1. 直接插入排序（insert sort）的基本思想和实现？

基本思想：往有序数组中插入新的元素到指定位置

①从第一个元素开始，该元素默认是排序的

②取出下一个元素，在已排序的元素序列中从后向前扫描

③如果该元素（已排序）大于新元素，将该元素移动到下一个位置

④重复步骤③，直到找到已排序的元素小于或者等于新元素的位置

⑤将新元素插入该位置

⑥重复步骤②-⑤

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(n2) | O(n) | O(n2) | O(1) | 是 |

优化：找需要插入的位置的时候可以使用二分查找，因为已经排好序了，所以变种的插入排序：二分查找插入排序

1. 希尔排序（shell sort）的基本思想和实现？

希尔排序是直接插入排序的改进，又称缩小增量排序。

基本思想：整个序列分成若干子序列分别进行直接插入排序，待基本有序时再对全体进行直接插入排序。

①选择一个增量序列t1 t2 ... tk（一般初次取数组半长，之后减半直到为1）

②按增量个数k=log2(n)进行k次排序

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(n1.25) | O(n) | O(n2) | O(1) | 不是 |

Hibbard增量的希尔排序的最坏时间复杂度比希尔排序最坏要好。

1. 选择排序（selection sort）的基本原理和实现？

基本思想：比较+交换

①从待排序序列中，找到关键字最小的元素

②如果最小元素不是待排序序列的第一个元素，将其和第一个元素互换

③从余下的n-1各元素中重复上述步骤

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(n2) | O(n2) | O(n2) | O(1) | 是 |

可以看出，它和差劲的冒泡排序、插入排序的思想差不多，都是要挨个比较。

1. 堆排序（heap sort）的基本原理和实现？

这就是在项目中所说的最小堆、最大堆。

基本思想：以大顶堆为例，将待排序的序列构造为一个堆，选出最大的移走，剩下的调整堆，重复...

①先将序列构建成堆，那么第一个元素就是最大的

②将最大元素与最后一个元素交换——原址排序

③现在堆已经打乱了，所以需要调整堆，重复上述步骤

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(n2) | O(n2) | O(n2) | O(1) | 不是 |

父节点i的左子节点是2\*i+1

父节点i的右子节点是2\*i+2

子节点i的父节点是floor((i-1)/2)

可以看出，堆排序可以用于数据量大、持续不断的数据进来，当时我的目的是想取topN，维护一个最大堆是可以的。

1. 快速排序（quick sort）的基本思想和实现？

基本思想：分治法

①从数列中挑选出一个元素作为基准pivot

②重新排序数列，所有比基准小的放在前后，比基准大的放在后面，这就是一个分区操作

③递归的对子序列进行①②

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(nlogn) | O(nlogn) | O(n2) | O(1) | 不是 |

对于基准的选择要注意，否则可能退化为冒泡排序，可以用中间位置作为基准。

1. 归并排序（merge sort）的基本原理和实现？

最近在看海量数据排序的面试题，经常出现hash+归并处理的情况。

基本思想：拆分+合并，将两个（或两个以上）有序表合并成一个新的有序表，即先化繁为简，后化散为整。

①将序列每相邻的两个元素分为一组，形成floor(n/2)个子序列，每个子序列中有两个元素

②将上述的子序列两两归并，减少为floor(n/4)个子序列，每个子序列有4个元素

③重复上述步骤

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(nlogn) | O(nlogn) | O(n2) | O(n) | 不是 |

归并排序需要额外空间。

1. 基数排序（radix sort）的基本原理和实现？

基本思想：将所有待比较正整数统一为同样长度的位数，不够补0，将整数按位数切割不同的数字，然后对每个数字分别比较。

两种方法，一种是MSD从高位到低位，另一种则是LSD从低位到高位，一般是MSD，比较适用于位数多情况。

①取得数组中最大数，并取得位数

②arr为原始数组，从高位开始取每个位组成radix数组

③按当前顺序取出（此时有一定顺序了）

④下一位比较，重复上述过程

注意：这里并不是基于比较的，而是有分桶（hash）的策略

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(d\*(n+k)) | O(d\*(n+k)) | O(d\*(n+k)) | O(n+k) | 是 |

注意：d肯定是位数，n是元素数，而k指的是桶的个数，正常来说桶应该是10个，但是可能因为数据少，并没有全部占起来，那么经过一次分桶后从桶中串数的时候就不会遍历那个空桶了。

1. 计数排序（counting sort）的原理与实现？

基本原理：三次扫描即可

①第一次扫描得到最大值最小值，开一个max-min+1长度的数组

②把arr中元素分配进数组，索引=value-min，值=频率

③再扫描一次，读取数组中的数据，值=索引+min

其实也是分桶的意思，只不过每个桶只代表一个元素或者相同值得元素

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(n+max-min) | O(n+max-min) | O(n+max-min) | O(max-min) | 不是 |

计数排序适用于max-min小的情况！

上面所说的计数排序、基数排序，其实都是桶排序的思想，下面看一下桶排序。

1. 桶排序（bucket sort）的基本思想和实现？

基本原理：和计数排序不同的时候桶排序每个桶里面可能不止一个元素，对每个桶进行排序（排序方法任选）

①找出min和max，规划同的个数k

②遍历一次把元素放入桶中，对每个桶排序

③在遍历一次读出来

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 稳定 |
| O(n+n(log(n/k)))假设桶内排序是快速排序 | O(n+n(log(n/k)))假设桶内排序是快速排序 | O(n+n\*(n/k)))假设桶内排序是快速排序 | O(n+k) | 不是 |

适用于元素分布比较均匀的情况，可以退化为计数排序

总结：关于桶排序的三种实现计数、基数、桶排序，他们突破了nlogn的极限，但是空间上做出了牺牲，要注意这点。

关于代码的实现，不管是递归还是非递归方式接下来都要自己实现一遍！！！以免面试时随便让自己手写一个排序算法的时候手足无措。