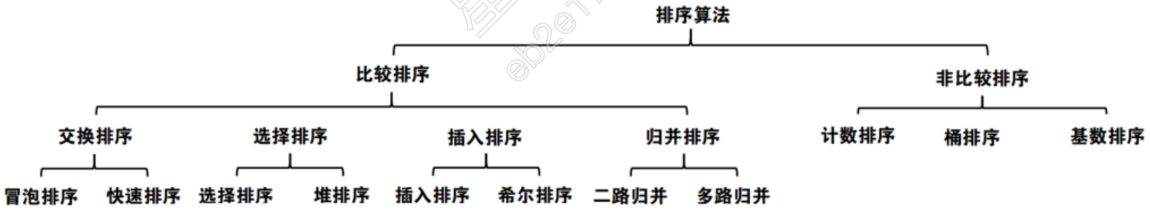
第六章 排序算法1

1. 排序算法

1.1 排序算法

- 排序算法
  排序就是将一组元素按照特定的顺序进行排列。 可以按照升序 （从小到大） 或降序 （从大到小） 的方式对元素进行排序。
  排序过程一般都要进行元素的比较和交换。
  在处理大量数据时，会对算法的效率有要求，需要考虑到数据的各种限制导我一种合适的排序算 法， 因此便演变出多种排序算法
- 常见的排序算法
  比较排序包括交换排序、 选择排序、 插入排序、 归并排序。 其中，交换排序包括冒泡排序和快速排序；选择排序包括选择排序和堆排序；插入排序包括插入排序和希尔排序； 归并排序包括二路归并排序和多路归并排序
  非比较排序包括计数排序 桶排序、 基数排序



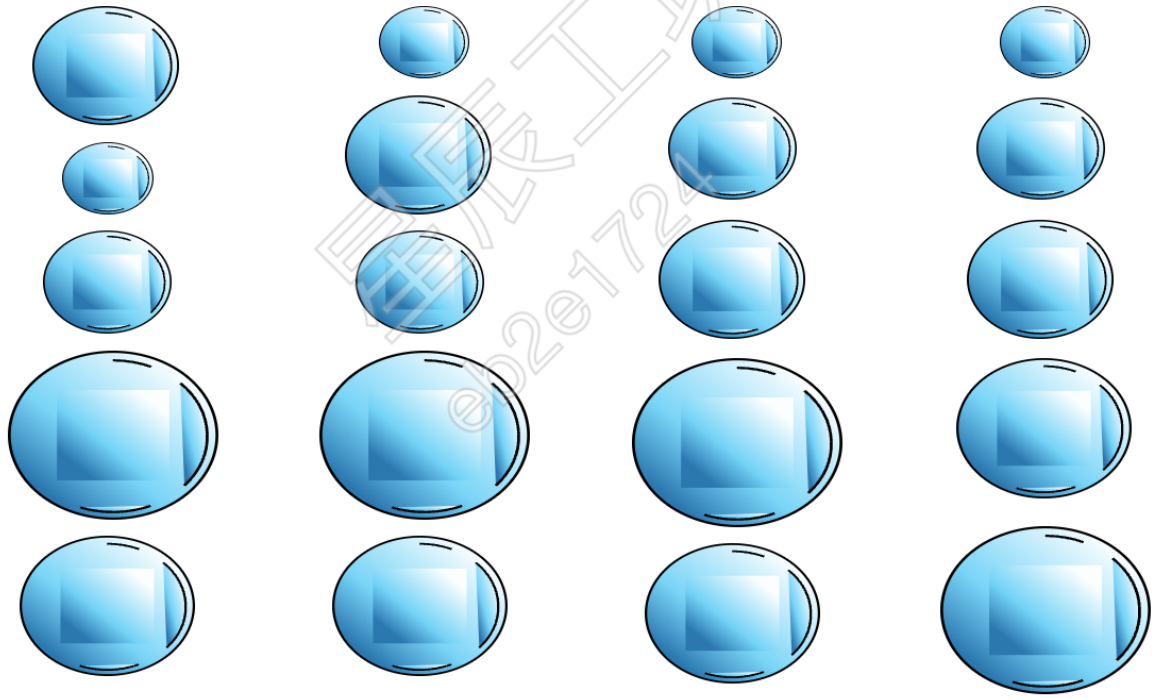
- 排序算法的优缺点
  各个排序算法都有各自的优缺点，这个优缺点主要是从三个方面来评价：时间复杂度， 空间复杂度 ，算法稳定性

排序算法	平均时间复杂度	最好情况	最坏情况	空间复杂度	稳定性
冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
插入排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
希尔排序	$O(n \log n)$	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(1)$	不稳定
归并排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	稳定
快速排序	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	不稳定
堆排序	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	不稳定
计数排序	$O(n + k)$	$O(n + k)$	$O(n + k)$	$O(k)$	稳定
桶排序	$O(n + k)$	$O(n + k)$	$O(n^2)$	$O(n + k)$	稳定
基数排序	$O(n \times k)$	$O(n \times k)$	$O(n \times k)$	$O(n + k)$	稳定

2. 冒泡排序

1. 概述

冒泡排序的名字由来是因为特定的元素会经由交换慢慢“浮”到数列的顶端， 就如同碳酸饮料中二氧化碳的气泡最终会上浮到顶端一样，故名“冒泡排序”



- 基本思想
  冒泡排序是一种简单的排序算法，假定从小到大排序，就从序列前边的第一个元素开始，依次对 相邻的两个元素比较，如果前边的元素大于后边的元素，就交换它们的位置，否则继续比较后边 相邻的元素。 直到所有元素成为有序序列
  降序排列原理类似， 如果前边的元素小于后边的元素，就交换它们的位置， 否则继续比较后边柜邻的元素。直到所有元素成为有序序列

- 排序步骤

1. 从待排序序列中的第一个元素开始，比较相邻的两个元素；
2. 如果前一个元素大于后一个元素，则交换这两个元素的位置；
3. 继续比较下一对相邻元素，重复步骤2，直到本轮元素操作完成
4. 重复上述1、2、3过程；
5. 直到所有元素都排序完成

- 代码实现

```
1. //arr:排序的序列; n:序列长度
2. void bubbleSort(int arr[], int n) {
3.     //外层循环表示循环的轮数，需要遍历n-1轮
4.     for (int i = n - 1; i > 0; i--) {
5.         //内层循环表示每一轮元素需要比较的次数
6.         for (int j = 0; j < i; j++) {
7.             //如果前边的元素大于后边元素，则交换它们的位置
8.             if (arr[j] > arr[j + 1]) {
9.                 //交换两个相邻元素的位置
10.                swap(arr[j], arr[j + 1]);
11.            }
12.        }
13.    }
14. }
```

2. 冒泡排序的性能分析

冒泡排序性能分析

1. 冒泡排序是一种简单但效率不高的排序算法，它的时间复杂度为 $O(n^2)$。 它适用于小规模数据的排序，但 对于大规模数据来说效率较低。 冒泡排序的优点是实现简单易懂，而缺点是效率较低。 在实际应用中，更常 更用更高效的排序算法，如快速排序 归并排序等。 2最好和最环及平均时间复杂度情况：
   1） 当输入序列已经有序时，冒泡排序只需进行一次完整的遍历，即可判断序列已经有序，不需要进行 任何交换操作。 因此，量最好情况下的时间复杂度为O(n)
   2） 当输入序列完全逆序排列时，冒泡排序需要进行n-1趟遍历，并且每超遍历都需要比较和交换相邻元素的位置。因此，最坏情况下的时间复杂度为 0（n²）。
   3） 对于长度为n的序列， 冒泡排序的平均情况下需要进行n/2 超遍历，每趟遍历需要比较和交换的次 数约为n/2。 因此， 平均情况下的时间复杂度为 0（n²）。
2. 冒泡排序只需要使用常数级别的额外空间来存储临时变量，因此空间复杂度为 0（1）。
   冒泡排序是一种稳定的排序算法 相同元素的相对位置在排序前后不会发生改变

2. 选择排序

选择排序的基本思想是每次从待排序的数据中选取最小 （或最大） 的元素，将其与待排序中的第 一个元素交换位置，直到排序完成

- 排序步骤

对数据 { k_1, k_2, k_3, ..., k_n } 进行升序排列：

第 1 趟从 k_1 到 k_n 中选择最小元素，将其与 k_1 交换；

第 2 趟从 k_2 到 k_n 中选择最小元素，将其与 k_2 交换；

第 3 趟从 k_3 到 k_n 中选择最小元素，将其与 k_3 交换；

以此类推；

最后一趟从 k_{n-1} 到 k_n 中选择最小元素，将其与 k_{n-1} 交换；

此时所有元素都排序完成。

- 参考代码

```
// arr: 序列; n: 序列长度
void selectionSort(int arr[], int n){
    // 外层循环表示循环的趟数，需要遍历 n-1 趟
    for (int i = 0; i < n - 1; i++){
        // 记录待排序部分的第一个元素索引
        int minIndex = i;
        // 内层循环表示每趟需要比较的元素次数
        for (int j = i + 1; j < n; j++){
            // 在序列中找到比当前最小元素还小的元素
            if (arr[j] < arr[minIndex]){
                // 将当前最小元素更新为找到的最小元素的索引
                minIndex = j;
            }
        }
        // 交换最小元素与待排序部分的第一个元素
        if(i != minIndex ){
            swap(arr[i], arr[minIndex]);
        }
        // 输出每一趟排序后的结果
        cout << "第" << i + 1 << "趟排序后为：";
        for (int i = 0; i < n; i++){
            cout << arr[i] << " ";
        }
        cout << endl;
    }
}
```

- 选择排序总结

1） 选择排序的优点是简单直观，实现起来比较容易，适用于小规模的排序任务。但是， 由于其时 复杂度较高，对于大规模数据的排序， 更高效的排序算法法 如（快速排序 归并排序） 更为适合；

2） 选择排序的时间复杂度 0（n^2），其中n 是待排序序列的长度 每次选择最小 （或最大） 元素 需要进行 n-i 次

比较，共需进行 n-1 轮选择操作

3） 选择排序只需要常数级别的额外空间来存储临时变量，不需要额外的空间来存储待排序序列，因此空间复杂度为O（1）

4） 选择排序是一种不稳定的排序算法