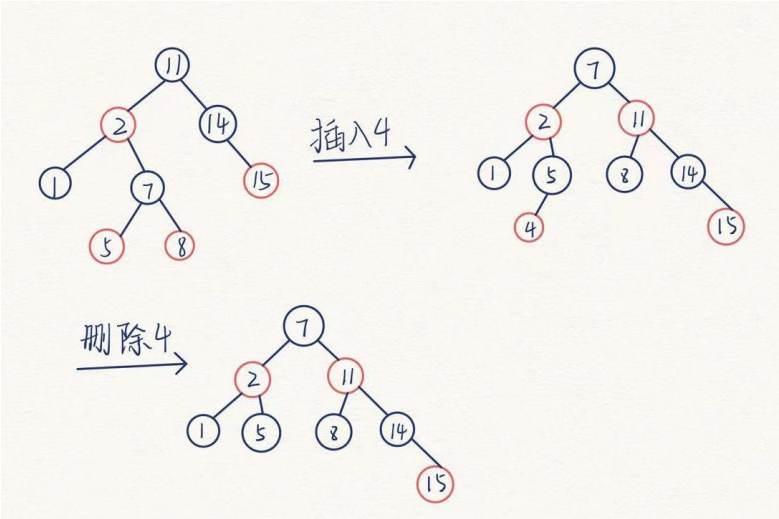
1. 选择题

5.1 可能的基准元素是 4，5，9

5.2 c

2. 简答题

（1）可能会改变，比如下图的例子就改变了其结构。



不改变结构的例子：只有一个根节点（黑色）的树，插入再删除后结构不变

（2）

排序算法	复杂度	优点	缺点	适用情况
冒泡排序	O(n^2)	简单易实现，不需要额外空间	效率低	不追求性能的简单情况
选择排序	O(n^2)	不需要额外空间	不稳定，效率低	不要求排序稳定性且不追求性能时
插入排序	O(n^2)	效率比选择排序、冒泡排序略高，稳定	效率仍然比较低	同冒泡排序
希尔排序	O(n*logn)	效率比前几个高，且不需要额外空间	不稳定，并且 d 的取值没有确切方法，只能依靠经验	适合数组规模较大的情况
快速排序	O(n*logn)	排序速度快，效率高	不稳定，且不适合初始序列基本有序的情况	是用于大多数情况，但数据规模较大时性能优势更明显
归并排序	O(n*logn)	效率比较高	需要占用较多的存储空间	适合数据量大，且对稳定性有要求的情况
堆排序	O(n*logn)	效率高	需要建堆和维护，不适合小规	适合大规模数据

			模序列	
基数排序	O(n*k)	正整数变化很大时优于计数排序	只针对整数，不能对小数排序	对整数进行排序的情况
计数排序	O(n+k)	对较小区间内的正整数排序效率很高	必须是正整数且数字不能太大	对小区间内整数排序的情况
桶排序	O(n+k)	线性时间复杂度，效率高，稳定	需要占用大量空间	适合数据非负且比较集中的情况

3. 算法设计题

思路：中序遍历的第 k 个数就是整个序列中第 k 小的数；设置一个 index 指示位置，递归进行中序遍历，使 index 依次增加直至达到 k，即得到结果

伪代码：

```
index = 0
BST_Select(TreeNode root, int k):
    if  !IsEmpty(root) :
        TreeNode node = BST_Select (root.left, k)
        if  !IsEmpty(node) :
            return node
        index++
        if  index == k :
            return root
        node = BST_Select (root.right, k)
        if  !IsEmpty(node) :
            return node
    else:
        return null
```