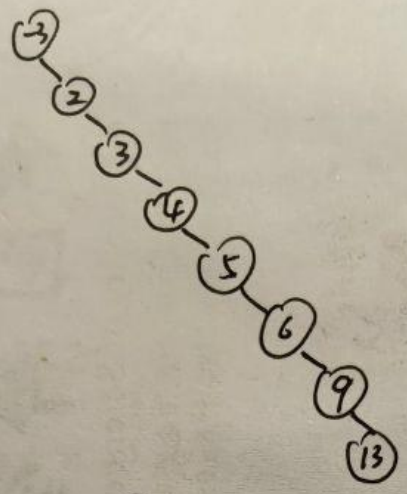
Q1. 1. 树高为n的二叉搜索树最多有(2^n-1)个结点，最少有$(n+1)$个结点

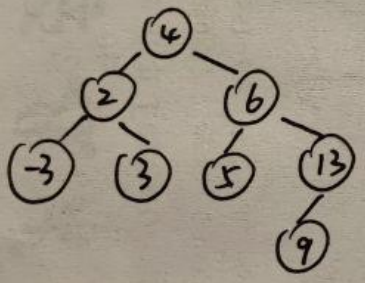
所以对于关键字集合{3,6,9,2,4,5,13,-3}

$H_{max}=7$　$H_{min}=3$

举例：

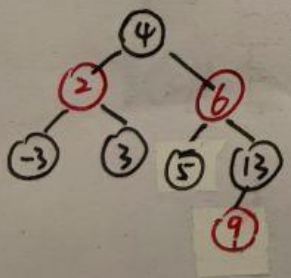


$H_{max}=7$

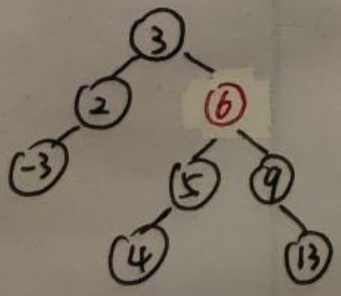


$H_{min}=3$

2.



$H_{black}=2$



$H_{black}=3$

Q2.

1. INORDERTREEWALK-LEAF:

```
if x==NIL then
    return;
if x ≠ NIL then
    INORDERTREEWALK-LEAF(left[x]);
    if(left[x]==NIL && right[x]==NIL)
        print key[x];
    INORDERTREEWALK-LEAF(right[x])
```

2. TREE-SEARCHk(x, k, Spath, leftnei, rightnei)

```
if  k==x.key
   {Spath ∈ x  //x加入Spath集合
    return Spath, leftnei, rightnei
   }
if  k< x.key
   {Spath∈ x.key
    leftnei∈x
    return TREE-SEARCHk(x.left,k,
                        Spath, leftnei, rightnei)
    }
if  k > x.key
   {Spath∈ x.key
    rightnei ∈ x
    return TREE_SEARCHk x.right, k,
                        Spath. leftnei, rightnei)
    }
```

```
TREE-STOREKEYS (x, set)
 if x!=NIL
     StoreKEYS (x.left, set)
     set ← x.key
     storekeYS (x.right, set)
```

TREE_SEARCHK 进程，递归期间经过遇到的结点构成二叉树上一条从根结点简单向下的路径，时间复杂度为 O(h)，其中 h 是树高.

```
TREESEARCH_SET (x, k)
{  Spath, leftnei, rightnei = TREE_SEARCHK (x, k, S1, S2, S3)
   Spath = S1, S4 = { }, S5 = ( )
   for i in len(S2)
       TREE_STOREKEYS (S2[i].left, S4)
   for j in len(S3)
       TREE_STOREKEYS (S3[j].right, S5)
   Sleft = S4
   Sright = S5
}
```

算法解释：TREE-SEARCHK 的作用是遍历树，找到在路径上遇到比 k 的关键字大的节点，就把它放入 leftnei 集合，并把对应关键字放入 Spath；遇到比 k 关键字小的节点，则把它及其关键字分别放入 rightnei 和 Spath。

y of Science and Technology of China
徽 合肥市金寨路96号 邮编：230026
3602184 传真：0551-63631760 Http://www.ustc.edu.cn

if k==...

TREE-STOREKEYS的作用是把以x为根结点的二叉树上的所有结点对应的关键字记录下来，存入Set集合。

~~TREE-SEARCH~~

TREE SEARCH_SET (x,k) 的作用是：调用 TREE-SEARCHk函数，得到以根结点到关键字为k的结点的路径上所有结点的关键字集合 Spath，和包含了所有~~在~~此路径在该结点处选择左子树的所有结点集合 rightnei，所有此路径在该结点处选择右子树的所有结点集合 leftnei。

对于 rightnei 中的每个元素的左孩子，调用 TREE-STOREKEYS 函数，求出以此左孩子为根结点的二叉树上的所有结点的关键字，保存到 S_{left}，此即为所求 Sleft。Sright 同理。

3. 时间复杂度：

TREE-SEARCHk步骤：O(h)

h为树高.

TREE-STOREKEYS 步骤：

最坏情况是~~所有关键~~除 Spath 外所有关键字都属于 Sleft 或 Sright.

故复杂度为 O(n) 其中 n 为所有结点数

~~TREESEARCH_SET(x,k)~~

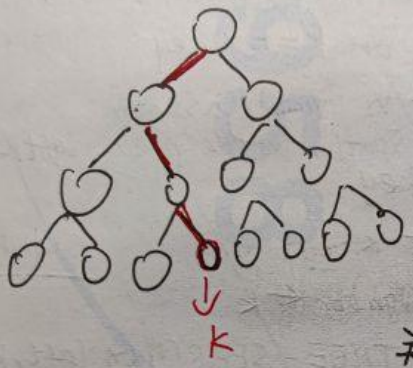
综上，时间复杂度为 O(n)，n为结点数

4. 命题 1、2 均正确，证明如下：

~~因为在 St~~

因为 Sleft 中的关键建字对应的结点可以理解为在 ~~[illegible]~~ 关键建字 k 对应结点所在的 Spath 路径中每~~[illegible]~~每一次选择了右子树作递归操作时该结点的左~~[illegible]~~孩子的根结点的二叉树上的所有结点的关键建字的总集合。

以这棵树为例：



~~由于根[illegible]~~ leftnei 中的结点均为在路径选择时选择了右子树的点，其左子树上所有结点关键建字均小于该结点关键建字，进而小于此路径上，该结点后续所有结点的关键建字。

rightnei 中的结点的关键建字同理也大于路径上在结点处选择了左子树继续递归的结点处的关键建字，进而大于此路径上，这些结点后续所有结点的关键建字。

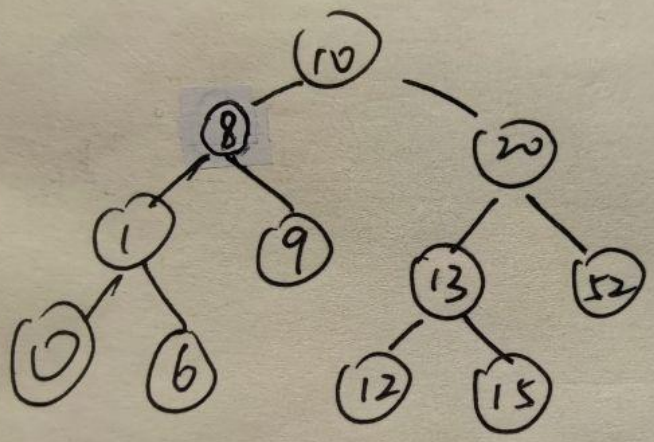
∴ k 大于路径上所有选择右子树继续递归的结点的关键建字，小于路径上所有选择左子树继续递归的结点的关键建字

∴ $\forall a \in S_{left}, b \in S_{right}$

有 $a \le k \le b$

故命题 1、命题 2 均正确

命题3错误，例如：



当k=15时，12包含在Sleft中，
而根结点10包含在Spath中
不满足 12<10
故命题3错误.

Q3.

首先证明：对任何一棵有n个结点的二叉搜索树中，恰有(n-1)种可能的旋转。

由于每个节点都可以与其父节点一起旋转，又有根节点没有父节点，因此有(n-1)种可能的轮换。

考虑将任意n-节点二叉树A旋转得到一个右向链：将根和根的所有连续右子项作为一条链的初始元素。由于根和根的所有连续右子项之外的节点数小于等于(n-1)，所以最多(n-1)次右旋可以得到此二叉树对应右向链。

对于另一棵n结点二叉搜索树B，它也可以经0 m (m≤ n-1)次右旋变成此右旋链，故将树A得到的右向链作树B m次右旋的逆序且反向旋转的操作即可还原树B。

两树旋转总次数小于2n

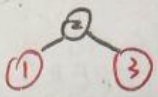
故任何一棵含n个结点的二叉搜索树可以通过O(n)次旋转，转变为其他任何一棵含n个结点的二叉搜索树。

Q4. 不一定一样，反例如下：

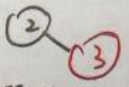
例如初始状态为：

向其中插入关键字为1的结点，

红黑树变为：



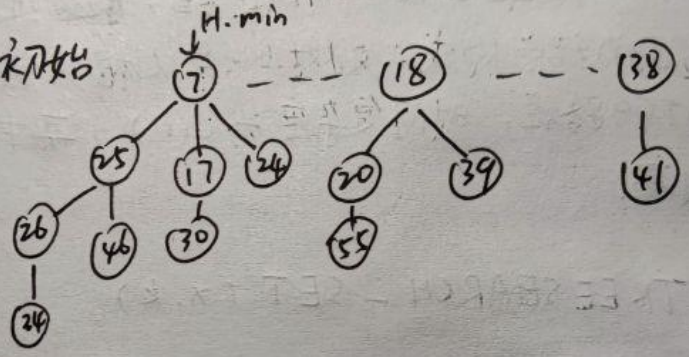
再从中删除1，红黑树变为：



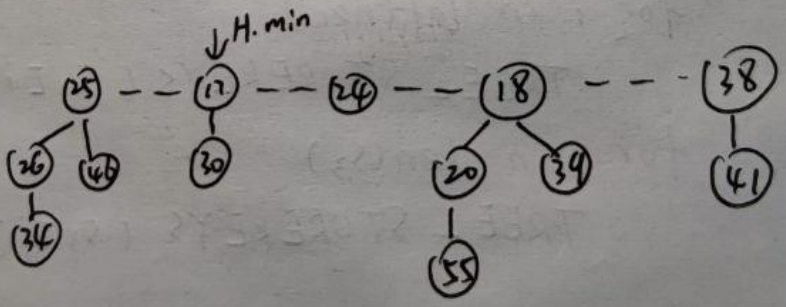
插入删除前后的红黑树不同。

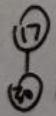
Q5.

初始



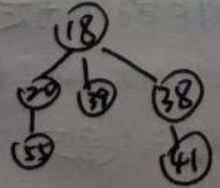
删除最小节点，并把它的所有孩子合并到堆的根结点表中，并更新min.





度为1　24度为0　18度为2

38度为1与前面重复，把38合并到18上。



29度为2，不重复

故最终结果为：