Lab6（2018-10-30）

董依菡 15302010054@fudan.edu.cn

耿同欣 15302010048@fudan.edu.cn

张星宇 15307110273@fudan.edu.cn

Lab6目标

在解决问题的过程中，我们经常会对数据进行排序，常见的排序算法有冒泡排序、选择排序、插入排序、快速排序、归并排序、基数排序、桶排序等等，我们要根据数据的类型和实际的需求选择合适的排序算法。

在本次Lab中，你需要使用 Java 编程语言，结合课堂知识，**用数组实现堆排序（Heap Sort）**。同学们如果对其他排序算法也感兴趣，可以在课后查阅相关资料自学，如果遇到问题，欢迎提问~

Example

```
Input:
5
23 56 234 98 -7

Output:
-7 23 56 98 234
```

用户首先输入一个正整数 n，表示待排序整数的个数，接下来需要输入的是 n 个整数；最后的输出是这 n 个整数从小到大排序后的结果。

堆排序（Heap Sort）

树

- 树是 n（n >= 0）个结点的组成的有限集合
- 当 n=0 时，集合为空，称为空树
- 在任意一颗非空树中，有且仅有一个特定的结点称为根
- 当 n > 1时，除根结点以外的其余结点可分为 m（m >= 0）个不相交的有限结点集合 T1，T2.....Tm，其中每个集合本身也是一棵树，称为根的子树

其中：

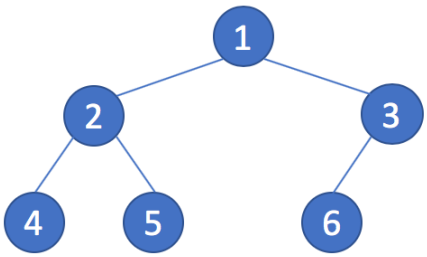
- 有且仅有一个根结点：没有前继结点，有 0 个或者多个后继结点
- 叶结点：有且仅有一个前继结点，而没有后继结点
- 其他结点：有且仅有一个前继结点，至少有一个后继结点

二叉树

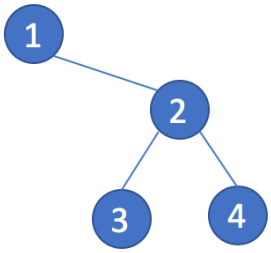
每个结点（父亲结点）**最多**有两个孩子（左孩子，右孩子）的树结构称为二叉树

完全二叉树

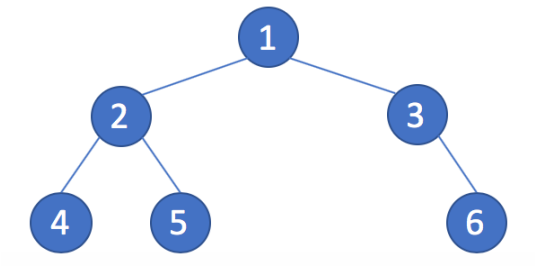
如果一棵二叉树共有 n 层，其中（1）前 n-1 层的结点个数达到最大（2）第 n 层所有结点都集中在最左边；这样的二叉树就称为完全二叉树，如下图**（数字表示结点序号）**：



以下两种都**不是**完全二叉树**（数字表示结点序号）**：



> 原因：第二层的结点个数没有达到最大，结点 1 还可以有一个左孩子。

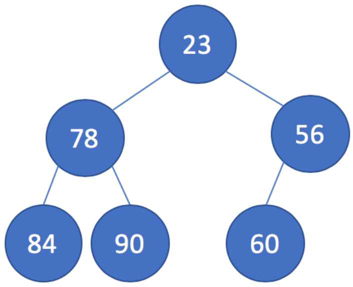


> 原因：最后一层的结点没有集中在最左边（结点 3 没有左孩子）

可以发现，用数组就可以表示一棵完全二叉树，比如：如果数组是从 1 开始标记的，那么对于一个下标为 k 的父亲结点，它左孩子的下标为 k * 2，右孩子的下标为 k * 2 + 1。（如果数组从 0 开始，也可以发现类似的性质）

小根堆

小根堆首先是一棵完全二叉树，其次在小根堆中每一个父亲结点的数值都不大于它的两个孩子（大根堆相反）。本次 Lab 要求是从小到大排序，所以我们需要用到的就是小根堆，如下图**（数字表示结点代表的数值）**：



构造初始堆

在构造初始堆的时候，我们需要从 n~1（下标） 倒着遍历数组，对于每一个结点，都需要执行一次下沉操作：

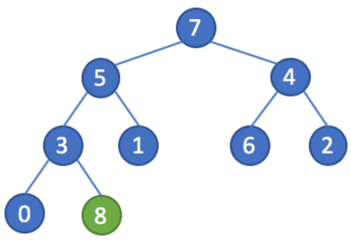
（1）如果当前结点是叶结点（没有孩子），退出

（2）如果左孩子小于等于右孩子和当前结点，则交换当前结点和左孩子，返回（1）

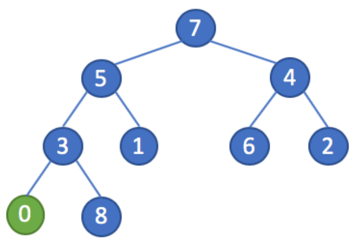
（3）如果右孩子小于等于左孩子和当前结点，则交换当前结点和右孩子，返回（1）

（4）如果当前结点小于等于左孩子和右孩子，退出

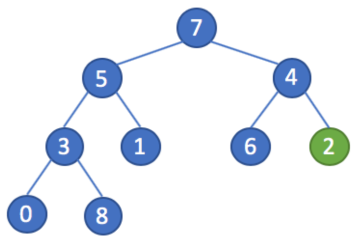
其实就是，把父亲结点和它孩子中的最小值变为新的父亲结点，因为文字表述不大容易理解，以下是一个例子：



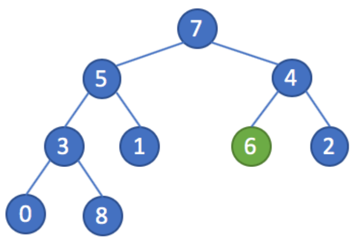
叶节点跳过



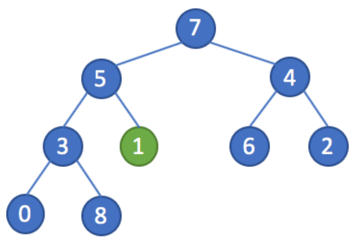
叶节点跳过



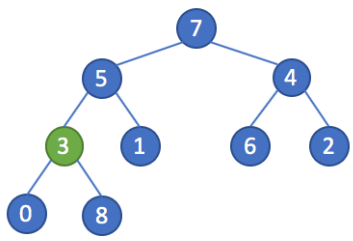
叶节点跳过



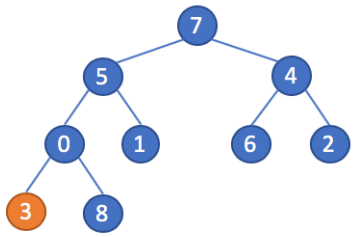
叶节点跳过



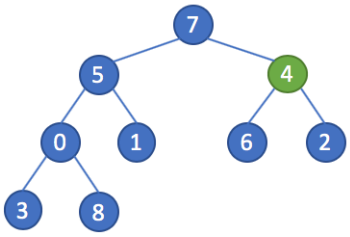
叶节点跳过



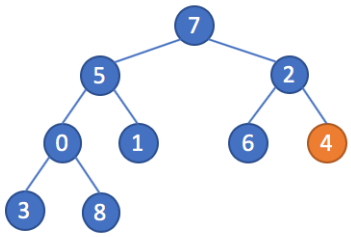
3 和 0 交换



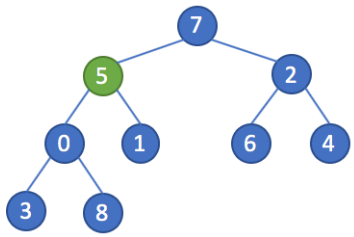
叶节点跳过



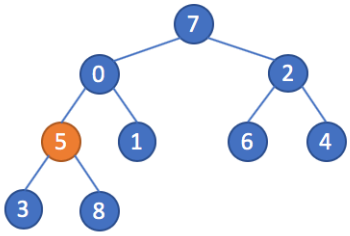
4 和 2 交换



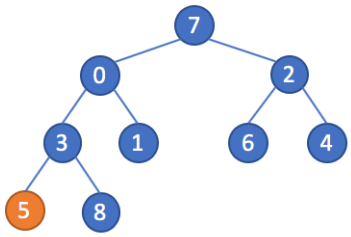
叶节点跳过



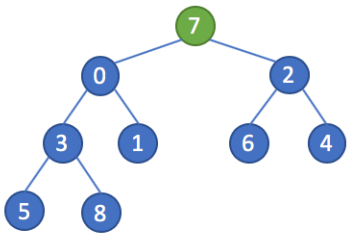
5 和 0 交换



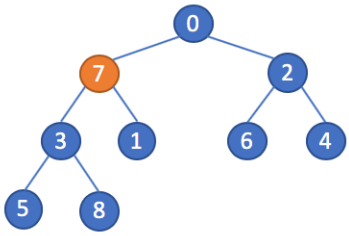
3 和 5 交换



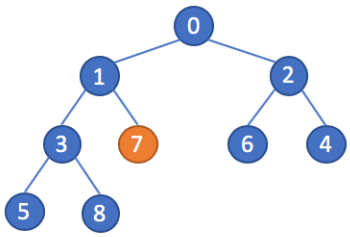
叶节点跳过



7 和 0 交换



7和 1 交换



叶节点跳过

> 至于为什么这样构造出来的完全二叉树就一定是一个小根堆，大家可以自己尝试证明一下。

排序

排序部分其实非常简单：

（1）输出根的值，因为很明显，它是当前小根堆中的最小值（上图中的 0）

（2）根输出后，它肯定需要被某个结点代替，我们选择最后那个结点（0 输出后，8 就代替 0 成为新的根），因为这样操作之后，可以保证它依然是一棵完全二叉树

（3）显然，新的完全二叉树可能并不满足小根堆的要求，这个时候对新的根做一次下沉操作即可（下沉操作在上面的示例中已经给出）

（4）返回（1），直到树为空

> 同样，至于为什么这样就完成了排序，大家可以自己尝试证明一下。

Tips

- 前面所有的描述中，都默认数组下标从 1 开始（从 0 开始类似）
- 本次 Lab 需要实现的主要有 3 个部分：（1）建堆（2）排序（3）下沉操作

- 以下代码仅供参考：

```
import java.util.Scanner;

public class HeapSort {

    public boolean isLeaf(int index, int length) {

        // TODO: 判断是否是叶子结点
        return true;
    }

    public void heapAdjust(int[] list, int index, int length) {

        while (!isLeaf(index, length)) {
            // TODO: 下沉操作
        }
    }


    public void heapSort(int[] list, int length) {

        for (int i = length; i >= 1; i--) {
            // TODO: 建堆
        }

        for (int i = length; i >= 1; i--) {
            // TODO: 排序
            // (1) 最后一个元素替代根;
            // (2) 对根做下沉操作
        }
    }

    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        HeapSort heapSort = new HeapSort();

        int length = input.nextInt();
        int[] list = new int[length + 1];

        // 数组下标从1开始
        for (int i = 1; i <= length; i++) {
            list[i] = input.nextInt();
        }

        heapSort.heapSort(list, length);

        // TODO: 输出排序后的结果
```

```
    }
}
```

关于PJ

上次 Lab 很多同学应该只是简单地用数组记录了一下游戏界面，总体逻辑跟非数组的版本类似，大家在做 PJ 的过程中可以考虑：

（1）把方块放到一个矩形中进行旋转，这样就可以大大减少代码量，避免过多的硬编码

（2）在游戏进入时，预处理每一种方块的旋转形式（也就是把方块的 4 种旋转状态事先计算好并保存下来），这样，之后只要记录方块的旋转中心、方块的类型和旋转方向就可以了

提交

提交地址：ftp://10.132.141.33/classes/18/181 程序设计A（戴开宇）/WORK_UPLOAD/lab6

提交内容：将java程序（.java文文件）压缩成zip格式，zip压缩包的名称：学号_姓名_lab6.zip

截止日期：2018/11/04 23:59:59