算法过程动态图
https://visualgo.net/

排序

如何分析一个排序算法

执行效率：

1.最好情况、最坏情况、平均情况时间复杂度

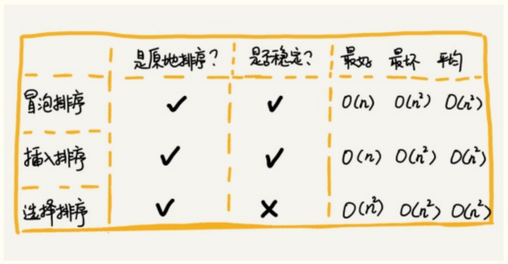
2.时间复杂度的系数、常数、低阶

3.比较次数和交换(或移动)次数

算法的内存消耗

排序算法的稳定性

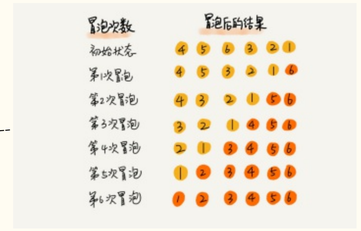
如果数据内有相同的值，在排序后它们的顺序没变，就叫做**稳定的排序算法**，如果前后顺序发生变化，那对应的排序算法就叫做**不稳定的排序算法**



冒泡排序

从头到尾，如果前一个比后一个大就交换，一轮走完，数组最后的都是已排定的最大数

原地排序、稳定、最好 O(n) 最坏 O(n^2)



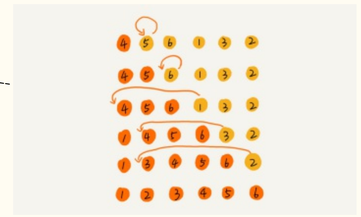
插入排序

将数组分为 **已排序** 跟 **未排序** 两个区间

原地排序、稳定排序、最好 O(n) 最坏 O(n^2) 平均 O(n)

冒泡单位时间是3，插入是1

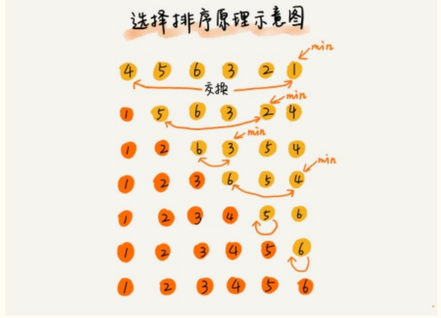
所以插入排序实际性能优于冒泡



选择排序

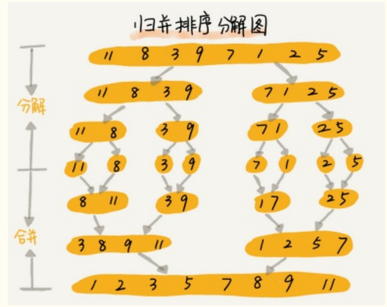
将数组分为 **已排序** 跟 **未排序** 两个区间，每次从未排序区间中找到最小元素，将其放到已排序区间的末尾

原地排序、不稳定、最好最坏都是 O(n^2)



分治思想

将一个大问题分解成小的子问题来解决。小的子问题解决了，大问题也就解决了



归并排序

将数组从中间分成前后两部分

对前后两部分分别排序

再将排序好的两部分合并在一起

稳定、最好最坏 O(nlogn)、空间 O(n)

递推公式：

```
merge_sort(p…r) = merge(merge_sort(p…q), merge_sort(q+1…r))
```

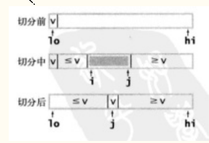
终止条件：

```
p >= r 不用再继续分解
```

快速排序

平均 O(nlogn)、最坏 O(n^2)

原地、不稳定的算法



要完成这个实现，需要实现切分方法。一般策略是先随意地取 a[lo] 作为切分元素，即那个将会被排定的元素，然后我们从数组的左端开始向右扫描直到找到一个大于等于它的元素，再从数组的右端开始向左扫描直到找到一个小于等于它的元素。这两个元素显然是没有排定的，因此我们交换它们的位置。如此继续，我们就可以保证左指针 i 的左侧元素都不大于切分元素，右指针 j 的右侧元素都不小于切分元素。当两个指针相遇时，我们只需要将切分元素 a[lo] 和左子数组最右侧的元素（a[j]）交换然后返回 j 即可。切分方法的大致过程如图 2.3.2 所示。

```
public static int partition(int[] a, int lo, int hi) {
    int i = lo, j = hi + 1;
    int p = a[lo];
    while (true) {
        while (a[++i] < p) if (i == hi) break;
        while (p < a[--j]) if (j == lo) break;
        if (i >= j) break;
        exch(a, i, j);
    }
    exch(a, lo, j);
    return j;
}
```

线性排序

时间复杂度是线性

非基于比较的排序算法，都不涉及元素之间的比较操作

对要排序的数据要求苛刻

桶排序

要求

1.要排序的数据要很容易能划分成 m 个桶

2.桶与桶之间有着天然的大小顺序

3.数据在各个桶之间的分布是比较平均的，在极端情况数据都被划分到一个桶中，复杂度则会退化为 O(nlogn)

桶排序比较适合用在外部排序中

所谓的外部排序就是数据存储在外部磁盘中，数据量比较大，内存有限，无法将数据全部加载到内存中。

计数排序

计数排序其实是桶排序的一种特殊情况

计数排序只能用在数据范围不大的场景中，如果数据范围 k 比要排序的数据 n 大很多，就不适合用计数排序了。而且，计数排序只能给非负整数排序，如果要排序的数据是其他类型的，要将其在不改变相对大小的情况下，转化为非负整数

比如把小数扩大成整数，最后再转换为小数