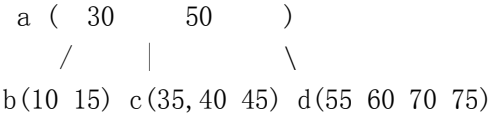
Sample solution

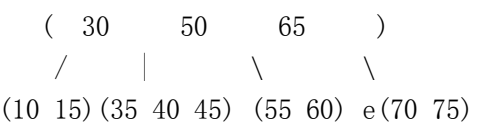
Final Exam, Fall 2002

1.

a)
After insert 75.



After insert 65



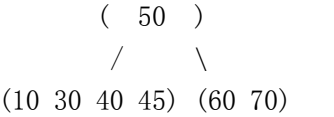
b)
We have the same assumption as in text that there is enough
internal memory to hold all nodes accessed during an operation.
Insert 75: 3 disk accesses
(2 to read a and d, and 1 to write updated node d)
Insert 65: 5 disk accesses
(2 to read a and d, and 3 to write nodes d, e and a)

c)

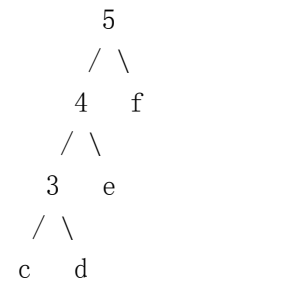
delete 15.

delete 55.

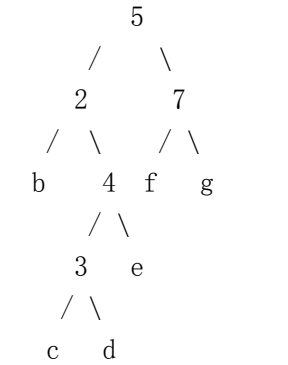
delete 35.



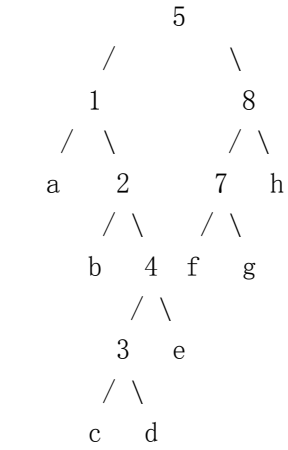
2.
Part 1:
After RR Rotation.



After LR Rotation



After RL Rotation.



Part B:

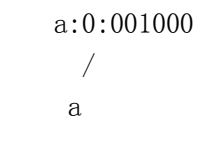
after searching for a key i., create a new node q, set q -> key
to some value other than i to indicate that there is no record in a tree
with key i, and then insert q into the search external node.
(i.e., id ther search stoped where p->LeftChild = 0,
insert q into p->LeftChild).

Because of splays, q will becaome the new root of a tree
and it easa splict.

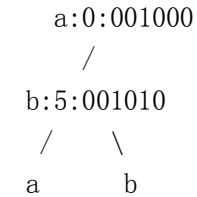
3.

node: <label>:<bitnumber>:<key>.

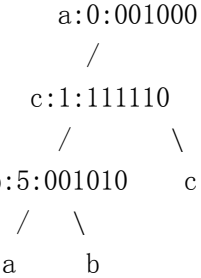
After inserting 001000:



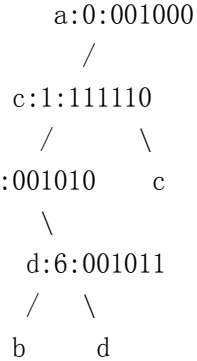
After inserting 001010:



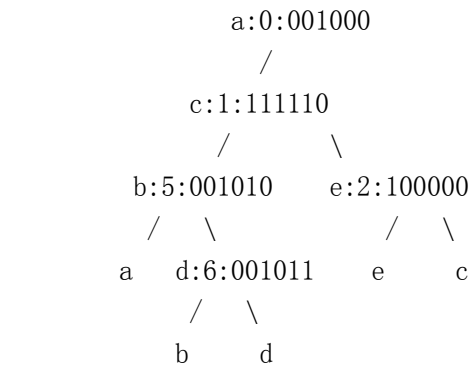
After inserting 111110:



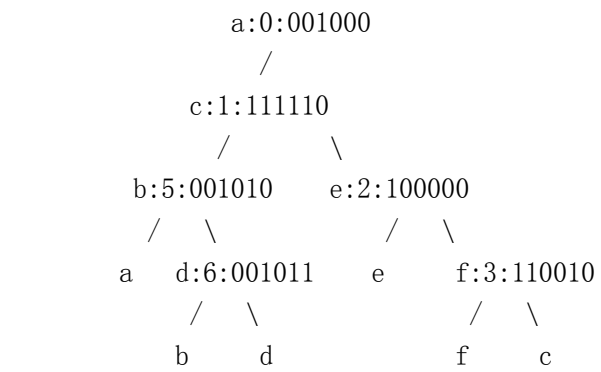
After inserting 001011:



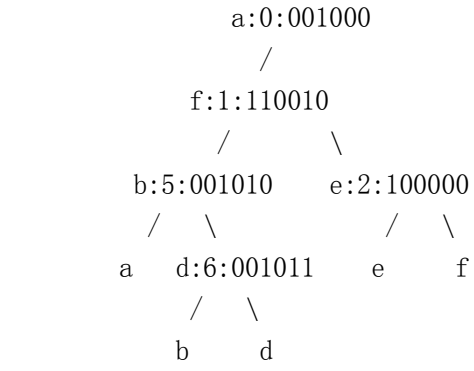
After inserting 100000:



After inserting 110010:



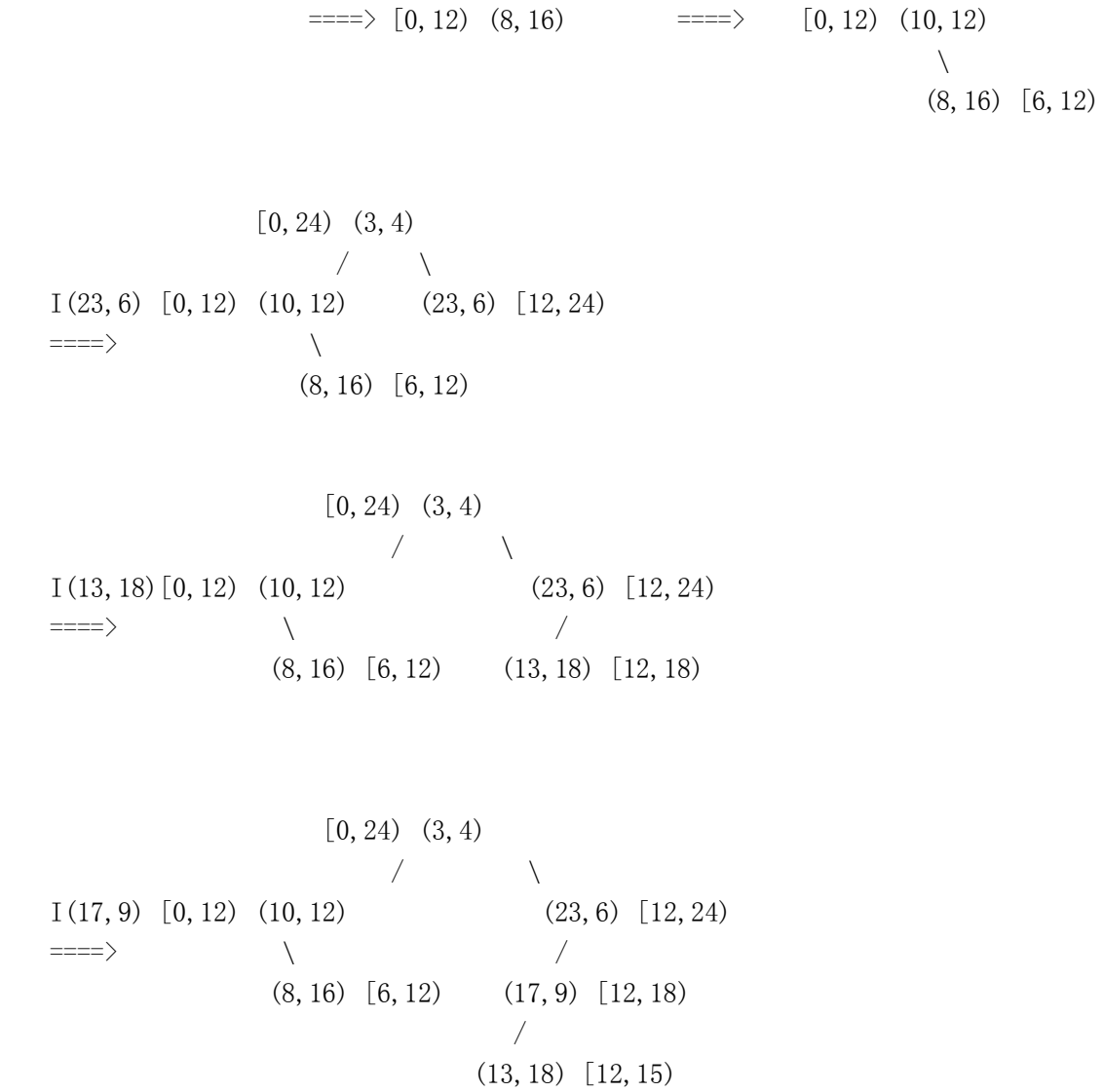
After deleting 0101:



4.

(10,12) [0,24) [0,24) (10,12) [0,24) (3,4)

I (8,16) / I(3,4) /



5.
(a)

(a) If the pixel [i,j] is to the NW,NE, SW, or SE of current partitioning line, go to NW, NE, SW, or SE child. repeat this until a leaf node is reached. This leaf node is the corresponding node to the pixel.
You can initialize all the leaf nodes of the quadtree using the scheme in the question (a). This takes O(NlogN) time where N(n^2) is the number of pixels in the image.
Then, all internal nodes can be initialized by just traversing in post-order
using black, white, and gray colors which takes O(N).
So, the total time complexity is O(NlogN).
While traversing the tree, you can add the number of pixels in a node if the node stands for white. The number of pixels in the node can be easily computed using tree level (that is, the number of pixels in a node

is 4^l where l is tree level).
Of course, if current node is gray, you'll need go down until white node.

(b)
```
find_points(kd_node node, int x, int y)
    if (node == NULL) return;
    if (node.x > X and node.y > Y)
       output node;
if ( x.discriminator( node ) )
    {
   if (node.x < X)
          find_points(node->right, x, y);
   if (node.x > X)
       {
       find_points(node->right, x, y);
       find_points(node->left, x, y);
   }
}
if ( y.discriminator( node ) )
    {
   if (node.y < Y)
          find_points(node->right, x, y);
   if (node.y > Y)
       {
       find_points(node->right, x, y);
       find_points(node->left, x, y);
   }
}
```