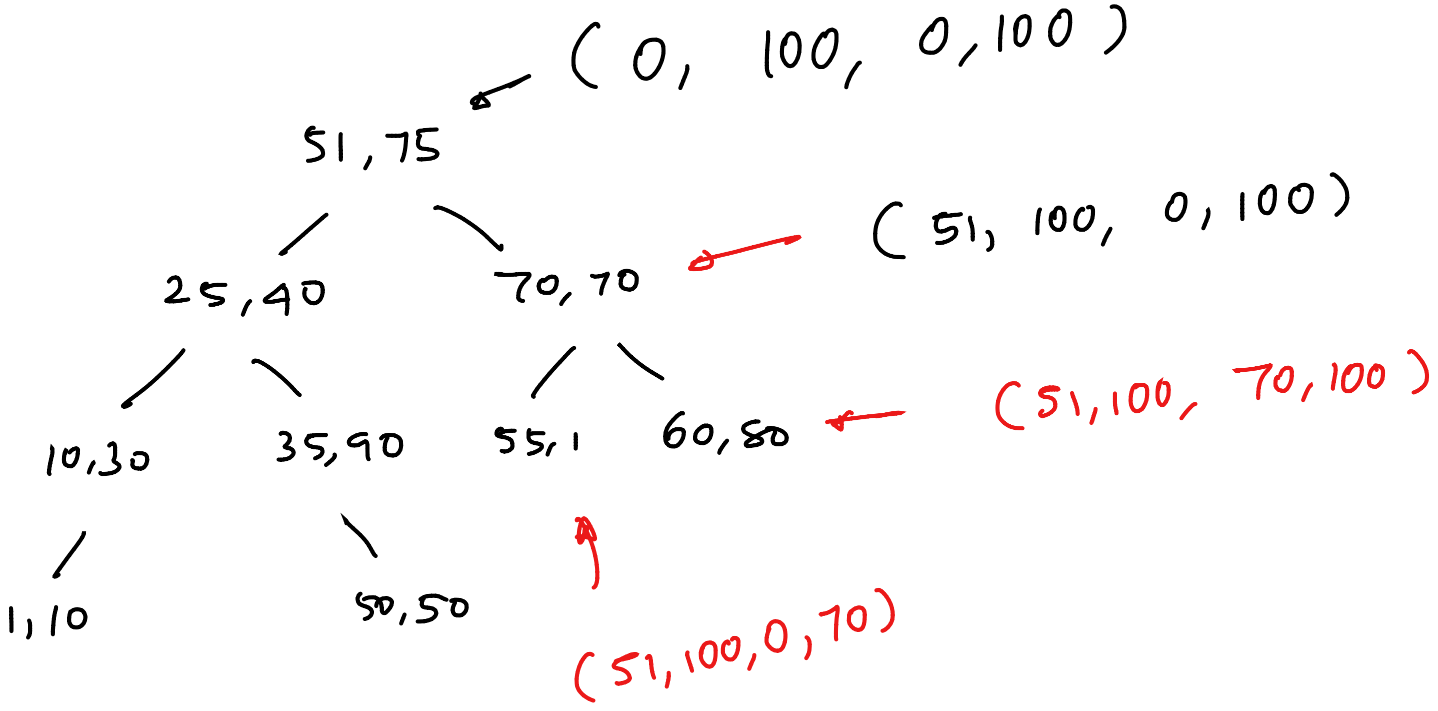
$\text{dist}(Q, BB(t)) > \text{dist}(Q, C)$ no need to search the sub-tree

$$\text{dist}(Q, BB) = (Q(1) - \underset{\sim}{y(1)})^2 + \dots + (Q(d) - y(d))^2$$

$$\text{dist}(x, y) = (x(1) - \underset{\uparrow}{y(1)})^2 + \dots + (x(d) - y(d))^2$$

Box

(start_dim-1, end-dim-1, start_dim_2, end-dim_2, ... , start-dim-d, end-dim



maintain best point, best-dist as global variables.

```
NN ( Query Q, node t, cd, BB)

    if distance ( BB, Q) > best. dist then return

    dist = distance (Q, t.data)

    if dist < best-dist :
        best = t.data
        best-dist = dist

    NN ( Q, t. left, (cd+1)% d, BB.trim left (cd, t.data) )
    NN ( Q, t. right, (cd+1)% d, BB.trim left (cd, t.data) )
```

```
# Import necessary library for distance calculation
import math

# Initialize global variables
best_point = None
best_dist = float('inf')

# Define the distance function (Euclidean distance)
def distance(point1, point2):
    return math.sqrt(sum((x - y) ** 2 for x, y in zip(point1, point2)))

# Define the nearest neighbor search function
def NN(Q, t, cd, BB):
    global best_point, best_dist

    # If the bounding box distance is greater than the best distance, prune this branch
    if BB.distance_to(Q) > best_dist:
        return

    # Calculate distance from the query point Q to the current node t's data
    if t is not None:
        dist = distance(Q, t.data)

        # If this point is closer, update best_point and best_dist
        if dist < best_dist:
            best_point = t.data
            best_dist = dist

        # Recursively traverse the left subtree
        if t.left:
            NN(Q, t.left, (cd + 1) % t.dimension, BB.trim_left(cd, t.data))

        # Recursively traverse the right subtree
        if t.right:
            NN(Q, t.right, (cd + 1) % t.dimension, BB.trim_right(cd, t.data))

# Define a simple BoundingBox class to represent BB
class BoundingBox:
    def __init__(self, lower, upper):
        self.lower = lower
        self.upper = upper

    def distance_to(self, point):
        # Calculate the minimum distance from the point to the bounding box
        dist = 0
        for i, p in enumerate(point):
            if p < self.lower[i]:
                dist += (self.lower[i] - p) ** 2
            elif p > self.upper[i]:
                dist += (p - self.upper[i]) ** 2
        return math.sqrt(dist)

    def trim_left(self, cd, data):
        # Update the upper bound for the left subtree
        new_upper = self.upper[:]
        new_upper[cd] = data[cd]
        return BoundingBox(self.lower, new_upper)

    def trim_right(self, cd, data):
        # Update the lower bound for the right subtree
```