研究生算法课课堂笔记

上课日期： 12 月 12 日　　第(1)节课

组长学号及姓名：施文娴 1601214479

组员学号及姓名：朱思文 1601214501

内容概要：

本节课主要总结了算法模拟考试的情况并讲解了三道题目（butterfly，Dynamic Median 及逆序对数）。笔记内容安排如下：

1. 考试相关。
2. Dynamic median
3. 逆序对数

详细内容：

1. **考试相关：**

Runtime Error：数组越界

Time Limit Exceeded：有死循环或者算法本身问题

考试时间：1 月 9 日星期一下午 2:00-5:00

考试地点：理科一号楼 1235

注意事项：封 u 盘，封网盘；推荐使用实验室给的键盘鼠标；金山词霸和 chrome 不兼容，有道可以使用；正式考试会采用新的上机登录账号；正式考试请自带笔，考场会提供草稿纸。

2. Dynamic Median（动态中位数）

问题描述：

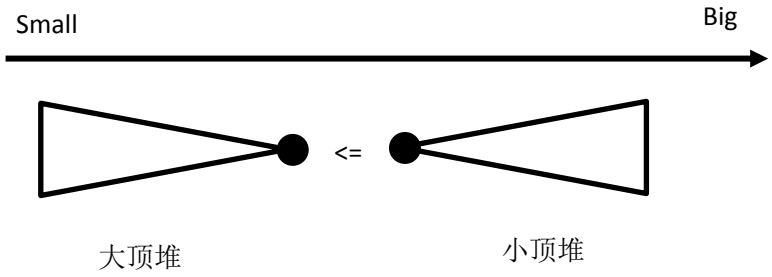
设计一个数据结构，初始为空，支持以下操作：

（1）增加一个元素，要求在 log(n)时间内完成，其中 n 是该数据结构中当前元素的个数。注意：数据结构中允许有重复的元素。

（2）返回当前元素集合的中位数，要求在常数时间内完成。如果当前元素的个数为偶数，那么返回下中位数（即两个中位数中较小的一个）。

（3）删除中位数，要求在 log(n)时间内完成。

这一题考察的是堆的使用。思路如下图所示，横线代表数组，从左往右，数组元素越来越大，构建两个堆，一个大顶堆（也叫最大堆，堆顶元素是堆中最大的），一个小顶堆（也叫最小堆，堆顶元素是堆中最小的），使得数组中较小的一半元素存在左边的大顶堆中，数组中较大的一半元素存在右边的小顶堆中。大顶堆的堆顶小于等于小顶堆的堆顶，两个堆的元素数目相差不超过 1，这样，可以保证中位数在两个堆的堆顶元素中取到，如果两个堆元素个数相等，中位数是大顶堆的堆顶元素（要求返回下中位数），如果两个堆元素个数不相等，中位数是元素多的那个堆的堆顶。由于题目中要求“如果当前元素的个数为偶数时，返回下中位数（即两个中位数中较小的一个）”，我们可以在建堆的过程中，保证左边堆（大顶堆）的元素个数始终和右边堆（小顶堆）的个数相等或者比右边堆的元素个数多一个，让中位数始终在左边堆的堆顶中取到。



具体步骤：

1. 初始化：如果有两个元素，小的加入大顶堆，大的加入小顶堆。

2. 插入：如果待插入的元素小于大顶堆的堆顶，加入左边的大顶堆，如果大于小顶堆的堆顶，加入右边的小顶堆。如果处于两者之间，哪一个堆元素少，插到哪里。如果两个堆一样多，插到左边的堆里。插入后，要调整使得两个堆的元素个数保持平衡（最多相差 1）。调整的方法是取出元素较多的那个堆的堆顶元素，加入另一个堆中，使得两边的元素个数保持平衡（最多相差 1）。如果每次插入一个元素后，我们都调整平衡，那么每次插入时最多调整一个元素，这是因为每次插入前的两个堆元素最多相差 1，插入后两个堆的元素最多相差 2，所以最多只要对堆顶元素做一次弹出插入的操作就可以了。

3. 查找返回中位数：返回元素个数多的那个堆的堆顶元素。如果一样多，返回左边大顶堆的堆顶元素。

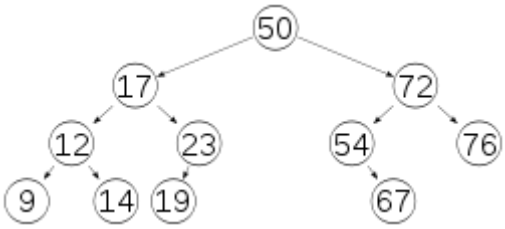
4. 删除中位数：按查找中位数的方法先找到中位数，删除对应的堆顶元素。如果原来两个堆的元素差 1，那么删除过后两个堆的元素个数相等，保持平衡性质。如果两个堆的元素个数相等，删除过后两个堆的元素个数相差不超过 1，也保持平衡性质。但是如果想要保持左边堆的个数始终等于右边堆的个数或者比右边堆的个数大 1，此时要调整一下，将右边堆的堆顶弹出，插入到左边堆中。

老师提问：有同学说这题可以用 STL 库的平衡二叉树做，效率更高：平衡二叉树的两个子树是平衡的，平衡二叉树的根就是中位数。请问可以吗？问题在哪里？

问题在于，平衡二叉树不能够保证左右子树的节点个数平衡（即不能保证根是中位数）。

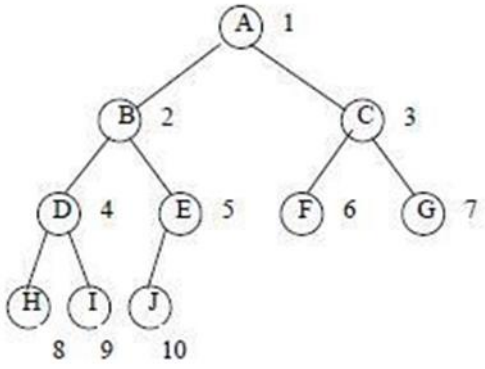
平衡二叉树用的比较多的是 AVL 树和红黑树。AVL 树的左右子树的高度差的绝对值不超过 1。红黑树的根节点到叶节点的最长路径和最短路径的长度相差在 2 倍以内。以上性质并不能保证左右子树的节点数目是平衡的（最多相差 1）。

举个例子，比如下面这棵树，它是平衡二叉树，但显然它的根的左右子树的节点个数相差超过 1。



之前的课上讲过堆和二叉树的区别：堆的实现是不依靠指针的，通过推算子女节点的数组下标访问子女节点。如果把堆看成二叉树，它是不是 AVL 树？是不是红黑树？

堆如果看成一棵二叉树，它是完全二叉树（如果不是完全二叉树，就不能通过推算得到子女节点的数组下标）。完全二叉树，形象来说，就是从上到下，一层一层填元素，填满上一层再填下一层，每一层从左往右依次填。例如下面这棵树就是一棵完全二叉树。完全二叉树是最平衡的二叉树，所以它一定是 AVL 树，也是红黑树。



(a) 一棵完全二叉树

3. 归并排序

动态规划：将蛮力算法（指数级算法）通过动态规划优化到多项式时间内。

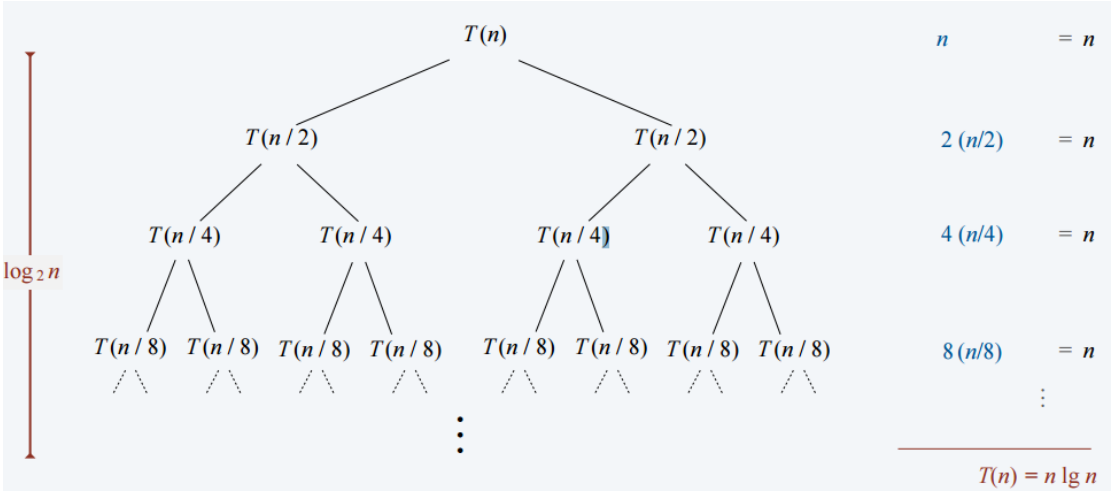
分治法：不做分治法，已经可在多项式时间内解决，通过分治法进行优化，变得更高效。

归并排序：

分解：将 n 个元素分成含 n/2 个元素的子序列。

解决：用归并排序法对两个子序列递归排序。

合并：合并两个已排序的子序列得到排序结果。



时间复杂度是 O（nlogn）。其中 n 是因为归并过程是线性的，分成左右两部分，合并在一起就是线性扫一遍。logn 是树的高度，归并排序树高度保证是 logn。快速排序，树的高度可能会退化成线性。

4. 逆序对

问题描述：

对于一个长度为 N 的整数序列 A，满足 i < j 且 Ai > Aj.的数对(i,j)称为整数序列 A 的一个逆序对。请求出整数序列 A 的所有逆序对个数

问题分析：

采用归并排序的方式计算。我们考虑把一个序列分成两部分，那么这个序列的逆序对数就等于每一部分的逆序对数加上合并的逆序对数。

关键步骤：

Count inversions (a, b) with $a \in A$ and $b \in B$, assuming A and B are sorted.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i < b_j$, then a_i is not inverted with any element left in B.
- If $a_i > b_j$, then b_j is inverted with every element left in A.
- Append smaller element to sorted list C.

当 ai>bi 时，A 中在 ai 之后的元素都比 ai 大，所以 bi 与 A 中 ai 及其之后的每个元素都构成逆序对，这时逆序对的数目就是 A 中 ai 及其之后元素的个数。

算法伪代码：

```
Merge-And-Count(A, B)
  初始化 count= 0，C =  空
  while A 和 B 都不为空
     令 ai 和 bj 分别为 A 和 B 中的首元素
     if ai <= bj
        把 ai 加入到输出表 C 中
        A = A - {ai}
     else
        把 bj 加入到输出表 C 中
        B = B - {bj}
        count += A 中剩下的元素的个数
     endIf
  endWhile
  if A  为空
     把 B 中剩下元素加入到 C
  else
     把 A 中剩下元素加入到 C
  endIf
  return  合并结果 C 和逆序对个数 count
```