HW #3

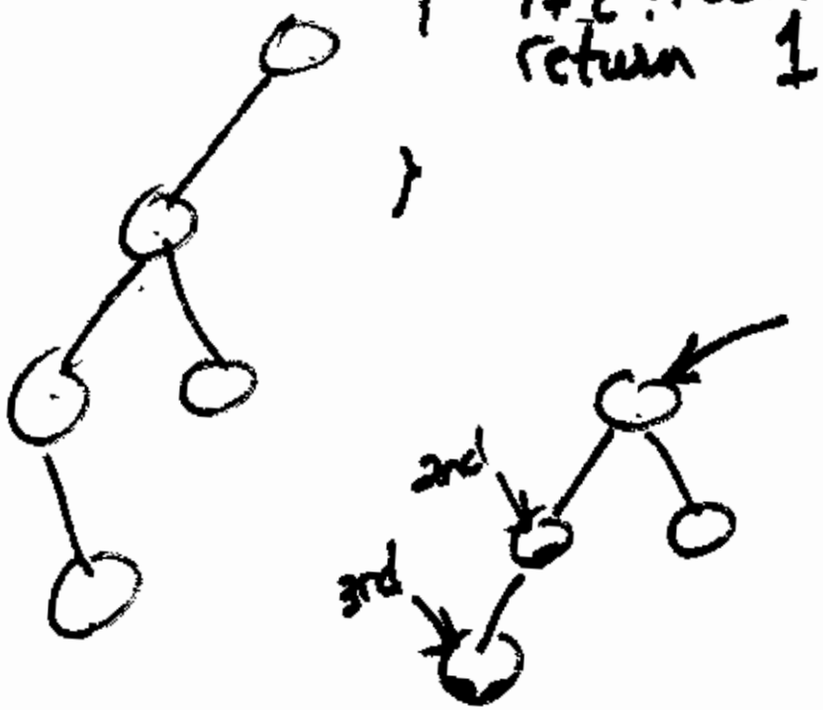
* Write a Function to Count the # nodes in a BST

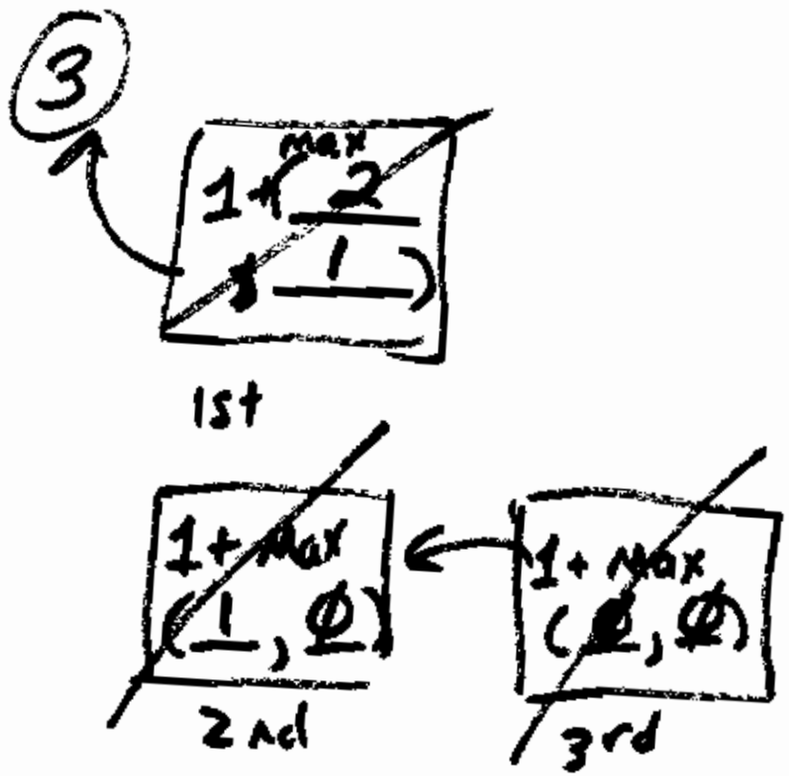
```
int   Count (node *  root)
{  if (!root)  return 0;
   return  1 + count (root->left) + count (root->right);
}
```

```
void (ount (node * root, int & cntr)
{   if (root)          //assumes calling routine sets cntr to zero
    {
       ++ cntr;
       count (root->left, cntr);
       count ( root->right, cntr);
} }
```

* Determine the Actual Height

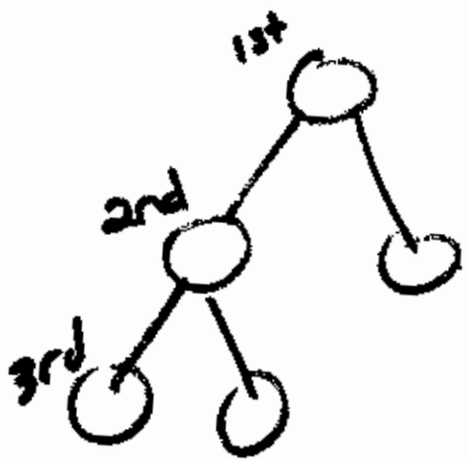
```
int Height ( node * root)
{   if (!root) return 0;
    return  1 + MAX ( Height(root->left),
                      Height(root->right) );
}
```

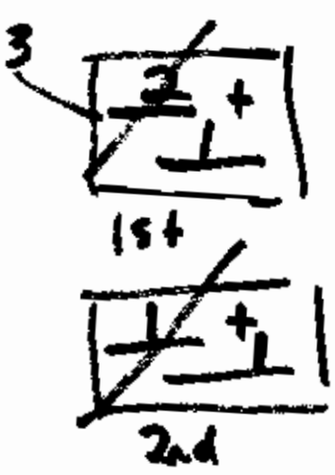


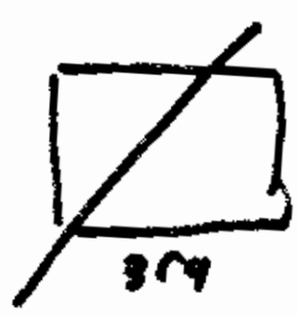


Count the nodes with no children

```
int countL (node * root)
{   if (!root) return 0;
    if (!root->left && !root->right)
        return 1;
    return countL(root->left) + countL(root->right);
}
```







Make a Complete Copy of a BST

```
Struct node
{     char * name;
      node * left;
      node * right;
};

  void  copy ( node * & newtree, node * old)
  {     if (!old)  newtree = NULL;
        else
        {     newtree = new node;
              newtree -> name = new char [strlen(
                                    old -> name ) + 1 ];
              strcpy ( newtree->name, old -> name);
              copy ( newtree -> left, old -> left);
        3     copy ( newtree -> right, old -> right);
```

Alternate Copy

```
int copy (node * & newtree, node * old)
{   int error1, int error2;
    if (!old) newtree = NULL;
    else
    {   newtree = new node;
        if (!newtree) return 0;
        :
        return copy(newtree -> left, old -> left) +
               copy( ... -> right, old -> right);
    }
    return 0;
}
```

```
        errorL = copy(newtree->left, old->left);
        errorR = copy( ...->right, ...right);
    }   if(!errorL || !errorR) return 0;
    ← return 1
```

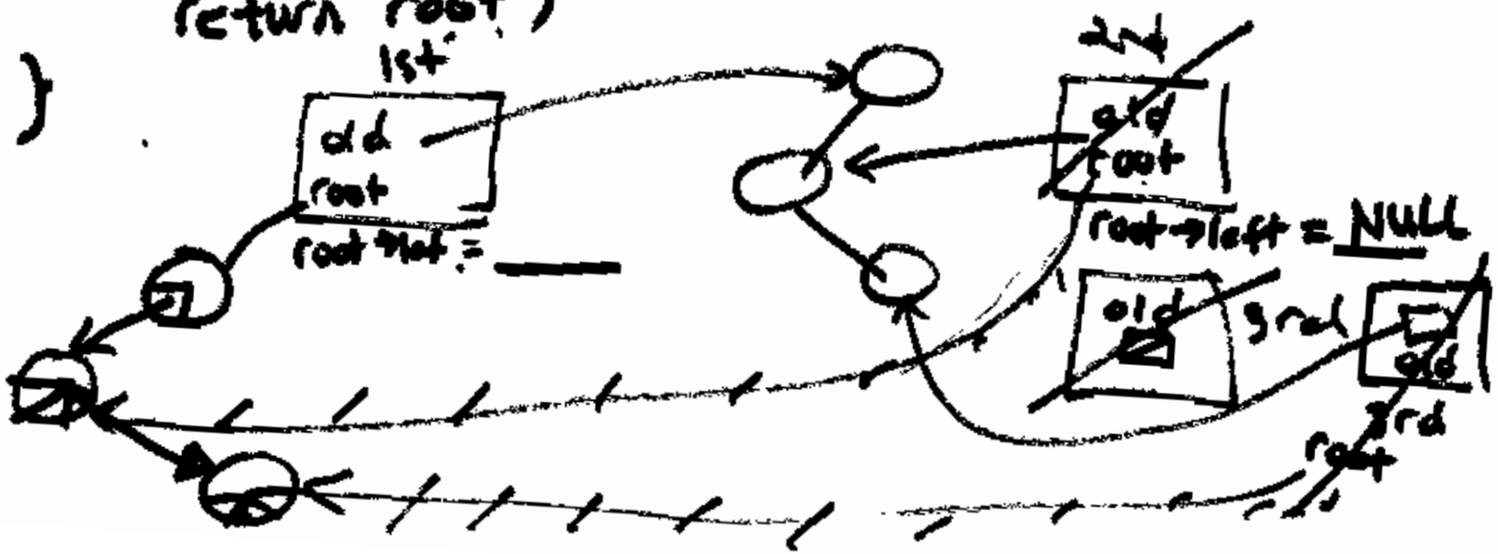
Alternate #2

Pass by value

```
node * copy (node * newtree, node * old)
{
    if (!old) return NULL;
    newtree = new node;
    newtree->name = new char [strlen(
                    old->name) + 1];
    strcpy (newtree->name, old->name);
    newtree->left = copy (newtree->left, old->left);
    newtree->right = copy (newtree->right, old->right);
    return newtree;
}
```

Alternate #3

```
node * copy ( node * old)
{   node * root;
    if (!old) return NULL;
    root = new node;
    root->name = .....
    strcpy(root->name, .....
    root->left = copy(old->left);
    root->right = copy(old->right);
    return root;
}
```

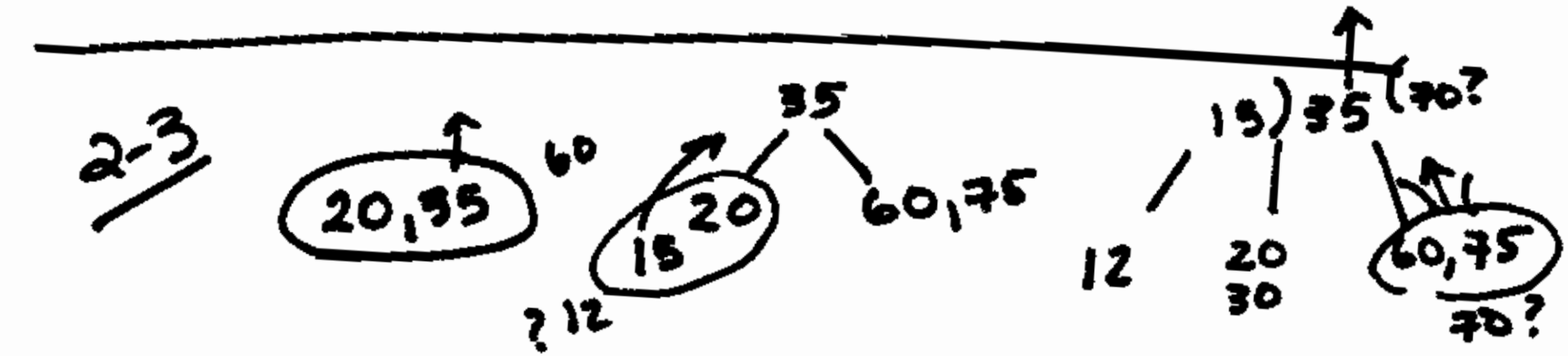


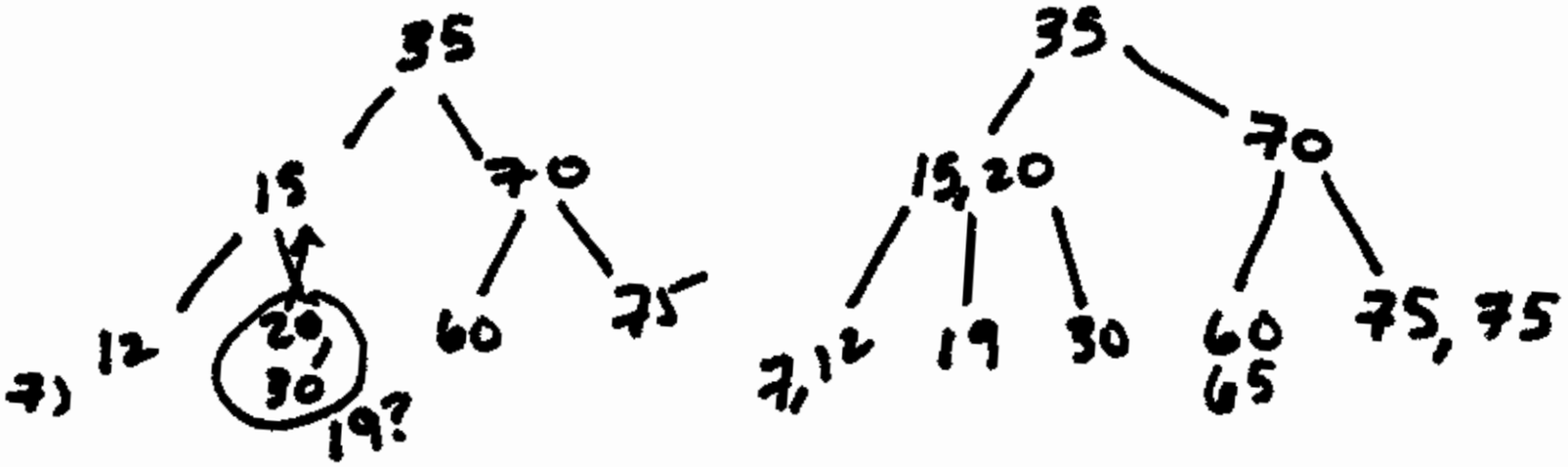
20 35 60 75 15 12 30

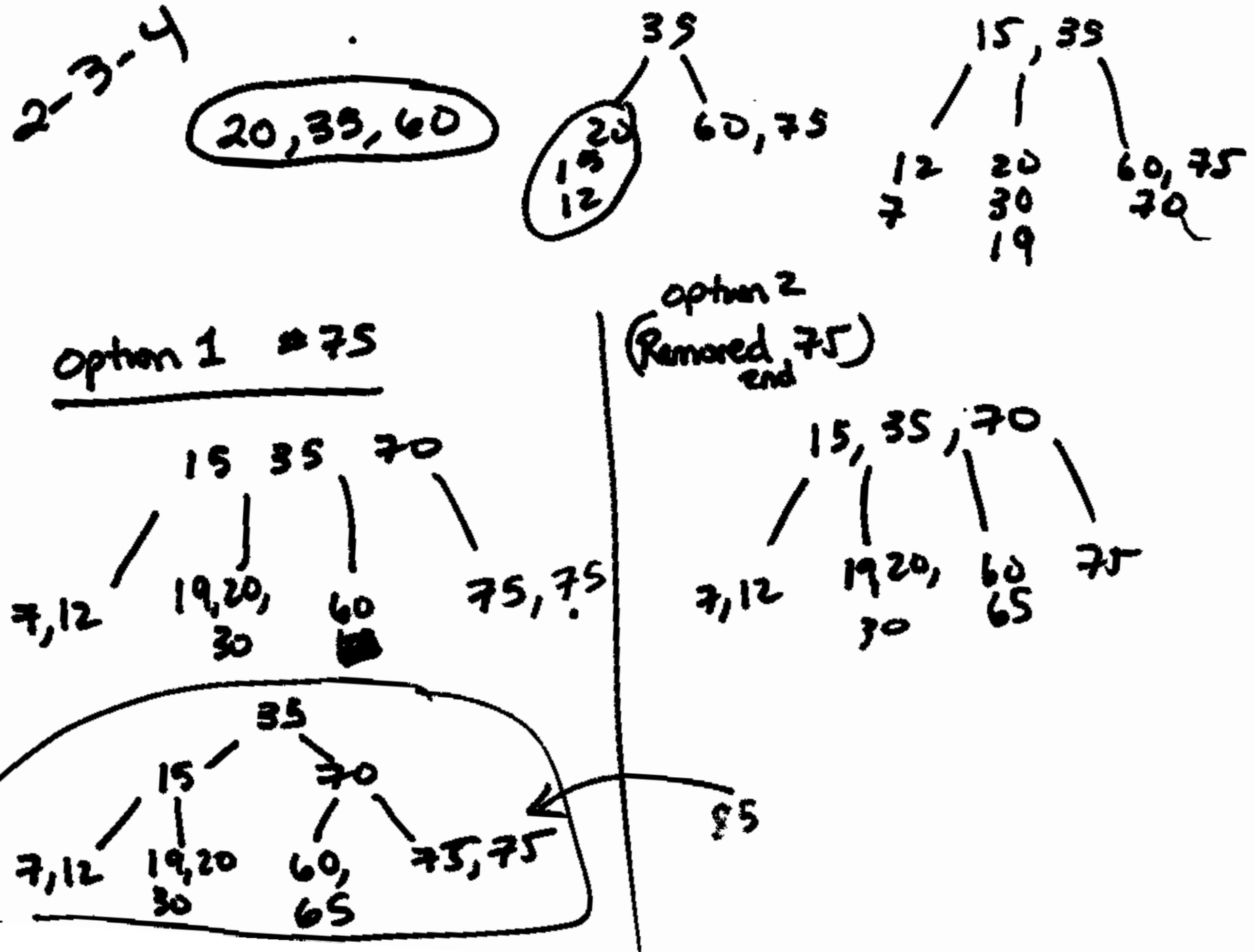
70 7 19 75 65

(85)

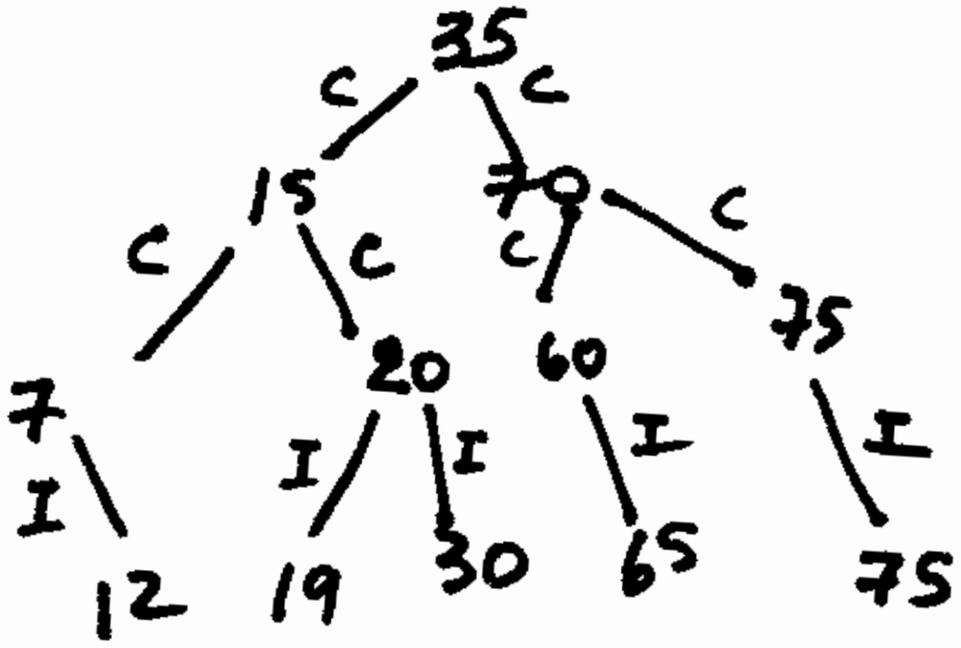
2-3



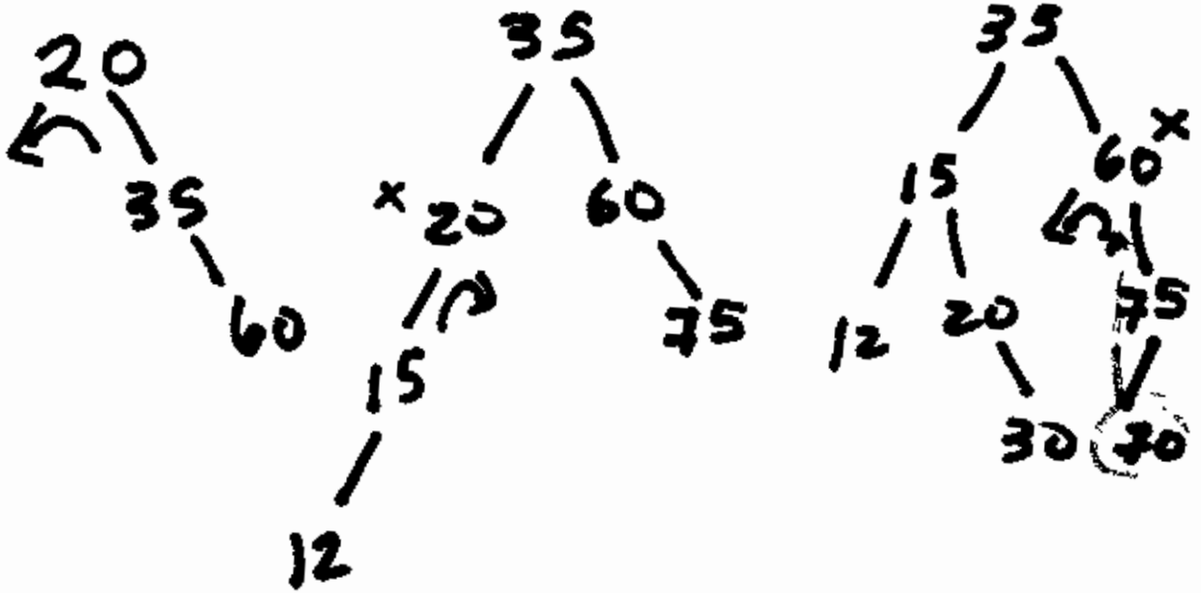


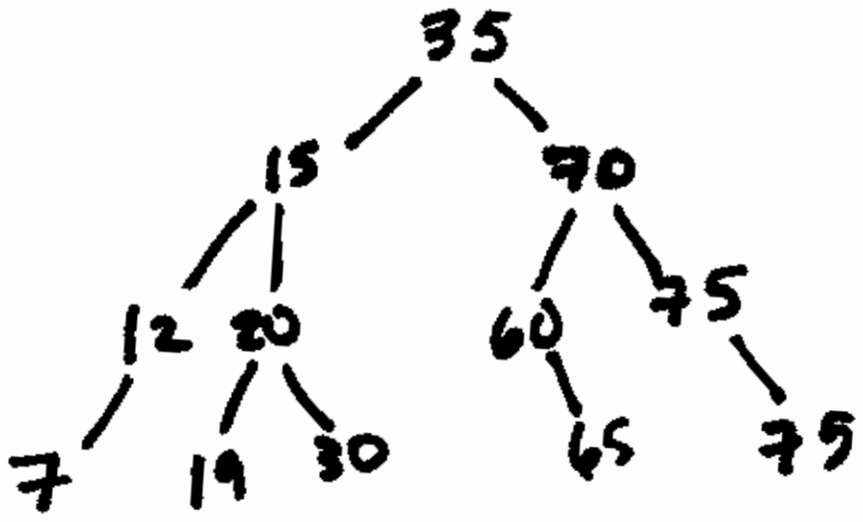

2-3-4
20,35,60
35
20
19
12
60,75
15, 35
12
20
30
19
60, 75
70
option 2
(Removed 75)
15, 35, 70
7,12
19,20,
60
75
option 1 #75
15 35 70
7,12
19,20,
30
60
75,75
35
15
70
7,12
19,20
30
60,
65
75,75
75

Red Black



AVL


20
35
60
35
x 20
60
15
75
12
35
15
60 x
12
20
75
30
70


35
15
70
12
20
60
75
7
19
30
65
79