4/3

The algorithm is recursive. It starts by coloring the root black.

Then it recursively colors the left and right subtrees. It does so

in such a way that the following property holds.

If the height of a subtree is h, it is colored so that the number

of black nodes on every path from its root to a null pointer is

h/2 + 1 (computed in integer arithmetic). So,

height #black nodes on paths

0 1

1 1

2 2

3 2

4 3

5 3

6 4

7 4

etc.

Clearly, this is easily done if the height is 0. Just color the root

black. Suppose the height of a root node is h > 0. The algorithm

recursively colors the left and right subtrees. Because it is an AVL

tree, one subtree must have height h - 1. The other could have height

h - 2 or h - 1. There are two cases to consider. If h (the height of

the root) is even, then the number of black nodes on paths should be

one greater than the number of black nodes on paths for both of the left

and right subtrees (which have heights h - 1 and h - 2). See the above

table and do the algebra to convince yourself of this. To achieve this,

just make the root black, and reference the already colored subtrees.

The other case to consider is if h is odd. In this case, the # blacks

on paths for the tree is the same as the number of blacks on paths

for subtrees with height h -1, and it is one greater than the # blacks

on paths for subtrees with height h - 2. (Again, look at the table

above and play with the math to convince yourself of this). To achieve

this, the root is still colored black, and already colored subtrees

with height h - 2 are referenced unchanged. But, height h -1 subtrees

are referenced, but with the color of their root changed from black to

red. This reduces by one the # blacks on paths by 1 in the subtrees

with height h - 1, which is what we need to get all black path lengths

the same in the tree. Now, can we safely change the root of a subtree

of height h - 1 to black. Yes, because if h is odd, then h -1 is even,

and in coloring that subtree, its children (if there are any) would

be black, because they would not have had their color changed.

So that is a description in words, and below is the algorithm:

public void AVLColor (AVLNode root)

{

/\* If root is null, nothing to do. \*/

if (root == null)

return;

/\* Colors the AVL tree rooted at the node root. \*/

root.color = black;

AVLColor (root.left);

AVLColor (root.right);

if (h % 2 == 1)

{

/\* height of tree rooted at root is odd. Change color

to red of roots of even height subtrees. \*/

if (root.left != null && root.left.height % 2 == 0)

root.left.color = red;

if (root.right != null && root.right.height % 2 == 0)

root.right.color = red;

}

}

**4/5 a**

A (2-4) Tree can have either 1, 2 or 3 keys at any particular node.

Also if we carefully think about it then the maximum height occurs when the tree is a perfectly binary tree that is the there is a single key at every node

Now a function which expresses the number of nodes in a perfect binary tree in terms of its height is

F(n) = 2h -1

So now the given number of nodes in the question is 106

So solving the equation

2h -1 = 106

we get

h = 9 (rounded of value)

Also now for the minimum height case the number of keys in each node should be 3.

then the function that would give us the total number of nodes in terms of the height of the tree is

F(n) = 4h – 1

Also as was before the total number of nodes is 106

So solving the equation

4h – 1 = 106

We get , h = 19(rounded off value)

**b**)  
The minimum number of keys that have to be inserted so that atleast >1 split operations are performed is 7. Please Note that this includes the last key inserted due to which the split operations were performed

**4/5 c )**

No we cannot do this for B+ trees because

Because all the leaf nodes are interlinked