# 归并排序

前端精进 - 科班方向

# 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

# 学算法什么最重要

技巧？编程语言？知道得多？

# 思路最重要

如何从零想出答案

# 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

# 数学思路

- 把未知变成已知
  - ✓ 五个数排序？不会！
  - ✓ 三个数排序？不会！
  - ✓ 两个数排序？会！

```
sort = ([a,b]) =>
  a > b ? [b, a] : [a, b]
```

  - ✓ 一个数排序？不用排！

- 分析
  - ✓ n = 1 和 n = 2 时可以搞定，n = 3、4、5 还不能搞定
  - ✓ 我们只需要找 n = 3、4、5 转化成 n = 2 的方法即可

# n = 3

- 分而治之
  - ✓ n = 3 可以化为两个数组 n = 2 和 n = 1
  - ✓ 将两个数组排好序，然后把两个数组结合起来
  - ✓ 怎么结合呢?
- 将 arr1 和 arr2 按顺序结合起来
  - ✓ arr1 = [2,5]  arr2 = [3,4]
  - ✓ arr3 = [2, 3, 4, 5]
  - ✓ 两只手分别指着两个数组的开头，谁小就复制谁

# n = 3, 4, 5...

- n = 4
  - ✓ 把数组分为 n = 2 和 n = 2，然后结合起来
- n = 5
  - ✓ 把数组分为 n = 2 和 n = 3，然后结合起来
- 找到了规律

# 排序算法搞定

$$sort(a_{1..n}) = \begin{cases} a_{1..n}, & n = 1 \\ a_1 > a_2 \ ? \ [a_2, a_1] : [a_1, a_2], & n = 2 \\ merge\left(sort\left(a_{1..n/2}\right), \ sort\left(a_{(n/2+1)..n}\right)\right), & n \geqslant 3 \end{cases}$$

实际上 n = 2 的情况是多余的，因为 merge 可以处理 n = 2 的情况

# merge 实现1

- 代码

```
merge = (a, b) => {
  let c = [], i = 0, k = 0
  while(i < a.length || k < b.length){
    if(k >= b.length)      { c.push(a[i]); i += 1 }
    else if(i >= a.length){ c.push(b[k]); k += 1 }
    else if(a[i] < b[k])  { c.push(a[i]); i += 1 }
    else                   { c.push(b[k]); k += 1 }
  }
  return c
}
```

- 如何证明这个代码是对的
  - ✓ 每次 push 的都是当前最小的，所以应该是成立的
  - ✓ 换成数学思路更好证明
- 操作次数
  - ✓ i + k

# merge 实现2

- 代码

```
merge = (a, b) =>
  a.length === 0 ? b :
  b.length === 0 ? a :
  a[0] > b[0] ?
     [b[0]].concat(merge(a, b.slice(1))) :
     [a[0]].concat(merge(a.slice(1), b))
```

- 数学归纳法
  - ✓ 当 i=0,k=0 时成立
  - ✓ 若 i=n,k=n 时成立，那么 i=n+1,k=n 时也成立
  - ✓ 若 i=n,k=n 时成立，那么 i=n,k=n+1 时也成立
- 操作次数
  - ✓ 有点难算，因为 concat、slice 的实现思路未知

# 接下来实现 sort

merge 已经搞定

# sort 实现1

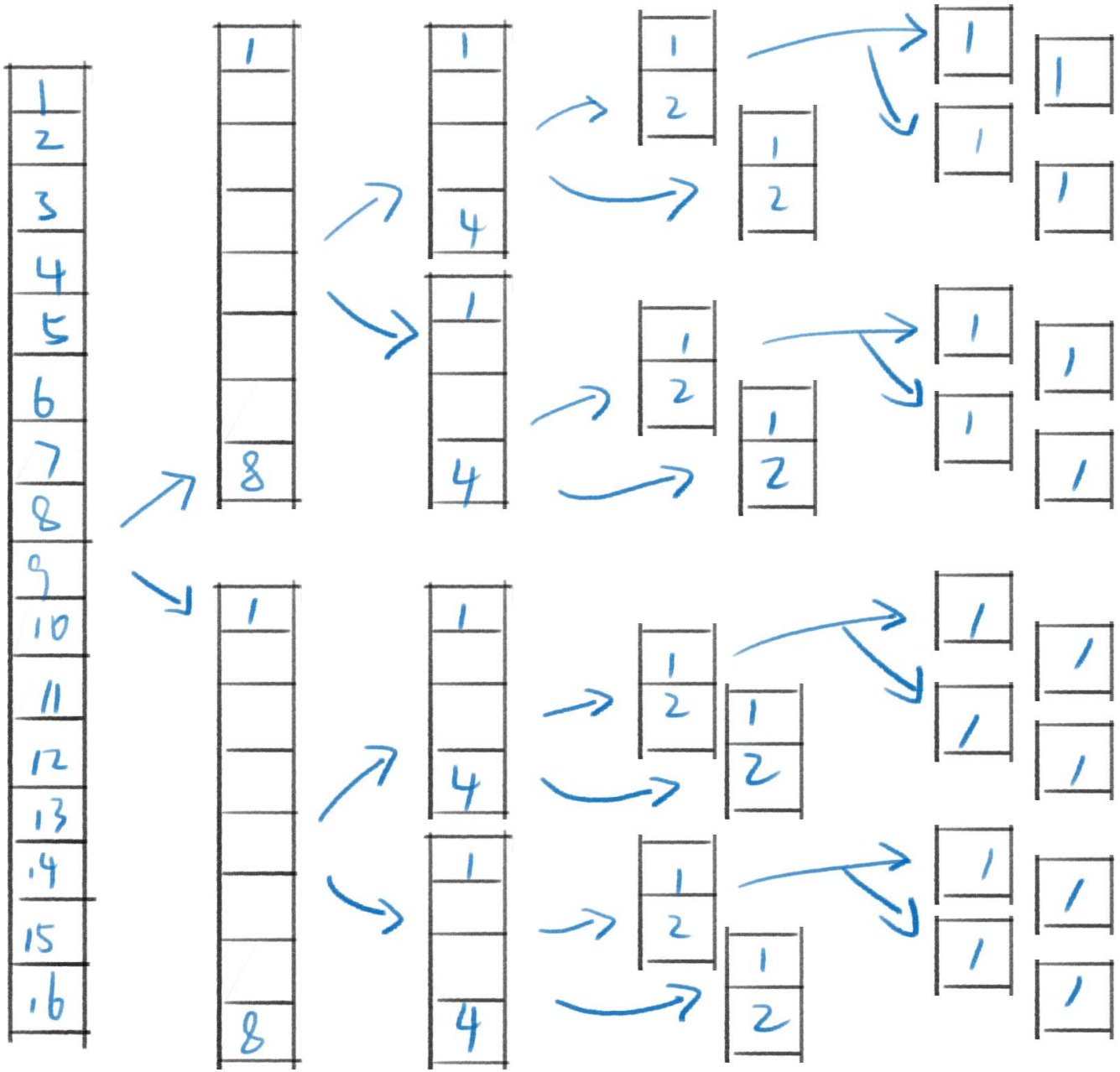
- 代码

```
sort = arr =>{
  let k = arr.length
  if(k===1){return arr}
  // 写到这里，发现 k=2 的情况没必要单独写
  let left = arr.slice(0, Math.floor(k/2))
  let right = arr.slice(Math.floor(k/2))
  return merge(sort(left), sort(right))
}
```

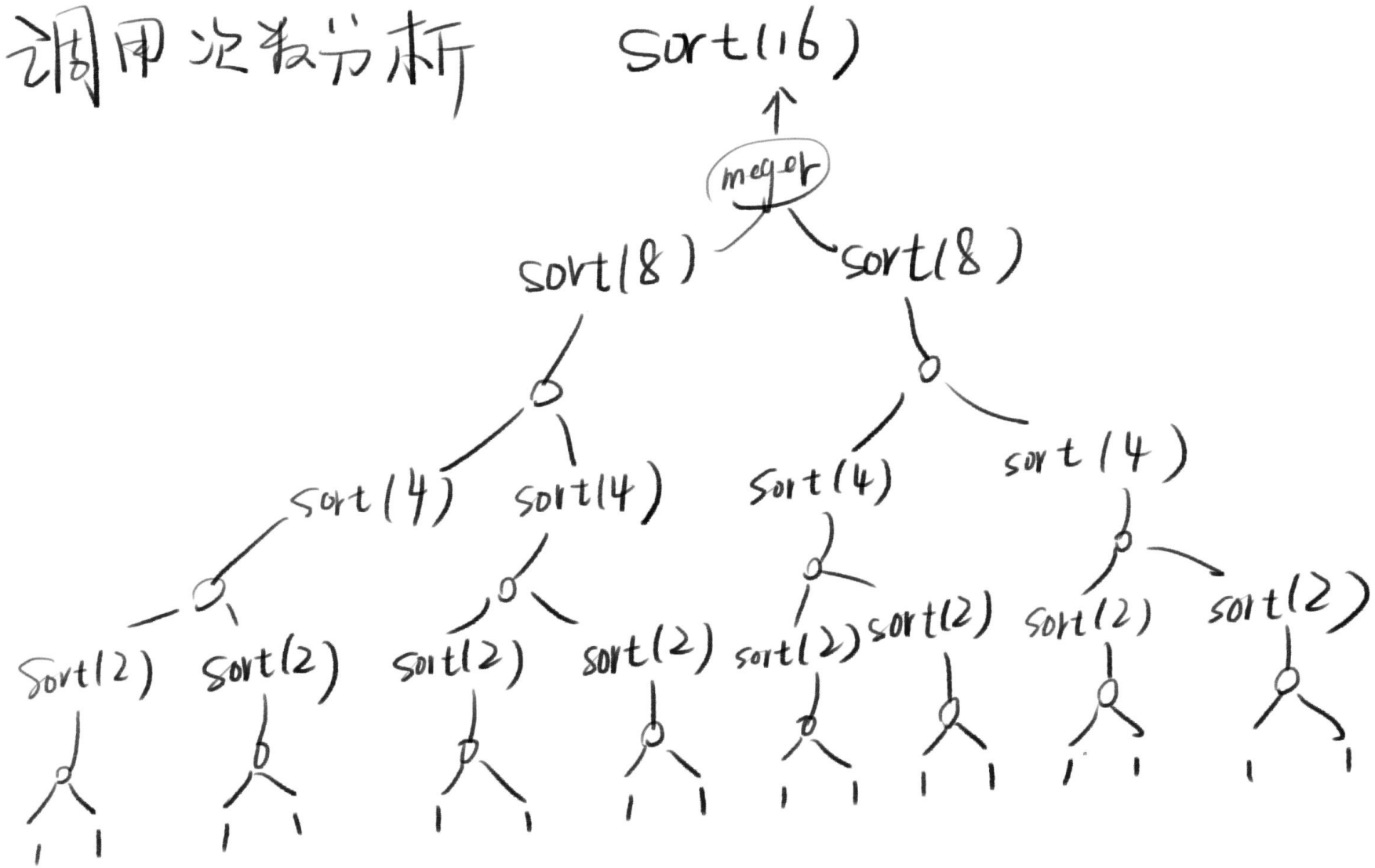
- 分析

✓ 每次 sort，会 slice 两次，内存翻倍，共翻 4 倍
✓ 每次 sort，会压栈三次，压栈很多次

# 内存分析


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
1
8
1
4
1
2
1
2
1
1
1
1
1
4
1
2
1
2
1
1
1
1
1
8
1
4
1
2
1
2
1
1
1
1
1
4
1
2
1
2
1
1
1
1

# sort 出现 31 次，merge 出现 15 次



# 又耗内存，又耗 CPU

这个 sort 你拿得出手吗

# 如何节约内存

就地操作

# 就地 merge

- 代码

```
merge = (a, middle) => {
  // [0..middle) 是排好序的
  // [middle..length) 也是排好序的
  // 要让 a 整体排好序
  let i = 0, k = middle
  while(i<middle && k<a.length){
    let w = 0
    while(a[i]<=a[k] && i<middle){i+=1}
    while(a[i]>=a[k] && k<a.length){k+=1;w+=1}
    let part = a.splice(k-w, w)
    a.splice(i, 0, ...part)
    i += w
    middle += w
  }
  return a
}
```

# 人类思维有什么问题

只有写代码的人看得懂……

# 人类思维如何改进

- 打 log
  - ✓ 将每一步用优美的log打出来分析
- 单元测试
  - ✓ 使用大量的测试用例保证代码大概正确
  - ✓ 目前我们先人肉测试，方便新手

# 就地 sort

```
sort = arr =>
  inplace_sort(arr, 0, arr.length)


inplace_sort = (arr, start, end) => {
  if(end - start <= 1) return arr
  let middle = parseInt((start+end)/2)
  inplace_sort(arr, start, middle)
  inplace_sort(arr, middle, end)
  merge(arr, start, middle, end)//需要改merge
  return arr
}
```

# 改 merge 以符合 sort 需求

```
merge = (a, start, middle, end) => {
  // [start..middle) 是排好序的
  // [middle..end) 也是排好序的
  let i = start, k = middle
  while(i<middle && k<end){
    let w = 0
    while(a[i]<=a[k] && i<middle){i+=1}
    while(a[i]>=a[k] && k<end){k+=1;w+=1}
    let part = a.splice(k-w, w)
    a.splice(i, 0, ...part)
    i += w
    middle += w
  }
  return a
}
```

# 内存优化好了

禁用 concat 和 slice，通过 in-place 搞定

# 调用次数需要优化么

目前复杂度已经很低了，所以可以不优化

# 空间换时间

如计数排序，计算次数很少，以后再讲

# 归并排序还有另一种思路

自顶向下 V.S. 自底向上

# 自底向上的归并排序

- 假设有 16 个数字
  - ✓ 先两两合并、再四四合并、再八八合并

[3,1,7,9, 16,14,8,2, 15,4,13,6, 5,10,11,12]
[1,3][7,9][14,16][2,8][4,15][6,13][5,10] [11,12]
[1,3,7,9] [2,8,14,16] [4,6,13,15] [5,10,11,12]
[1,2,3,7,8,9,14,16]  [4,5,6,10,11,12,13,15]
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]

# sort 实现3

```
sort = arr => {
  for(let size=1;size<arr.length;size*=2){
    console.log(`size:${size}`)
    for(let w=0;w<arr.length;w+=size*2){
      console.log(`w:${w}`)
      console.log(`arr:${arr}`)
      merge(arr,w,w+size,w+size*2)
      console.log(`arr:${arr}`)
    }
  }
  return arr
}
```

# 改 merge 以符合 sort 需求

```
merge = (a, start, middle, end) => {
  if(start >a.length){start =a.length-1}
  if(middle>a.length){middle=a.length}
  if(end   >a.length){end   =a.length}
  let i = start, k = middle
  while(i<middle && k<end){
    let w = 0
    while(a[i]<=a[k] && i<middle){i+=1}
    while(a[i]>=a[k] && k<end){k+=1;w+=1}
    let part = a.splice(k-w, w)
    a.splice(i, 0, ...part)
    i += w
    middle += w
  }
  return a
}
```

# 总结

- 思路
  - ✓ 遇到问题
  - ✓ 分析 n = 1、n = 2、n = 3 的区别
  - ✓ 发现大问题可以化成三个小问题
  - ✓ 总结成数学公式
  - ✓ 转化成代码，一个 merge 一个 sort
  - ✓ 写完代码发现性能很差
  - ✓ 使用 in-place 节省内存
  - ✓ 清空大脑，发现还有其他思路