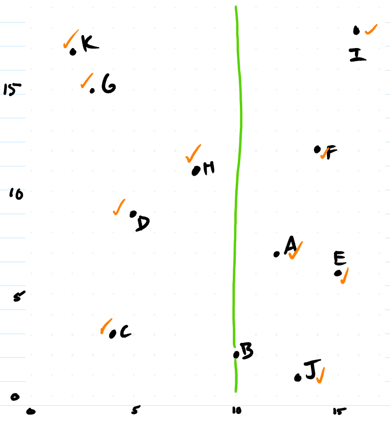
kd-Trees

```
build balanced   preprocessing   sort by x  to get list X        O(n log n)
                                 sort by y  to get list Y
                                 build (t->root, X, Y)

                 build (X, Y)
                    find the median in the cutting dimension
                    make that the root of the current subtree
  ∴  O(n^2)   O(n) [ split X into X_L, X_R   -> points in right subtree sorted by x
                   |                          -> points in left, sort on x
                   [ split Y into Y_L, Y_R   -> points in right subtree sorted y
                                              -> points in left, sort on y
                    build (t->left, X_L, Y_L)
                    build (t->right, X_R, Y_R)
```



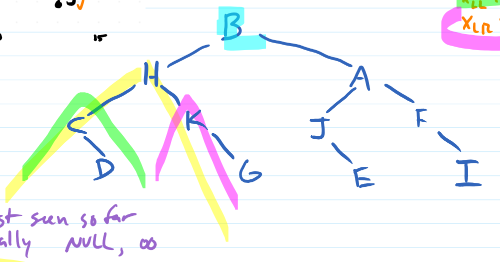
X = KGCDHBAJFEI
Y = JBCEADHFGKI

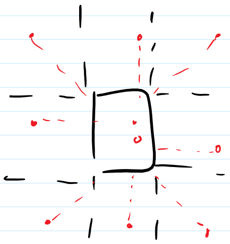
X_L : KGCDH
X_R : AJFEI

Y_L : CDHGK
Y_R : JEAFI

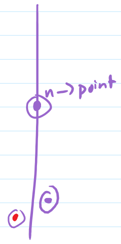
Y_{LL} : CD
Y_{LR} : GK

X_{LL} : CD
X_{LR} : KG

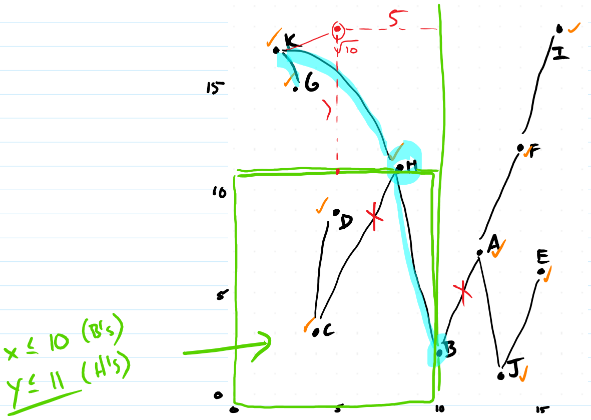


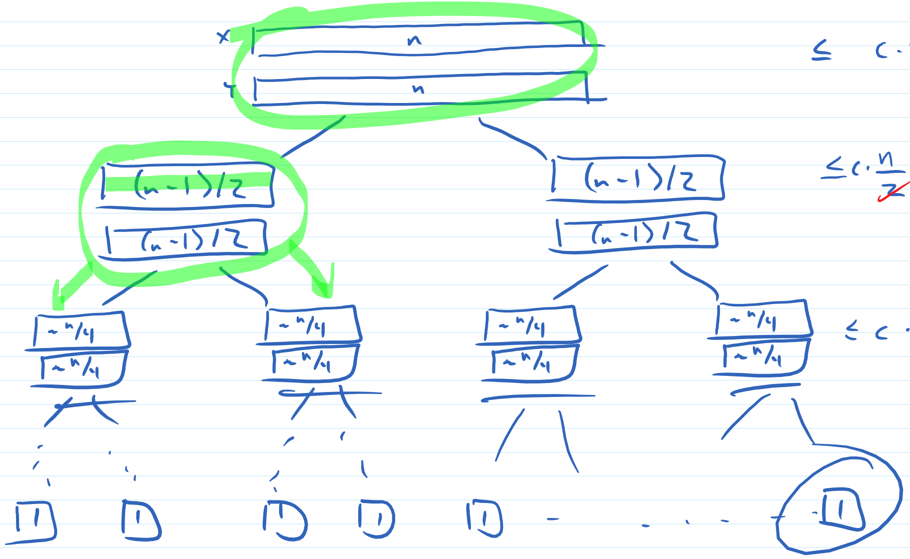


```
          (current node, initially the root)
             |   query point
             v   v
nearest (n, p, nearest, d)
                  -> closest seen so far
                     initially NULL, ∞
    if n == NULL or d == 0
        return
    if closest dist from p to region bounding n's subtree ≥ d
        return
    if |p - n->point < d|
        nearest <- n->point
        d <- |p - n->point|
    if p on left
        nearest (n->left, p, nearest, d)
        nearest (n->right, p, nearest, d)
    if p on right
        nearest (n->right, p, nearest, d)
        nearest (n->left, p, nearest, d)
```



total work @ this level





$\leq c \cdot n$ $\quad$ $c \cdot n$

$\leq c \cdot \frac{n}{\cancel{2}} \cdot \cancel{2}$ $\quad$ $c \cdot n$

$\leq c \cdot \frac{n}{\cancel{4}} \cdot \cancel{4}$ $\quad$ $c \cdot n$

TOTAL $\quad$ $c \cdot n \cdot$ (# levels of recursion)

$= c \cdot n \cdot \log_2 n$

$O(n \log n)$

Red-Black Trees

Binary Search Trees such that

1) each node has a color: red or black

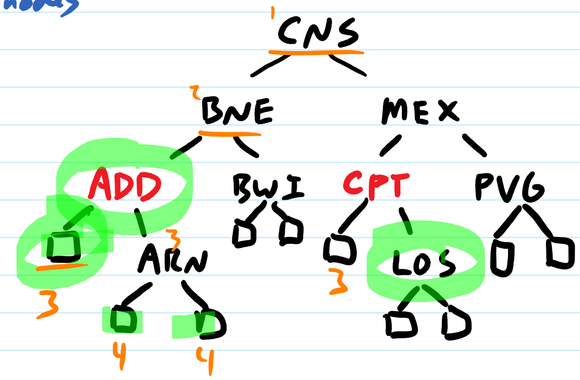
2) root is colored black

3) all leaves are colored black

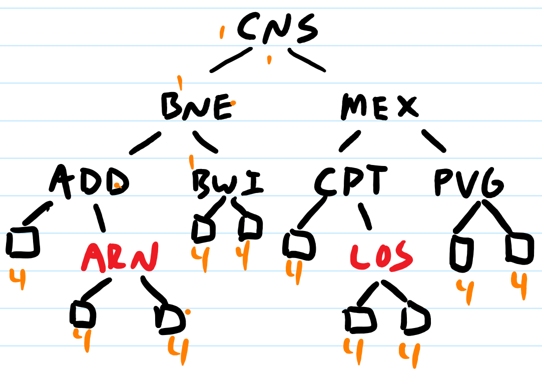
4) children of red nodes are black

5) every path root → leaf has same # black nodes

add empty leaves as "missing" children of all data nodes



not a red-black tree



a valid red-black tree

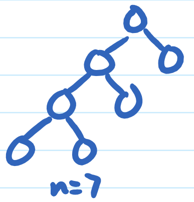
→ 0 or 2 children per node

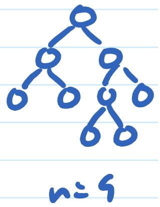
leaves in a non-empty full binary tree = # internal nodes + 1
(non-leaves)

if height is $O(\log_2 \text{total nodes}) = O(\log_2 2 \cdot \text{data nodes} + 1)$

$= O(\log_2 \text{data nodes})$

(black height)

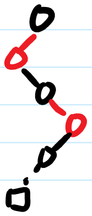


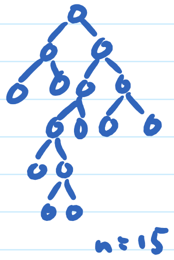


suppose # black nodes in path root → leaf is

shortest possible path has nodes

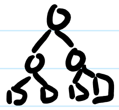
longest possible path has

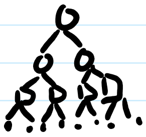




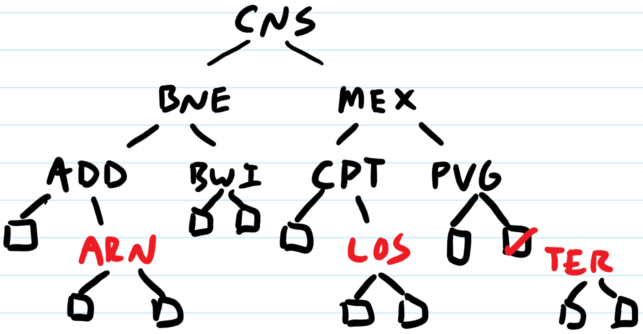
Red-black tree with black height b has ≥ $2^b - 1$ nodes

$n \geq 2^b - 1$





$n+1 \geq 2^h$

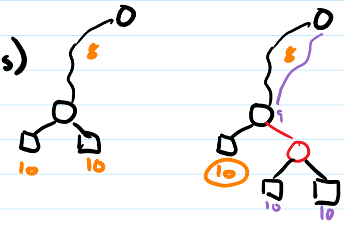


Insert

1) Do normal BST insert, color new node red (w/ black leaves)

2) If parent of new node exists and is black, DONE

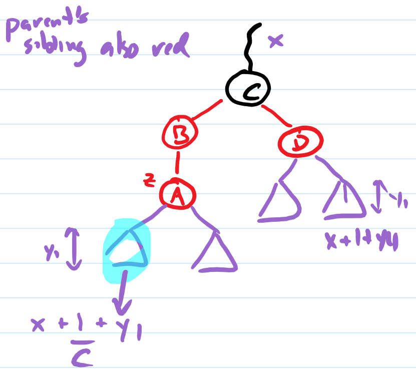
Else



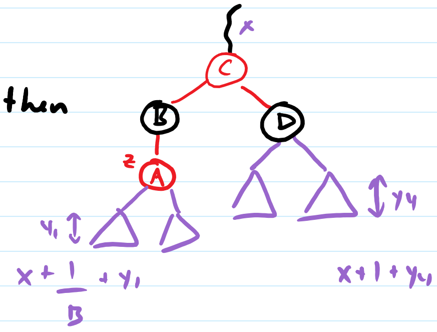
let z = new node

while z is not root and z -> parent -> color = red

if

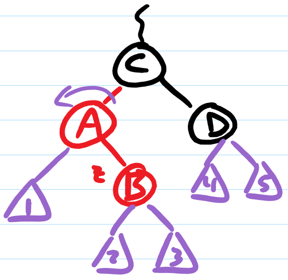


then

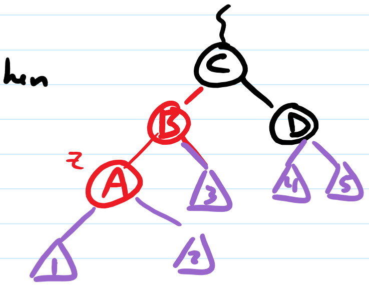


else

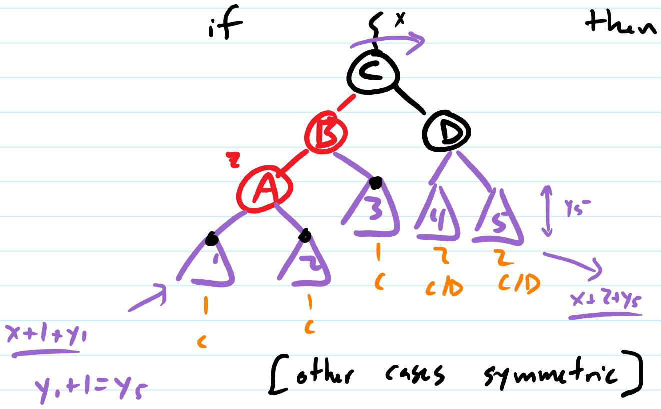
if



then

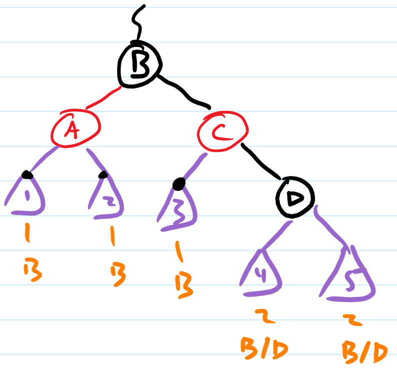


if



[other cases symmetric]

then



DONE

t -> root -> color = black