第八章 排序

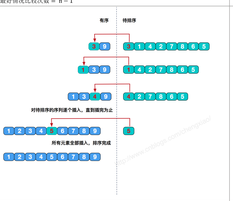
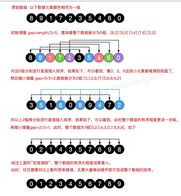
2022年10月12日 星期日 22:56

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	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 • 快速排序得到的把原始元素定位最终位置（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，第二天至少2个） • 最大递归深度 = 枢纽值每次都等于表等分 = 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路归并排序序列的趟数满足$k^m = N$
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 平均	时间复杂度 最坏	额外空间复杂度	描述
冒泡排序	数组	✓	$O(n^2)$		$O(1)$	元序区，有序区）。 从元序区通过交换找出最大元素放到有序区前端。
选择排序	数组	✗	$O(n^2)$		$O(1)$	元序区，无序区）。 在元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	✓	$O(n^2)$		$O(1)$	元序区，无序区）。 把元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	✓	$O(n\log^2 n)$		$O(n)$	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。 如果不是从下往上，可从上到下递归下去，进行。
	链表		$O(n\log n)$		$O(1)$	
快速排序	数组	✗	$O(n\log n)$	$O(n^2)$	$O(\log n)$	分成，基准元素，大部分。 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，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，间隔会依次减小，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，于是该元素就放在目标数组的索引位 $(n-1)$ 。
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。

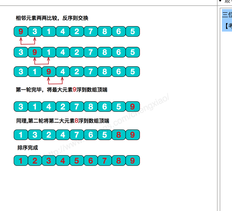
第一类：插入排序

	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	`void insertion_sort(int arr[], int len)`
	`{`
	`  int j, 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`
	`  }`
	`}`
		`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	https://www.runoob.com/w3cnote/insertion-sort.html	https://www.runoob.com/w3cnote/shell-sort.html https://www.cnblogs.com/chenqiaiao/p/8104371.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】

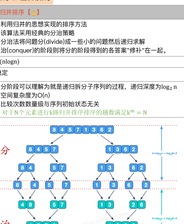
	1.冒泡排序【...】	2.快速排序【...】
原理	• 比较相邻的元素。如果第一个比第二个大，就交换他们两个 • 对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对 • 上一步做完后，最后的元素会是最大的数 • 针对所有的元素重复以上的步骤，除了最后一个 • 直到没有任何一对数字需要比较	• 通过一趟排序将要排序的数据分割成独立的两部分 • 其中一部分的所有数据都比另外一部分的所有数据都要小 • 然后再按此方法对这两部分数据分别进行快速排序 • 整个排序过程可以递归进行 • 以此达到整个数据变成有序序列
平均复杂度	$O(n^2)$	O(nlogn)
稳定性	稳定	稳定
常考点	• 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 $= n-1$	• 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 • 快速排序得到的把原始元素定位最终位置（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，第二天至少2个） • 最大递归深度 = 枢纽值每次都等于表等分 = 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		

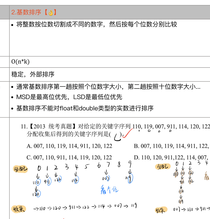
代码	`#include <stdio.h>`
	`void bubble_sort(int arr[], int len)`
	`{`
	`  int i, j, temp;`
	`  for (i = 0; i < len - 1; i++)`
	`  for (j = 0; j < len - 1 - i; j++)`
	`  if (arr[j] > arr[j + 1])`
	`  {`
	`  temp = arr[j];`
	`  arr[j] = arr[j + 1];`
	`  arr[j + 1] = temp;`
	`  }`
	`}`
	`int main()`
	`{`
	`  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;`
	`}`
		`void swap(int *x, int *y)`
		`{`
		`  int t = *x;`
		`  *x = *y;`
		`  *y = t;`
		`}`
		`void quick_sort_recursive(int arr[], int start, int end)`
		`{`
		`  if (start >= end)`
		`  return;`
		`  int mid = arr[end];`
		`  int left = start, right = end - 1;`
		`  while (left < right)`
		`  {`
		`  while (arr[left] < mid && left < right)`
		`  left++;`
		`  while (arr[right] >= mid && left < right)`
		`  right--;`
		`  swap(&arr[left], &arr[right]);`
		`  }`
		`  if (arr[left] >= arr[end])`
		`  swap(&arr[left], &arr[end]);`
		`  else`
		`  left++;`
		`  if (left)`
		`  quick_sort_recursive(arr, start, left - 1);`
		`  quick_sort_recursive(arr, left + 1, end);`
		`}`
		`void quick_sort(int arr[], int len)`
		`{`
		`  quick_sort_recursive(arr, 0, len - 1);`
		`}`

动画图链接	https://www.runoob.com/w3cnote/bubble-sort.html https://www.cnblogs.com/chenqiaiao/p/8100002.html	https://www.runoob.com/w3cnote/quick-sort-2.html https://www.cnblogs.com/chenqiaiao/p/9262208.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.将无需序列构建成一个堆，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 • 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





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)的阶段将阶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路归并排序序列的趟数满足$k^m = N$	• 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

第五类：外部排序