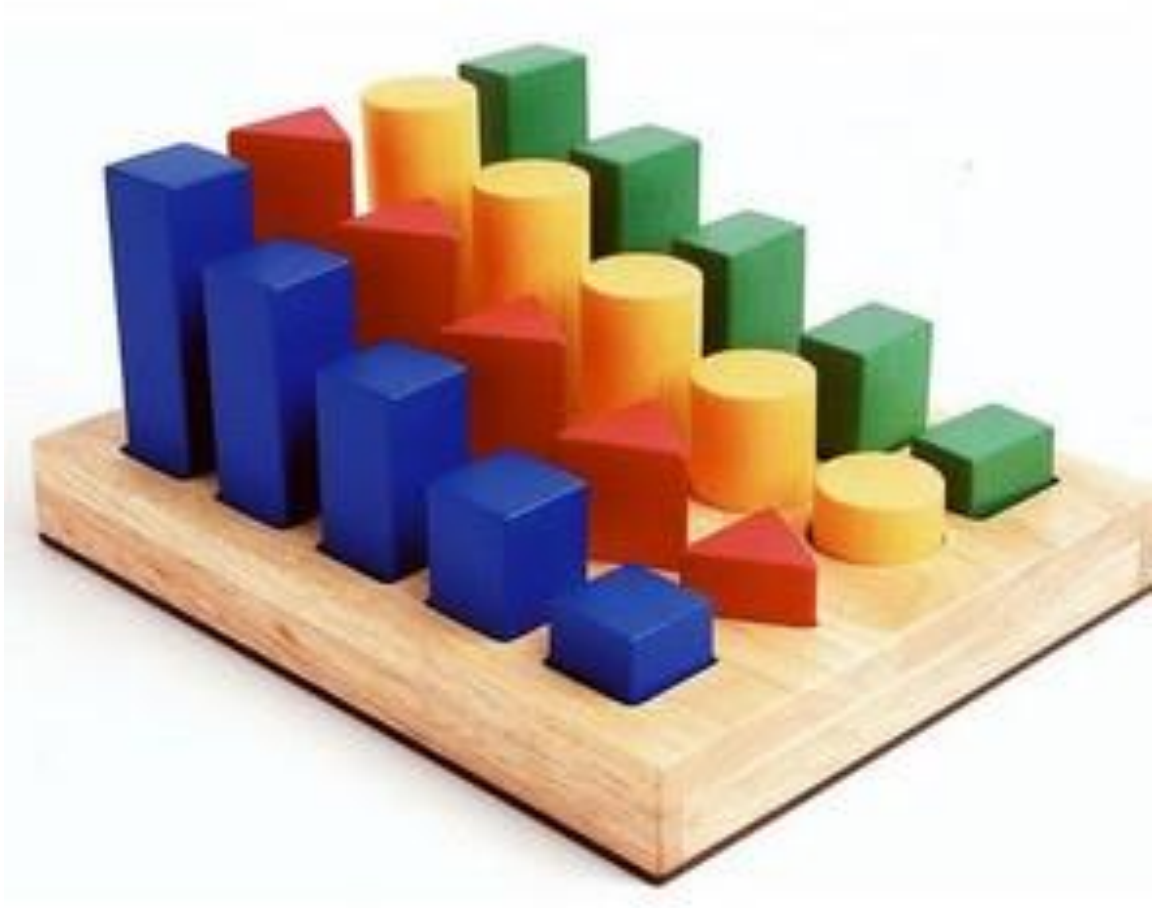
亲子算法课（7）
——简单排序算法

作者：叶蒙蒙

简单排序算法
(Simple Sorting)

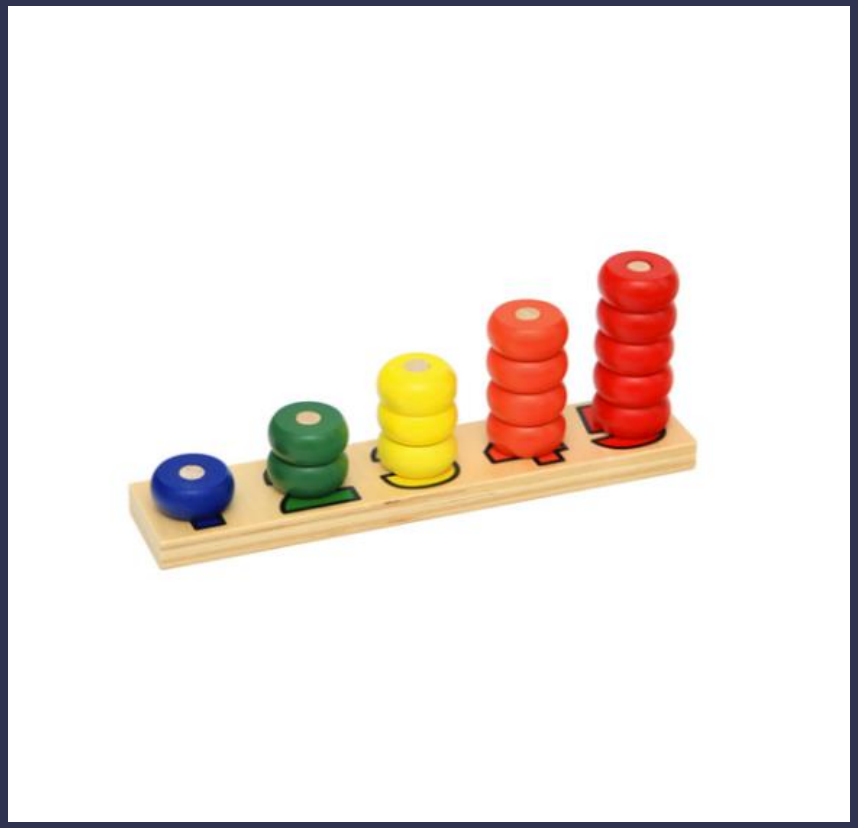
排序（Sorting）

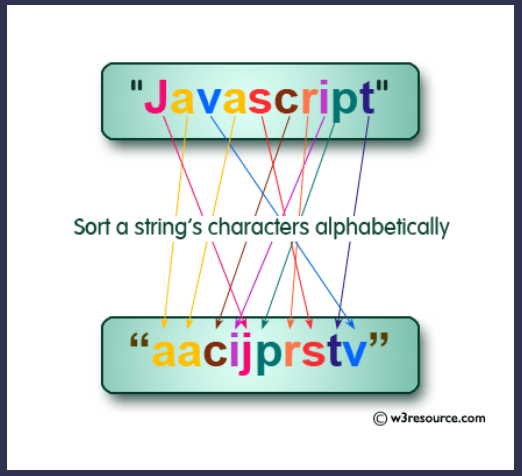
- 把很多个体按顺序排列
- 个体——同类，或者有共同的性质
- 顺序——根据某一种可以比大小的指标



现实中的排序

- 按大小个站队
- 考试成绩排名
- 福布斯排行
- 大学排行榜
- ……





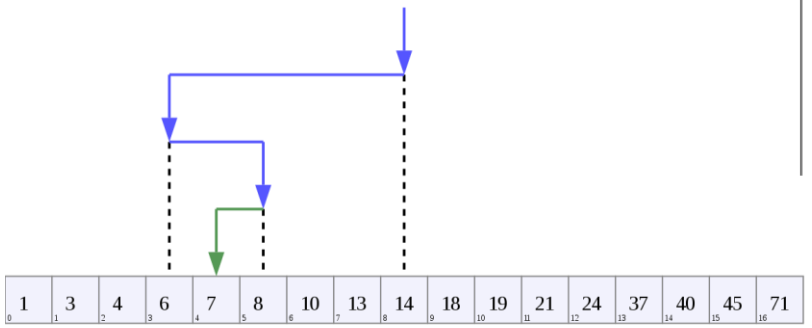
计算机中的排序

任何在现实中排序的元素，都可以在计算机中排序

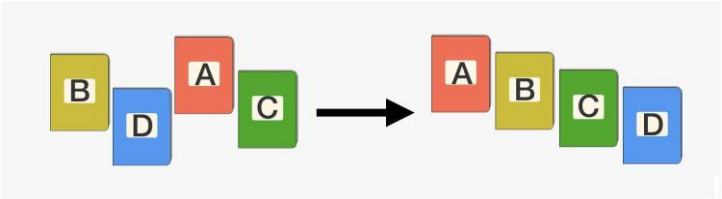
最简单的：整数排序

查找和排序的异同

- 查找的基本操作
  - i）比大小
  - ii）确定下一个要比大小的元素的位置
- 算法完成后，原序列顺序不变



- 排序的基本操作
  - i）比大小
  - ii）找到正确位置
  - iii）插入正确位置
- 算法完成后，原序列顺序很可能改变



排序算法

- 非常重要！！！
- 算法的基础
- 种类多样
- 算法的时间复杂度考量
  - 随机
  - 基本正序
  - 倒序
  - 大量相同元素

用扑克牌来尝试排序，分别尝试:

1. 取出某一花色的1 – 10，打乱顺序，进行排序（已经直到是1-10共10张牌）
2. 从1-10里随便抽掉 1 – 3 张牌，再次进行排序

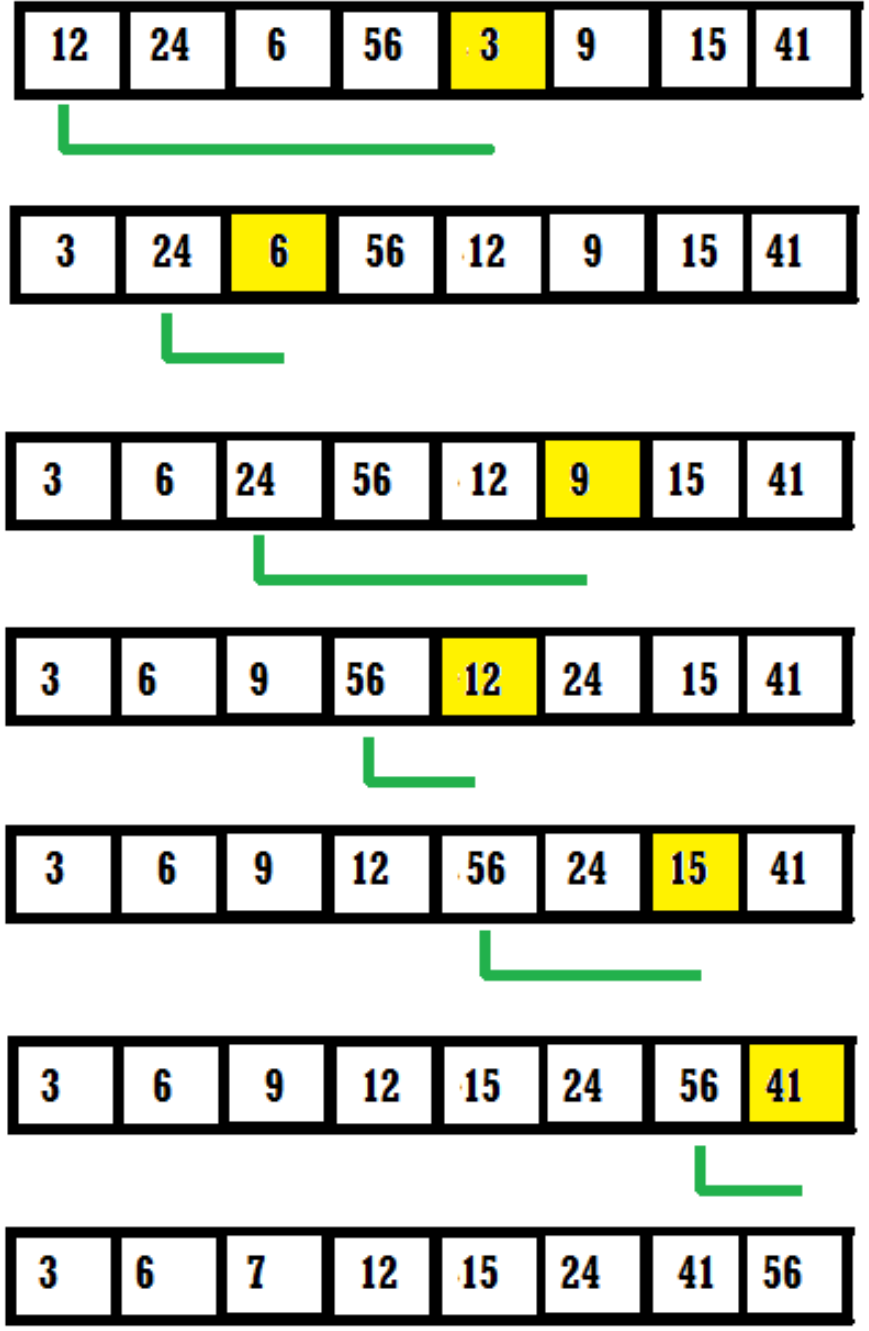
初步尝试排序

选择排序

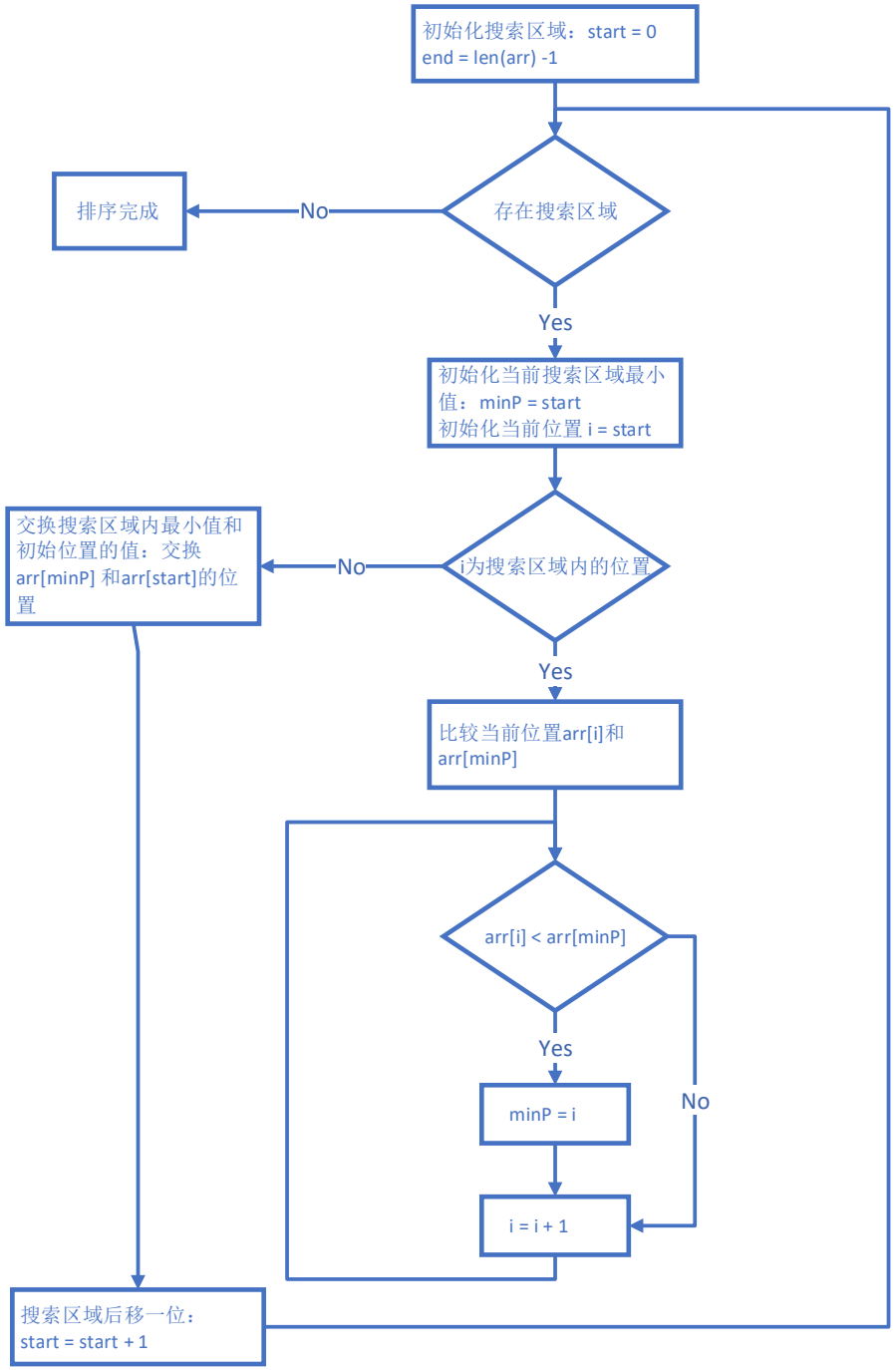
选择排序

n个数字排序，默认要求排列为正序

- 原理：反复迭代，每次在未排序的所有元素里选中最小的那个
- 迭代步骤：
  i. 选中当前选择区域内最小的元素
  ii. 和当前区域第一位的元素交换
  iii. 将选择区域起始位置后移动一位



选择排序流程图

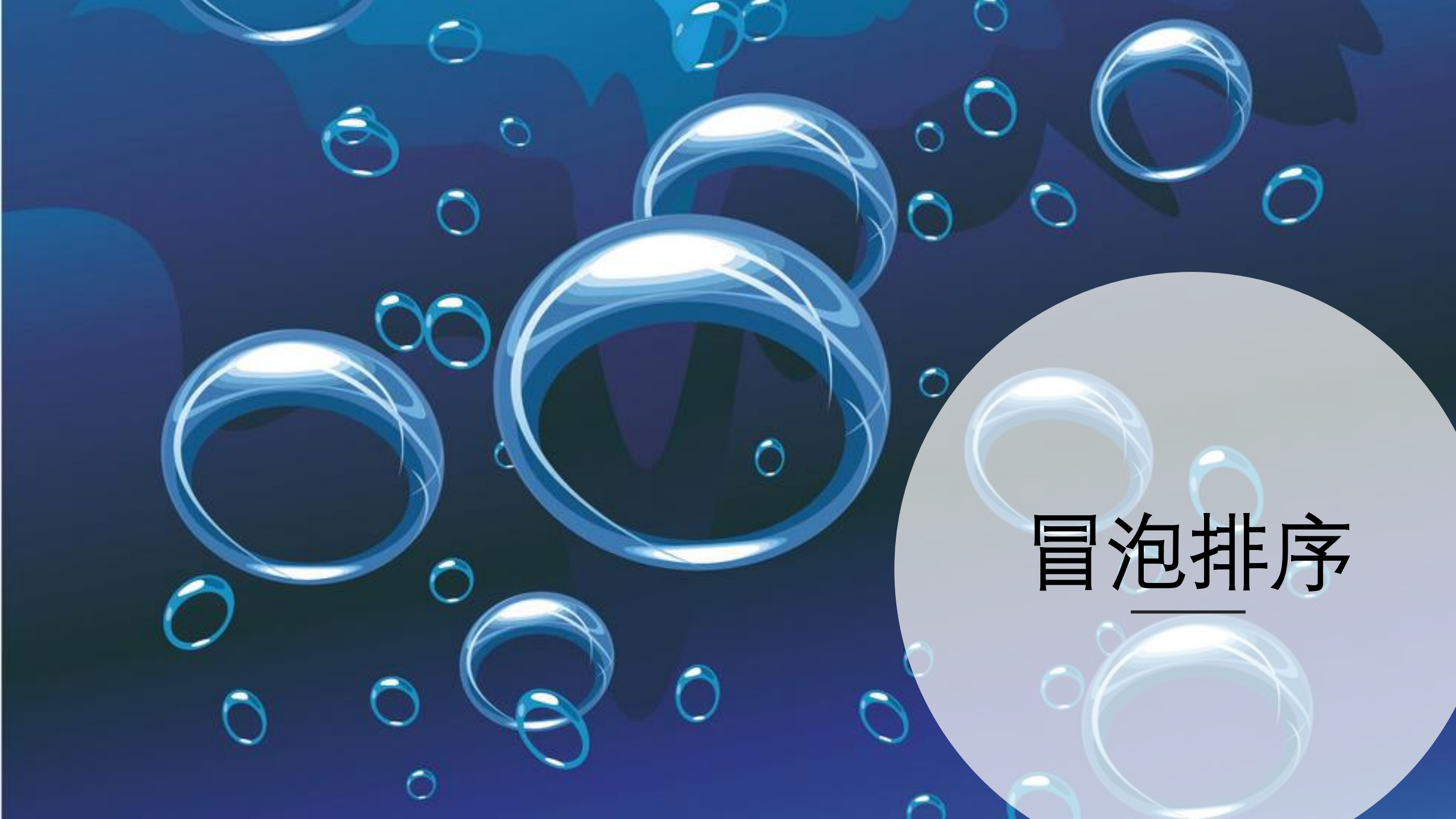

初始化搜索区域：start = 0
end = len(arr) -1
存在搜索区域
No
排序完成
Yes
初始化当前搜索区域最小值：minP = start
初始化当前位置 i = start
i为搜索区域内的位置
No
交换搜索区域内最小值和初始位置的值：交换arr[minP] 和arr[start]的位置
Yes
比较当前位置arr[i]和arr[minP]
arr[i] < arr[minP]
Yes
No
minP = i
i = i + 1
搜索区域后移一位：start = start + 1

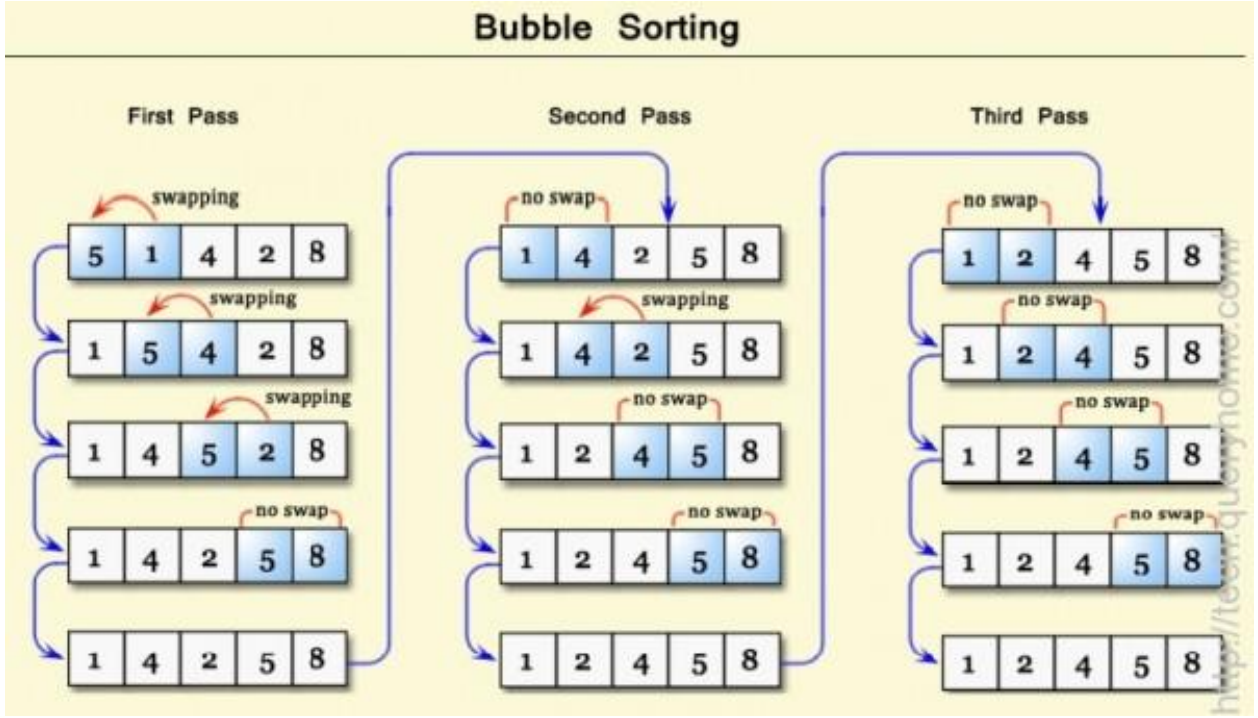
选择排序代码

```
def ssort(arr):
    for start in range(0, len(arr)):
        minP = start
        for i in range(start+1, len(arr)):
            if (arr[i] < arr[minP]):
                minP = i

        tmp = arr[start]
        arr[start] = arr[minP]
        arr[minP] = tmp

if __name__ == "__main__":
    array = [54,26,93,17,77,31,44,55,20]
    ssort(array)
    print(array)
```

选择排序代码（2）

```
def lookforFewest(arr, start, end):
    minP = start
    for i in range(start,end + 1):
        if (arr[i] < arr[minP]):
            minP = i
    return minP
def switch(arr, i, j):
    tmp = arr[i]
    arr[i] = arr[j]
    arr[j] = tmp

def selectionSort(arr):
    end = len(arr) - 1
    for start in range(0,end):
        fewest = lookforFewest(arr, start, end)
        switch(arr, start,fewest)
```

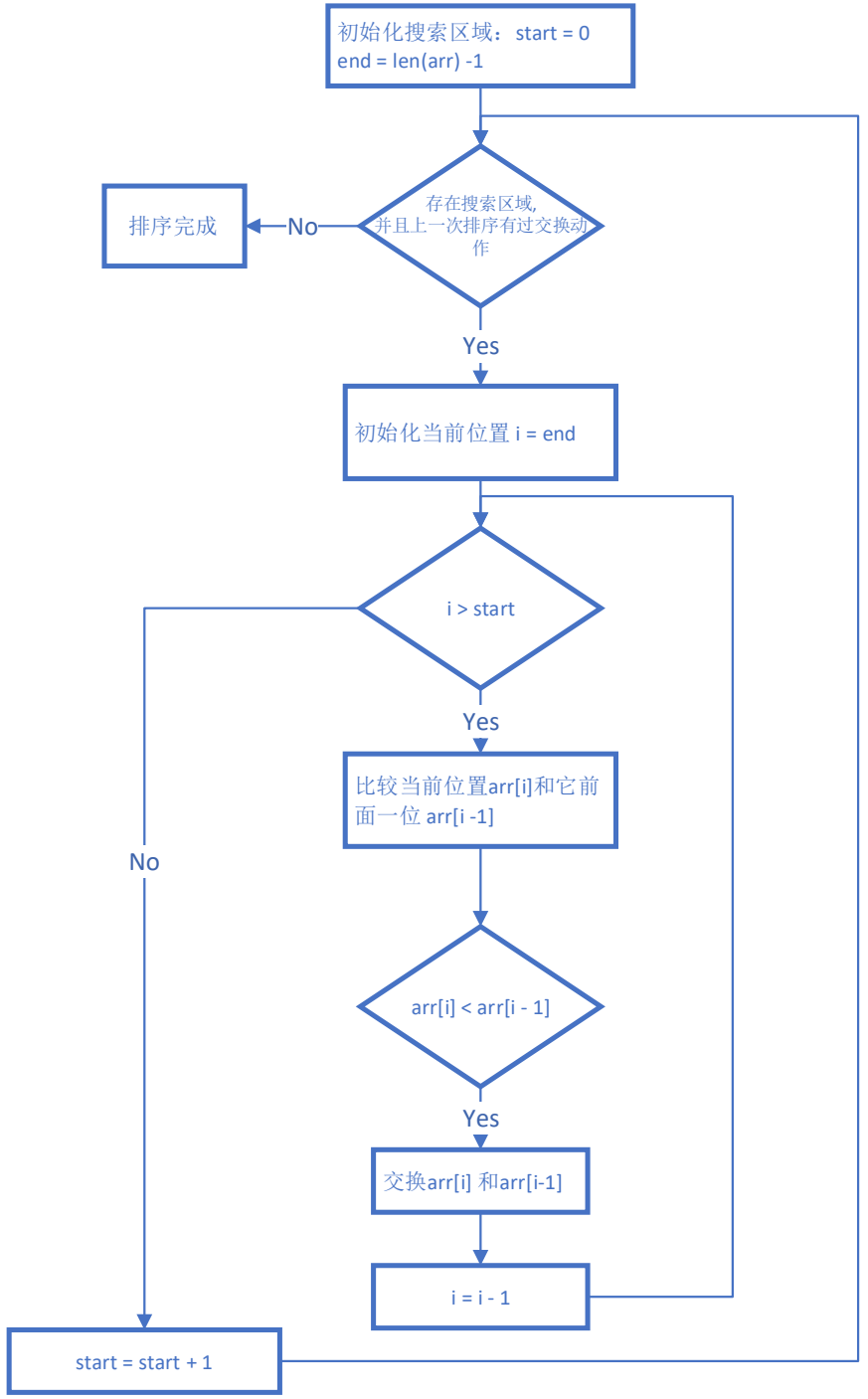

冒泡排序



- 原理： 每次选择未排序的元素中最小的那个，让它“冒”上来
- 迭代步骤：
  I. 当前选择区域内，从最后一个元素开始，重复如下迭代到第一个元素位置
     i. 选中当前元素，将其和其前一位的元素比较，如果小于前面一个，则两者交换
     ii. 当前元素前移一位
  II. 当前选择区域后移一位

冒泡排序

冒泡排序流程图


初始化搜索区域：start = 0
end = len(arr) -1
存在搜索区域，并且上一次排序有过交换动作
No
排序完成
Yes
初始化当前位置 i = end
i > start
No
Yes
比较当前位置arr[i]和它前面一位 arr[i -1]
arr[i] < arr[i - 1]
Yes
交换arr[i] 和arr[i-1]
i = i - 1
start = start + 1

冒泡排序代码

```
def switch(arr, i, j):
    tmp = arr[i]
    arr[i] = arr[j]
    arr[j] = tmp

def bsort(arr):

    for start in range(0, len(arr)):
        switched = False
        for i in range(len(arr)-1, start, -1):
            if (arr[i] < arr[i-1]):
                switch(arr, i, i-1)
                switched = True
        if not switched:
            break

if __name__ == "__main__":
    array = [54,26,93,17,77,31,44,55,20]
    bsort(array)
    print(array)
```

选择排序 vs 冒泡排序

	选择排序	冒泡排序
最好情况（正序） 时间复杂度	O(n)^2	O(n) NOTE：如果原序列已排序，能够在第一次循环中识别出来
最坏情况（倒序） 时间复杂度	O(n)^2	O(n)^2
平均时间复杂度	O(n)^2	O(n)^2
最坏（需要交换） 空间复杂度	O(1)	O(1)
稳定性	不稳定	稳定
速度	通常比冒泡排序快，而且写入次数少	慢，一般只用于少量数据排序，不实用

谢谢