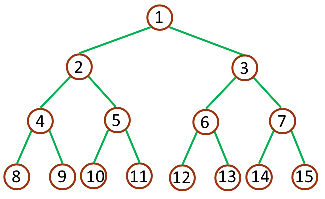
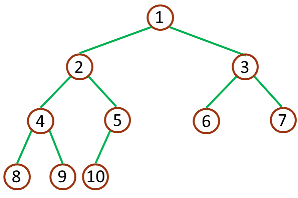
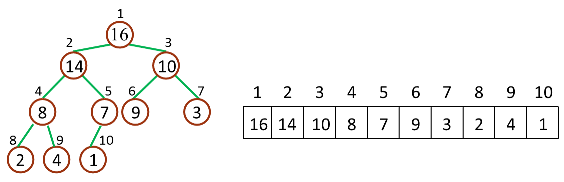
# 堆排序

## 知识储备

1. 满二叉树：一棵深度为 k，且有 2k - 1 个节点称之为满二叉树 
2. 完全二叉树：深度为 k，有 n 个节点的二叉树，当且仅当其每一个节点都与深度为 k 的满二叉树中序号为 1 至 n 的节点对应时，称之为完全二叉树 
3. 堆：堆（二叉堆）可以视为一棵完全的二叉树，完全二叉树的一个“优秀”的性质是，除了最底层之外，每一层都是满的，这使得堆可以利用数组来表示（普通的一般的二叉树通常用链表作为基本容器表示），每一个结点对应数组中的一个元素。 
4. 堆和数组的关系，对于给定的某个结点的下标 i，可以很容易的计算出这个结点的父结点、孩子结点的下标（因为下一行是上一行\*2，所以就单个节点来说也是如此）：
   * Parent(i) = floor(i/2)，i 的父节点下标
   * Left(i) = 2i，i 的左子节点下标
   * Right(i) = 2i + 1，i 的右子节点下标

## 二叉堆分为最大堆和最小堆

### 最大堆

* 最大堆中的最大元素值出现在根节点（对顶）
* 堆中的每个父节点的元素值都大于等于其孩子节点（如果存在）

### 最小堆

* 最小堆的最小元素值出现在根节点（堆顶）
* 堆中每个父节点的元素值都小于等于其孩子节点（如果存在）

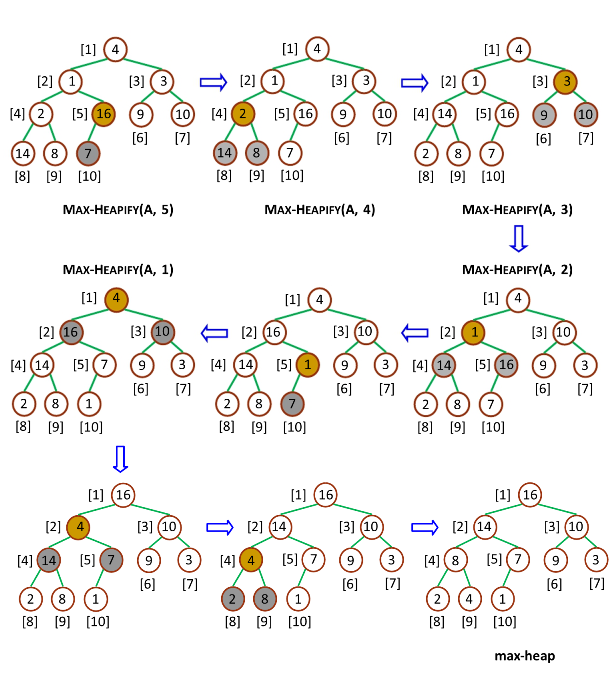
## 堆排序原理

堆排序就是把最大堆堆顶的最大数取出，将剩余的堆继续调整为最大堆，再次将堆顶的最大数取出，这个过程持续到剩余数只有一个时结束。在堆中定义以下几种操作：

* 最大堆调整（Max-Heapify）：将堆的末端子节点作调整，使得子节点永远小于父节点
* 创建最大堆（Build-Max-Heap）：将堆所有数据重新排序，使其成为最大堆
* 堆排序（Heap-Sort）：移除位在第一个数据的根节点，并做最大堆调整的递归运算

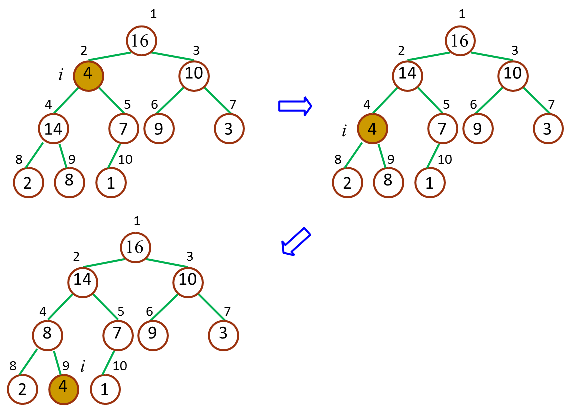
### 创建最大堆

使用自下而上的方式调用Max\_Heapify改造狐族，建立最大堆，这样保证下表i之后的节点都满足最大堆性质。

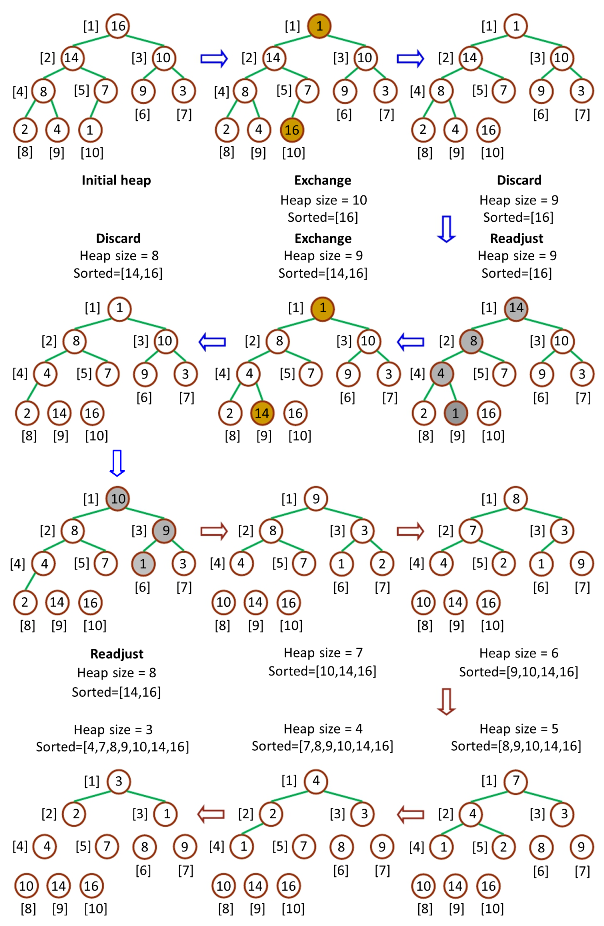


### 最大堆调整

保持最大堆的性质



### 堆排序



## 代码

package com.go2going.sort;  
  
import java.util.Arrays;  
  
/\*\*  
 \* 项目名称： testcode<br>  
 \* 类名称： HeapSort<br>  
 \* 描述：堆排序<br>  
 \* <a href="http://bubkoo.com/2014/01/14/sort-algorithm/heap-sort/">参考文章，需要理解文章中的变化过程</a>  
 \* <a href="https://zh.wikipedia.org/wiki/%E5%A0%86%E6%8E%92%E5%BA%8F#Java">wiki</a>  
 \*  
 \* @author wangqiang  
 \* 创建时间： 2017/11/7 0007 11:10  
 \*/  
public class HeapSort {  
  
 private int[] arr;  
  
 public HeapSort(int[] arr) {  
 this.arr = arr;  
 }  
  
 /\*\*  
 \* 测试用例  
 \* <p>  
 \* 输出：  
 \* [0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 9]  
 \*/  
 public static void main(String[] args) {  
 int[] arr = new int[]{3, 5, 3, 0, 8, 6, 1, 5, 8, 6, 2, 4, 9, 4, 7, 0, 1, 8, 9, 7, 3, 1, 2, 5, 9, 7, 4, 0, 2, 6};  
 new HeapSort(arr).sort();  
 System.out.println(Arrays.toString(arr));  
 }  
  
 /\*\*  
 \* 堆排序的主要入口方法，共两步。  
 \*/  
 public void sort() {  
  
 //1.创建最大堆  
 int length = arr.length;  
 int beginIndex = (length - 1) >> 1;  
 //i都是有子节点的根节点  
 for (int i = beginIndex; i >= 0; i--) {  
 heapMax(i, length);  
 }  
  
 //构建玩最大堆之后，还不能保证左右节点的大小，只知道根节点最大  
  
 //2.找到最大值，方队数组的最后，那么下次循环的时候就不用排序这个了  
 for (int i = length-1; i > 0; i--) {  
 swap(0, i);  
 //将最大值放到最后，然后重新构建  
 heapMax(0, i);  
 }  
 }  
  
 /\*\*  
 \* left和right+1的原因是在计算beginIndex的位置时就少了1，因为下标是从0开始的  
 \*  
 \* @param i  
 \* @param length  
 \*/  
 public void heapMax(int i, int length) {  
 int left = (i << 1) + 1;//左节点  
 int right = left + 1;//右节点  
 int maxIndex = i;  
  
 if (left < length && arr[left] > arr[maxIndex]) {  
 maxIndex = left;  
 }  
  
 if (right < length && arr[right] > arr[maxIndex]) {  
 maxIndex = right;  
 }  
  
 if (maxIndex != i) {  
 swap(maxIndex, i);  
 //递归调整  
 heapMax(maxIndex, length);  
 }  
  
 }  
  
 public void swap(int a, int b) {  
 int temp = arr[a];  
 arr[a] = arr[b];  
 arr[b] = temp;  
 }  
  
}