第八章 排序

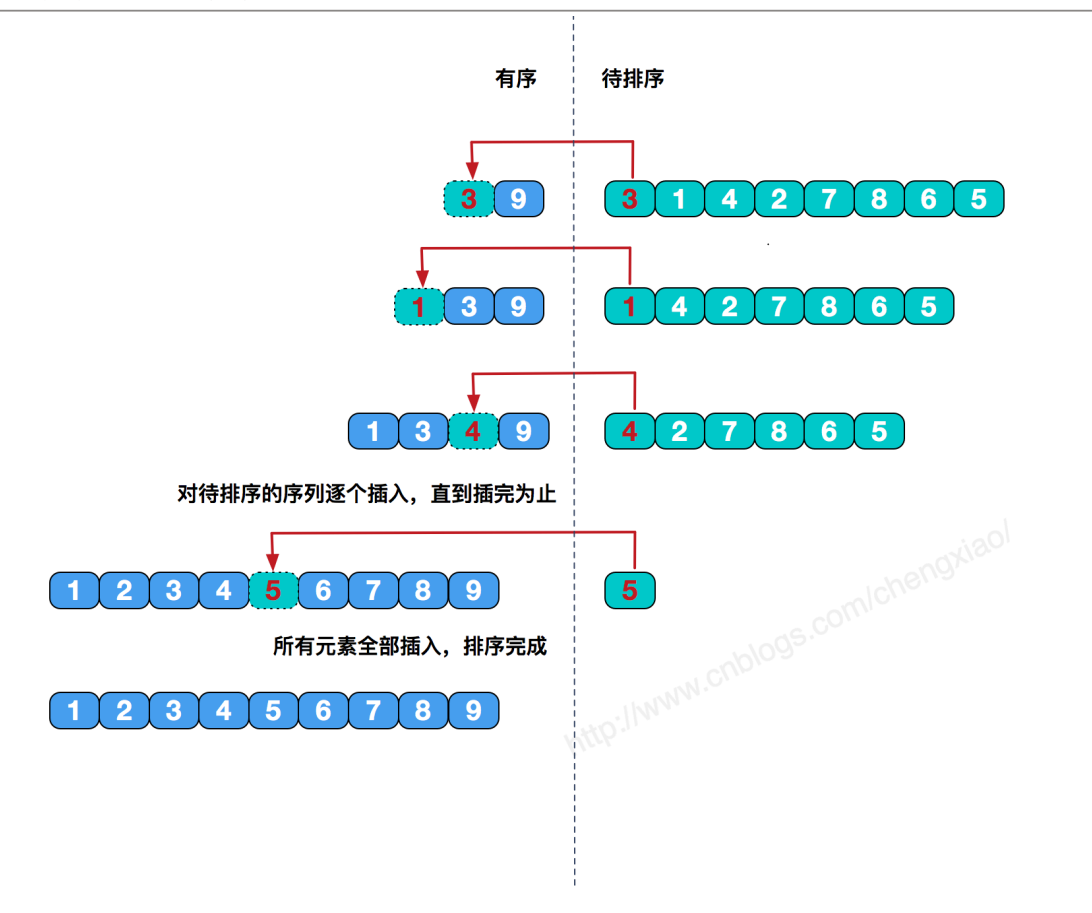
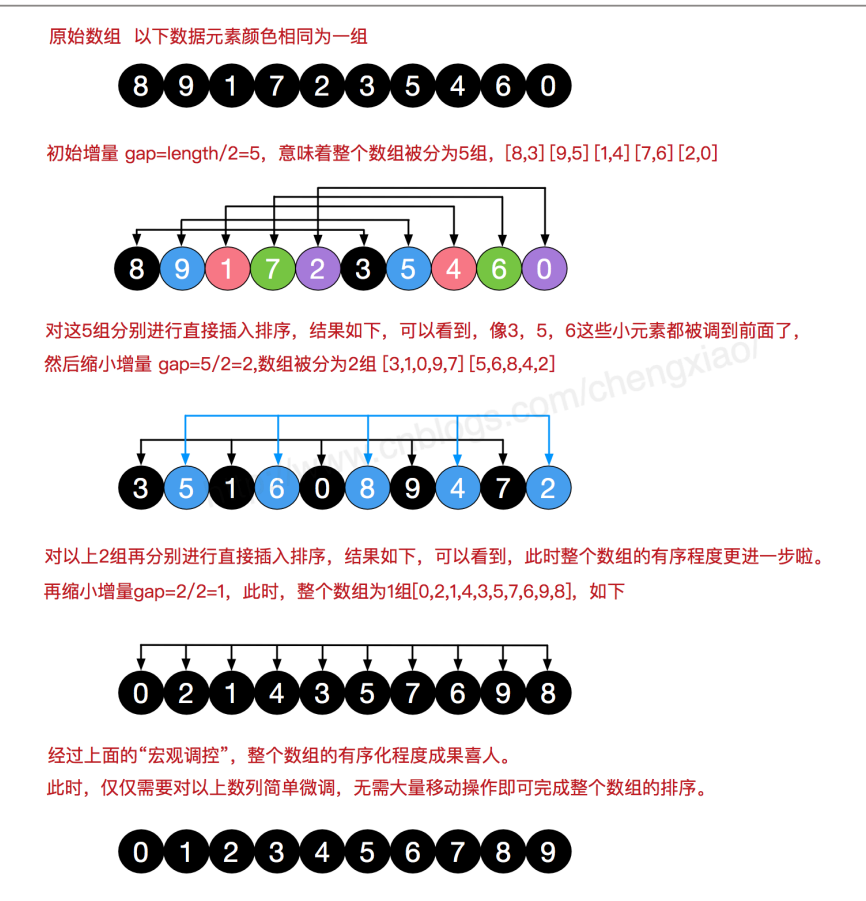
2022年10月2日 星期日 22:56

总结

	最好情况	平均情况	最坏情况	空间复杂度	稳定性	内外排序	数据对象	简述(后续补上)	常考点
直接插入排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定	内排序	数组,链表		• 可能出现: 在最后一趟开始前, 所有元素都不在最终位置 • 待排序序列基本有序的情况下，该方法效率最高 • 最坏情况比较次数 $= \frac{n\times(n-1)}{2}$ • 最好情况比较次数 $= n-1$
希尔排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	O(1)	不稳定	内排序	数组		• 计算希尔排序的增量大小
冒泡排序	O(n)	$O(n^2)$	$O(n^2)$	O(1)	稳定	内排序	数组		• 每一趟最后一个元素都是最大的元素（从小到大的序列） • 元素从大到小时 = 最坏情况比较次数 $= \frac{n\times(n-1)}{2}$ • 元素从小到大时 = 最好情况比较次数 $= n-1$
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$	不稳定	内排序	数组		• 蕴含了分而治之的思想 • 平均性能而言，目前最好的内部排序方法 • 当数据随机或者数据量很大的时候，适合快速排序; 当排序的数据已基本有序，不适合快速排序 • 每次的枢纽值把表分为等长的部分时，速度最快 • 快速排序每趟都把基准元素放在最终位置（常用来计算是不是某一趟排序结果的题目，如第一趟至少有1个元素在最终位置，第二趟至少2个） • 最大递归深度 = 枢纽值每次都将子表等分 = 树高位 $\log_2 n$ • 最小递归深度 = 枢纽值每次都是子表的最大值或最小值 = 单链表 = 树高为n
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	不稳定	内排序	数组，链表		• 不怎么考 • 比较次数数量级与序列初始状态无关
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	O(1)	不稳定	内排序	数组		• 取一大堆数据中的k个最大(最小)的元素时，都优先采用堆排序 • 可以将堆视作一颗完全二叉树，采用顺序存储方式保护堆 • 插入和删除一个新元素的时间复杂度都为 $O(\log_2 n)$ • 构造n个记录的初始堆，时间复杂度为O(n)
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	O(n)	稳定	外排序	数组，链表		• 分阶段可以理解为就是递归拆分子序列的过程，递归深度为$\log_2 n$ • 空间复杂度为O(n) • 比较次数数量级与序列初始状态无关 • 对于N个元素进行k路归并排序排序的趟数满足$k^m = N$
基数排序	O(d(n+r))	O(d(n+r))	O(d(n+r))	O(r)	稳定	外排序	数组，链表		• 通常基数排序第一趟按照个位数字大小，第二趟按照十位数字大小... • MSD是最高位优先，LSD是最低位优先 • 基数排序不能对float和double类型的实数进行排序

名称	数据对象	稳定性	时间复杂度		额外空间复杂度	描述
			平均	最坏		
冒泡排序	数组	✓	$O(n^2)$		$O(1)$	（无序区，有序区）。 从无序区透过交换找出最大元素放到有序区前端。
选择排序	数组	✗	$O(n^2)$		$O(1)$	（有序区，无序区）。 在无序区里找一个最小的元素跟在有序区的后面。对数组：比较得多，换得少。
	链表	✓				
插入排序	数组、链表	✓	$O(n^2)$		$O(1)$	（有序区，无序区）。 把无序区的第一个元素插入到有序区的合适的位置。对数组：比较得少，换得多。
堆排序	数组	✗	$O(n\log n)$		$O(1)$	（最大堆，有序区）。 从堆顶把根卸出来放在有序区之前，再恢复堆。
归并排序	数组	✓	$O(n\log^2 n)$		$O(1)$	把数据分为两段，从两段中逐个选最小的元素移入新数据段的末尾。 可从上到下或从下到上进行。
			$O(n\log n)$		$O(n)+O(\log n)$ 如果不是从下到上	
	链表				$O(1)$	
快速排序	数组	✗	$O(n\log n)$	$O(n^2)$	$O(\log n)$	（小数，基准元素，大数）。 在区间中随机挑选一个元素作基准，将小于基准的元素放在基准之前，大于基准的元素放在基准之后，再分别对小数区与大数区进行排序。
希尔排序	数组	✗	$O(n\log^2 n)$	$O(n^2)$	$O(1)$	每一轮按照事先决定的间隔进行插入排序，间隔会依次缩小，最后一次一定要是1。
计数排序	数组、链表	✓	$O(n+m)$		$O(n+m)$	统计小于等于该元素值的元素的个数i，于是该元素就放在目标数组的索引i位（i≥0）。
桶排序	数组、链表	✓	$O(n)$		$O(m)$	将值为i的元素放入i号桶，最后依次把桶里的元素倒出来。
基数排序	数组、链表	✓	$O(k\times n)$	$O(n^2)$		一种多关键字的排序算法，可用桶排序实现。

第一类：插入排序

	1.直接插入排序【👌】	2.希尔排序【👌】	3.折半插入排序【👌】
原理	某节点前是排好序的，节点后是未排序的 未排序数据从后向前一一比较大小并交换数据 边比较边移动元素	先将整个待排序的记录序列按增量分割成若干子序列 每个子序列分别进行直接插入排序 然后不断减小增量的大小，直到序列有序	先折半查找出元素的待插入位置 然后统一地移动待插入位置后的所有元素
平均复杂度	$O(n^2)$	$O(n\log_2 n)$	$O(n^2)$
稳定性	稳定	不稳定	稳定
常考点	• 可能出现: 在最后一趟开始前, 所有元素都不在最终位置 • 待排序序列基本有序的情况下，该方法效率最高 • 最坏情况比较次数 $= \frac{n\times(n-1)}{2}$ • 最好情况比较次数 $= n-1$	• 计算希尔排序的增量大小	• 折半插入排序的总趟数 = 直接插入排序的 = n–1 • 折半插入排序元素的移动次数 = 元素的移动次数 • 折半插入排序使用辅助空间的数量 = 直接插入排序的 • 两者元素之间的比较次数不同
例图			

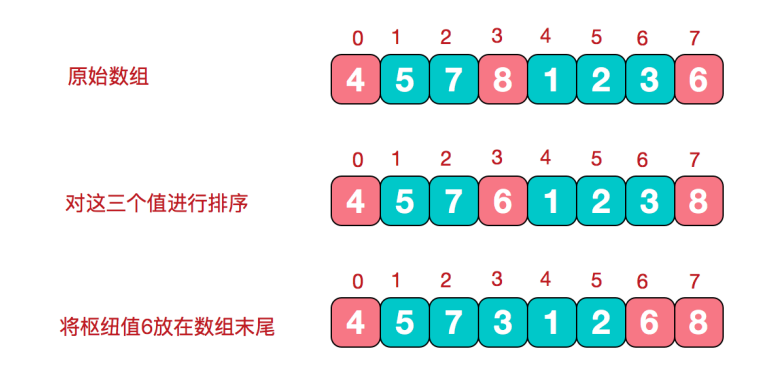
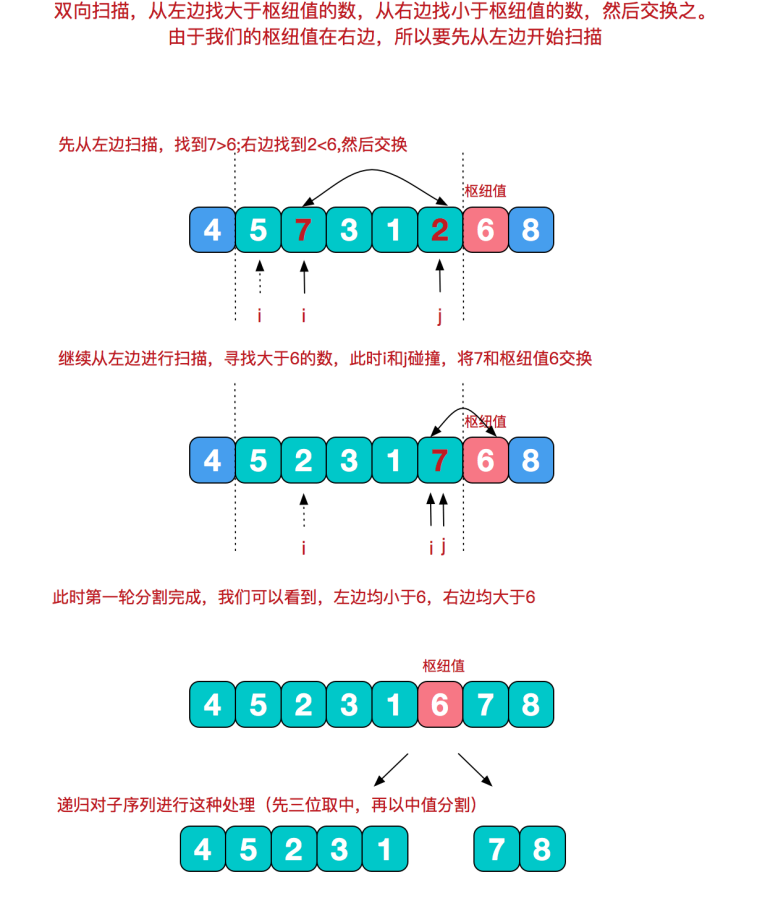
代码

```c
void insertion_sort(int arr[], int len)
{
    int i, j, key;
    for (i = 1; i < len; i++)
    {
        key = arr[i];
        j = i - 1;
        while ((j >= 0) && (arr[j] > key))
        {
            arr[j + 1] = arr[j]; //把大的数值往右换
            j--;
        }
        arr[j + 1] = key; //把小的数值往左换
    }
}
```

```c
void shell_sort(int arr[], int len)
{
    int gap, i, j;
    int temp;
    for (gap = len >> 1; gap > 0; gap >>= 1)
        for (i = gap; i < len; i++)
        {
            temp = arr[i];
            for (j = i - gap; j >= 0 && arr[j] > temp; j -= gap)
                arr[j + gap] = arr[j];
            arr[j + gap] = temp;
        }
}
```

参考链接	1.直接插入排序	2.希尔排序	3.折半插入排序
	https://www.runoob.com/w3cnote/insertion-sort.html	https://www.runoob.com/w3cnote/shell-sort.html https://www.cnblogs.com/chengxiao/p/6104371.html	

第二类：交换排序【交换类排序的排序趟数与原始状态有关】

	1.冒泡排序【👌】	2.快速排序【👌】
原理	• 比较相邻的元素，如果第一个比第二个大，就交换他们两个 • 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对 • 上一步做完后，最后的元素会是最大的数 • 针对所有的元素重复以上的步骤，除了最后一个 • 直到没有任何一对数字需要比较	• 通过一趟排序将要排序的数据分割成独立的两部分 • 其中一部分的所有数据都比另外一部分的所有数据都要小 • 然后再按此方法对这两部分数据分别进行快速排序 • 整个排序过程可以递归进行，以此达到整个数据变成有序序列
平均复杂度	$O(n^2)$	O(nlogn)
稳定性	稳定	不稳定
常考点	• 每一趟最后一个元素都是最大的元素（从小到大的序列） • 元素从小到大时 = 最坏情况比较次数 $= \frac{n\times(n-1)}{2}$ • 元素从小到大时 = 最好情况比较次数 $= n-1$	• 蕴含了分而治之的思想 • 平均性能而言，目前最好的内部排序方法 • 当数据随机或者数据量很大的时候，适合快速排序; 当排序的数据已基本有序，不适合快速排序 • 每次的枢纽值把表分为等长的部分时，速度最快 • 快速排序每趟都把基准元素放在最终位置（常用来计算是不是某一趟排序结果的题目，如第一趟至少有1个元素在最终位置，第二趟至少2个） • 最大递归深度 = 枢纽值每次都将子表等分 = 树高位 $\log_2 n$ • 最小递归深度 = 枢纽值每次都是子表的最大值或最小值 = 单链表 = 树高为n
例图		三位取中 【考试通常取最左边的元素为枢纽值】  根据枢纽值进行分割 【考试一般先从右到左交换比枢纽值小的,再从左到右交换比枢纽值大的】  根据枢纽值进行分割 

代码

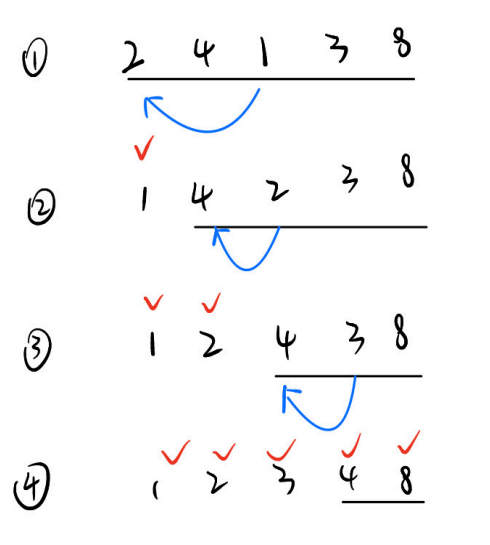
```c
#include <stdio.h>
void bubble_sort(int arr[], int len)
{
    int i, j, temp;
    for (i = 0; i < len - 1; i++)
        for (j = 0; j < len - 1 - i; j++)
            if (arr[j] > arr[j + 1])
            {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
}
int main()
{
    int arr[] = {22, 34, 3, 32, 82, 55, 89, 50, 37, 5, 64, 35, 9, 70};
    int len = (int)sizeof(arr) / sizeof(*arr);
    bubble_sort(arr, len);
    int i;
    for (i = 0; i < len; i++)
        printf("%d ", arr[i]);
    return 0;
}
```

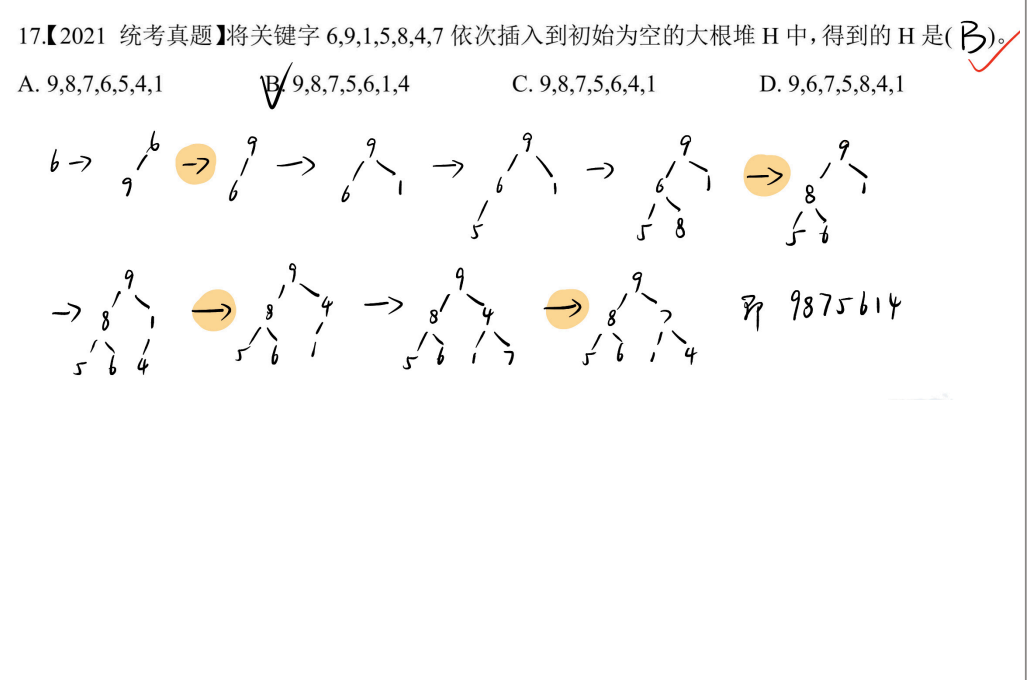
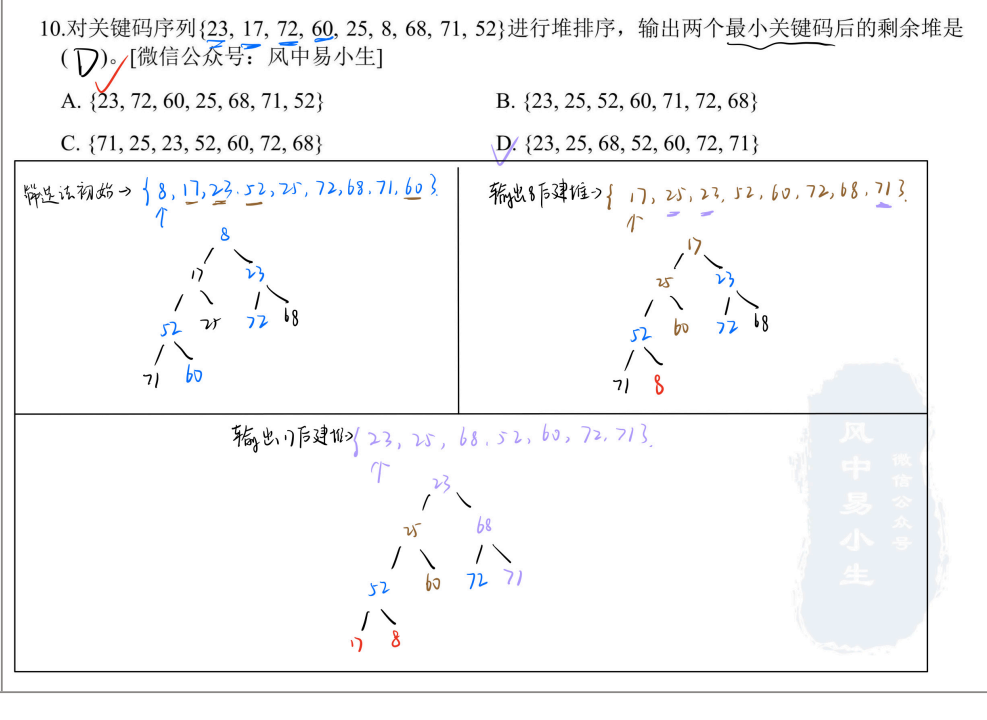
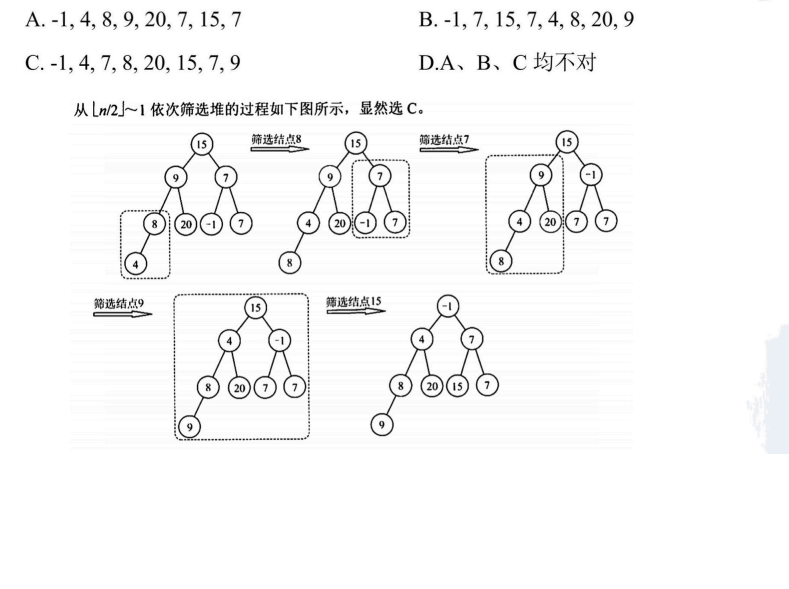
```c
void Qsort(int A[], L, R){       //a 数组保存数据，L 和 R 是边界
  if (L>=R) return;              //当前区间元素个数<=1 则退出
  int key, i=L, j=R;             //i 和 j 是左右两个数组下标移动
  把 A[L~R]中随机一个元素和 A[L]交换  //快排优化，使得基准值的选取随机
  key=A[L];                      //key 作为枢值参与比较
  while (i<j){
    while (i<j && A[j]>key)
        j--;
    while (i<j && A[i]<=key)
        i++;
    if (i<j)
        swap(A[i], A[j]);        //交换 A[i]和 A[j]
  }
  swap(A[L], A[i]);
  Qsort(A, L, i-1);              //递归处理左区间
  Qsort(A, i+1, R);              //递归处理右区间
}
```

【背这个代码】

动画图链接	1.冒泡排序	2.快速排序
	https://www.runoob.com/w3cnote/bubble-sort.html https://www.cnblogs.com/chengxiao/p/6103002.html	https://www.runoob.com/w3cnote/quick-sort-2.html https://www.cnblogs.com/chengxiao/p/6262208.html

第三类：选择排序

	1.简单选择排序【👌】	2.堆排序【👌】
原理	• 每一趟从待排序的数据元素中选择最小(或最大)的一个元素作为首元素，直到所有元素排完为止	• 堆是具有特殊性质的完全二叉树 • 大顶堆：arr[i] >= arr[2i+1] && arr[i] >= arr[2i+2] 【根最大】 • 小顶堆：arr[i] <= arr[2i+1] && arr[i] <= arr[2i+2] 【根最小】 • 堆的基本思想： a.将无序序列构建成一个堆，根据升序降序需求选择大顶堆或小顶堆 b.将堆顶元素与末尾元素交换，将最大元素"沉"到数组末端 c.重新调整结构，使其满足堆定义，然后继续交换堆顶元素与当前末尾元素 d.反复执行调整+交换步骤，直到整个序列有序
平均复杂度	$O(n^2)$	O(nlogn)
稳定性	不稳定	不稳定
常考点	• 不怎么考 • 比较次数数量级与序列初始状态无关	• 取一大堆数据中的k个最大(最小)的元素时，都优先采用堆排序 • 堆排序的序列相当于二叉树的层次序列 • 可以将堆视作一颗完全二叉树，采用顺序存储方式保护堆 • 插入和删除一个新元素的时间复杂度都为 $O(\log_2 n)$ • 构造n个记录的初始堆，时间复杂度为O(n)
例图		https://www.cnblogs.com/chengxiao/p/6129630.html
动画图链接	https://www.runoob.com/w3cnote/selection-sort.html	https://www.runoob.com/w3cnote/heap-sort.html

掌握建堆的过程	掌握堆增加一个元素的过程 掌握堆删除/输出一个元素的过程	掌握大顶堆和小顶堆的概念
		

17.【2021 统考真题】将关键字 6,9,1,5,8,4,7 依次插入到初始为空的大根堆 H 中，得到的 H 是(B)。

A. 9,8,7,6,5,4,1　B. 9,8,7,5,6,1,4　C. 9,8,7,5,6,4,1　D. 9,6,7,5,8,4,1

10.对关键码序列{23, 17, 72, 60, 25, 8, 68, 71, 52}进行堆排序，输出两个最小关键码后的剩余堆是(D)。[微信公众号：风中易小生]

A. {23, 72, 60, 25, 68, 71, 52}　B. {23, 25, 52, 60, 71, 72, 68}

C. {71, 25, 23, 52, 60, 72, 68}　D. {23, 25, 68, 52, 60, 72, 71}

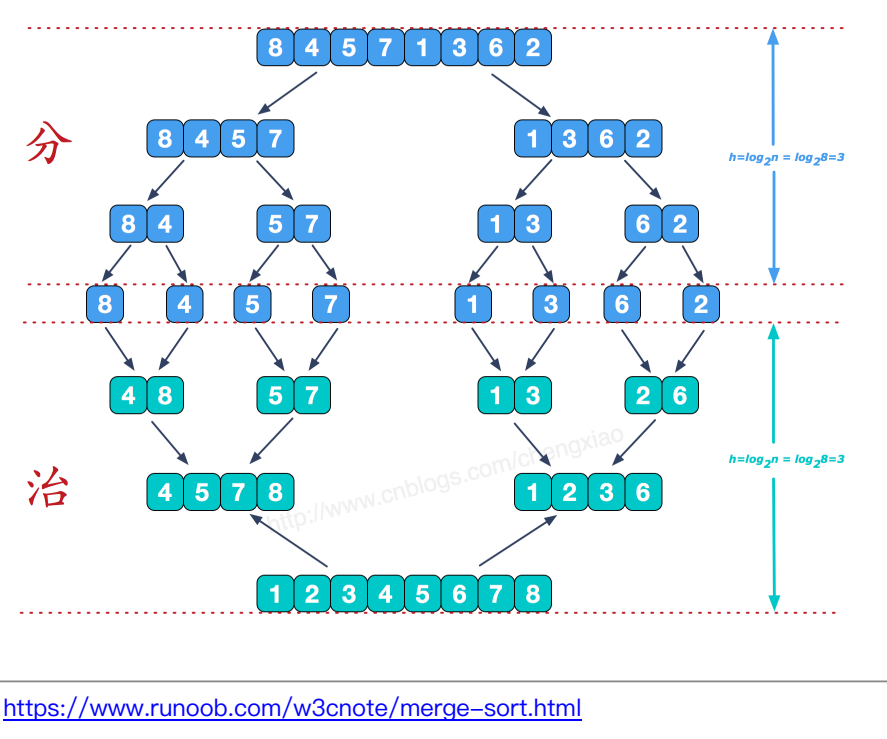
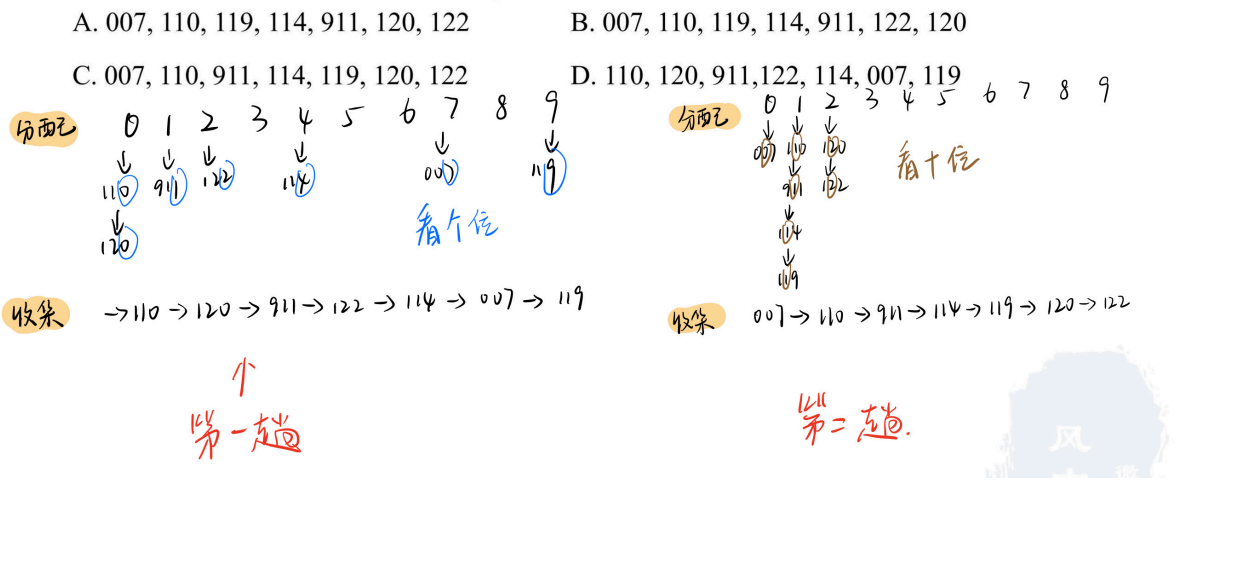
6.有一组数据(15, 9, 7, 8, 20, -1, 7, 4)，用堆排序的筛选方法建立的初始小根堆为(C)。

A. -1, 4, 8, 9, 20, 7, 15, 7　B. -1, 7, 15, 7, 4, 8, 20, 9

C. -1, 4, 7, 8, 20, 15, 7, 9　D.A、B、C 均不对

从⌊n/2⌋~1 依次筛选堆的过程如下图所示，显然选 C。

第四类：归并排序和基数排序

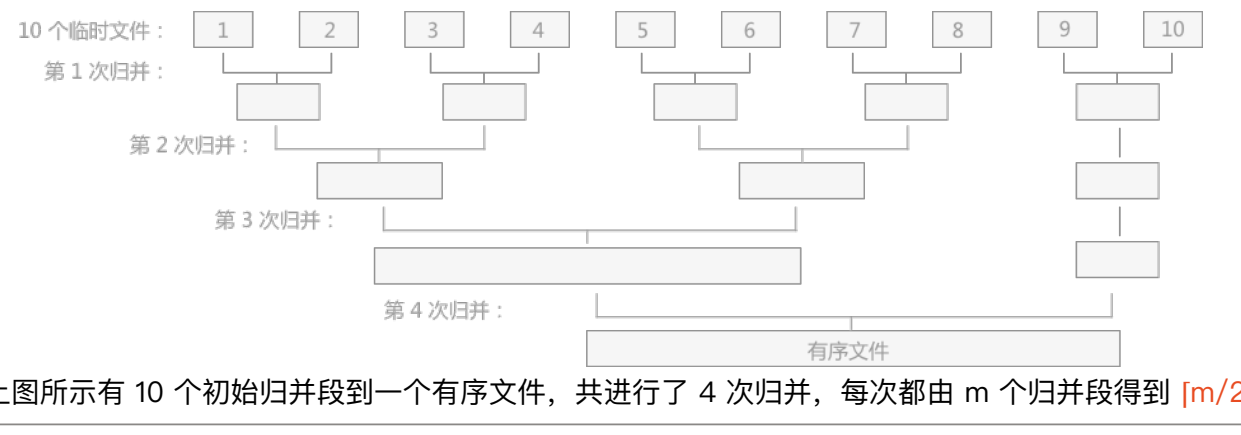
	1.归并排序【👌】	2.基数排序【👌】
原理	• 利用归并的思想实现的排序方法 • 该算法采用经典的分治策略 • 分治法将问题分(divide)成一些小的问题然后递归求解 • 治(conquer)的阶段则将分的阶段得到的各答案"修补"在一起。	• 将整数按位数切割成不同的数字，然后按每个位数分别比较
平均复杂度	O(nlogn)	O(n*k)
稳定性	稳定	稳定，外部排序
常考点	• 分阶段可以理解为就是递归拆分子序列的过程，递归深度为$\log_2 n$ • 空间复杂度为O(n) • 比较次数数量级与序列初始状态无关 • 对于N个元素进行k路归并排序排序的趟数满足$k^m = N$	• 通常基数排序第一趟按照个位数字大小，第二趟按照十位数字大小... • MSD是最高位优先，LSD是最低位优先 • 基数排序不能对float和double类型的实数进行排序
例图		
动画图链接	https://www.runoob.com/w3cnote/merge-sort.html https://www.cnblogs.com/chengxiao/p/6194356.html	https://www.runoob.com/w3cnote/counting-sort.html

11.【2013 统考真题】对给定的关键字序列 110, 119, 007, 911, 114, 120, 122 进行基数排序，第 2 趟分配收集后得到的关键字序列是(C)。

A. 007, 110, 119, 114, 911, 120, 122　B. 007, 110, 119, 114, 911, 122, 120

C. 007, 110, 911, 114, 119, 120, 122　D. 110, 120, 911,122, 114, 007, 119

第五类：外部排序【未完全搞懂】

概念	• 当待排序的文件比内存的可使用容量还大时，文件无法一次性放到内存中进行排序 • 需要借助于外部存储器（例如硬盘、U盘、光盘），这时就需要用到外部排序算法来解决
外部排序的阶段	1. 按照内存大小，将大文件分成若干长度为 l 的子文件（l 应小于内存的可使用容量） 2. 然后将各个子文件依次读入内存，使用适当的内部排序算法对其进行排序(排好序的子文件统称为"归并段"或者"顺段") 3. 将排好序的归并段重新写入外存，为下一个子文件排序腾出内存空间 4. 对得到的顺段进行合并，直至得到整个有序的文件为止
举例	有一个含有 10000 个记录的文件，但是内存的可使用容量仅为 1000 个记录，需要使用外部排序算法，具体步骤如下： 1. 将整个文件其等分为 10 个临时文件（每个文件中含有 1000 个记录） 2. 然后将这 10 个文件依次进入内存，采取适当的内存排序算法对其中的记录进行排序，将得到的有序文件（初始归并段）移至外存 3. 对得到的 10 个初始归并段进行如下图的两两归并，直至得到一个完整的有序文件  如上图所示有 10 个初始归并段到一个有序文件，共进行了 4 次归并，每次都由 m 个归并段得到 ⌈m/2⌉ 个归并段，这种归并方式被称为 2–路平衡归并
效率分析	• 影响整体排序效率的因素主要取决于读写外存的次数，即访问外存的次数越多，算法花费的时间就越多，效率就越低 • 对于同一个文件来说，对其进行外部排序时访问外存的次数同归并的次数成正比，即归并操作的次数越多，访问外存的次数就越多 • 提高效率的方法：1、增加 k–路平衡归并中的 k 值 2、尽量减少初始归并段的数量 m，即增加每个归并段的容量；
常用公式	• 对于具有 m 个初始归并段进行 k–路平衡归并时，归并的次数为：s=⌊logk m⌋（其中 s 表示归并次数，注意是向下取整）
参考链接	什么是外部排序算法 http://data.biancheng.net/view/76.html 多路平衡归并排序算法 http://data.biancheng.net/view/77.html 置换选择排序算法 http://data.biancheng.net/view/78.html 最佳归并树 http://data.biancheng.net/view/79.html