类别	排序方法	最好情况 最短时间	平均情况	最坏情况 最长时间	辅助空间	稳定性	特点
插入排序	直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定	• 最坏情况： $O(n^2)=\frac{n(n-1)}{2}$
	希尔排序	$O(n^{1.3})$	$O(nlog_2n)$	$O(n^2)$	$O(1)$	不稳定	
	折半插入排序	$O(nlog_2n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定	
交换排序	冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定	• n 较小时好，用时间换空间的排序方法 • 最坏情况：把顺序变逆序，或把逆序变顺序，时间复杂度$O(n^2)$只是表示其操作次数的数量级 • 最好情况：数据本来就有序，复杂度为$O(n)$
	快速排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(n^2)$	$O(\log_2n)$~$O(n)$	不稳定	• n 较大时好 • 极端情况：划分之后一边是 1 个，一边是 n-1 个，时间复杂度为$O(n^2)=\frac{n(n-1)}{2}$ • 最好情况：每次都能均匀的划分序列，$O(nlog_2n)$
选择排序	选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定	• n 较小时好
	堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定	• n 较大时好
	归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定	• n 较大时好 • 比较占用内存，内存随 n 的增大而增大 • 效率高且稳定
非比较排序	基数排序	$O(d(r+n))$	$O(d(r+n))$	$O(d(r+n))$	$O(rd+n)$	稳定	
	计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定	
	桶排序	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n\times k)$	--	• 最好情况：每个元素恰好落入不同的桶中，使得桶内元素无需再排序，只需要对各个桶进行合并。即桶的数量接近于元素数量 • 最坏情况：所有元素都被分到同一个桶中，此时桶内元素需要使用其他排序方法进行排序，得到有序数组 • 桶排序的稳定性与桶内排序使用的方法有关

- r 代表关键字基数，d 代表长度，n 代表关键字个数，k 代表整数范围、桶的数量
- 归并排序每次递归都要用到一个辅助表，长度与待排序的表长度相同，虽然递归次数是$O(log_2n)$，但每次递归都会释放掉所占的辅助空间
- 归并排序可以通过手摇算法将空间复杂度降到$O(1)$，但是时间复杂度会提高
- 快速排序空间复杂度通常情况下为$O(log_2n)$，如最坏情况就要$O(n)$的空间，可通过随机化选择 pivot 来将空间复杂度降低到$O(log_2n)$
- 桶排序每个桶需要用到外部存储空间，所以不是原地排序算法

- 时间复杂度记忆

  冒泡、选择、直接排序需要两个 for 循环，每次只关注一个元素，平均时间复杂度为$O(n^2)$，一遍找元素$O(n)$，一遍找位置$O(n)$

  快速、归并、希尔、堆排序基于二分思想，log 以 2 以为底，平均时间复杂度为$O(nlog_2n)$，一遍找元素$O(n)$，一遍找位置$O(log_2n)$
- 不稳定--“快希选堆”

  排序稳定性：排序前后相同元素的相对位置不变，则称排序算法是稳定的；否则排序算法是不稳定的

1、时间复杂度

时间复杂度可以认为是对排序数据的总的操作次数。反映当 n 变化时，操作次数呈现什么规律

常见的时间复杂度有：常数阶$O(1)$，对数阶$O(log_2n)$，线性阶$O(n)$，线性对数阶$O(nlog_2n)$，平方阶$O(n^2)$

时间复杂度$O(1)$：算法中语句执行次数为一个常数，则时间复杂度为$O(1)$

2、空间复杂度

空间复杂度是指算法在计算机内执行时所需存储空间的度量，它也是问题规模 n 的函数

空间复杂度$O(1)$：当一个算法的空间复杂度为一个常量，即不随被处理数据量 n 的大小而改变时，可表示为$O(1)$

空间复杂度$O(log_2n)$：当一个算法的空间复杂度与以 2 为底的 n 的对数成正比时，可表示为$O(log_2n)$

$$a^x=N(a>0，a\neq1)，\ x=log_aN$$

空间复杂度$O(n)$：当一个算法的空间复杂度与 n 成线性比例关系时，可表示为$O(n)$

	顺序表	链表
优点	• 方法简单，各种高级语言中都有数组，容易实现 • 不用为表示结点间的逻辑关系而增加额外的存储开销，存储密度大 • 具有按元素序号随机访问的特点，查找速度快	• 插入、删除时，只要找到对应前驱结点，修改指针即可，无需移动元素 • 采用动态存储分配，不会造成内存浪费和溢出
缺点	• 插入删除操作时，需要移动元素，平均移动大约表中一半的元素，元素较多的顺序表效率低 • 采用静态空间分配，需要预先分配足够大的存储空间，会造成内存的浪费和溢出	• 在有些语言中，不支持指针，不容易实现 • 需要用额外空间存储线性表的关系，存储密度小 • 不能随机访问，查找时要从头指针开始遍历

数据结构	查找		插入		删除		遍历
	平均	最坏	平均	最坏	平均	最坏	
数组	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	--
有序数组	$O(log_2n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
链表	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	--
有序链表	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
二叉查找树	$O(log_2n)$	$O(n)$	$O(log_2n)$	$O(n)$	$O(log_2n)$	$O(n)$	$O(n)$
红黑树	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(n)$
平衡树	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(n)$
二叉堆/优先队列	$O(1)$	$O(1)$	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(log_2n)$	$O(n)$
哈希表	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(n)$

复杂度类型	意义	举例
$O(1)$	耗时与数据规模无关，一次计算后即可找到目标	哈希算法（不考虑冲突）
$O(n)$	数据增大n倍，耗时也增大n倍	遍历算法
$O(n^2)$	对n个数排序需要扫描$n \times m$次	冒泡排序
$O(log_2n)$	当数据增大n倍时，耗时增大log_2n倍。当数据增大 256 倍时，耗时只增大 8 倍	二分查找，256 个数据中查找只要找 8 次就可以
$O(nlog_2n)$	复杂度高于线性低于平方。当数据增大 256 倍时，耗时增大$256 \times 8 = 2048$倍	归并排序