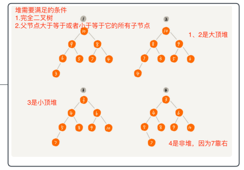
17.堆和堆排序

1.堆的概念

堆需要满足的条件
*堆是一个完全二叉树；
*堆中每一个节点的值都必须大于等于（或小于等于）其子树中每个节点的值

堆化：在插入和删除的过程中会破坏以上原则，恢复的过程叫做堆化

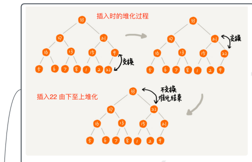


2.堆的操作

堆的操作有插入和删除

堆化的复杂度
插入和删除的主要过程就是堆化的过程，一个包含 n 个节点的完全二叉树，高度不会超过 log2n。堆化的过程是顺着节点所在路径比较交换的，所以堆化的时间复杂度跟树的高度成正比，也就是 O(logn)。

插入和删除复杂度
插入数据和删除堆顶元素的主要逻辑就是堆化，所以，往堆中插入一个元素和删除堆顶元素的时间复杂度都是 O(logn)。

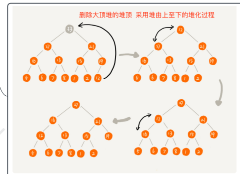


插入操作

```
public class Heap {
  private int[] a; // 数组，从下标1开始存储数据
  private int n;  // 堆可以存储的最大数据个数
  private int count; // 堆中已经存储的数据个数

  public Heap(int capacity) {
    a = new int[capacity + 1];
    n = capacity;
    count = 0;
  }

  public void insert(int data) {
    if (count >= n) return; // 堆满了
    ++count;
    a[count] = data;
    int i = count;
    while (i/2 > 0 && a[i] > a[i/2]) { // 自下往上堆化
      swap(a, i, i/2); // swap()函数作用：交换下标为i和i/2的两个元素
      i = i/2;
    }
  }
}
```



删除操作

```
public void removeMax() {
  if (count == 0) return -1; // 堆中没有数据
  a[1] = a[count];
  --count;
  heapify(a, count, 1);
}

private void heapify(int[] a, int n, int i) { // 自上往下堆化
  while (true) {
    int maxPos = i;
    if (i*2 <= n && a[i] < a[i*2]) maxPos = i*2;
    if (i*2+1 <= n && a[maxPos] < a[i*2+1]) maxPos = i*2+1;
    if (maxPos == i) break;
    swap(a, i, maxPos);
    i = maxPos;
  }
}
```

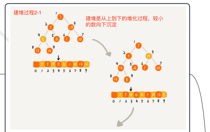
3.堆排序

堆排序可以分为两个步骤：
1.建堆 2.排序

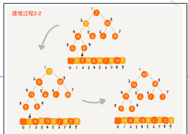
1.建堆也有两种思路
第一种是借助我们前面讲的，在堆中插入一个元素的思路。尽管数组中包含 n 个数据，但是我们可以假设，起初堆中只包含一个数据，就是下标为 1 的数据。然后，我们调用前面讲的插入操作，将下标从 2 到 n 的数据依次插入到堆中。这样我们就将包含 n 个数据的数组，组织成了堆

第二种实现思路，跟第一种截然相反，也是我这里要详细讲的。第一种建堆思路的处理过程是从前往后处理数组数据，并且每个数据插入堆中时，都是从下往上堆化。而第二种实现思路，是从后往前处理数组，并且每个数据都是从上往下堆化

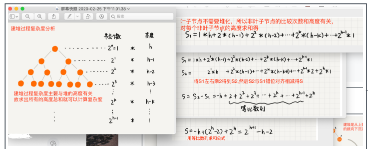
2.排序过程如图



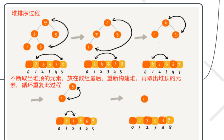
第二种方法建堆2-1



第二种方法建堆2-2



建堆的复杂度分析
h=log2n，代入公式 S，就能得到 S=O(n)，所以，建堆的时间复杂度就是 O(n)



排序过程

```
// n表示数据的个数，数组a中的数据从下标1到n的位置。
public static void sort(int[] a, int n) {
  buildHeap(a, n);
  int k = n;
  while (k > 1) {
    swap(a, 1, k);
    --k;
    heapify(a, k, 1);
  }
}
```

4. 堆排序的复杂度分析

堆排序包括建堆和排序两个操作，建堆过程的时间复杂度是 O(n)，排序过程的时间复杂度是 O(nlogn)，所以，堆排序整体的时间复杂度是 O(nlogn)。

堆排序不是稳定的排序算法，因为在排序的过程，存在将堆的最后一个节点跟堆顶节点互换的操作，所以就有可能改变值相同数据的原始相对顺序

5. 练习：

1.O(nlogn)，甚至堆排序比快速排序的时间复杂度还要稳定，但在实际的软件开发中，快速排序的性能要比堆排序好，这是为什么呢?

2.我们今天讲了堆的一种经典应用，堆排序。关于堆，你还能想到它的其他应用吗?

3.10 亿个搜索关键词的日志文件，1GB单机内存，求Top10

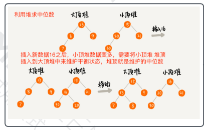
1.思路：
第一点，堆排序数据访问的方式没有快速排序友好。对于快速排序来说，数据是顺序访问的。而对于堆排序来说，数据是跳着访问的。这样对 CPU 缓存是不友好的

第二点，对于同样的数据，堆排序算法的数据交换次数要多于快速排序。对于基于比较的排序算法来说，整个排序过程就是比较和交换（或移动）组成的。快速排序数据交换的次数不会比逆序度多。但是堆排序的第一步是建堆，会打乱数据原有的相对先后顺序，导致原数据的有序度降低。比如，对于一组已经有序的数据来说，经过建堆之后，数据反而变得更无序了

2. 思路：

第一类：优先级对列
1.合并有序的小文件：
类似归并排序的合并过程，只不过合并的是100个有序文件，先将100个文件的第一个字符串取出，放入一个数组中，对数组排序找出最小的，将其放到合并文件中，假如最小的这个字符串是13.txt文件，接下来再从这个文件中取出第二个字符串放入数组，循环继续比较，对于数组的排序过程可以使用最小堆的插入和删除操作，构建一个优先级对列，将堆顶的最小数删除，再取新数据插入，堆的操作复杂度是O(log(n))，比排序要省时间
2.高效定时器：
一个定时器里面维护了很多任务，常规最法是没个很小的时间去扫描任务时间，达到了定时时间就去执行，如果构建一个最小堆驱动态的插入和删除，堆顶的就是马上要执行，对定任务和当前时间做差就得出时间间隔T，过了这个间隔直接执行任务就可以了，不需要循环扫描整个任务列表
3.求Topn：
静态topn：数据集是静态例如长度为m的数组，构建一个n大小的小顶堆，遍历数组，小于堆顶不处理，大于堆顶就把堆顶删除，然后将这个数据插入堆中，这个堆中的数据就是Topn，时间复杂度最坏情况所有数据都入堆一次即O(nlog(n))
动态topn：动态数据集，可以一直维护一个n大小的小顶堆，当有数据添加到集合时，就拿他和堆顶作对比，小于堆顶不处理，大于则删除顶堆，再将其插入堆，这就是一个实时topn

第二类：使用堆求中位数
思路：将数据集分为两个堆，前半部分构建大顶堆，后半部分数据构建小顶堆，大顶堆堆顶就是中位数，对于持续加入数据集中的元素，可以将小顶堆的堆顶插入到大顶堆中，达到持续流动更新堆顶的操作，让堆顶一直保持是中位数
类似的求99%的响应时间是一样的道理，大顶堆中保存n*99%的数据，小顶堆中保存n*1%的数据，不满足比例时要移动数据，直到满足比例



思路：
hash分成小文件-->对每一个小文件使用散列表统计访问次数-->在针对每一个文件构建小顶堆
这样就统计出了多个文件的Topn,再从这多个文件中选出n个就是要求的数据