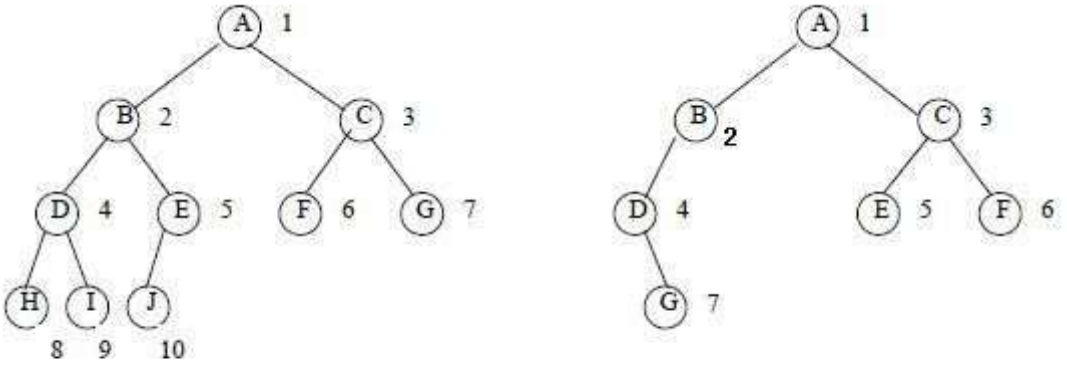
大根堆

一个大根堆（小根堆）既是大根树（小根树）也是完全二叉树。

大根树（小根树）：每个节点的值都大于（小于）或等于其子节点（如果有子节点的话）的值。

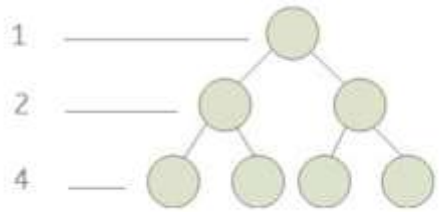
完全二叉树：若设二叉树的深度为h，除第 h 层外，其它各层 (1～h-1) 的结点数都达到最大个数，第 h 层所有的结点都连续集中在最左边，这就是完全二叉树。



(a) 一棵完全二叉树　　(b) 一棵非完全二叉树

图 6.3　完全二叉树和非完全二叉树示意图

满二叉树：除最后一层无任何子节点外，每一层上的所有结点都有两个子结点的二叉树。



初始化大根堆

```
#include <stdio.h>
// 分类 -------------- 内部比较排序
// 数据结构 ---------- 数组
// 最差时间复杂度 ---- O(nlogn)
// 最优时间复杂度 ---- O(nlogn)
// 平均时间复杂度 ---- O(nlogn)
// 所需辅助空间 ------ O(1)
// 稳定性 ------------ 不稳定
//交换
void Swap(int A[], int i, int j){
    int temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}
void Heapify(int A[], int i, int size)  // 从A[i]向下进行堆调整{
    int left_child = 2 * i + 1;          // 左孩子索引
    int right_child = 2 * i + 2;         // 右孩子索引
    int max = i;                         // 选出当前结点与其左右孩子三者之中的最大值
    if (left_child < size && A[left_child] > A[max])
        max = left_child;
```

```
    if (right_child < size && A[right_child] > A[max])
        max = right_child;
    if (max != i){
        Swap(A, i, max);                // 把当前结点和它的最大(直接)子节点进行交换
        Heapify(A, max, size);          // 递归调用，继续从当前结点向下进行堆调整
    }
}

int main(){
    int A[9] = { 5, 2, 9, 4, 7, 6, 1, 3, 8 };
    int n = sizeof(A) / sizeof(int);
    for (int i = n / 2; i >= 0; i--)//从最后一个具有孩子的节点开始检查
        Heapify(A, i, n);

    printf("堆排序结果：");
    for (int i = 0; i < n; i++)
        printf("%d ", A[i]);
    printf("\n");
    return 0;
}
```