排序算法

朱睿

June 13, 2019

大纲

注意：除基数排序使用链表实现外，本文中其他排序方式默认使用数组实现，与课本相同。

直接插入排序

- 时间复杂度：最优 $O(n)$，平均最坏 $O(n^2)$
- 额外空间复杂度：$O(1)$
- 稳定性：✓
- 适用场景：
  - 需要排序的数据量很小
  - 已知输入元素大致上按照顺序排列
  - 在 STL 的 sort 算法中，将插入排序作为快速排序的补充，用于少量元素的排序（通常为 8 个或以下）

3	5	2	1	6	8	7	4
3	5	2	1	6	8	7	4
2	3	5	1	6	8	7	4
1	2	3	5	6	8	7	4
1	2	3	5	6	8	7	4
1	2	3	5	6	8	7	4
1	2	3	5	6	7	8	4
1	2	3	4	5	6	7	8

直接插入排序

```
void insertion_sort(int arr[], int len) {
  for (int i = 1; i < len; i++) {
    int key = arr[i];
    int j = i - 1;
    while ((j >= 0) && (key < arr[j])) {
      arr[j + 1] = arr[j];
      j--;
    }
    arr[j + 1] = key;
  }
}
```

二分插入排序

- 将顺序查找优化为二分查找
- 时间复杂度：最优 $O(n)$，平均最坏还是$O(n^2)$：比较 $O(n \log n)$，移动 $O(n^2)$
- 额外空间复杂度：$O(1)$
- 稳定性：✓
- 减少比较次数但不减少移位次数
- 代码略

希尔排序

- 为啥要有希尔排序：
  - 插入排序中如果一个元素与正确位置相隔较远，需要多次移位
  - 希尔排序先比较较远的元素，减少移位次数
- 关键点：一个 h_k 有序的数组经过 h_j 排序后仍然是 h_k 有序的 $(j < k)$，证明略

初始	12	11	10	9	8	7	6	5	4	3	2	1
5 排序	2	1	5	4	3	7	6	10	9	8	12	11
3 排序	2	1	5	4	3	7	6	10	9	8	12	11
1 排序	1	2	3	4	5	6	7	8	9	10	11	12

希尔排序

- 为啥要有希尔排序：
  - 插入排序中如果一个元素与正确位置相隔较远，需要多次移位
  - 希尔排序先比较较远的元素，减少移位次数
- 关键点：一个 h_k 有序的数组经过 h_j 排序后仍然是 h_k 有序的 $(j < k)$，证明略
- 时间复杂度：比 $O(n^2)$ 要好，比 $O(n \log n)$ 要差
- 额外空间复杂度：$O(1)$
- 稳定性：✗

希尔排序

```
void shell_sort(int arr[], int len) {
  int h = 1, tmp;
  while (h < len / 3) {
    h = 3 * h + 1;       // e.g. 增量序列为 {1, 4, 13, ...}
  }
  while (h >= 1) {
    for (int i = h; i < len; i++) {
      for (int j = i; j >= h && arr[j] < arr[j - h];
           j -= h) {
        tmp = arr[j];
        arr[j] = arr[j - h];
        arr[j - h] = tmp;
      }
    }
    h = h / 3;
  }
}
```

直接选择排序

- 时间复杂度：最优平均最坏都是 $O(n^2)$
- 额外空间复杂度：$O(1)$
- 稳定性：✗

3	5	2	1	6	8	7	4
1	5	2	3	6	8	7	4
1	2	5	3	6	8	7	4
1	2	3	5	6	8	7	4
1	2	3	4	6	8	7	5
1	2	3	4	5	8	7	6
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

直接选择排序

```
void selection_sort(int arr[], int len) {
  for (int i = 0; i < len - 1; i++) {
    int min = i, tmp;
    for (int j = i + 1; j < len; j++)
      if (arr[j] < arr[min])
        min = j;
    if (i != min) {
      tmp = arr[i];
      arr[i] = arr[min];
      arr[min] = tmp;
    }
  }
}
```

堆排序

- 算法：
  1. 建堆：对每个非叶子节点调用向下过滤
  2. 出堆：执行 $n-1$ 次 deQueue()
- 时间复杂度：最优平均最坏都是 $O(n \log n)$：建堆 $O(n)$（教材 P220 定理 7.1）+ 出堆 $O(n \log n)$
- 额外空间复杂度：$O(1)$
- 稳定性：✗
- 代码：略（堆那章全都有）
- 适用场景：元素较多

冒泡排序

- 时间复杂度：最优 $O(n)$，平均最坏 $O(n^2)$
- 额外空间复杂度：$O(1)$
- 稳定性：✓
- 鸡尾酒排序：来回冒泡

3	5	2	1	6	8	7	4
3	2	1	5	6	7	4	8
2	1	3	5	6	4	7	8
1	2	3	5	4	6	7	8
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8

冒泡排序

```c
void bubble_sort(int arr[], int len) {
  int i, j, tmp;
  for (i = 0; i < len - 1; i++)
    for (j = 0; j < len - 1 - i; j++)
      if (arr[j] > arr[j + 1]) {
        tmp = arr[j];
        arr[j] = arr[j + 1];
        arr[j + 1] = tmp;
      }
}
```

快速排序

- 思想：挖坑填坑，递归分治
- 时间复杂度：最优平均 $O(n \log n)$，最坏 $O(n^2)$（证明见课本 P338-340）
- 额外空间复杂度：$O(\log n)$
- 稳定性：✗

3	5	2	1	6	8	7	4
	5	2	1	6	8	7	4
l						⇐	r
1	5	2		6	8	7	4
l	⇒		r				
1		2	5	6	8	7	4
	l	⇐	r				
1	2		5	6	8	7	4
	l	r					
1	2	3	5	6	8	7	4

快速排序

```
void quick_sort(int s[], int l, int r) {
  if (l < r) {
    int i = l, j = r, x = s[l];
    while (i < j) {
      while (i < j && s[j] >= x) // 从右向左找第一个小于 x 的数
        j--;
      if (i < j)
        s[i++] = s[j];
      while (i < j && s[i] < x) // 从左向右找第一个大于等于 x 的数
        i++;
      if (i < j)
        s[j--] = s[i];
    }
    s[i] = x;
    quick_sort(s, l, i - 1); // 递归调用
    quick_sort(s, i + 1, r);
  }
}
```

归并排序

- 时间复杂度：最好平均最坏均为 $O(n \log n)$
- 额外空间复杂度：$O(n)$
- 稳定性：✓
- 应用：求逆序数

3	5	2	1	6	8	7	4
3	5	1	2	6	8	4	7
1	2	3	5	4	6	7	8
1	2	3	4	5	6	7	8

归并排序

```
void merge_sort_recursive(int arr[], int reg[], int start, int end) {
 if (start >= end)
  return;
 int len = end - start, mid = (len >> 1) + start;
 int start1 = start, end1 = mid;
 int start2 = mid + 1, end2 = end;
 merge_sort_recursive(arr, reg, start1, end1);
 merge_sort_recursive(arr, reg, start2, end2);
 int k = start;
 while (start1 <= end1 && start2 <= end2)
  reg[k++] = arr[start1]<arr[start2] ? arr[start1++] : arr[start2++];
 while (start1 <= end1)
  reg[k++] = arr[start1++];
 while (start2 <= end2)
  reg[k++] = arr[start2++];
 for (k = start; k <= end; k++)
  arr[k] = reg[k];
}
```

基数排序（口袋排序）

- 从低位到高位按位比较
- 时间复杂度：$O(digit * (n + decimal)) = O(digit * n)$
- 额外空间复杂度：$O(decimal)$
- 稳定性：✓
- 优点：不用比较，直接往口袋里装，某些场景下（比如比较耗时久）优于快速排序
- 代码：略，见课本 P345

总结

名称	数据对象	稳定性	时间复杂度			额外空间复杂度
			最好	平均	最坏	
直接插入排序	数组	✓	$O(n)$	$O(n^2)$		$O(1)$
二分插入排序	数组	✓	$O(n)$	$O(n^2)$		$O(1)$
希尔排序	数组	✗	$O(n \log^2 n) \sim O(n^2)$			$O(1)$
直接选择排序	数组	✗	$O(n^2)$			$O(1)$
堆排序	数组	✗	$O(n \log n)$			$O(1)$
冒泡排序	数组	✓	$O(n)$	$O(n^2)$		$O(1)$
快速排序	数组	✗	$O(n \log n)$		$O(n^2)$	$O(\log n)$
归并排序	数组	✓	$O(n \log n)$			$O(n)$
基数排序	链表	✓	$O(digit * n)$			$O(decimal)$