**问题描述：**

这是在网上找到的一道百度的面试题：  
搜索引擎会通过日志文件把用户每次检索使用的所有检索串都记录下来，每个查询串的长度为1-255字节。假设目前有一千万个记录，这些查询串的重复度比较 高，虽然总数是1千万，但如果除去重复后，不超过3百万个。一个查询串的重复度越高，说明查询它的用户越多，也就是越热门。请你统计最热门的10个查询 串，要求使用的内存不能超过1G。

解答转自：http://blog.redfox66.com/post/2010/09/23/top-k-algoriyhm-analysis.aspx

**问题解析：**

【分析】：要统计最热门查询，首先就是要统计每个Query出现的次数，然后根据统计结果，找出Top 10。所以我们可以基于这个思路分两步来设计该算法。下面分别给出这两步的算法：

第一步：Query统计

**算法一：直接排序法**

首先我们能想到的算法就是排序了，首先对这个日志里面的所有Query都进行排序，然后再遍历排好序的Query，统计每个Query出现的次数了。但是 题目中有明确要求，那就是内存不能超过1G，一千万条记录，每条记录是225Byte，很显然要占据2.55G内存，这个条件就不满足要求了。

让我们回忆一下数据结构课程上的内容，当数据量比较大而且内存无法装下的时候，我们可以采用外排序的方法来进行排序，这里笔者采用归并排序，是因为归并排序有一个比较好的时间复杂度O(NlgN)。

排完序之后我们再对已经有序的Query文件进行遍历，统计每个Query出现的次数，再次写入文件中。

综合分析一下，排序的时间复杂度是O(NlgN)，而遍历的时间复杂度是O(N)，因此该算法的总体时间复杂度就是O(NlgN)。

**算法二：Hash Table法**

在上个方法中，我们采用了排序的办法来统计每个Query出现的次数，时间复杂度是NlgN，那么能不能有更好的方法来存储，而时间复杂度更低呢？  
题目中说明了，虽然有一千万个Query，但是由于重复度比较高，因此事实上只有300万的Query，每个Query255Byte，因此我们可以考虑 把他们都放进内存中去，而现在只是需要一个合适的数据结构，在这里，Hash Table绝对是我们优先的选择，因为Hash Table的查询速度非常的快，几乎是O(1)的时间复杂度。  
那么，我们的算法就有了：维护一个Key为Query字串，Value为该Query出现次数的HashTable，每次读取一个Query，如果该字串 不在Table中，那么加入该字串，并且将Value值设为1；如果该字串在Table中，那么将该字串的计数加一即可。最终我们在O(N)的时间复杂度 内完成了对该海量数据的处理。  
本方法相比算法一：在时间复杂度上提高了一个数量级，但不仅仅是时间复杂度上的优化，该方法只需要IO数据文件一次，而算法一的IO次数较多的，因此该算法比算法一在工程上有更好的可操作性。

**第二步：找出Top 10**

**算法一：排序**

我想对于排序算法大家都已经不陌生了，这里不在赘述，我们要注意的是排序算法的时间复杂度是NlgN，在本题目中，三百万条记录，用1G内存是可以存下的。

**算法二：部分排序**

题目要求是求出Top 10，因此我们没有必要对所有的Query都进行排序，我们只需要维护一个10个大小的数组，初始化放入10Query，按照每个Query的统计次数由 大到小排序，然后遍历这300万条记录，每读一条记录就和数组最后一个Query对比，如果小于这个Query，那么继续遍历，否则，将数组中最后一条数 据淘汰，加入当前的Query。最后当所有的数据都遍历完毕之后，那么这个数组中的10个Query便是我们要找的Top10了。  
不难分析出，这样的算法的时间复杂度是N\*K， 其中K是指top多少。

**算法三：堆**

在算法二中，我们已经将时间复杂度由NlogN优化到NK，不得不说这是一个比较大的改进了，可是有没有更好的办法呢？  
分析一下，在算法二中，每次比较完成之后，需要的操作复杂度都是K，因为要把元素插入到一个线性表之中，而且采用的是顺序比较。这里我们注意一下，该数组 是有序的，一次我们每次查找的时候可以采用二分的方法查找，这样操作的复杂度就降到了logK，可是，随之而来的问题就是数据移动，因为移动数据次数增多 了。不过，这个算法还是比算法二有了改进。  
基于以上的分析，我们想想，有没有一种既能快速查找，又能快速移动元素的数据结构呢？回答是肯定的，那就是堆。  
借助堆结构，我们可以在log量级的时间内查找和调整/移动。因此到这里，我们的算法可以改进为这样，维护一个K(该题目中是10)大小的小根堆，然后遍历300万的Query，分别和根元素进行对比。。。  
那么这样，这个算法发时间复杂度就降到了NlogK，和算法而相比，又有了比较大的改进。

**结语：**

至此，我们的算法就完全结束了，经过步骤一和步骤二的最优结合，我们最终的时间复杂度是O（N） + O（N’）logK。如果各位有什么好的算法，欢迎跟帖讨论。