**冒泡排序**

把小的元素往前调或者把大的元素往后调，比较是相邻的两个元素比较，交换也发生在这两个元素之间。（类似于气泡上浮过程）

口诀：

外层循环-1

内层循环-1-i

内循环相邻的2个比较大小

邻里交换位置

**优化方式：**

记录当前轮次是否发生过交换，没有发生过交换表示数组已经有序；

记录上次发生交换的位置，下一轮排序时只比较到此位置。

**代码实现：**

function bubbleSort(){

//普通的冒泡排序没有bSwitch这一步

//外层控制排序的轮数   n个数排n-1轮

for(var i = 0;i < arr.length -1 ; i++){

// 内层控制每轮比较的次数,每次比较都会少一个数

//所以-i,因为是和j+1比较 所以-1 防止下标溢出

var bSwitch = false;

//假设在第n轮的时候数组已经排好序了

//则内层循环结束时bSwitch仍为false 没有发生交换

for(var j = 0; j < arr.length - 1 - i;j++){

if(arr[j] > arr[j+1]){

bSwitch = true;

temp = arr[j];

arr[j] = arr[j+1];

arr[j+1] = temp;

}

}

//如果一轮内循环结束后bSwitch为true 则表示发生了交换

//下面的if条件不成立 开始下一轮排序

if (!bSwitch){

return arr;

}

}

}

外层循环为什么要 - 1

外层循环，控制比较的轮数，5个数比较4轮，2个数，比较1轮，所以要 - 1

内循环为什么要 - 1，而且还要 - i?

内循环为什么要-1?

防止下标越界，因为内循环和[j+1]比较

而且还要 - i?

每比较一轮，就会少1个数（找到了最大的数,不参与下一轮的比较)

**选择排序**

从未排序序列中找到最小（最大），放在已排序序列尾部

找到排序队列最小（最大）元素，存放在序列起始位置

在未排序序列找到最小（最大）元素，放在已排序序列尾部

重复1、2步骤

**代码实现：**

function selectSort(arr) {

//因为n个数在第n轮的时候 前面的数已经完全排好序了

//没有必要再循环一次 所以外层循环条件减1

for (var i = 0; i < arr.length - 1; i++) {

var indexOfMax = i;

//内层循环不-1

for (var j = i; j < arr.length; j++) {

if (arr[j] > arr[indexOfMax]) {

indexOfMax = j;

}

}

if (indexOfMax != i) {

var temp = arr[i];

arr[i] = arr[indexOfMax];

arr[indexOfMax] = temp;

}

}

return arr;

}

**选择排序的优化**

二元选择排序：一次遍历选出两个值——最大值和最小值

**快速排序**

以一个元素为基数，将小于基数的元素移到基数前面，大于基数的元素移到基数后面，对左右区间递归以上步骤，直到区间只有一个数

定义一个函数

在函数内部判断传入的参数必须是一个数组，而且长度要大于等于2

找到数组的中间下标，数组的长度/2，向下取值

找到中间下标，找到中间数，必须是从原来的数组中抠出

定义两个空数组

循环遍历原来的数组，如果大于中间数，则放到右边数组，否则就放到左边数组

左边数组+中间数+右边数组

**代码实现**

function quickSort(arr) {

// 先判断数组长度是否为1 如果为1则返回数组本身

if (arr.length < 2) {

return arr;

}

//获取中间的数下标

var middleIndex = Math.ceil(arr.length / 2);

// 获取中间数的值

var middleVal = arr.splice(middleIndex, 1)[0];

//定义两个空数组

var arr1 = [],

arr2 = [];

//循环遍历数组 如果比中间的数大则存入右边的数组

//如果比中间的数小则存入左边的数组

for (var i = 0; i < arr.length; i++) {

if (arr[i] > middleVal) {

arr2.push(arr[i]);

} else {

arr1.push(arr[i]);

}

}

//左边和右边的数组还是一个未排序的状态 所以递归调用

//拼接成一个排好序的数组

return quickSort(arr1).concat(middleVal, quickSort(arr2));

}

**O(n^2)的排序算法（选择、插入、冒泡）**

选择、插入、冒泡是入门级的排序算法，虽然性能不怎么样，但是属于基础，为后面的排序也提供良好的思路。

**三种排序比较**

**相同点**

时间复杂度都是 O(n^2)，空间复杂度都是 O(1)。

都需要采用两重循环。

**不同点**

选择排序是不稳定的，冒泡排序、插入排序是稳定的；

假定在待排序的记录序列中，存在多个具有相同的关键字的记录，

若经过排序，这些记录的相对次序保持不变，即在原序列中，r[i] = r[j]，且 r[i] 在 r[j] 之前，

而在排序后的序列中，r[i] 仍在 r[j] 之前，则称这种排序算法是稳定的；否则称为不稳定的。

在这三个排序算法中，选择排序交换的次数是最少的

**选择排序**

选择排序可以演变为二元选择排序：

二元选择排序：一次遍历选出两个值——最大值和最小值；

二元选择排序剪枝优化：当某一轮遍历出现最大值和最小值相等，表示数组中剩余元素已经全部相等。

**插入排序**

插入排序有两种写法：

交换法：新数字通过不断交换找到自己合适的位置；

移动法：旧数字不断向后移动，直到新数字找到合适的位置。

**冒泡排序**

冒泡排序有两种优化方式：

记录当前轮次是否发生过交换，没有发生过交换表示数组已经有序；

记录上次发生交换的位置，下一轮排序时只比较到此位置。