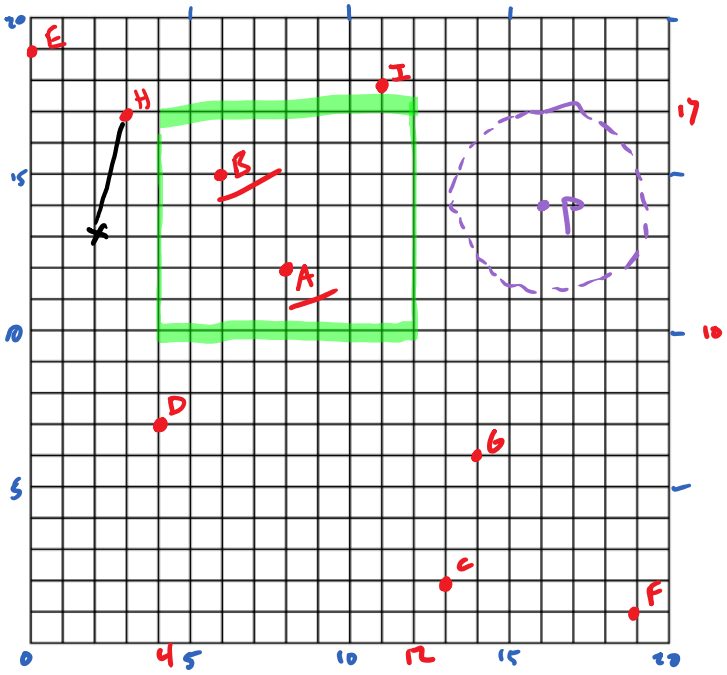
Spatial Data Structures



Spatial Data

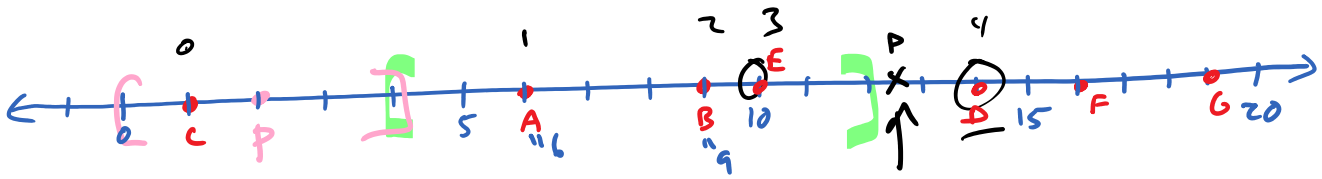
info about objects that includes location

queries: find all objects with locations in a given region

within radius r of pt p

find the object closest to a location

1 Dimension



find in range [a, b] A,B,E

find within d of p → [p-d, p+d]

[0, 4] → C

Find closest to p → D

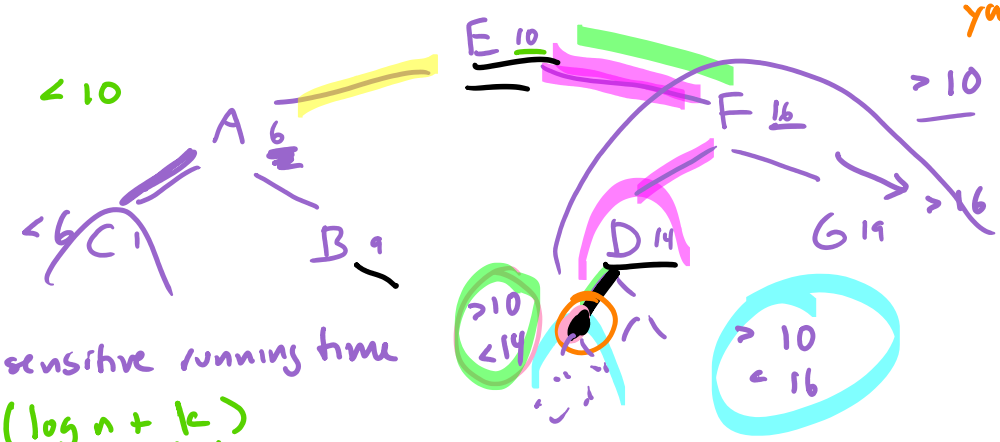
Search for p
return closer of
1) inorder predecessor
2) inorder successor
next highest value in tree
(leftmost in right subtree if there is a right subtree
otherwise, most recent ancestor you went left from
O(log n)

struct with upper/lower bound

find_range(n, r, a, b) → bounds on things in n's subtree

if n = NULL or no intersection between r, [a,b]
return

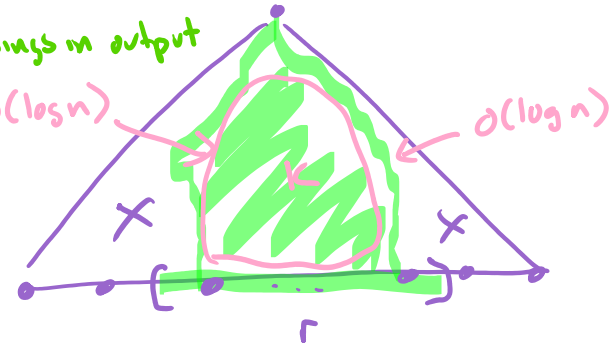
else if n->data in r, process
find-range(n->left, r, a, n->data)
find-range(n->right, r, n->data, b)

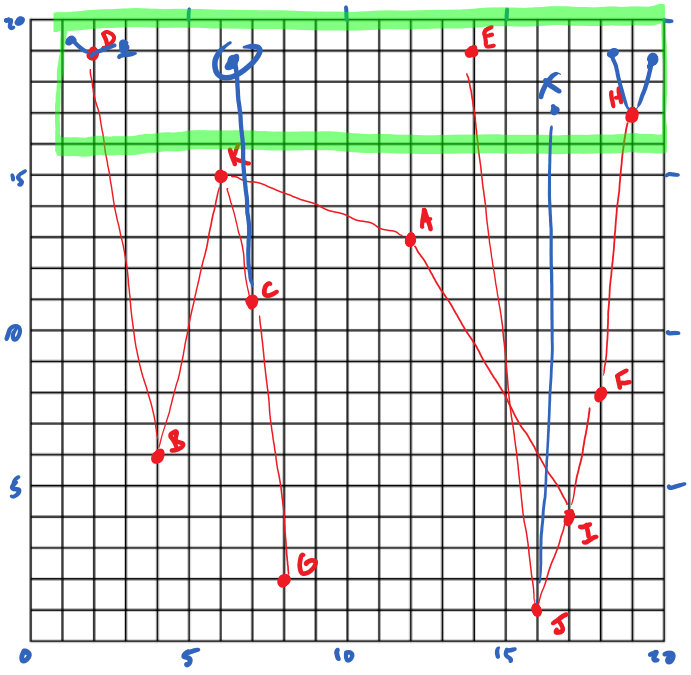


output-sensitive running time

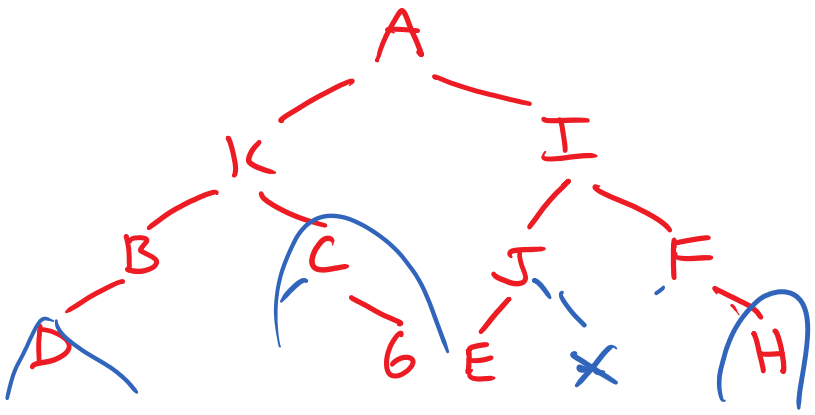
O(log n + k)

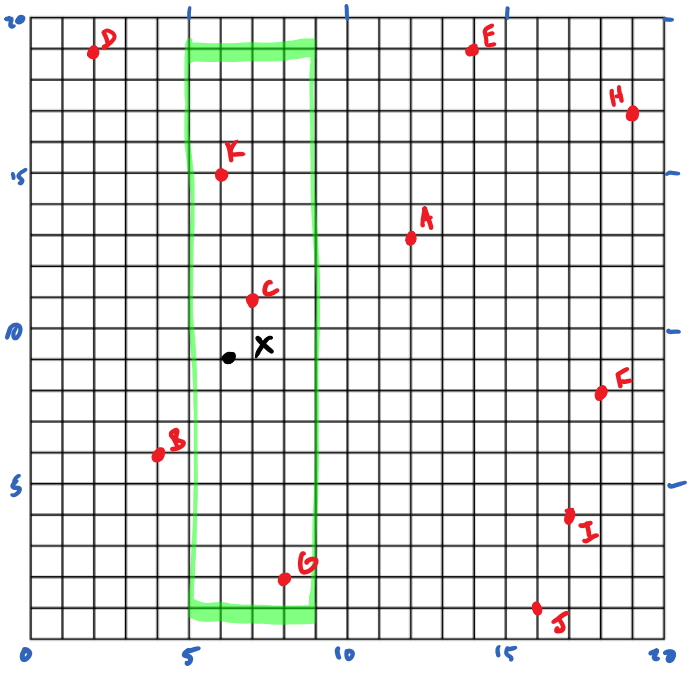
k = # of things in output



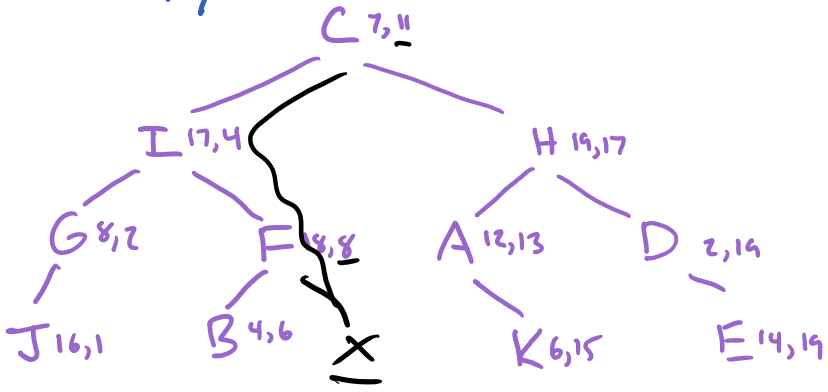


Idea: BST ordered by x-coord?





Order by y?



kd-Tree

k=2 for (x,y) coords

kd tree: 0 1 nodes at level l obey BST order property for

dimension 0 if l is a multiple of k

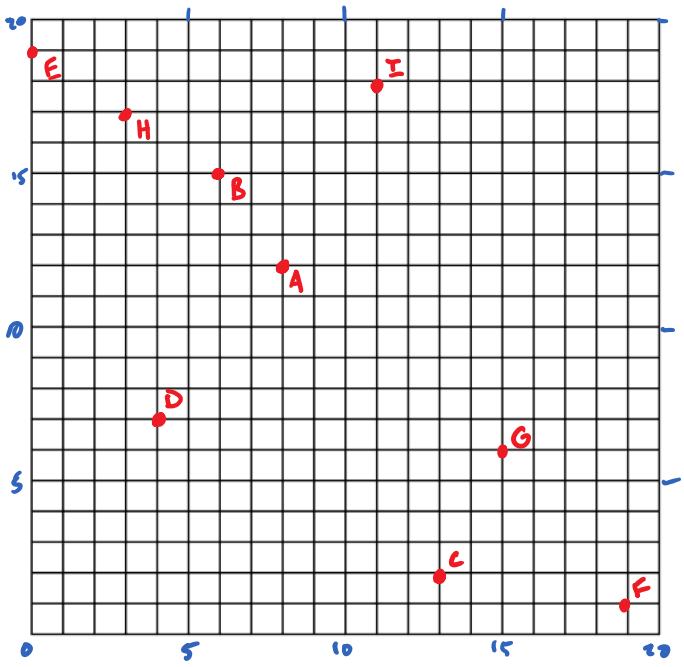
dimension 1 if l is one more than mult of k

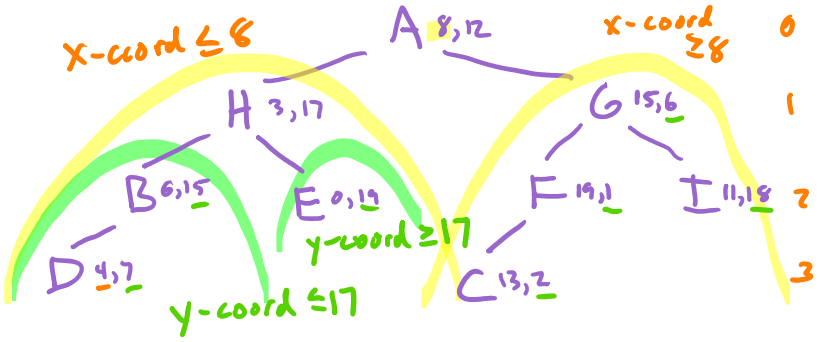
dimension 2 if l is 2 more than ...

⋮

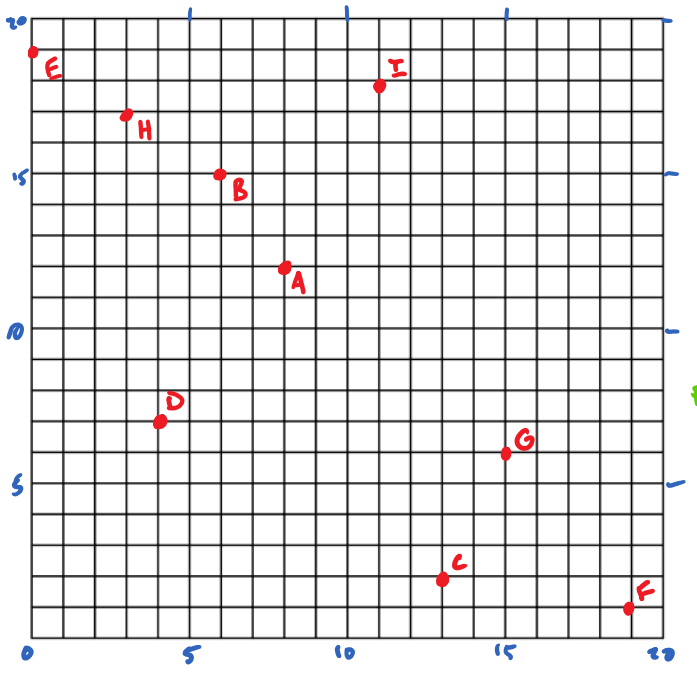
x at even levels

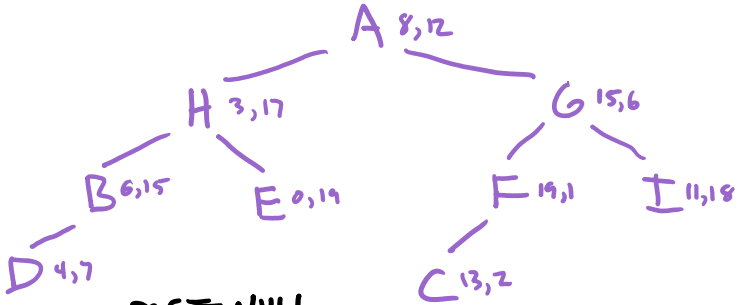
y at odd levels





Adding to kd-Tree





par = NULL
curr = root
dim = 0

iterations ≤ h

while curr ≠ NULL and curr->data ≠ p

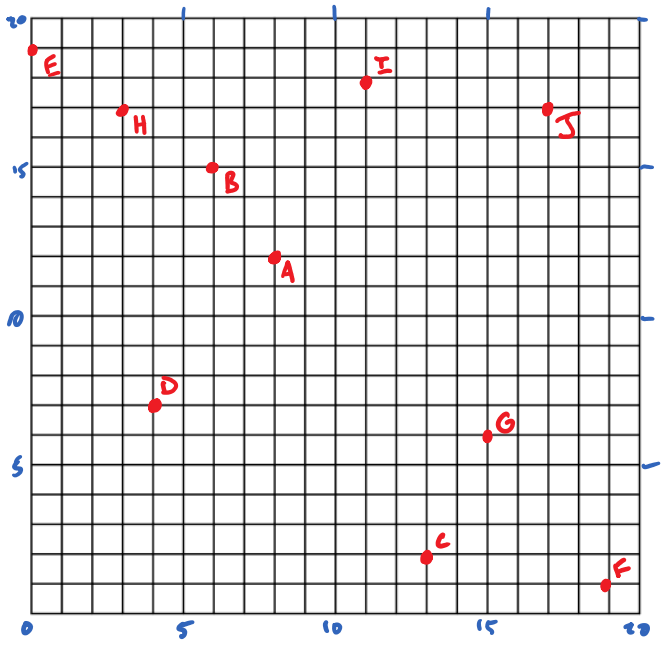
O(1)

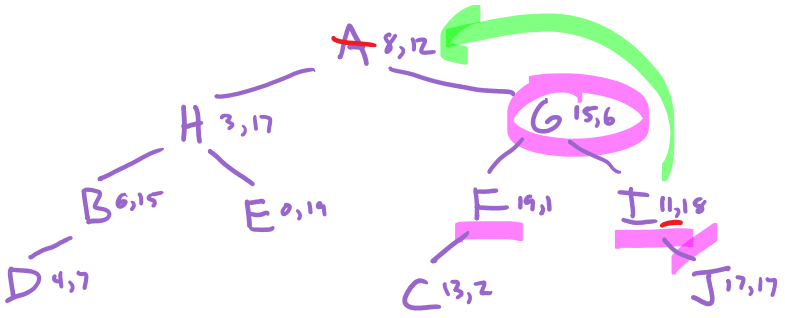
par = curr
if p[dim] < curr->data[dim]
    curr = curr->left
else if p[dim] > curr->data[dim]
    curr = curr->right
dim = (dim+1) % k

if curr = NULL
    add new node as appropriate child of parent

O(log n) if balanced

Removing From kd-Tree





find node n containing p, and its parent, cut dimd

delete (n, parent, d)

if n is a leaf, remove

if n→right ≠ NULL,

find node n' w/ lowest value in dim d in n's right subtree and find its parent p', cut d'

n→data = n'→data

delete (n', p', d')

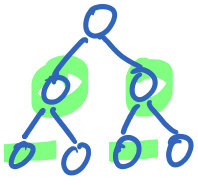
else

find node n' w/ highest value in dim d in n's left subtree (and its parent p', cut dim d')

n→data = n'→data

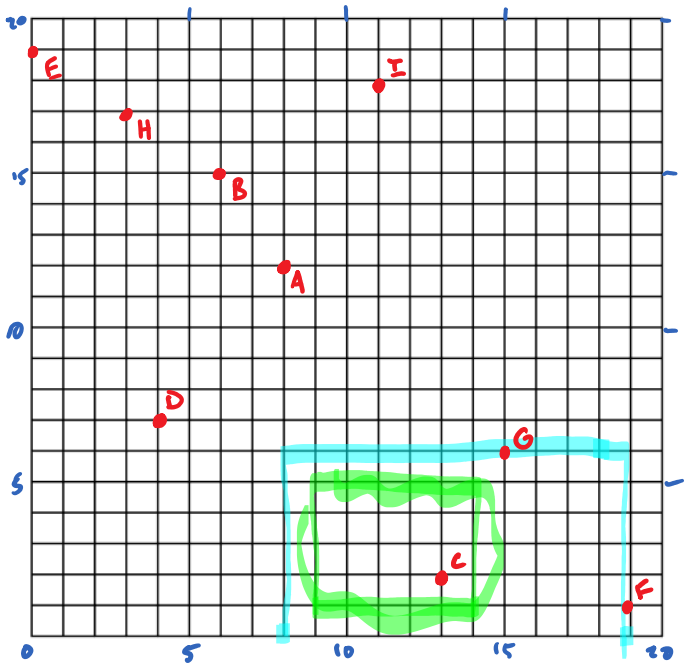
delete (n', p', d')

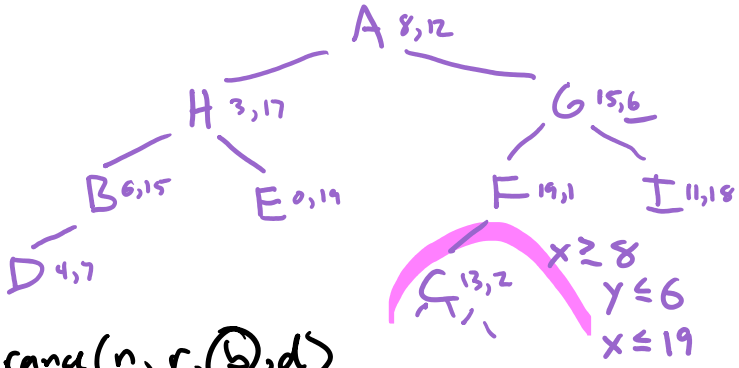
```
find_min(n, d)
    if n == NULL return NULL
    if n's cut is dim d
        if n→left ≠ NULL
            return find_min(n→left, d)
        else
            return n
    else
        return min(n, find_min(n→left, d),
                   find_min(n→right))
```



$O(\sqrt{n})$

Range Query in kd-Tree






```
range(n, r, b, d)
  if n == NULL or r ∩ b = ∅
      return
  if n->data in r, process
  range(n->left, r, b ∩ dim ≤ n->data[dim], (d+1)%k)
  range(n->right, r, b ∩ dim ≥ n->data[dim], (d+1)%k)
```

$O(\sqrt{n} + k)$ if balanced