5.1:解：设 n_1 为度为 1 的节点，n_2 为度为 2 的节点，则由题意有 $n = n_1+n_2$;又由于二叉树的性质有 $n_0 = n_2+1$;则有 $n_1+ n_0-1 = n$; $n_0 = n -n_1+1$; 当 n1 = 0 时，n0 有最大值 n + 1 即一个包含 n 个分支节点（非叶节点）的非空二叉树，它的叶节点数目最多为 n + 1，此时 n 个分支节点都是含有两个分支的结点，符合满二叉树的条件，所以在有 n 个分支结点的所有二叉树中，满二叉树的叶结点的数目（或者叶结点与全部结点数的比例）是最高的。

6.16:解：根据例 6.5 可得，给定数的线性表示为：CA/BG///FED///H/I//
代码思想：用中序遍历的方法构建书，遇到 / 就返回空节点，遍历完字符串

```
TreeNode * bulit(char * list){
    int index = 0;
    return add(list,index);
}

TreeNode * add(char * list,int & index){
    if(lisr[index] == '/'){
        cur++;
        return NULL;
    }

    TreeNode node = new TreeNode(list[index++],NULL,NULL);
    node->left = add(list,index);
    node->right = add(list,index);

    return node;

}
```

算法的时间复杂度为 $\Theta(\log N)$.

5.6 : 解：题目要求遍历一个非二叉查找数，看有没有存在给定的 key 值，主要算法思想是用递归，要遍历整棵树，就是查看当前节点然后一次遍历左右节点，如此反复，就能遍历整颗树。具体步骤：

1):先判断当前节点是否为空，为空返回 fall
2):判断当前节点 key 是否为给定的 k;如果是返回 true
3):遍历左右节点

```
bool search(Tree * root , key k){
    if(root == null){
            return false;
    }else if(root->key == k){
            return true;
    }else{
```

```
                return (search(root->left,k)||search(root->right,key));
        }
}
```

由于给定的树不是二叉查找数，所以时空复杂度为 Θ（n）。