1. Creating KD-Trees: Insert (50,30), (30,25), (40,25), (70,30), (70,25) into a KD-Tree

```
   iter-1: (30,25), (40,25), (50,30), (70,30), (70,25)
                              (50,30)
   iter-2: (30,25), (40,25) -- (70,25), (70,30)
                              (50,30)
                              /      \
                           (40,25)   (70,30)
   iter-3: (30,25) -- (70,25)
                              (50,30)
                              /      \
                           (40,25)   (70,30)
                            /         /
                        (30,25)    (70,25)
```

2. Inserting into an R*-Tree

```
   (1,3), (10,3), (4,8), (7,4), (4,5), (7,5):

   After first 4 inserts:
    - [(1,3), (10,3), (4,8), (7,4)] - use Y axis for split

   After split and final 2 inserts:

                 [{(1,3), (4,8)}, {(7,4),(10,5)}, {}, {} ]
                     /                にbackslash
```

3. Forced reinsert is a maintenance technique in R* trees in which nodes of the tree are removed and reinserted into the tree to allow the ChooseSubtree algorithm a chance to better optimiize the directory rectangles of the tree. Early nodes can establish sub-optimal directory structures when more nodes are added later in the tree's life-span. Since no regular maintenance (aside from very localized splits) takes place to ensure the overall query performance for an R*Tree during insertion, these structures may never be recomputed leaving the tree in a sub-optimal state. Reinsertion allows these structures to be recomputed optimizing for the most up to date set of data.

4.

```
   a) MinDist and MinMaxDist
      Q->M1)
         MinDist: |5-4|^2 = 1
         MinMax:  |5-2|^2 + |3-2|^2 = 10

      Q->M2)
         MinDist: |5-7|^2 = 4
         MinMax:  |5-7|^2 + |3-5|^2 = 8

      Q->M3)
         MinDist: |5-8|^2 + |3-4|^2 = 10
         MinMax:  |5-8|^2 + |3-6|^2 = 13
```

b) M3 can be discarded because it's MinDist is greater than the MinMaxDist of M2

c) M1 (dist of 1) followed by M2 (dist of 4)

d) First iteration would access M1 (O1, O2, O8)

- Algorithm would pick O8 as the closest node with distance |5-4|^2 + |3-2|^2 = 2

e) Upon returning from accessing the first leaf node in part d, an object has been found at a distance of 2. Applying heuristic 3, M2 may be discarded since it's MinDist is greater than 2. Since M3 has already been discarded and M1 has already been explored, the algorithm for 1NN would return with a result of O8 at a distance of 2.