少儿编程

21工作室出品

四大算法思想

- 贪心算法
- 分治算法
- 动态规划
- 穷举算法

贪心算法

四大算法之一

贪心算法

- 把原问题分解出几个部分，这几个部分一般是按时间先后或者任务先后划分，分别求出子问题最优解再合并在一起。
- 在每一步中，选择目前最优的策略而并不考虑长远，短视的算法
- 如01背包问题
  - 有一个背包，最多能承载150斤的重量，现在有7个物品，
  - 重量分别为[35, 30, 60, 50, 40, 10, 25]，
  - 价值分别为[10, 40, 30, 50, 35, 40, 30]，
  - 现在想要用这个背包背走最多价值的物品，怎么背

子问题的最优解

- 方案一：
  - 每次都选当前价值最高的物品
- 方案二：
  - 每次都选当前重量最轻的物品
- 方案三：
  - 每次都选当前“价值密度”最高的物品

- 方案一
  - [4, 2, 6, 5]
  - 总重量130，总价值165
- 方案二
  - [6, 7, 2, 1, 5]
  - 总重量140，总价值155
- 方案三
  - [6, 2, 7, 4, 1]
  - 总重量150，总价值170
- 可以看出最后一种定义方法最优

- 定义子问题的最优方案，一种定义方式往往可能只在一种特定条件下是最优，新条件下可能就不是最优了
- 为什么不直接求全局最优解，因为问题过于复杂数学模型难以建立

分治算法

四大算法之一

二分查找法

- 第一周第二节课
- 第一个猜数小程序就用到了

快速排序算法

- 将原问题的规模减小，变成更容易解决的问题
- 是动态规划的退化算法
- 递归
- 如：
  - 快速排序算法
  - 与同簇（交换排序）的冒泡排序比，快速排序算法是用空间换取了时间，所以更快

动态规划算法

四大算法之一

数学归纳法

- 知道最基础的第一步
- 知道第一步之后第二步以及任意一步
- 那么就可以退出所有的数都满足一个定理
- 这就是数学归纳法证明

- 等差数列求和公式

$$1+2+3+\ldots+n=\frac{n(n+1)}{2}$$

- 证明：
- 第一步：验证该公式在 $n = 1$ 时成立。

$$1 = \frac{1(1+1)}{2}$$

- 第二步，需要证明**假设**$n = m$ 时公式成立，**推导**出$n = m+1$ 时公式也成立。

$$1+2+3+\ldots+m=\frac{m(m+1)}{2}$$

- 然后在等式两边**同时***分别*加上$m+1$

$$1+2+3+\ldots+m+m+1?\frac{(m+1)[(m+1)+1]}{2}$$

$$1+2+3+\ldots+m+m+1=\frac{m(m+1)}{2}+m+1$$

$$=\frac{m(m+1)}{2}+\frac{2(m+1)}{2}$$

$$=\frac{m(m+1)+2(m+1)}{2}$$

$$=\frac{(m+2)(m+1)}{2}$$

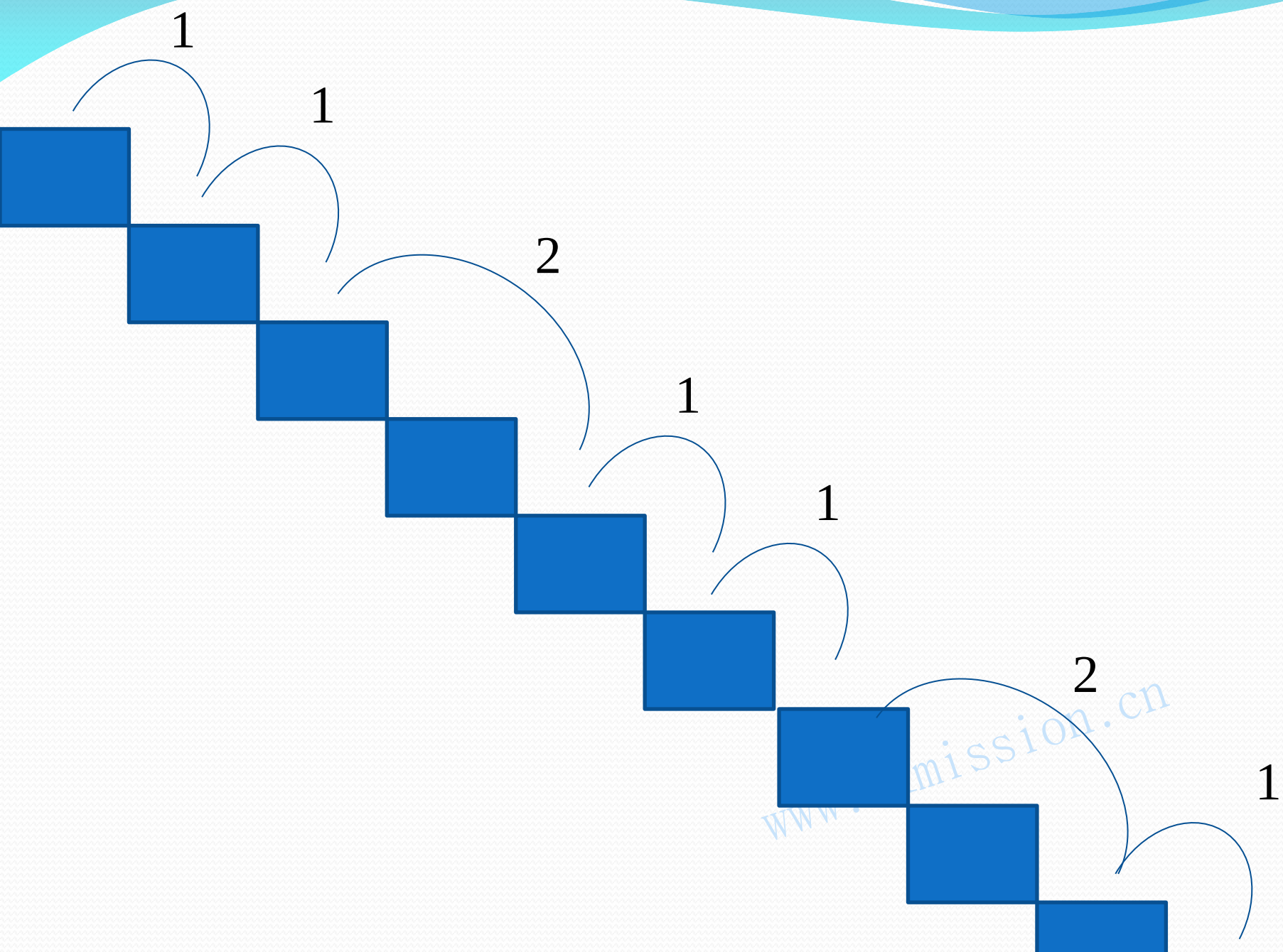
$$= \frac{(m+2)(m+1)}{2}$$

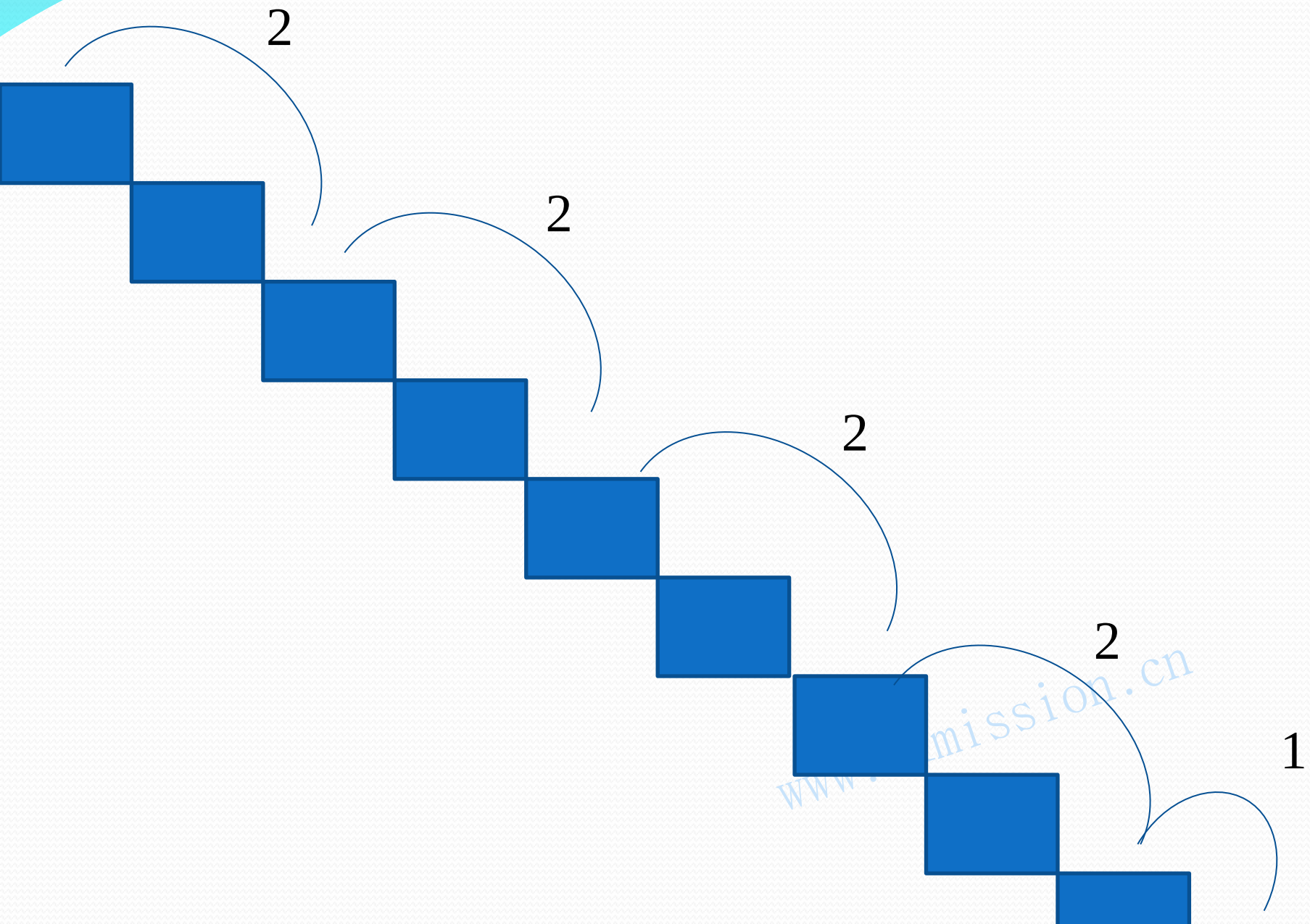
$$= \frac{(m+1)[(m+1)+1]}{2}$$

$$1+2+3+\ldots+m+m+1 = \frac{(m+1)[(m+1)+1]}{2}$$

动态规划

- 擅长解决“多阶段决策问题”，利用各个阶段之间的递推关系，逐个确定每个阶段的最优决策，并最终得到原问题最优决策。
- 它是分治算法的进阶，子问题是有重叠的。存储中间结果的表叫备忘录
- 递归
- 使用状态转移方程
- 如：
  - 上楼梯问题
  - 一个10级的台阶，每次只能上1级或2级，问：共有几种上法


1
1
2
1
1
2
1



- 状态方程：F(n) = F(n – 1) + F(n – 2)
- F(10) = F(9) + F(8)
- F(9) = F(8) + F(7)
- 类推，最终问题被逐渐降低了规模，越来越简单

穷举法

四大算法之一

穷举搜索算法

- 把所有解空间都算一遍，得到最优解
- 如：
  - 求质数
  - 求完美数
  - 国际象棋
  - 中国象棋
  - 围棋

- 遍历（经历一遍）所有元素，一一排查
- 比如求100是不是质数，那么就用100除以2到100之间的所有的自然数，看余数是否等于0。
- 如果这个数除以2到自身之间的所有的自然数，而余数没有等于0，那么这个数就是质数
- 比如71
- 不厌其烦，不怕辛苦，不怕重复

排序算法

交换排序的两种算法

各种排序算法的时间复杂度和空间复杂度

类别	排序方法	时间复杂度			空间复杂度	稳定性
		平均情况	最好情况	最坏情况	辅助存储	
插入排序	直接插入	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	Shell排序	$O(n^{13})$	$O(n)$	$O(n^2)$	$O(1)$	不稳定
选择排序	直接选择	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆选择	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n^2)$	$O(n)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	不稳定
归并排序		$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
基数排序		$O(d(r+n))$	$O(d(n+rd))$	$O(d(r+n))$	$O(rd+n)$	稳定

- 十种常见排序算法可以分为两大类：
  - 非线性时间比较类排序：通过比较来决定元素间的相对次序，由于其时间复杂度不能突破O(nlogn)，因此称为非线性时间比较类排序。
  - 线性时间非比较类排序：不通过比较来决定元素间的相对次序，它可以突破基于比较排序的时间下界，以线性时间运行，因此称为线性时间非比较类排序。

排序算法

- 我们只介绍第一类中的四种排序算法
  - 三种简单、一种较复杂

- 交换排序
  - 冒泡排序——简单
  - 快速排序——较复杂
- 选择排序——简单
- 插入排序——简单

冒泡排序	• 交换排序的一种
选择排序	• 找出最高分排第一名，次高分排第二名
插入排序	• 玩扑克牌经常用到
快速排序	• 交换排序的一种 • 冒泡排序的升级版

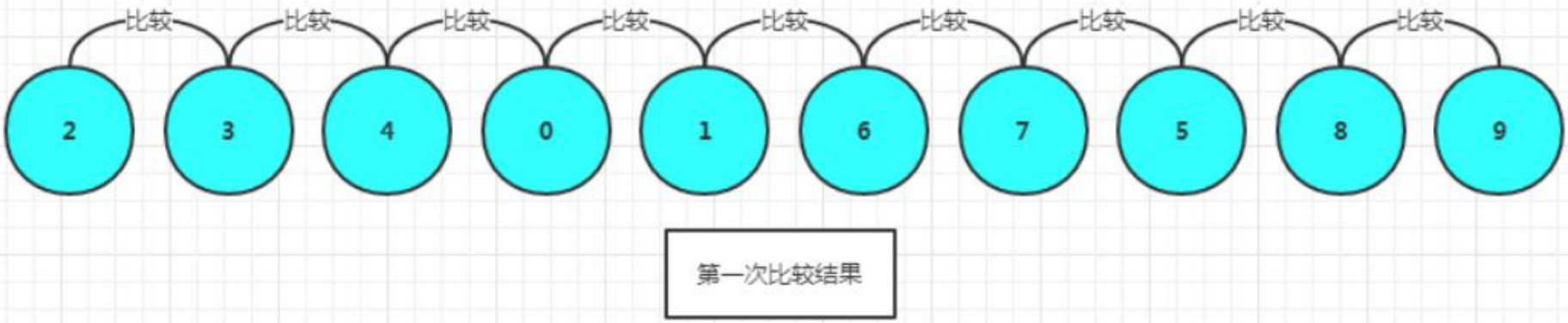
交换排序

- 比较两个数，然后根据大小交换这两个数的位置
- 经过不断使用交换规则达到排序的目的

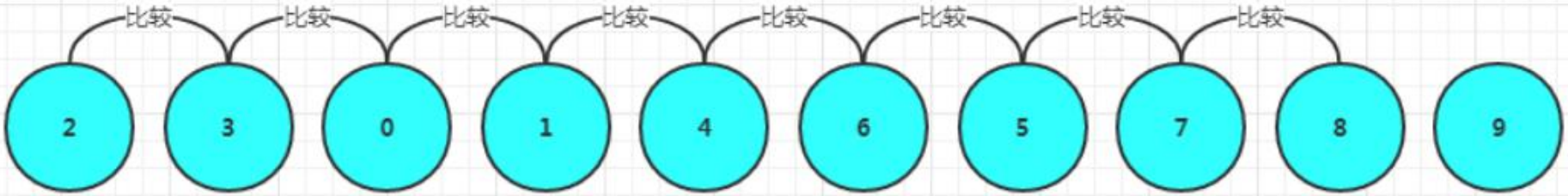
交换排序——冒泡排序

- 利用泡泡，每次带上来一个最大的数
- 多用几个泡泡，就能完成排序

- 过程是交换



第一次比较结果



第二次比较结果

一个气泡会带走一个最大的数到右边

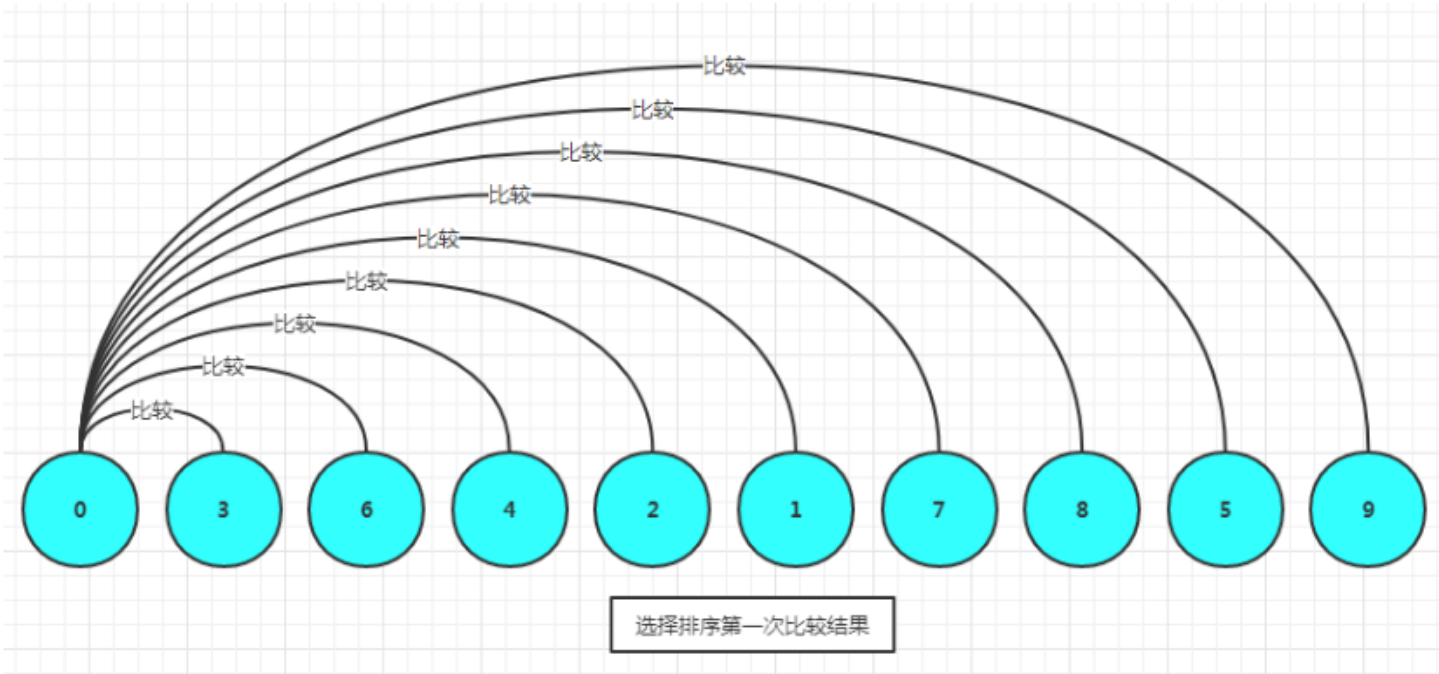
冒泡排序算法-20220223-过程版-pre2.py
```
原来的列表: [9, 13, 1, 56, 44, 21, 19, 87, 98, 65]
经过第0遍排序, 列表是[9, 13, 1, 56, 44, 21, 19, 87, 98, 65]
经过第1遍排序, 列表是[9, 1, 13, 56, 44, 21, 19, 87, 98, 65]
经过第2遍排序, 列表是[9, 1, 13, 56, 44, 21, 19, 87, 98, 65]
经过第3遍排序, 列表是[9, 1, 13, 44, 56, 21, 19, 87, 98, 65]
经过第4遍排序, 列表是[9, 1, 13, 44, 21, 56, 19, 87, 98, 65]
经过第5遍排序, 列表是[9, 1, 13, 44, 21, 19, 56, 87, 98, 65]
经过第6遍排序, 列表是[9, 1, 13, 44, 21, 19, 56, 87, 98, 65]
经过第7遍排序, 列表是[9, 1, 13, 44, 21, 19, 56, 87, 98, 65]
经过第8遍排序, 列表是[9, 1, 13, 44, 21, 19, 56, 87, 65, 98]
第1个气泡完成了一轮
经过第0遍排序, 列表是[1, 9, 13, 44, 21, 19, 56, 87, 65, 98]
经过第1遍排序, 列表是[1, 9, 13, 44, 21, 19, 56, 87, 65, 98]
经过第2遍排序, 列表是[1, 9, 13, 44, 21, 19, 56, 87, 65, 98]
经过第3遍排序, 列表是[1, 9, 13, 21, 44, 19, 56, 87, 65, 98]
经过第4遍排序, 列表是[1, 9, 13, 21, 19, 44, 56, 87, 65, 98]
经过第5遍排序, 列表是[1, 9, 13, 21, 19, 44, 56, 87, 65, 98]
经过第6遍排序, 列表是[1, 9, 13, 21, 19, 44, 56, 87, 65, 98]
经过第7遍排序, 列表是[1, 9, 13, 21, 19, 44, 56, 65, 87, 98]
第2个气泡完成了一轮
经过第0遍排序, 列表是[1, 9, 13, 21, 19, 44, 56, 65, 87, 98]
经过第1遍排序, 列表是[1, 9, 13, 21, 19, 44, 56, 65, 87, 98]
经过第2遍排序, 列表是[1, 9, 13, 21, 19, 44, 56, 65, 87, 98]
经过第3遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
经过第4遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
经过第5遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
经过第6遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
第3个气泡完成了一轮
经过第0遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
经过第1遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
经过第2遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
经过第3遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
经过第4遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
经过第5遍排序, 列表是[1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
第4个气泡完成了一轮
排序后的列表: [1, 9, 13, 19, 21, 44, 56, 65, 87, 98]
```

98

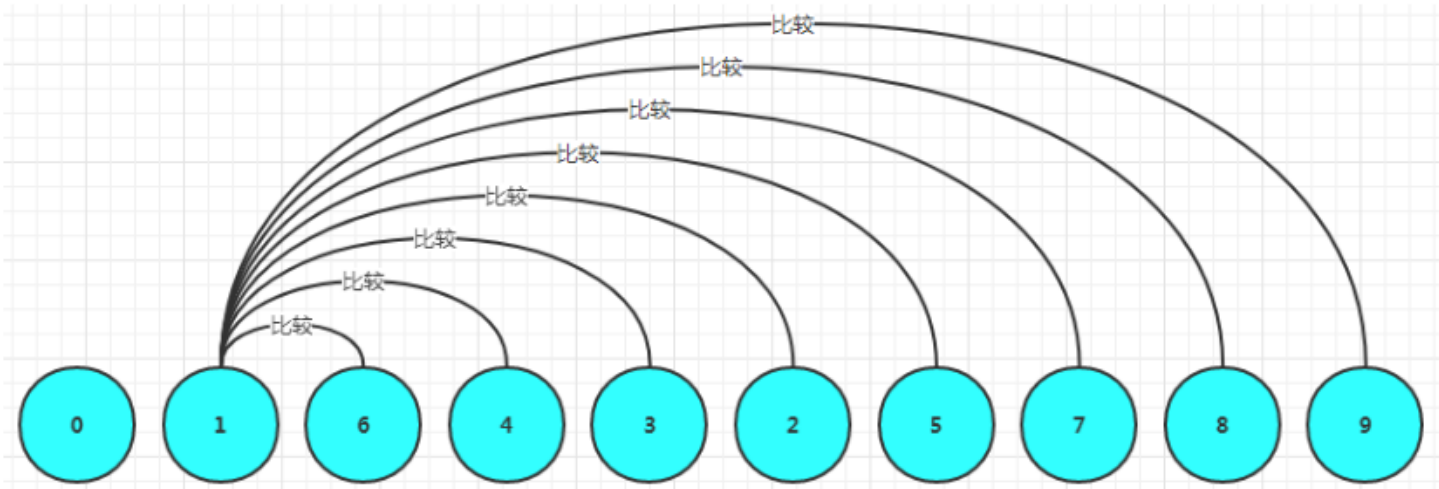
选择排序

- 过程是选定一个数，然后作比较

- 第一个位置我选择了最小的数
- 第二个位置我要选择次小的数
- 第三个位置……



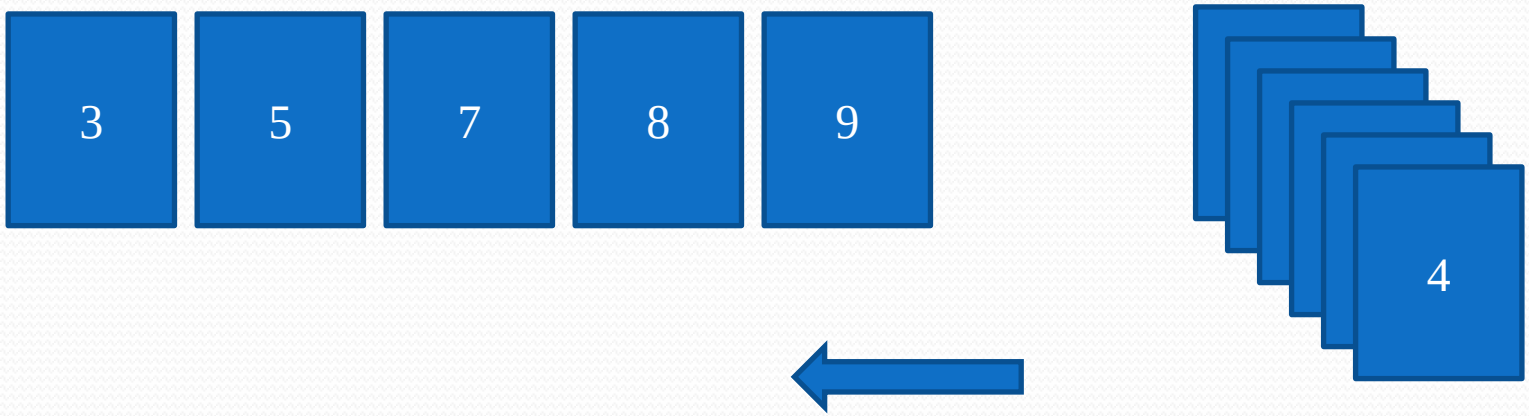
选择排序第一次比较结果

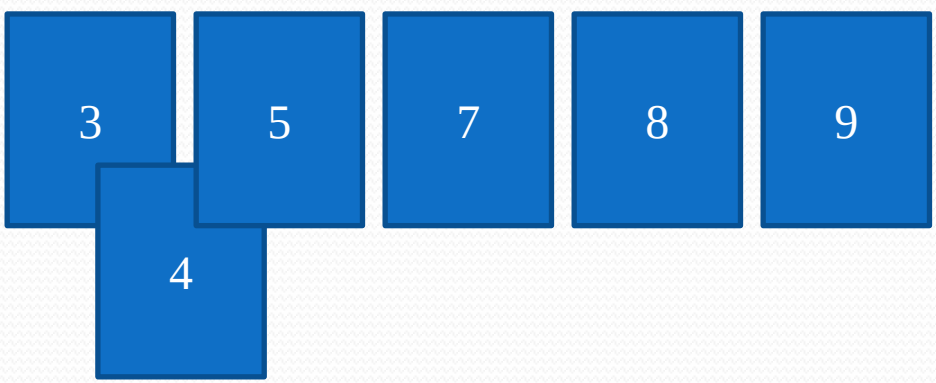


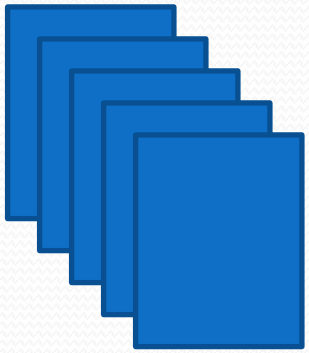
选择排序第二次比较结果

插入排序

- 摸到一张牌
- 然后从左到右，去和手上所有的比他小的牌进行比较，
- 找到空位，然后插入







快速排序

- 排序速度最快的算法

- 与冒泡排序比较，冒泡排序用时18.14秒
- 快速排序用时：4秒

- 快速为什么快？
  - 因为它用到了分治算法——分而治之。
  - 用基准数分开了两个区。每次处理比较仅在小分区内进行
  - 冒泡排序和选择排序为什么慢，因为每一轮都要处理所有的元素。从头到尾。

- 基准值
- 游标

总结——到处可见的排序算法

- 资源分配（供不应求的情况下，凭抢到的先后顺序号来排队领取）
- 查找抽取（工具标号太多快速get到匹配号，比杂乱摆放，做事井井有条、有条不紊）
- 优先级（给每个事务分配一个优先级别号，号数越小/大越优先处理）