# 快速排序

quick sort

# 版权声明

# 联系方式

如果你想要购买本课程
请微信联系 xiedaimala02 或 xiedaimala03

如果你发现有人盗用本课程
请微信联系 xiedaimala02 或 xiedaimala03

# 一个经典问题

- 正整数排序
  - arr = [2341, 982,394,124,9901]
  - 请写出一个函数 sort，输入为 arr，输出为一个从小到大排序好的数组

# merge sort 思路

- 人类思维
  - ✓ 将数组对半分，不停对半分
  - ✓ 直到数组长度为一，即已排好序的数组
  - ✓ 用 merge 将长度为1的数组合并为，长度为2的排好序的数组
  - ✓ 用 merge 将长度为2的数组合并为，长度为4的排好序的数组
  - ✓ ……
- 数学思维

$$sort(a_{1..n}) = \begin{cases} a_{1..n}, & n = 1 \\ merge(sort(a_{1..n/2}), sort(a_{(n/2+1)..n})), & n \geqslant 2 \end{cases}$$

# quick sort 思路

- 人类思维
  - ✓ 以某个元素为基准 pivot，小的往左走，大的往右走
  - ✓ 递归此操作，直到各部位只剩一个元素
- 数学思维

$$sort(a_n) = \begin{cases} a_n, n = 1 \\ sort((a_n - a_p).filter(value < a_p)) \\ \quad + [a_p] \\ \quad + sort((a_n - a_p).filter(value \geqslant a_p)), n >= 2 \end{cases}$$

其中 an - ap 表示从数组 an 中删除 ap 元素

# 数学思维写快排

- 代码

```
quickSort = (arr) => {
  const [pivot, ...rest] = arr
  return arr.length <= 1 ? arr :
    [
      ...quickSort(rest.filter( n=> n < pivot)  ),
      pivot,
      ...quickSort(rest.filter( n=> n >= pivot) )
    ]
}
```

- 分析

- ✓ 太简单，而且是对的（可用数学归纳法证明）
- ✓ 至少遍历了四次：filter x 2，... x 2
- ✓ 至少复制了五次：rest x 1，filter x 2，... x 2
- ✓ 目标：减少遍历（一次遍历多做些事），减少复制（就地操作）

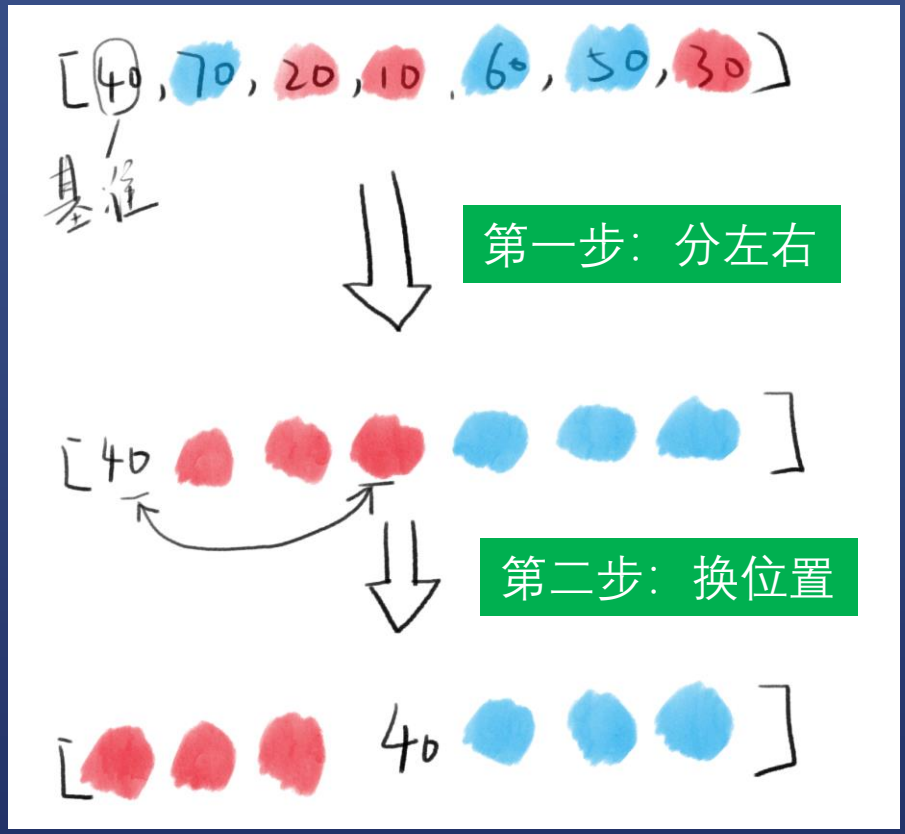
# 小的往左走，大的往右走

先来优化这一步

# 需求

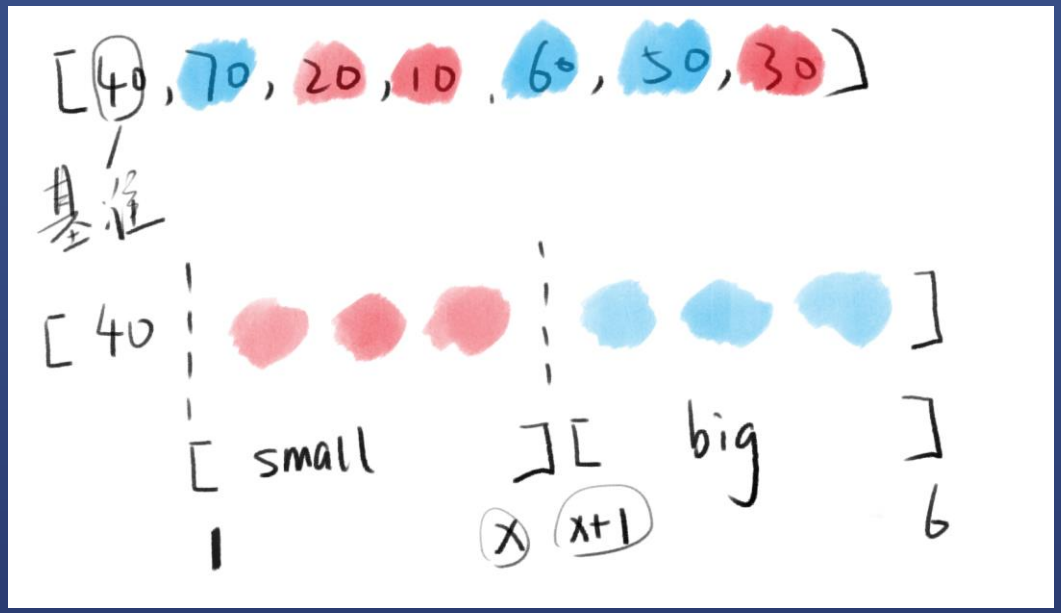
- 需求
  - 数组 arr = [40,70,20,10,60,50,30]
  - 函数 handlerPivot 以 40 为基准
  - 小的往左走，大的往右走
  - 使得数组 arr 变为 [...小于40的数, 40，...大于40的数]
  - 返回值为 pivot 的新下标，也就是 3
  - 不要求小于40的数是排好序的，大于的也不要求
  - 不要创建新数组，必须就地操作
- 怎么做
  - 不会耶……
  - 我尝试想了几种办法后，找到了一个相对简单的做法
  - 然后想了一种比较好理解的讲法

# handlerPivot 分析




找到比 40 小的数字，不要移动到 40 左边，因为这会导致整体位移；
只需要在第二步把 40 和最后一个小数字调换位置即可。
接下来我们研究第一步应该怎么做。

# 思路




遍历 arr[1] 到 arr[6]
比40小的数放到左边 small，比40大的数放到右边 big

我们知道 small 的起点，但不知道 small 的终点 x
我们知道 big 的终点，但不知道 big 的起点 x+1

那么这样吧：每次换到 small 的后面，big 的前面
直到 small.length + big.length === 6 就说明任务完成

```
const handlePivot = (arr, start, end) => {
 //start 和 end 表示数组的开始和结尾（不含结尾）
 if(end - start <= 1){ return arr.length-1}
 let pivot=arr[start]
 let smallEnd=start // small 的终点就在起点
 let bigStart=end // big 起点就在终点
 let i=start+1 // i 用于遍历
 while(bigStart - smallEnd>1){ // 两个数组没挨着
  if(arr[i]<pivot){
   smallEnd+=1; swap(arr, i, smallEnd)
   i+=1
  }else if(arr[i]>=pivot){
   bigStart-=1; swap(arr, i, bigStart)
  }
 }
 swap(arr, start, smallEnd)
 return smallEnd
}
```

# 代码续

```
const swap = (arr, a, b)=> {
  [arr[a], arr[b]] = [arr[b], arr[a]]
}

const arr = [40, 70,20,10, 60,50,30]
console.log(handlePivot(arr, 0, 7))
// 3
console.log(arr)
// [10, 30, 20, 40, 50, 60, 70]
```

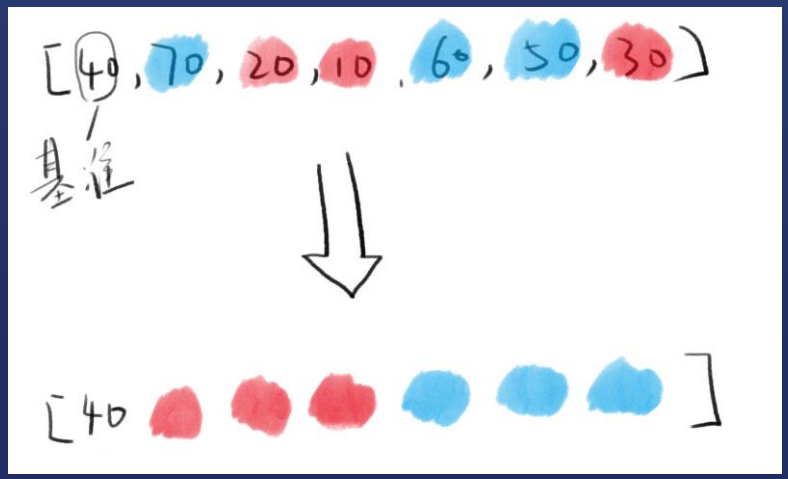
# 我的思考过程（都是错的）

思路一：从 i=1 开始遍历，发现比 40 大的就跟 arr[6] 交换位置
有问题：不能总是跟 arr[6] 交换位置，第二次要跟 arr[5] 交换位置
思路二：从 i=1 开始遍历，发现比 40 大的就跟 arr[x] 交换位置，x自减1

送分题：如果发现比 40 小的数字，你该怎么做？
答：放在原地不动，然后 i+=1

新问题：那你的遍历什么时候结束？i 大于 6 的时候结束显然不对
思路三：记录小数和大数的个数和，当个数和等于6的时候，遍历结束
有问题：大数的个数知道，但你怎么知道小数的个数？
答：那就加个变量 y 记录一下

以上是我在大脑里思考的问题
但，你不一定能理解我的想法



# 解决问题的方法就是先尝试解决问题

在每次尝试中，你会了解更多细节

大概尝试 10 次，你就得到了最终答案

```
const quickSort = arr =>
 _quickSort(arr, 0, arr.length)

const _quickSort = (arr, start, end) => {
 // start end 表示数组的开始和结尾，不含结尾
 if(end-start<=1){return arr}
 const pivotIndex = handlePivot(arr, start, end)
 _quickSort(arr, start, pivotIndex)
 _quickSort(arr, pivotIndex+1, end)
 return arr
}
```

# 总结

- 快速排序
  - ✓ 第一步：找到基准，把小的放左边，大的放右边
  - ✓ 第二步：对左右两边的数组执行第一步，直到遇到长度为 0 或 1 的数组才停止递归
- 难点
  - ✓ 第一步如何做到就地操作（通过 small 和 big）
  - ✓ 第二部如何做到就地操作（通过 start 和 end）
- 回顾
  - ✓ 数学思维简单而低效（可证明，可优化）
  - ✓ 人类思维复杂而高效（难证明，容易错）
  - ✓ 问思路就答数学思维，问效率就答人类思维

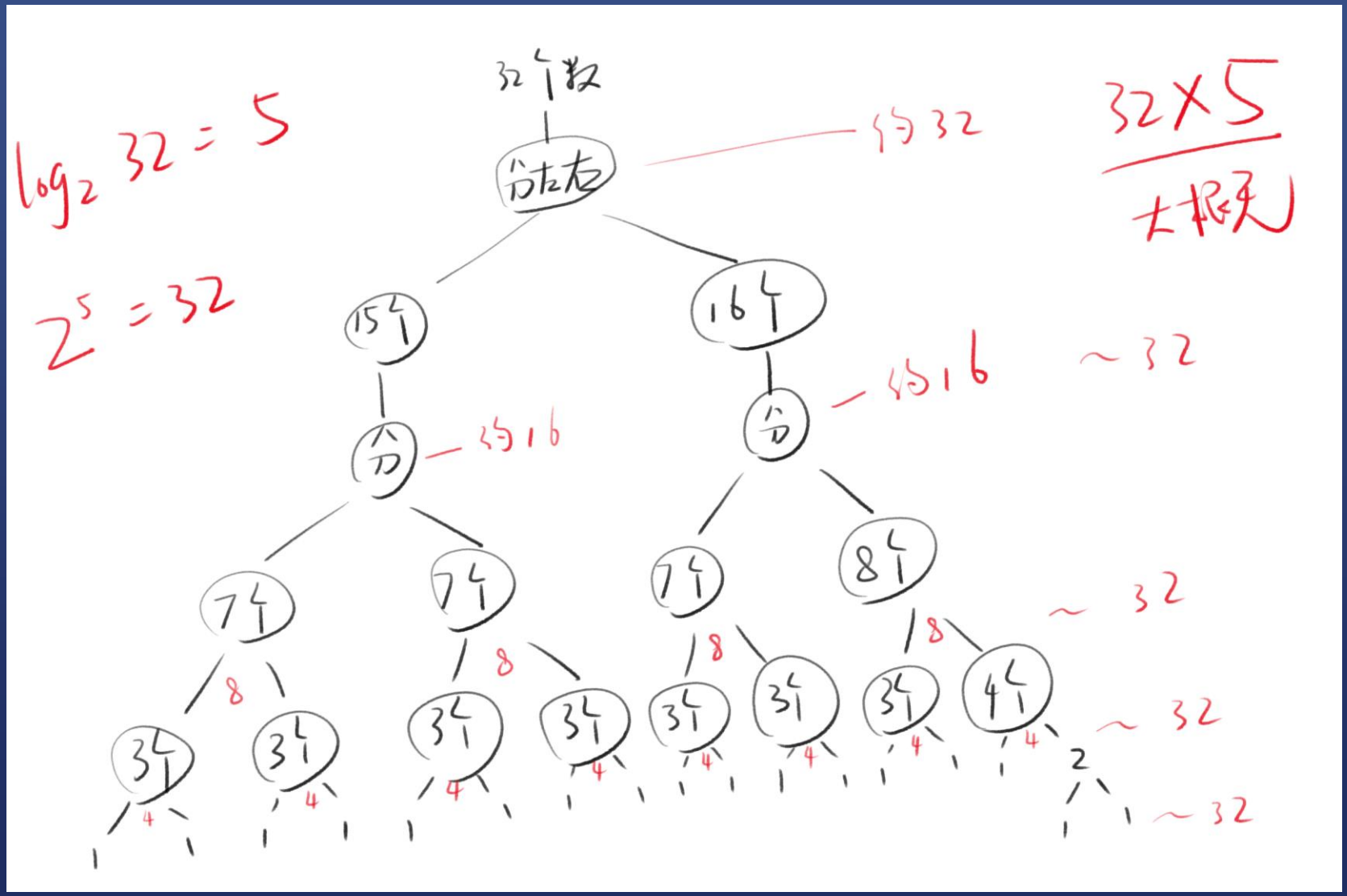
# 你自己写不出来代码

- 如果你想不出思路
  - ✓ 那就学(chao)习别人的思路
- 如果你的代码总是出错
  - ✓ 请拿出纸笔把排序过程走一遍、两遍、三遍
  - ✓ 看我如何用 console.log，这非常重要！
  - ✓ 你的代码一般都错在少加了个1或少减了个1

# 随机化快排

优化平均效率

# 目前的时间复杂度是多少



# 但是，如果你的运气不好

- 会遇到这种情况
  - 每次你选的基准数都是最小的或者最大的
  - 比如原数组为 [1,2,3,4,5,6,7]
  - 那么每次你都无法将数组分成两半
  - 每次，你只能选出最小或最大的数
  - n + (n-1) + (n-2) + ... 约为 n*n
  - 你的快排变成了选择排序，复杂度 O(n^2)
- 如何避免?
  - 每次随机选基准，总不可能次次都是最小的吧!
  - 只需要每次将 arr[random] 与 arr[0] 调换位置即可!

# 快排讲完了

有没有觉得算法很枯燥/很有意思?

老是类似的算法就很枯燥~

搞点新算法就很有意思!