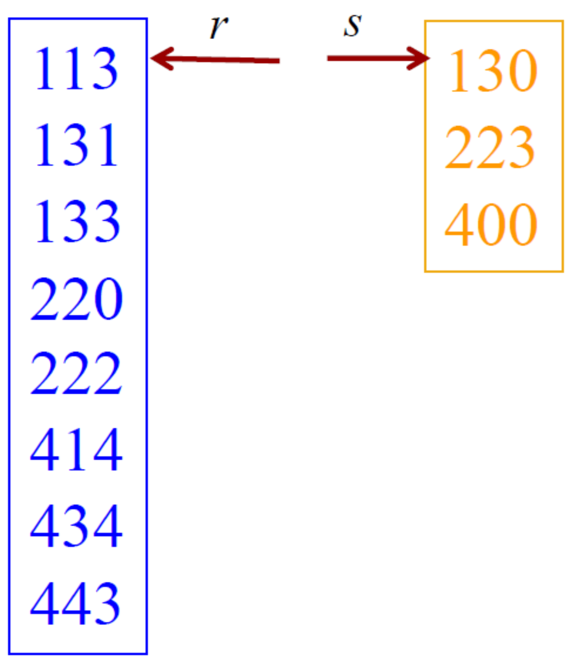
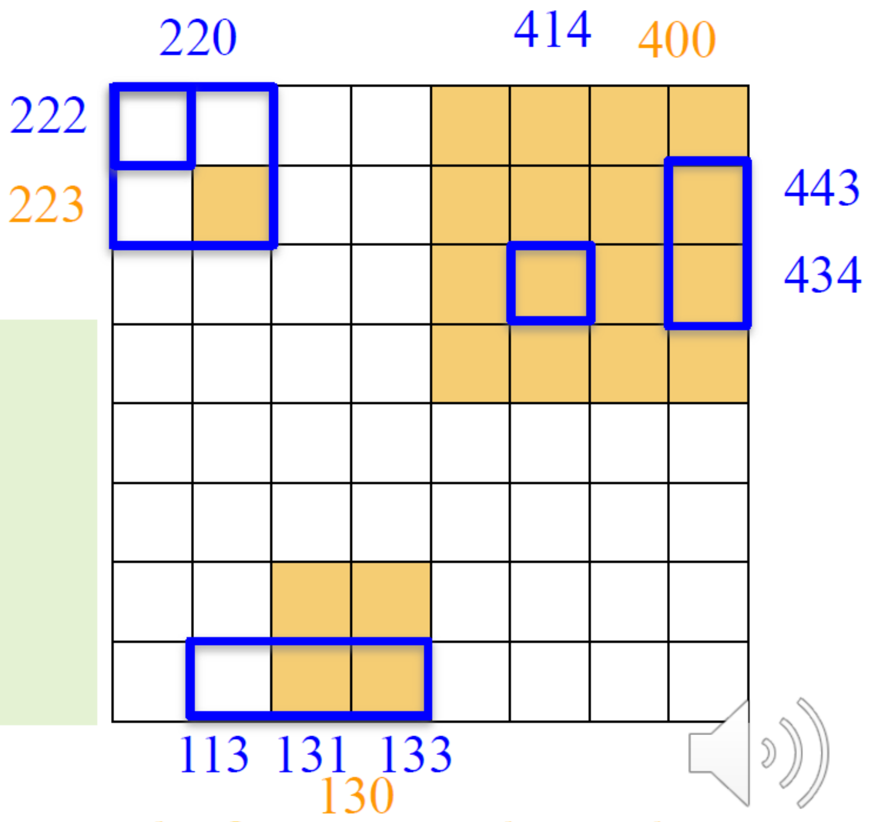
Q1

Sort-Merge Join using Z-values

+ Sort-Merge Join using Z-values

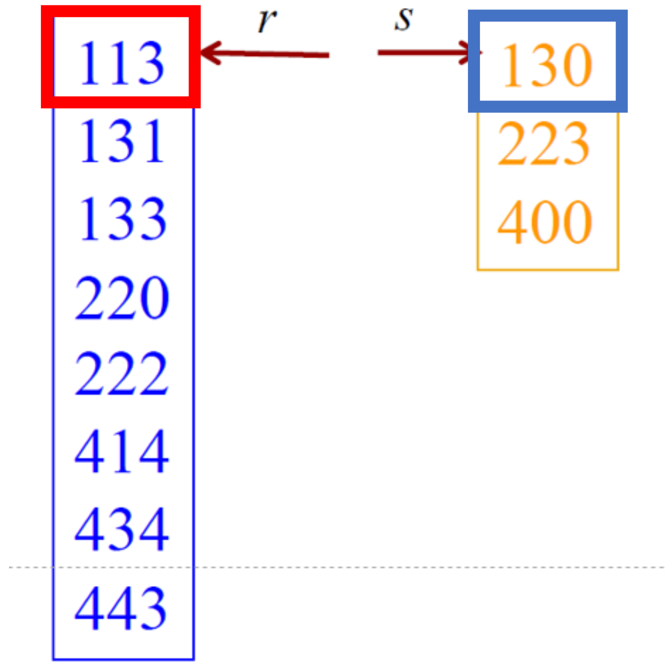


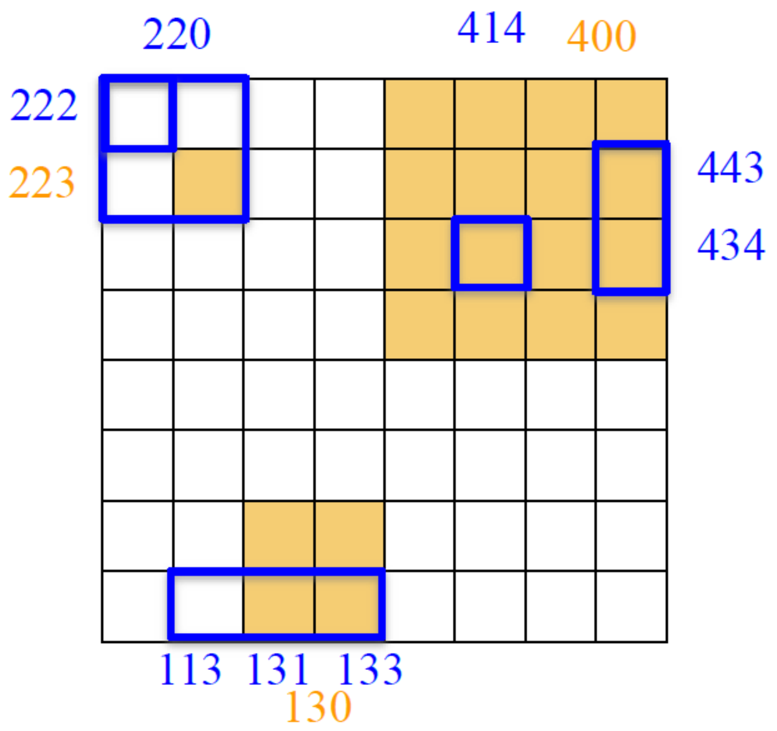
- Differences
  1. $131 \neq 130$
  2. Cannot move from 130 to 223 immediately



Algorithm Sketch:
1) Two sorted lists and two pointers
2) Synchronized traversal
   - overlap(*r*, *s*)?
   - increase min(*r*, *s*)
3) Some values in stack

Reading: the paper by Orenstein and Manola 1988.



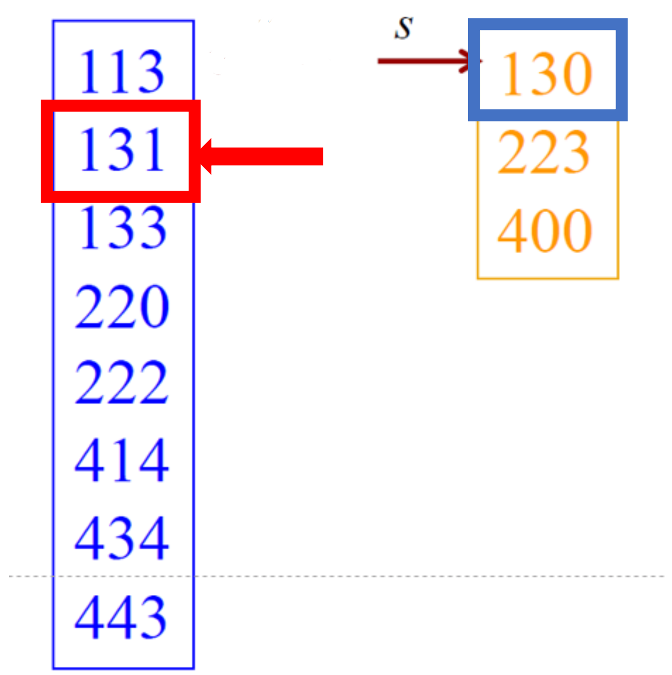


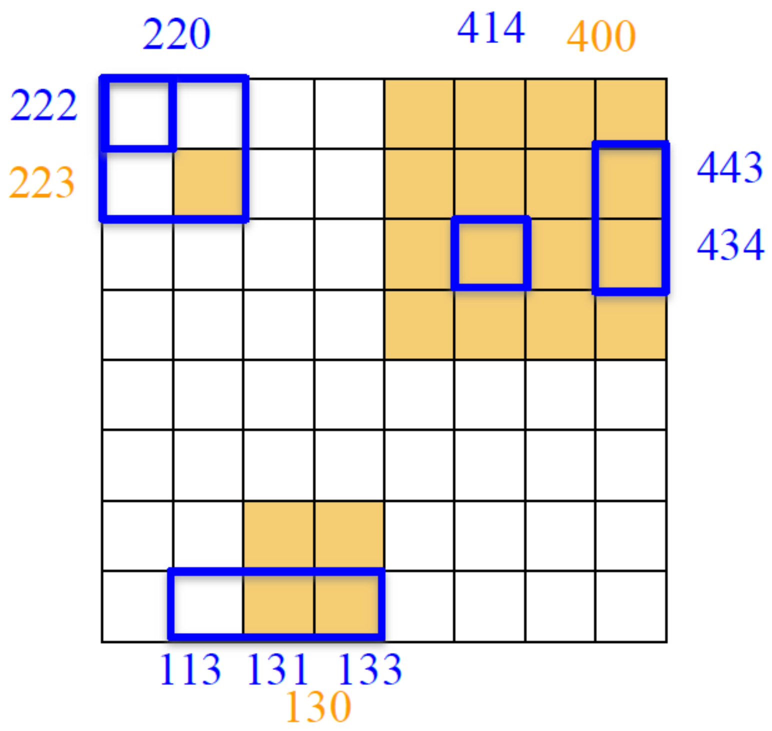
Step 1:

Take 113 and 130, because 13 and 30 do not intersect, we skip this pair.

Because 113 is smaller than 130, and 130 is at a higher level, we visit the next value in r.

C = {}

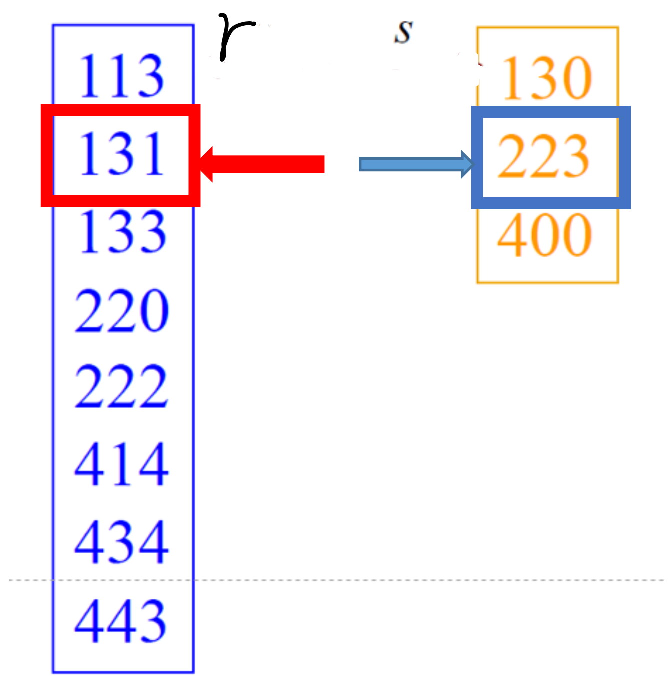


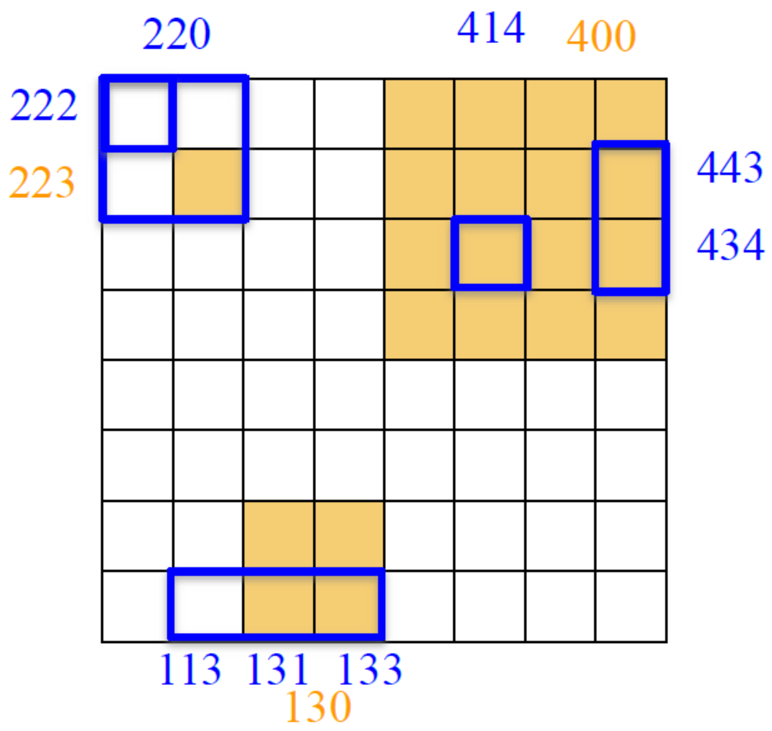


Step 2:

Take 131 and 130, because 30 covers 31, we add (131, 130) into the candidate list. Because 131 is larger than 130, we visit the next value in s. Meanwhile, because 130 is at a higher level, we have to keep 130 in stack (do not remove it from the comparison list).

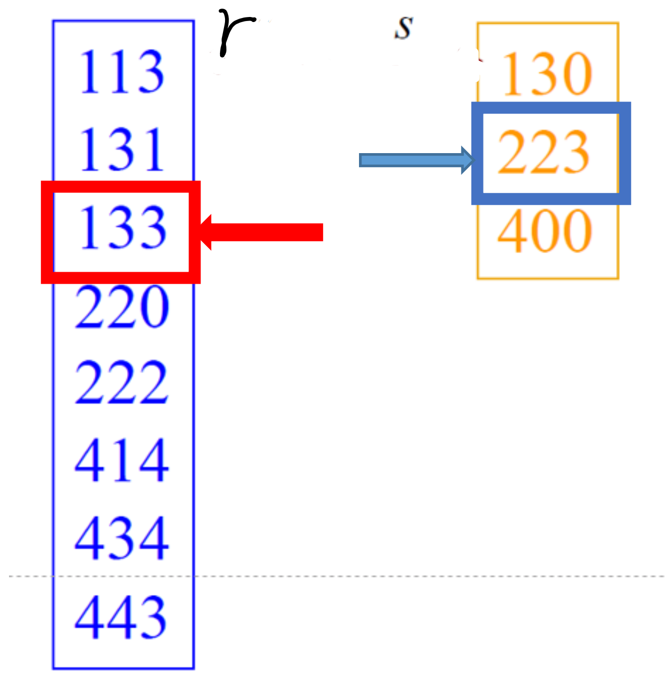
C = {(131,130)}   S = {130}

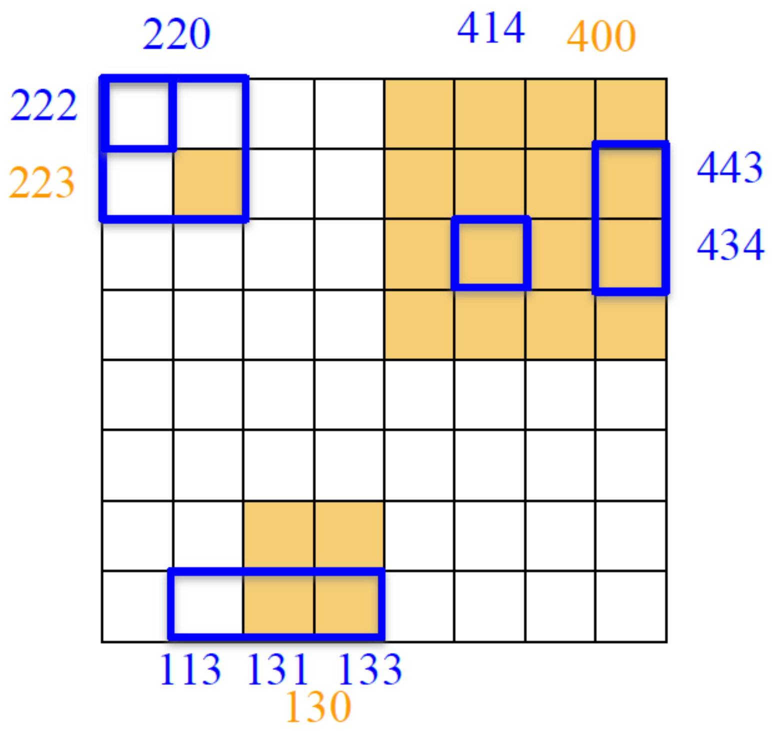




Step 3:
Take 131 and 223, they do not intersect, and 223 is larger than 133, we visit the next value in r.

C = {(131,130)}   S = {130}



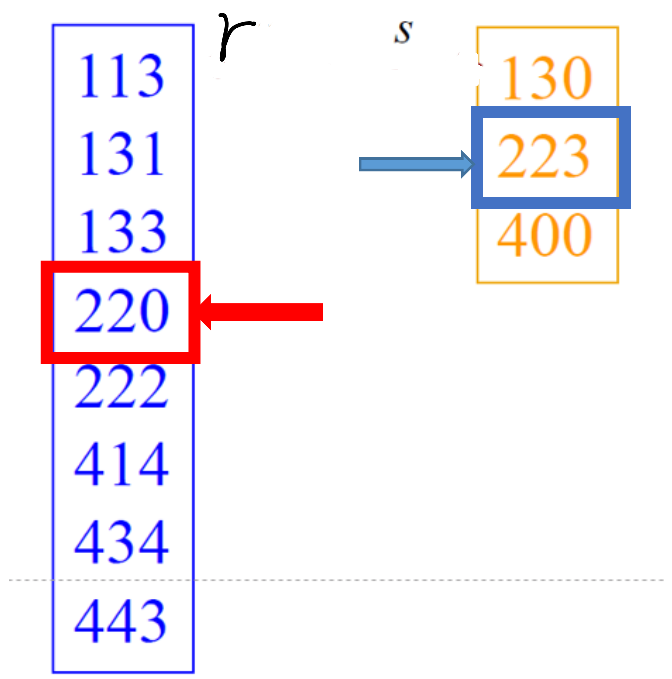


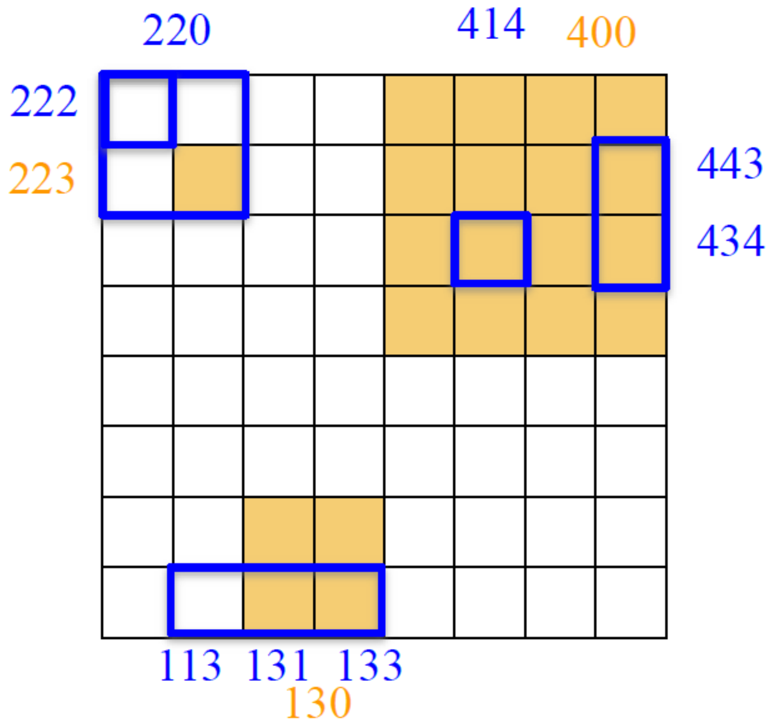
Step 4:

Take 133 and 130, because 30 covers 33, we add (133, 130) into the candidate list.

Take 133 and 223, they do not intersect. Because 133 is smaller than 223, we visit the next value in r.

C = {(131,130), (133, 130) }   S = {130}

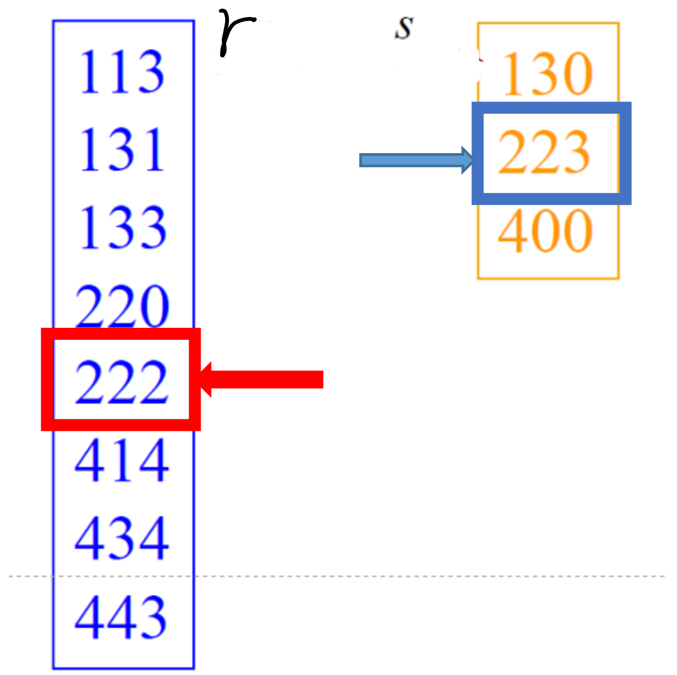


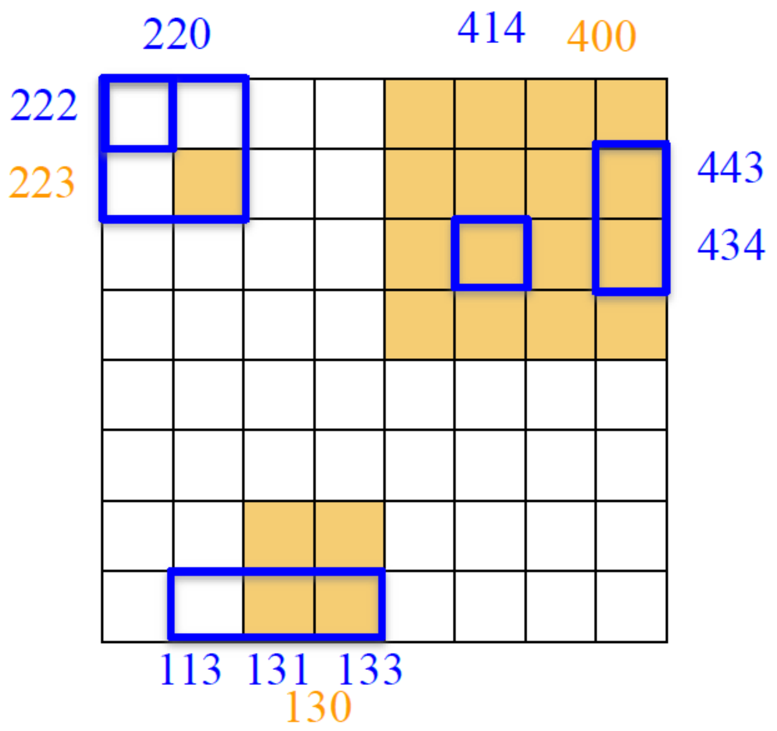


Step 5:

Take 220 and 130, they are totally different, and 130 is smaller than 220, they are at the same level, so it is guaranteed that no value in R will intersect with 130. Therefore, we remove 130 out of stack. Next, we compare 220 and 223, because 220 covers 223, we add (220, 223) into C. 220 is at a higher level than 223, we put 220 into stack, and we visit next value in r.

C = {(131,130), (133, 130), (220, 223) }   S = {220}

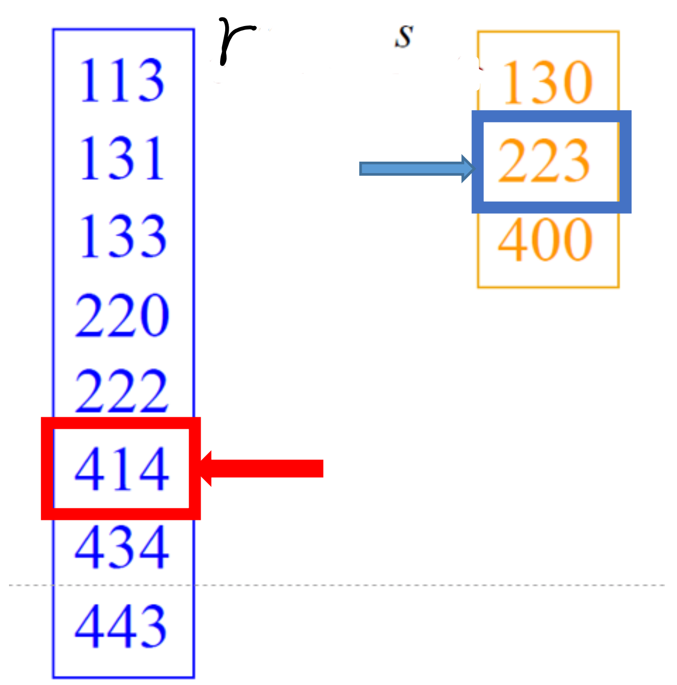


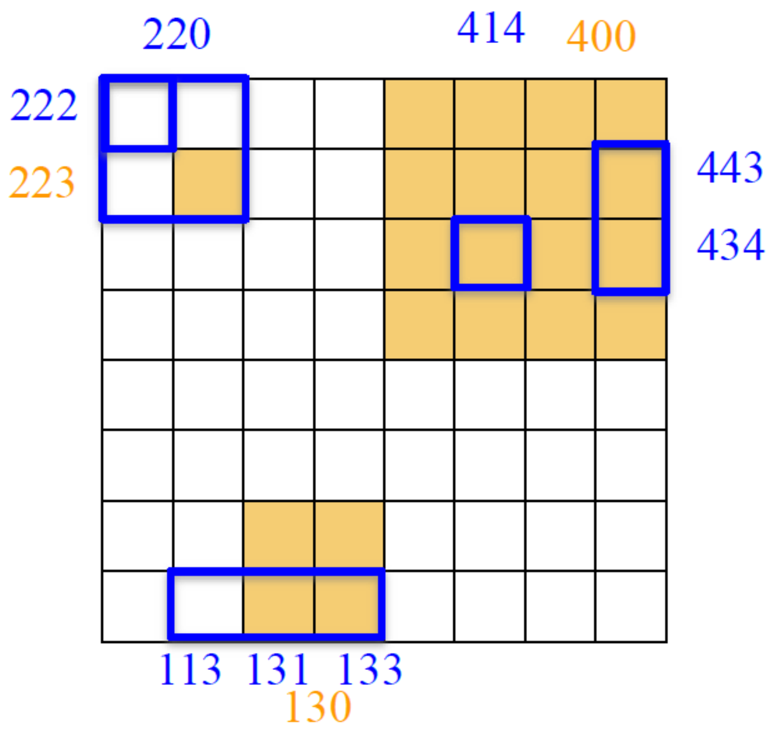


Step 6:

Take 222 and 223, they do not overlap, and we visit the next value in r.

C = {(131,130), (133, 130), (220, 223) }   S = {220}

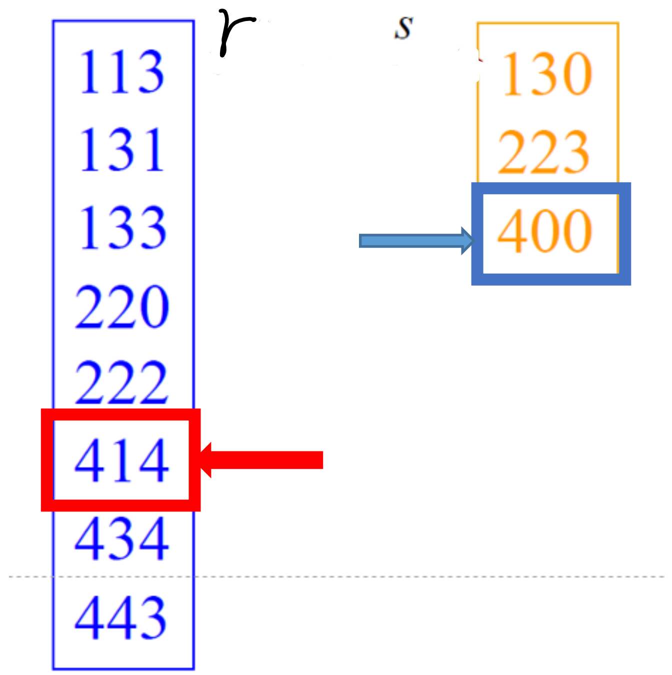


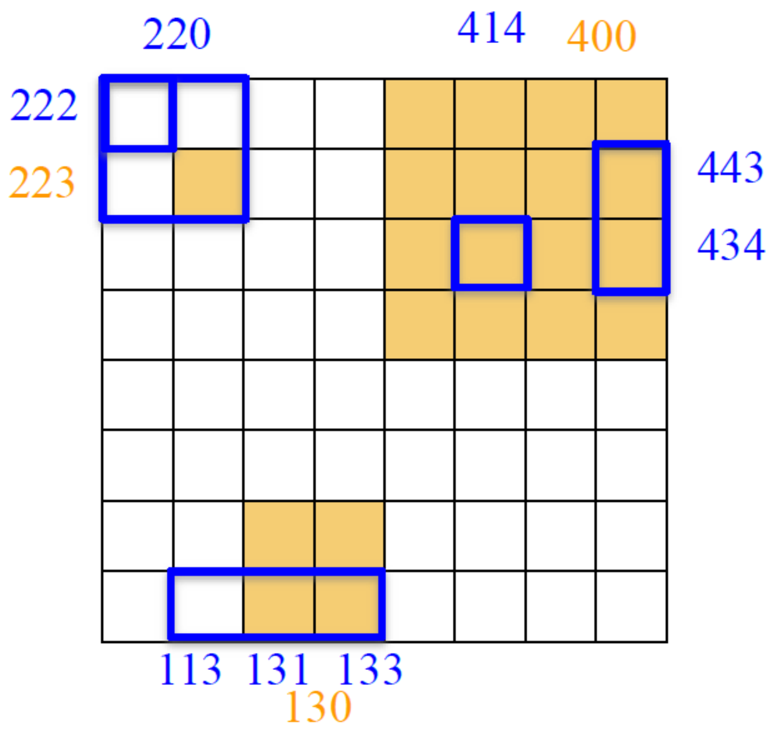


Step 7:

414 and 223, they do not intersect, and 414 is larger than 223, we visit the next value in s.

C = {(131,130), (133, 130), (220, 223) }   S = {220}



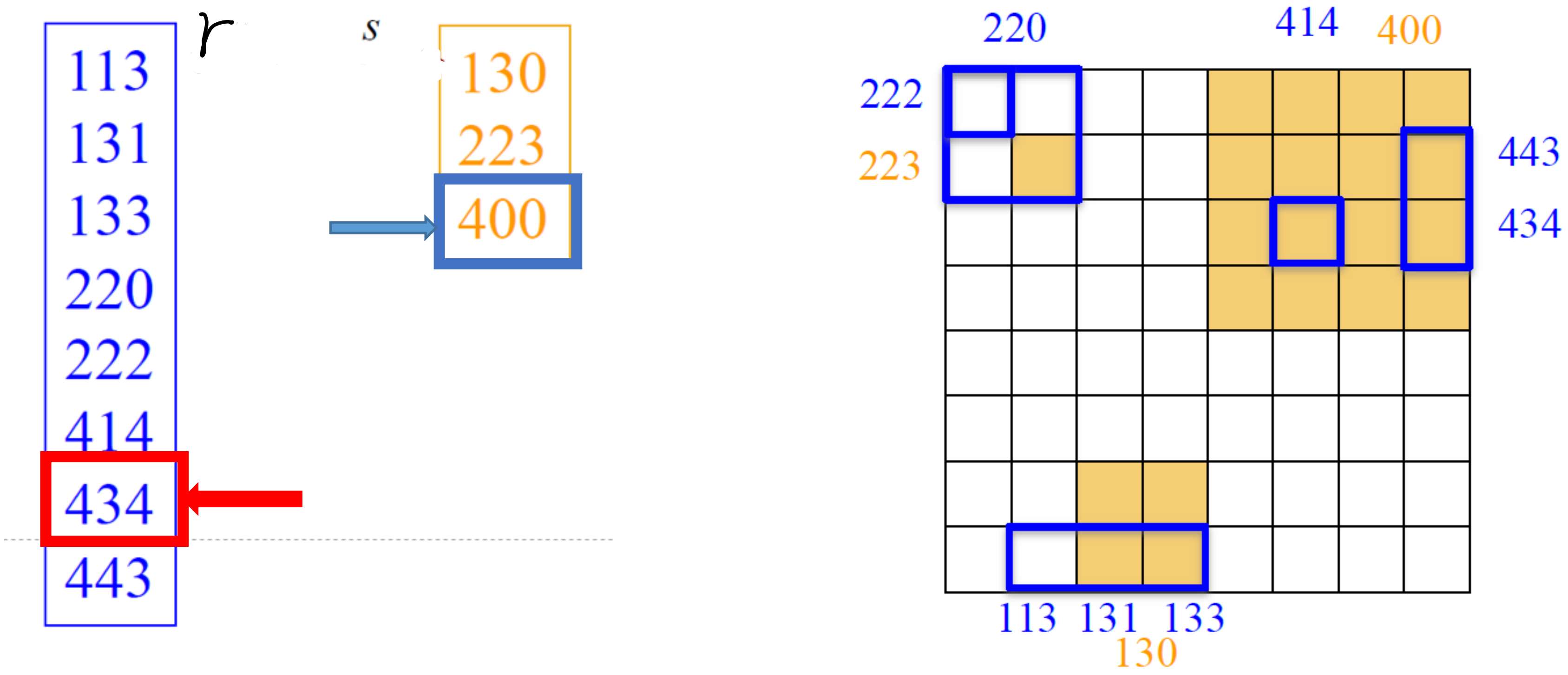


Step 8:

Take 220 and 400, they do not intersect, we move 220 out of stack.

Take 414 and 400, 414 is covered by 400, so we put (414, 400) into the candidate list. 400 is the last value of R, and it is at a higher level, so we put 400 into the stack, and visit the next value in r.

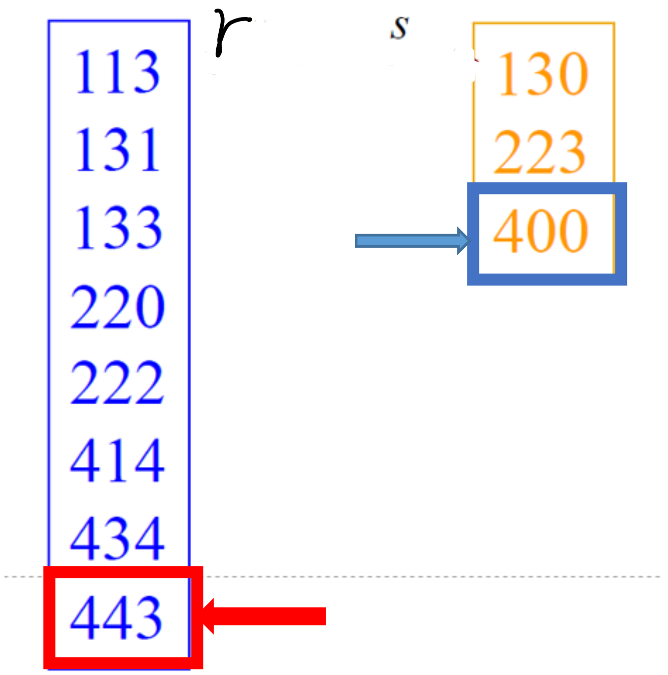
C = {(131,130), (133, 130), (220, 223), (414, 400) }   S = {400}

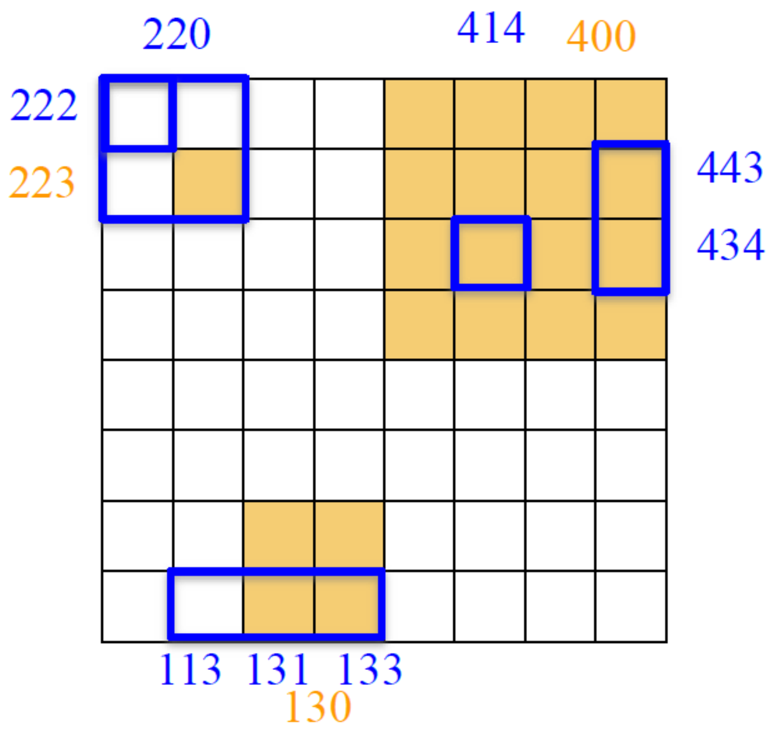


Step 9:

Take 434 and 400, 434 is covered by 400, so we add (434, 400) into the candidate list, and we visit the next value in r.

C = {(131,130), (133, 130), (220, 223), (414, 400), (434, 400)}
S = {400}





Step 10:

443 is covered by 400, so we add (443, 400) into the candidate list, and the merge finishes.

C = {(131,130), (133, 130), (220, 223), (414, 400), (434, 400), (443, 400)}

S = {400}

C = {(131,130), (133, 130), (220, 223), (414, 400), (434, 400), (443, 400)}

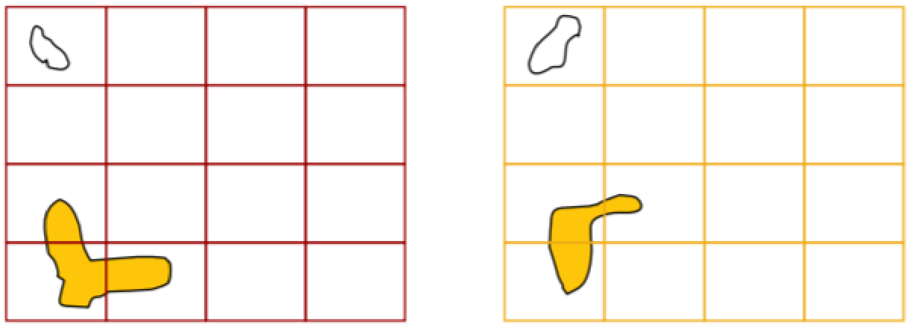
We might do some housekeep check because some of the testing pairs are duplicated, like (131,130) and (133,130). Both 131 and 133 points to the same object, so we only need to retrieve and test the actual intersection once.

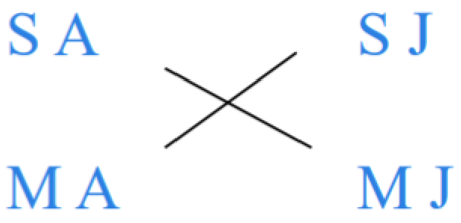
Q2

Spatial Hash Join: When we do the hash based spatial join, we have two choices of hash functions: single assignment and multiple assignment, and two join procedures: single join and multiple join. Different index would require different join procedure. Please discuss which kind of join procedure should be used by **R-Tree**, **Z-order**, and **R+-Tree**, and explain why.

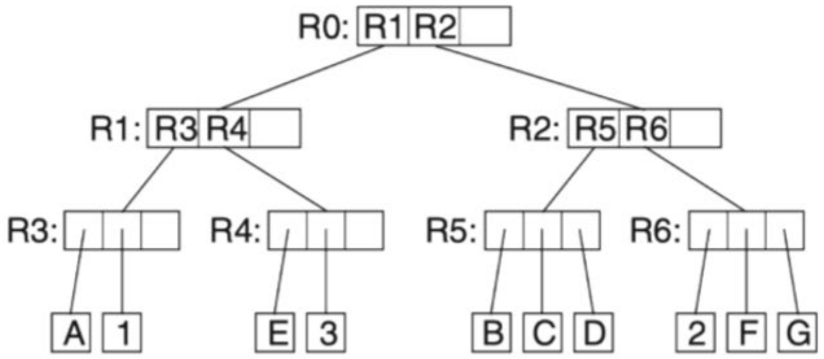
+ Spatial Hash Join

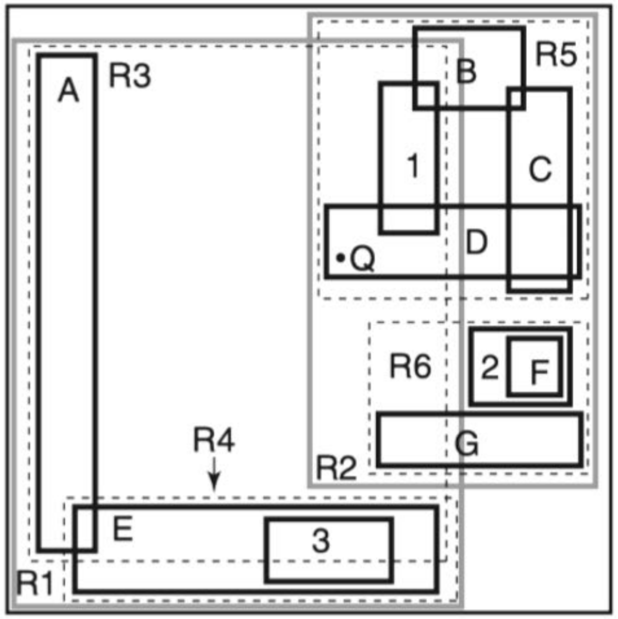
- Bucket Assignment
  - SA: a polygon is mapped to a cell by its centroid
  - MA: a polygon is mapped to all cells it overlaps with
- Bucket Join
  - SJ: one R bucket joins with one corresponding S bucket
  - MJ: one R bucket joins with many S buckets
    - Question: which buckets to join with?





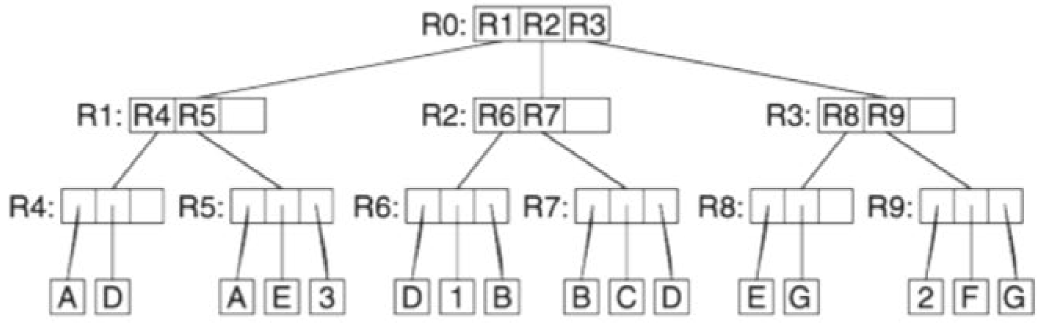
R tree

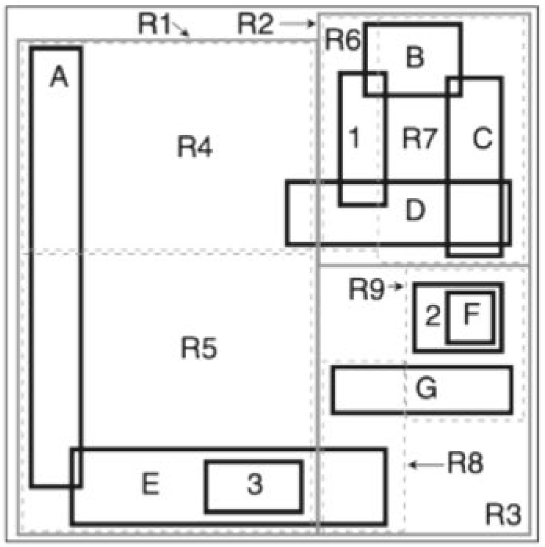




each object is enclosed by **only one MBR**, so it belongs to the **single assignment**. When we use the single assignment hash function, we have to run the **multiple join procedure** to guarantee the correctness. Therefore, to get the result, we have to join all the intersected objects together.

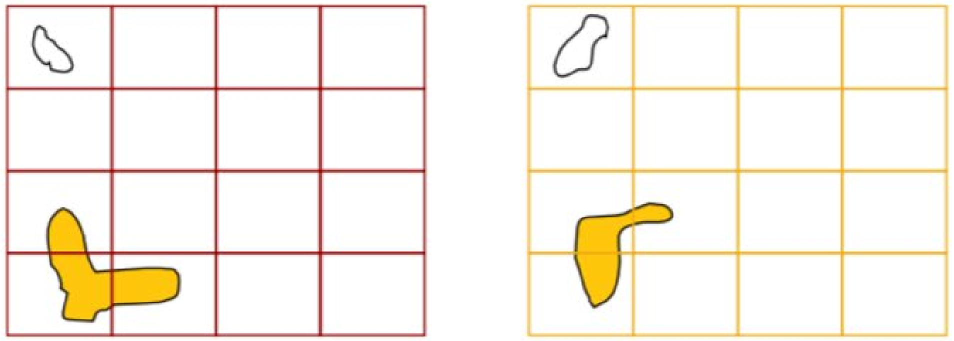
R+ tree

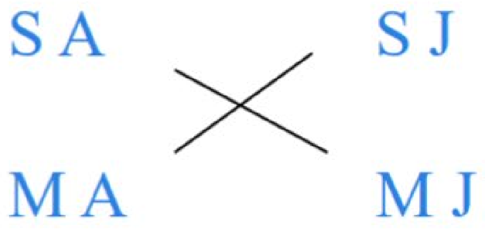




one polygon can exist in **several MBRs**, so it belongs to the **multiple assignment**. When we use the multiple assignment hash function, we only need to join once for each object to obtain the result. (single join)

Z-order





For Z-order, one polygon can exist in **several cells**, so it belongs to the **multiple assignment**. When we use the multiple assignment hash function, one cell joins with one corresponding cell. (single join)

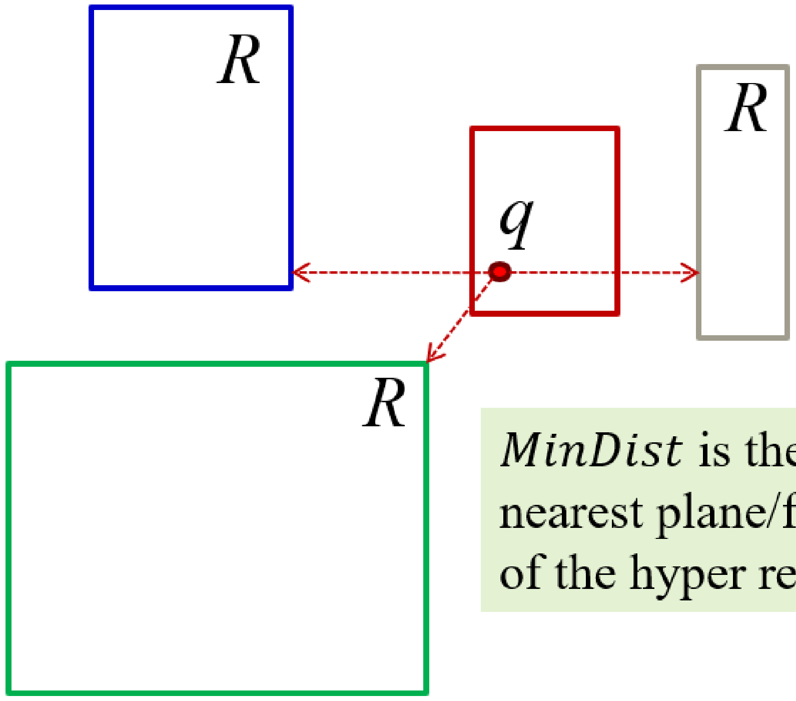
Q3

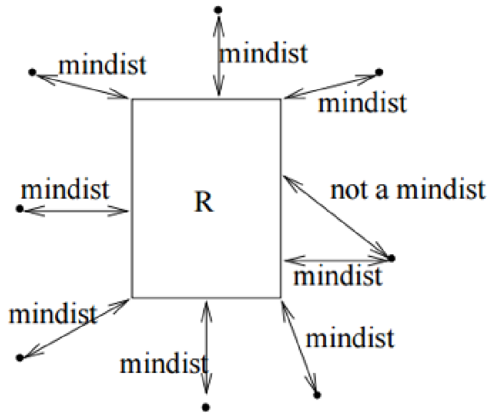
R tree NN search: When using the R-tree to search for the nearest neighbor, we have two metrics to determine the search order: **MINDIST** and **MINMAXDIST**. The MINDIST is the optimistic choice, while the MINMAXDIST is the pessimistic one. Is the MINDIST always a better ordering than using the MINMAXDIST?

+ MinDist

A lower bound distance for any point in R to q

- If q is inside R, then $MinDist = 0$
- If q is outside of R, $MinDist$ is the distance of q to the closest point of R (one point of the perimeter)





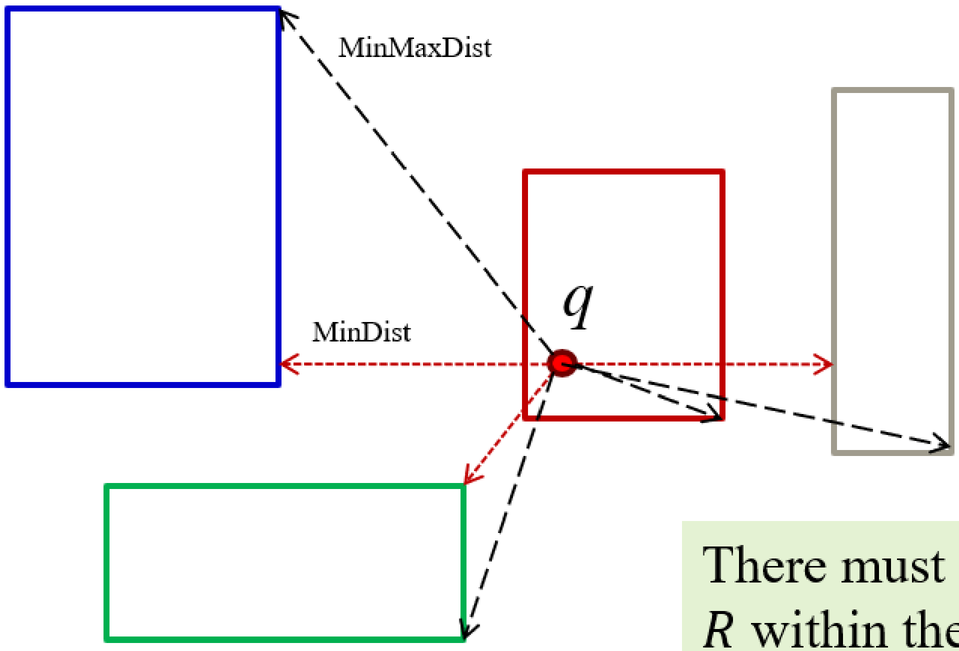
$MinDist$ is the distance of the point q to the nearest plane/face (nearest side in 2 dimension) of the hyper rectangle R.

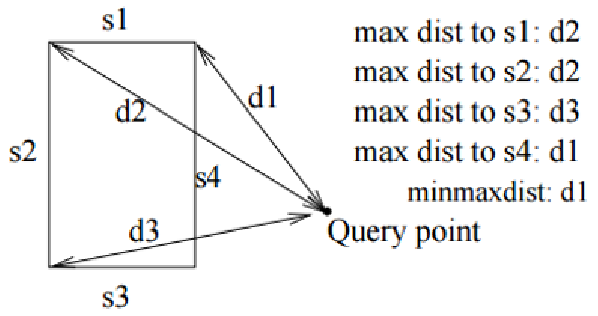
...to order and prune the search in an R-tree

+ MinMaxDist

An upper bound of distance from q to R

- For *each* dimension, find the closest face, compute the distance to the furthest point on this face and take the minimum of all these distances
- It is the smallest possible upper bound of distances from q to R
- There exists at least one point p in $R, d(p,q) \leq MinMaxDist$





There must exist a point of some spatial object in R within the distance $MinMax$ from the point q

+ Pruning Strategies

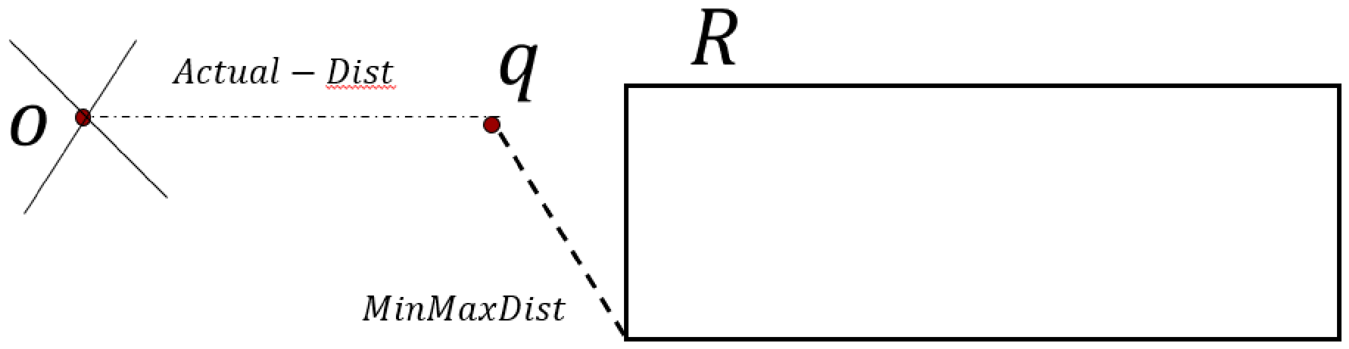
- Key observation

$$MinDist(q,R) \leq NN(q) \leq MinMaxDist(q,R)$$

- Strategy:
  1. An MBR R is discarded if there exists another R' such that $MinDist(q,R) > MinMaxDist(q,R')$
  2. An object *q* is discarded if there exists an R such that $d(q,p) > MinMaxDist(q,R)$
  3. An MBR R is discarded if a point p is found such that $MinDist(q,R) > d(q,p)$

+ Pruning 2 Example

- An object o is discarded if there exists an R s.t. the $Actual - Dist(q, o) \; > \; MinMaxDist(q, R)$



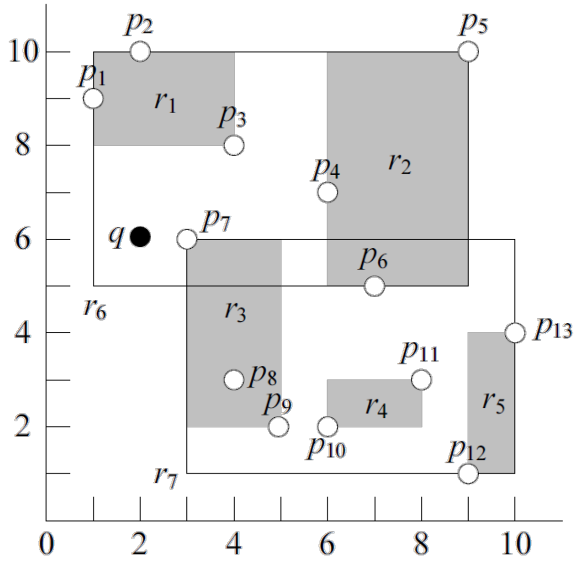
Q4

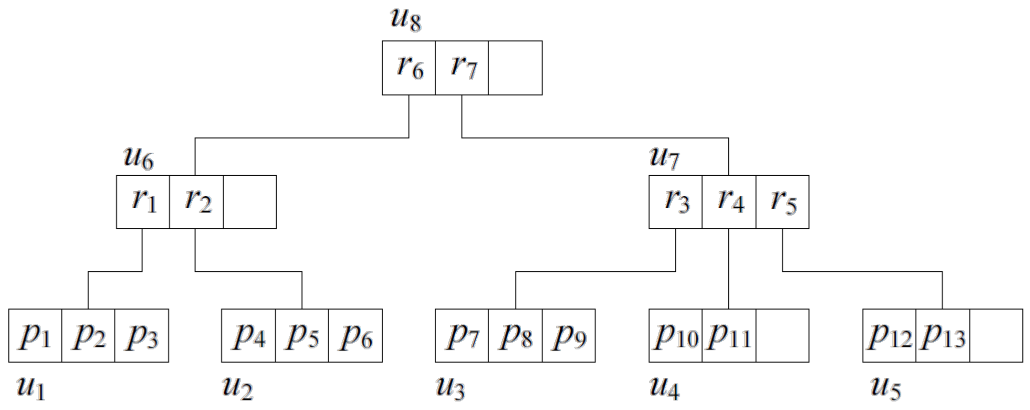
KNN BF: How to extend the **Best-First Nearest Neighbor** algorithm to answer the kNN query?

+ kNN Best First (BF)

- H: A sorted list with $MinDist$ as key
  - Use a heap
  - r_6 and r_7
  - $MinDist(q, r_6) = 0 < MinDist(q, r_7) = 1$

$r_6, 0$
$r_7, 1$



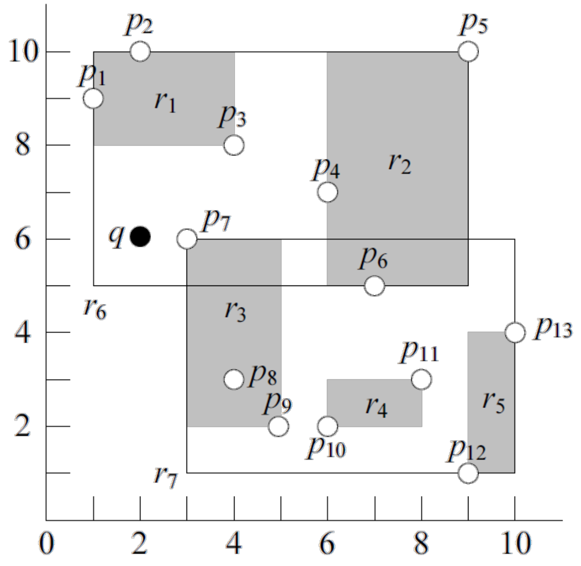


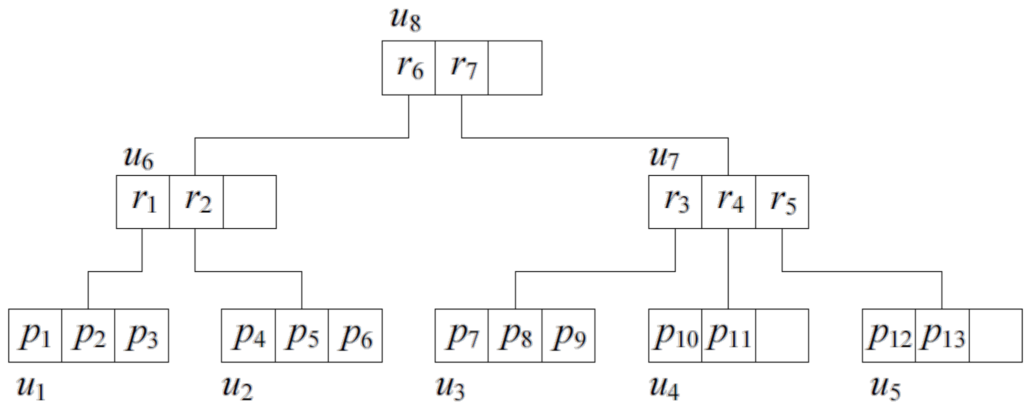
+ kNN Best First (BF)

- Visit u_6
  - r_1 and r_2
    - $MinDist(q, r_1) = 2$
    - $MinDist(q, r_2) = 4$

$r_6, 0$
$r_7, 1$

$r_7, 1$
$r_1, 2$
$r_2, 4$



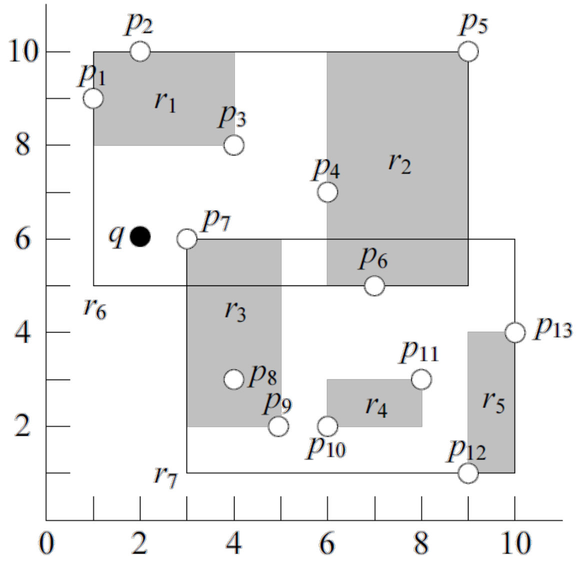


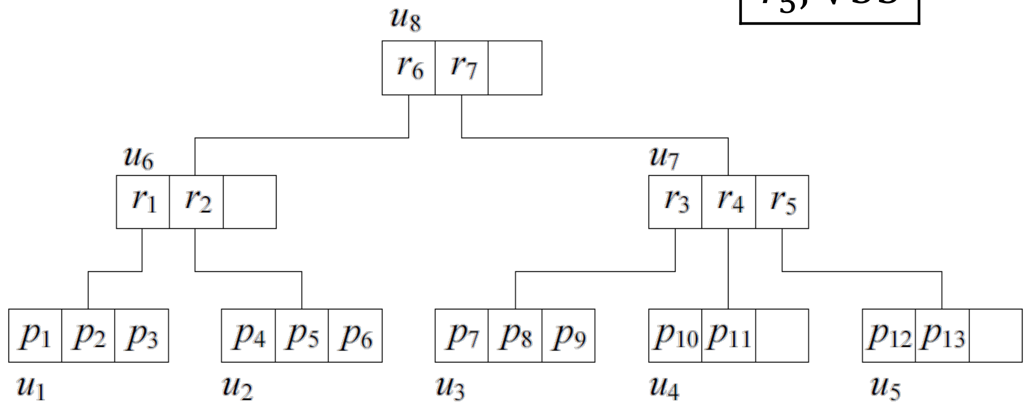
+ kNN Best First (BF)

- Visit u_7
  - r_3, r_4 and r_5
    - $d(q, r_3) = 1$
    - $d(q, r_4) = 5$
    - $d(q, r_5) = \sqrt{53}$

$r_7, 1$
$r_1, 2$
$r_2, 4$

$r_3, 1$
$r_1, 2$
$r_2, 4$
$r_4, 5$
$r_5, \sqrt{53}$



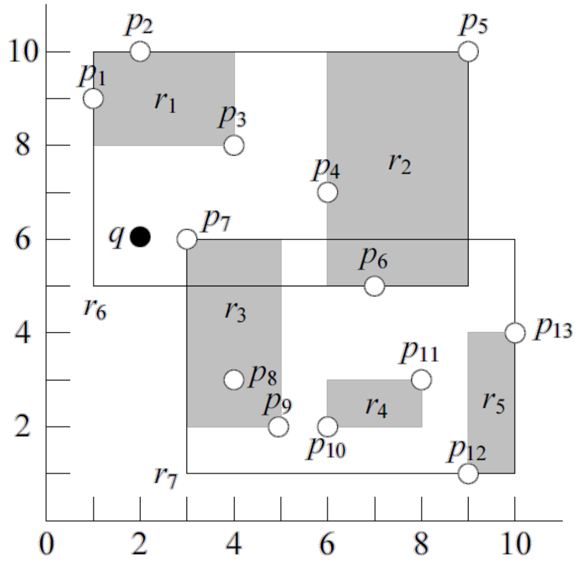


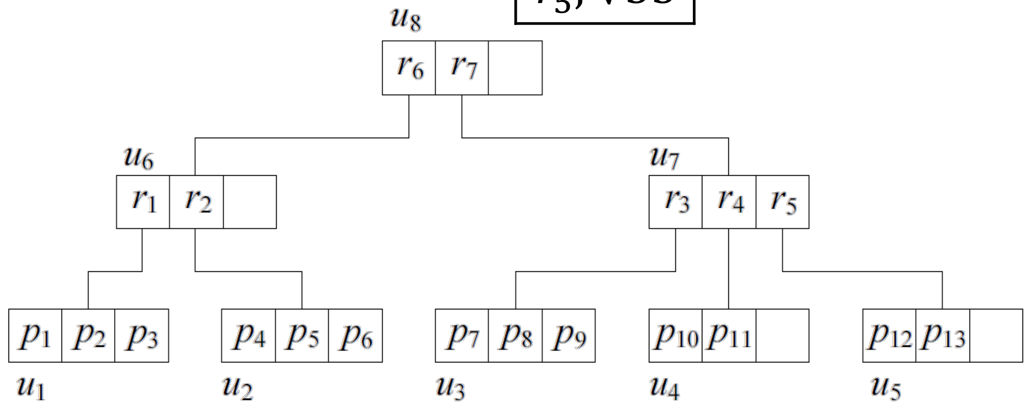
+ kNN Best First (BF)

- Visit u_3
  - p_7, p_8 and p_9
    - $d(q, p_7) = 1$
    - $d(q, p_8) = \sqrt{13}$
    - $d(q, p_9) = 5$
  - 1<2, p_7 is the nearest neighbour

$r_3, 1$
$r_1, 2$
$r_2, 4$
$r_4, 5$
$r_5, \sqrt{53}$

$r_1, 2$
$r_2, 4$
$r_4, 5$
$r_5, \sqrt{53}$





+ K-NN search

- Just a simple extension
  - Keep the sorted buffer of at most k current nearest neighbors
  - Pruning is done using the k-th distance

Q5

Skyline: Why the skyline result will appear in all monotonically increasing function-based NN results?

+ Monotonically Increasing Functions

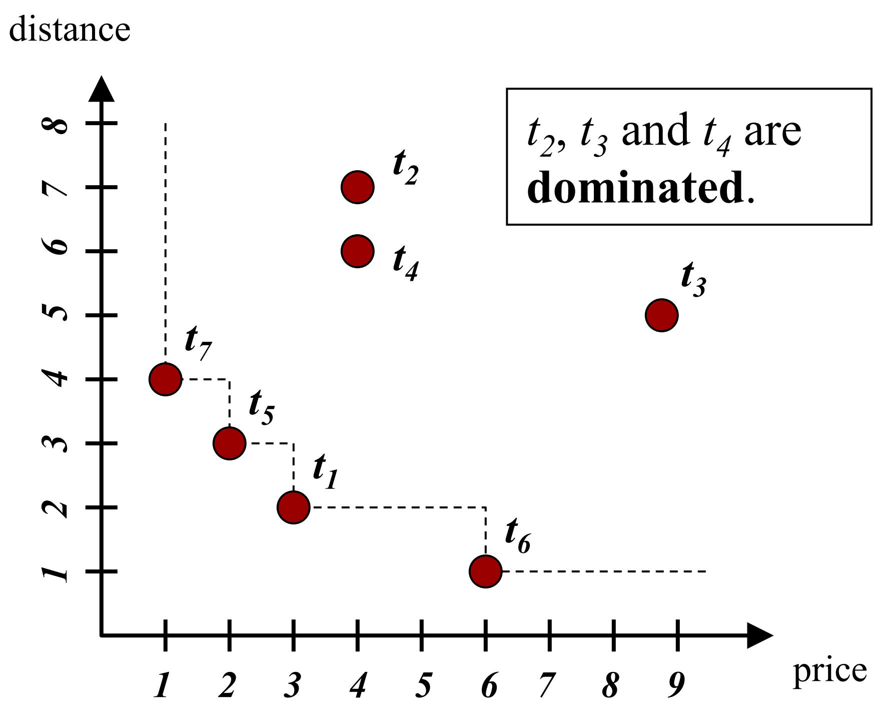
- Monotonically Increasing Functions
  - Let p be a d-dimensional point in $\mathbb{R}^d$
  - Let $f : \mathbb{R}^d \rightarrow R$ a function that calculates a score $f(p)$ for p
  - We say that f is monotonically increasing if the score increases when any coordinate of p increases.

A Formal Definition

- Assume the preference is smaller (e.g., closer to beach, cheaper hotel etc) – other semantics can be supported too
- For two d-dimensional points p and q, p dominates q iff
  - (1) $\forall 1 \leq i \leq d, p_i \leq q_i$ and
  - (2) $\exists 1 \leq i \leq d, p_i < q_i$
- Informally, a point dominates another point if it is as good or better in all dimensions and better in at least one dimension
  - Dominance is transitive
- A **skyline query** is to find all points from a dataset which are not dominated by any other points in the dataset

…Pareto dominance

+ Finding a Hotel…



t_2, t_3 and t_4 are **dominated**.

Skyline query: retrieve all points that are not dominated

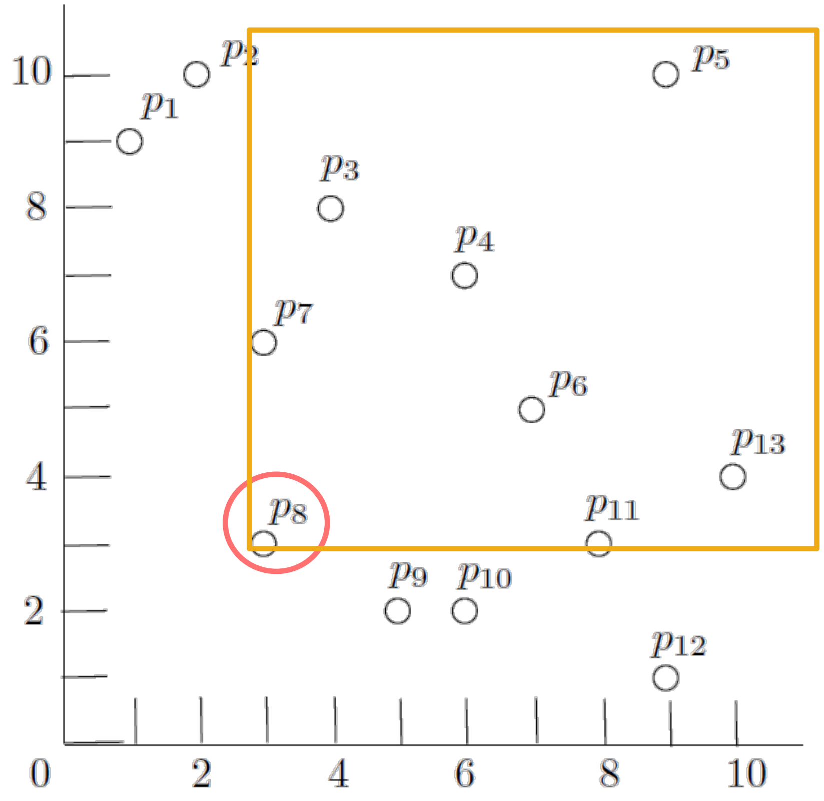
+ Skyline with NN

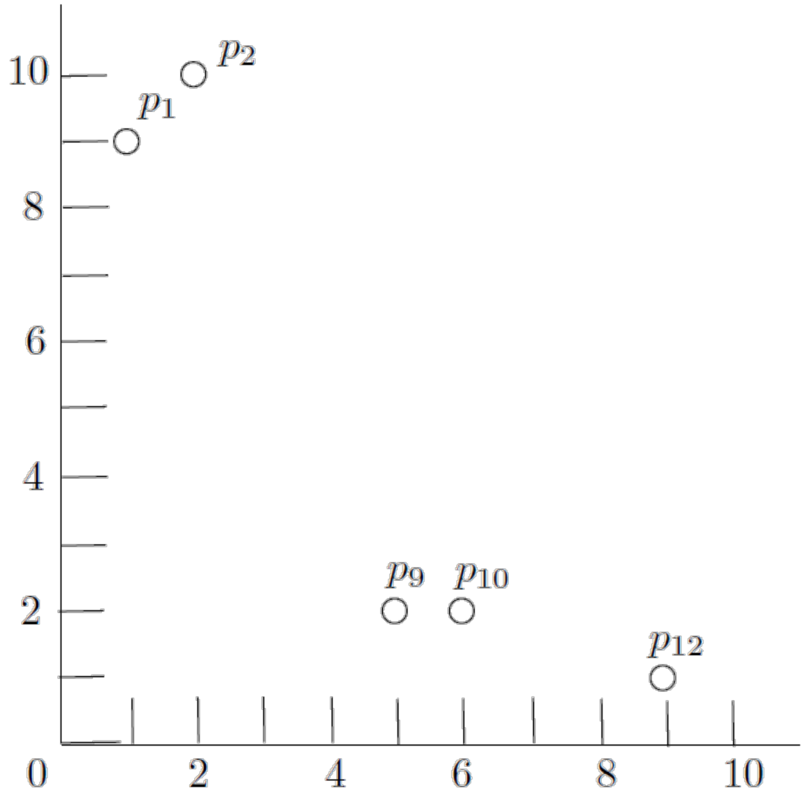
- Let p be the (Euclidean) NN of the origin of the data space, among all the points in the dataset P. Then, p must be in the skyline of P
  - Proof
    - Suppose it is not true, then p is dominated by at least another point p'
    - This, however, means that p' must be closer to the origin than p
    - Contradict
  - Use NN as a Black Box
    - Repeat the following operations until R-tree is empty

    1. Find the NN p of the origin of the data space. Include p into the skyline
    2. Delete p from the tree, as well as all the points that are dominated by p

+ Skyline with NN Example

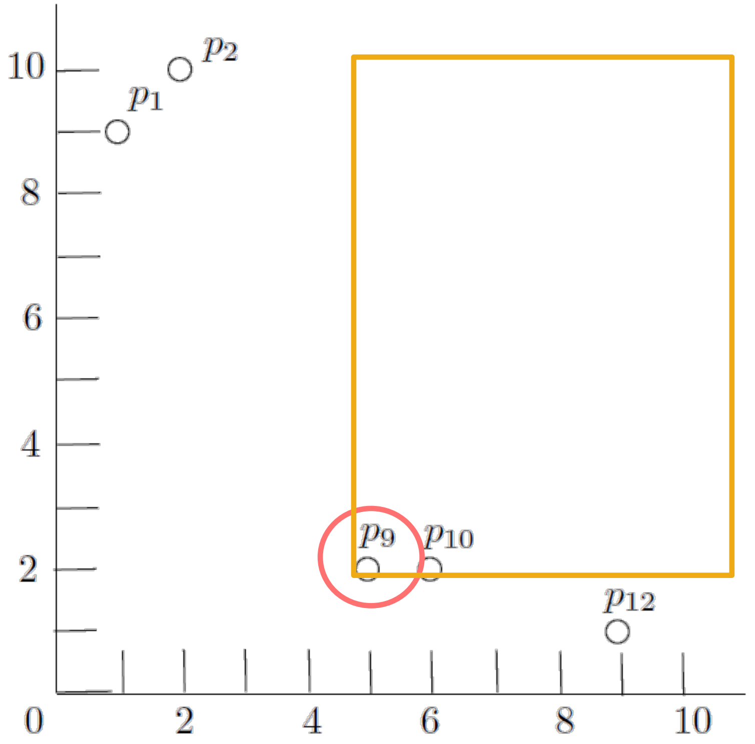
- The first NN query finds p_8
- The figure on the right shows the dataset after the corresponding deletions

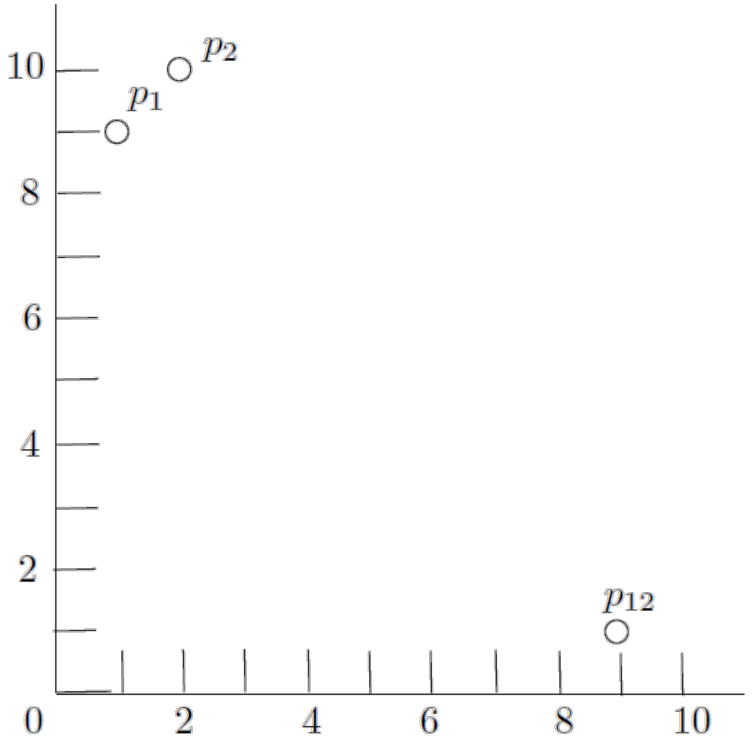




+ Skyline with NN Example

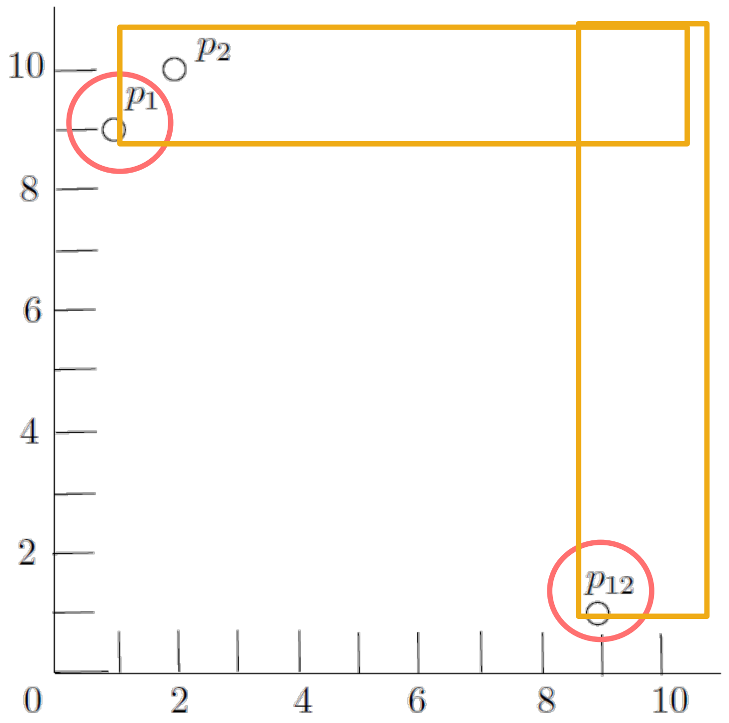
- The second NN query finds p_9
- The figure on the right shows the dataset after the corresponding deletions

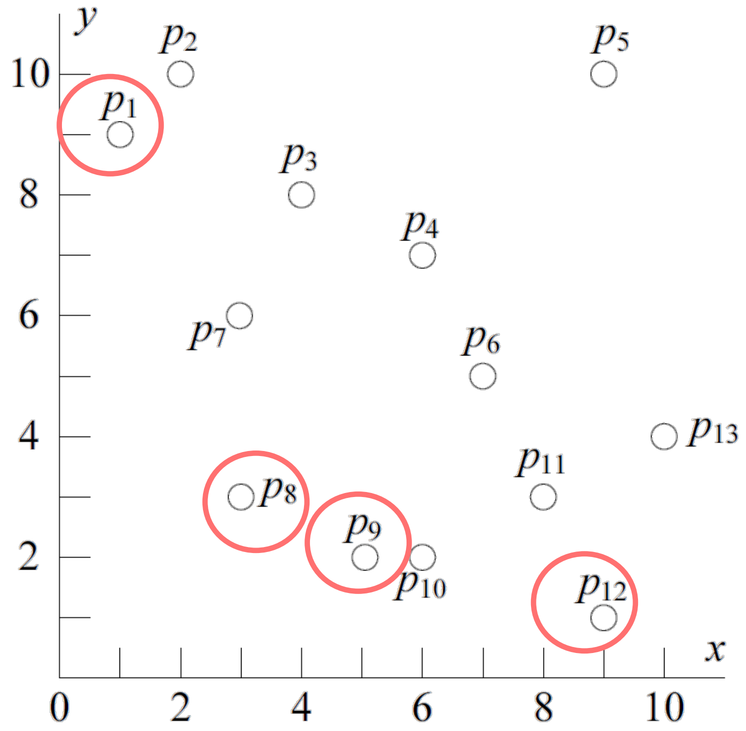




+ Skyline with NN Example

- The third NN query finds p_1 and p_{12}
  - The dataset becomes empty after corresponding deletions
  - Final skyline points=$\{p_8, p_9, p_1, p_{12}\}$


p_1
p_2
p_{12}
0
2
4
6
8
10


y
x
p_1
p_2
p_3
p_4
p_5
p_6
p_7
p_8
p_9
p_{10}
p_{11}
p_{12}
p_{13}
0
2
4
6
8
10