

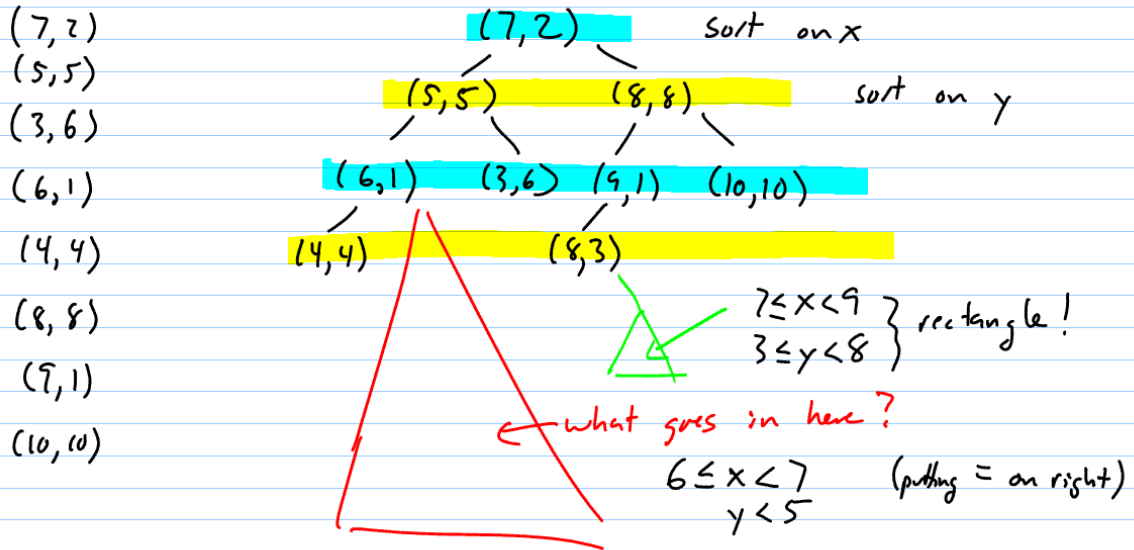
4 billion cells
* 1 byte/cell (256 terrain types)
4 GiB

decompose until each cell
has 1 terrain type

use pseudorandom numbers
for texture

kD-tree (2D-tree = "xy tree")

- binary search tree where component to sort on changes at each level



$\text{findInRadius}(x, y, r) = \text{findRect}(x, y, r, \text{root}, (-\infty, -\infty), (\infty, \infty), \text{true})$

$\text{findInRadius}(x, y, r, n, \text{rect}, \text{onX})$

if $n.\text{point}$ is within r of (x, y)
 $\text{process}(n.\text{point})$

if $(n.\text{left exists})$

 if (onX)

$\text{leftRect} = \text{rect} \cap x \leq n.\text{point}.x$

 else $\text{leftRect} = \text{rect} \cap y \leq n.\text{point}.y$

 compute distance (x, y) to leftRect

 if $< r$

$\text{findInRadius}(x, y, r, n.\text{left}, \text{leftRect}, !\text{onX})$

if $(n.\text{right exists})$

 // same as \uparrow except $\text{left} \rightarrow \text{right}; \leq \rightarrow >$

KNN (x, y, k)

init priority queue Q
 init list L to empty

Q.enqueue((root, (∞, -∞) - (∞, ∞), true), 0)

while !Q.empty() and L.size() < k

 p ← Q.dequeue()

 if (p is a point) then L.add(p)

 else

 (n, rect, onX) ← p

 if (n.left exists)

 compute left rectangle + distance
 Q.enqueue((n.left, leftRect, !onX), d)

 if (n.right exists)

 compute right rectangle + distance
 Q.enqueue((n.right, rightRect, !onX), d)