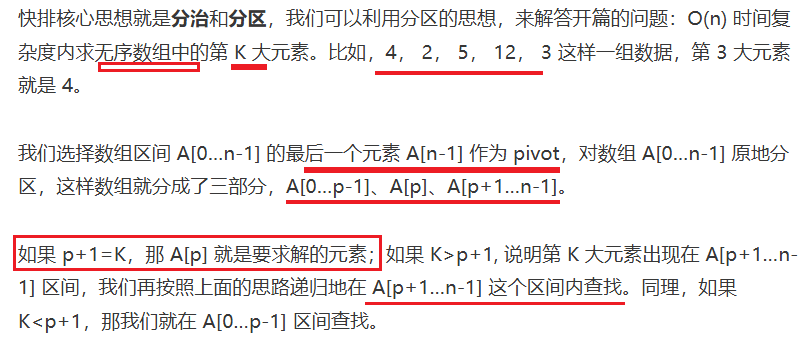
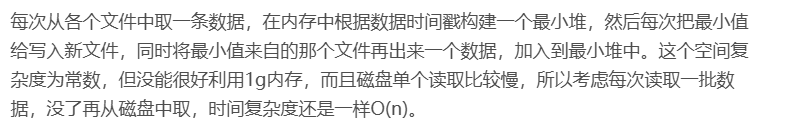
## 12.如何在 O(n) 的时间复杂度内查找一个无序数组中的第 K 大元素？



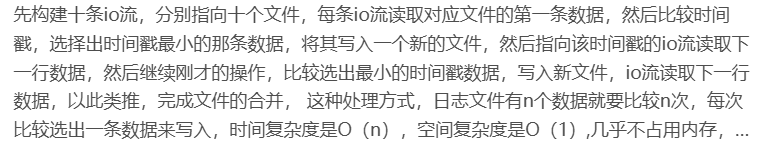
思考：

现在你有 10 个接口访问日志文件，每个日志文件大小约 300MB，每个文件里的日志都是按照时间戳从小到大排序的。你希望将这 10 个较小的日志文件，合并为 1 个日志文件，合并之后的日志仍然按照时间戳从小到大排列。如果处理上述排序任务的机器内存只有 1GB，你有什么好的解决思路，能“快速”地将这 10 个日志文件合并吗？

参考1



参考2



## 13. 如何根据年龄给 100 万用户排序？

实际上，根据年龄给 100 万用户排序，就类似按照成绩给 50 万考生排序。我们假设年龄的范围最小 1 岁，最大不超过 120 岁。我们可以遍历这 100 万用户，根据年龄将其划分到这 120 个桶里，然后依次顺序遍历这 120 个桶中的元素。这样就得到了按照年龄排序的 100 万用户数据。

## 15. 如何在 1000 万个整数中快速查找某个整数？

我们的内存限制是 100MB，每个数据大小是 8 字节，最简单的办法就是将数据存储在数组中，内存占用差不多是 80MB，符合内存的限制。可以先对这 1000 万数据从小到大排序，然后再利用二分查找算法，就可以快速地查找想要的数据了。

## 16.如何快速定位出一个 IP 地址的归属地？

如果 IP 区间与归属地的对应关系不经常更新，我们可以先预处理这 12 万条数据，让其按照起始 IP 从小到大排序。如何来排序呢？我们知道，IP 地址可以转化为 32 位的整型数。所以，我们可以将起始地址，按照对应的整型值的大小关系，从小到大进行排序。

然后“在有序数组中，查找最后一个**小于等于**某个给定值的元素”了。

当我们要查询某个 IP 归属地时，我们可以先通过二分查找，找到**最后一个起始 IP** **小于等于**这个 IP 的 IP 区间，然后，检查这个 IP 是否在这个 IP 区间内，如果在，我们就取出对应的归属地显示；如果不在，就返回未查找到。

## 24有了如此高效的散列表，为什么还需要二叉树？

散列表的插入、删除、查找操作的时间复杂度可以做到常量级的 O(1)，非常高效。

而二叉查找树在比较平衡的情况下，插入、删除、查找操作时间复杂度才是 O(logn)，

相对散列表，好像并没有什么优势，那我们为什么还要用二叉查找树呢？

有以下面几个原因：

第一，散列表中的数据是无序存储的，如果要输出有序的数据，需要先进行排序。而对于二叉查找树来说，我们只需要中序遍历，就可以在 O(n) 的时间复杂度内，输出有序的数据序列。

第二，散列表扩容耗时很多，而且当遇到散列冲突时，性能不稳定，尽管二叉查找树的性能不稳定，但是在工程中，我们最常用的平衡二叉查找树的性能非常稳定，时间复杂度稳定在 O(logn)。

第三，笼统地来说，尽管散列表的查找等操作的**时间复杂度**是常量级的，但因为**哈希冲突**的存在，这个**常量不一定比 logn 小**，所以实际的查找速度可能不一定比 O(logn) 快。加上**哈希函数的耗时**，也不一定就比平衡二叉查找树的效率高。

第四，**散列表的构造比**二叉查找树要复杂，需要考虑的东西很多。比如散列函数的设计、冲突解决办法、扩容、缩容等。平衡二叉查找树**只需要考虑平衡性这一个问题**，而且这个问题的解决方案比较成熟、固定。

最后，为了避免过多的散列冲突，散列表**装载因子不能太大**，特别是基于开放寻址法解决冲突的散列表，不然会浪费一定的存储空间。