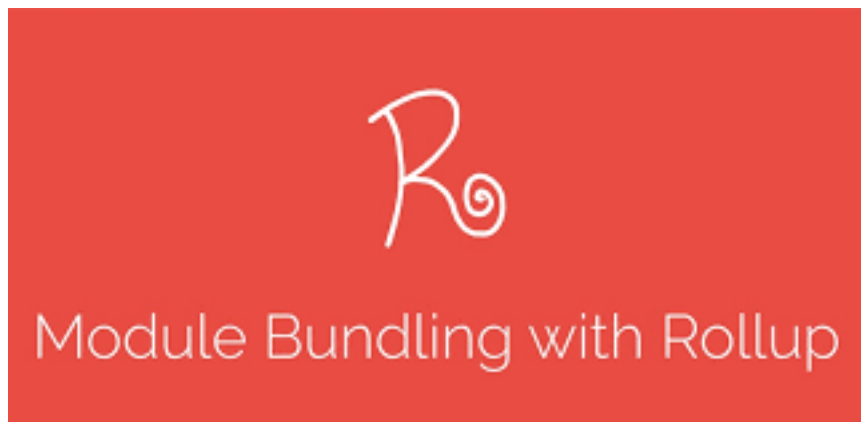


# 前端算法数据结构场景 介绍

例举前端算法应用场景和讲解学习算法的重要性

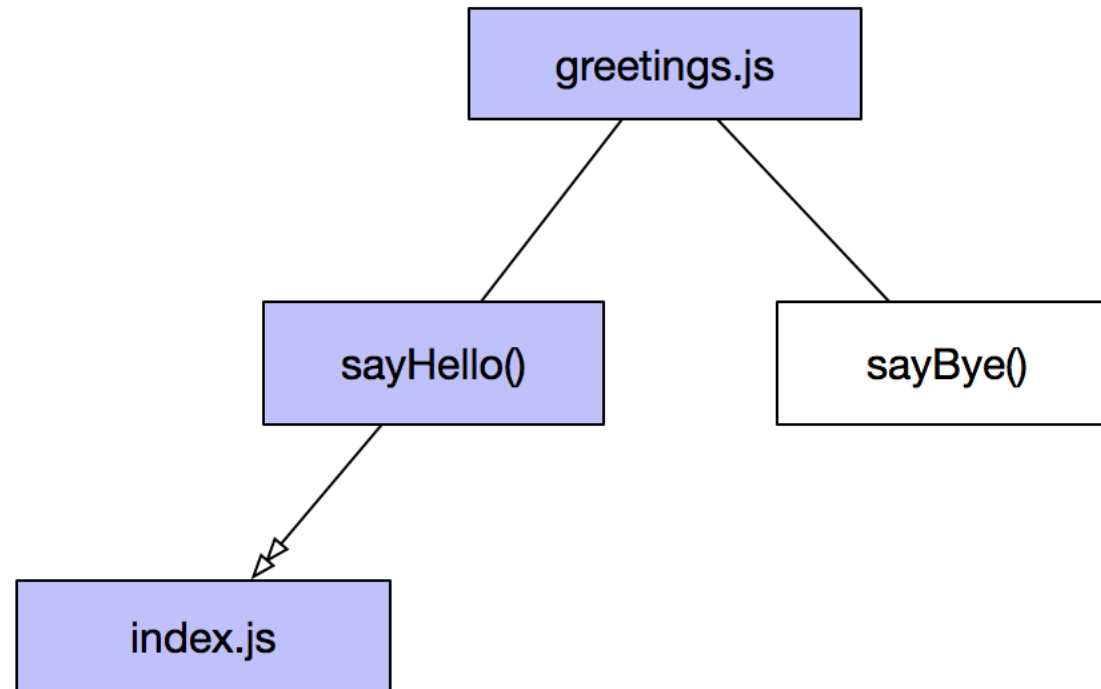
# 图相关算法



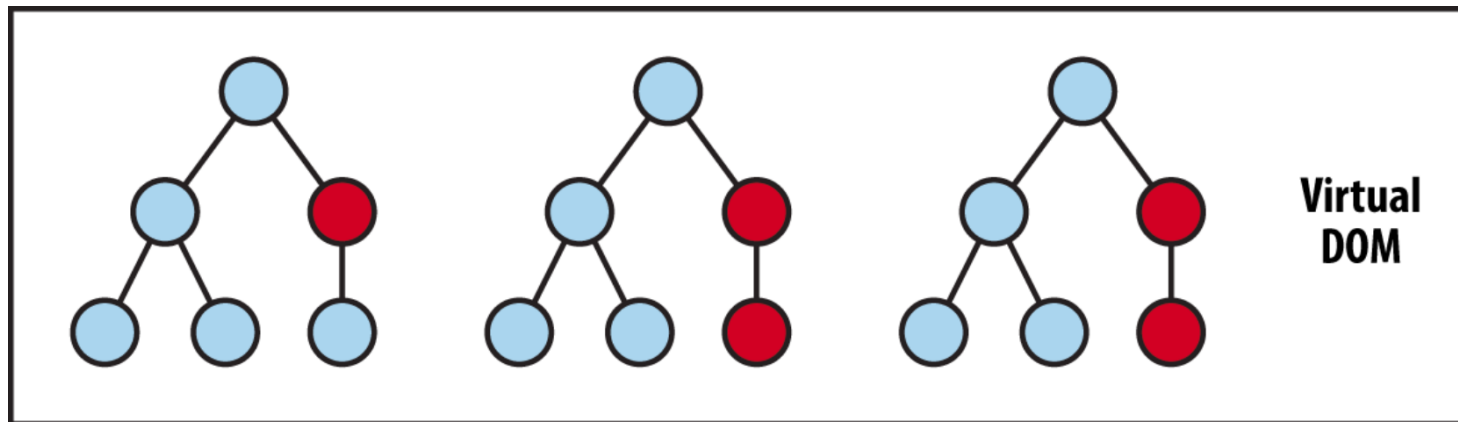
rollup使用tree-shaking算法，检测用不到的代码，减小包的大小

```
/* greetings.js */  
  
export function sayHello() {  
  console.log('Hello')  
}  
  
export function sayBye() {  
  console.log('Bye')  
}
```

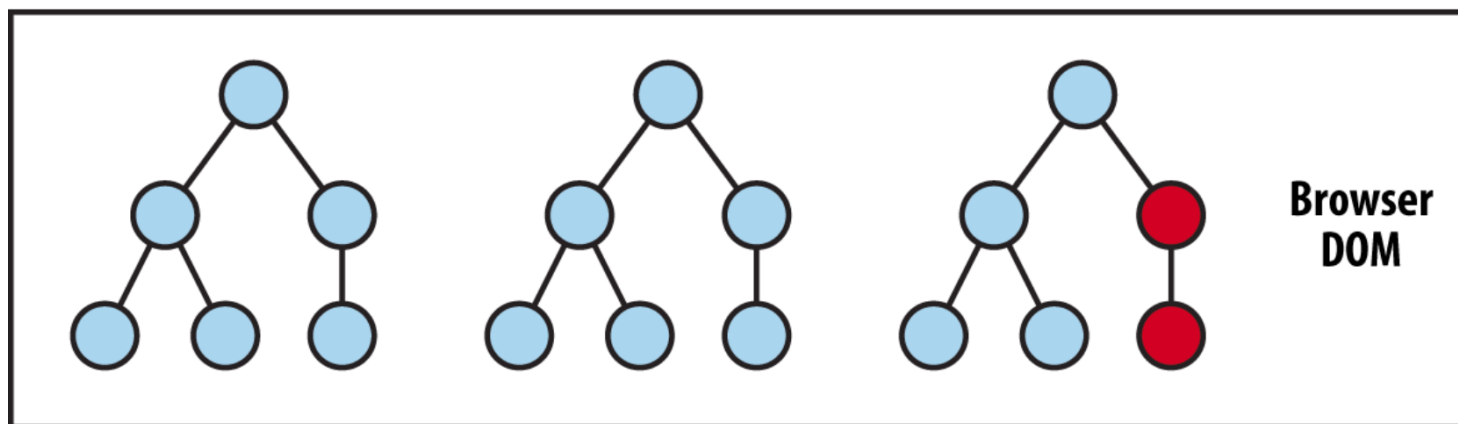
```
/* index.js */  
  
import { sayHello } from './greetings'  
  
sayHello()
```



# 树 ( DOM-DIFF ) 算法



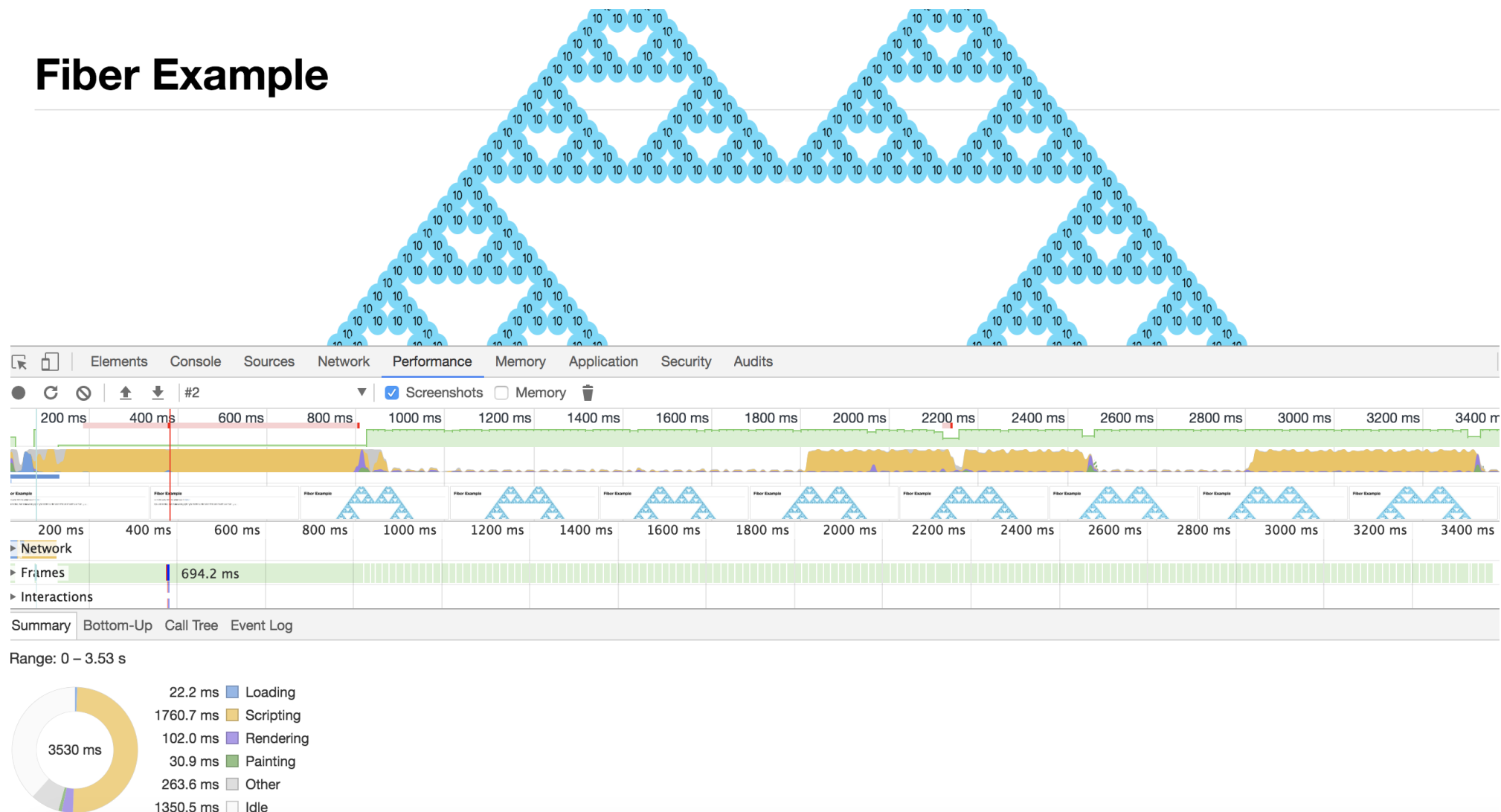
状态变化 —————> 计算变化 —————> 重新渲染



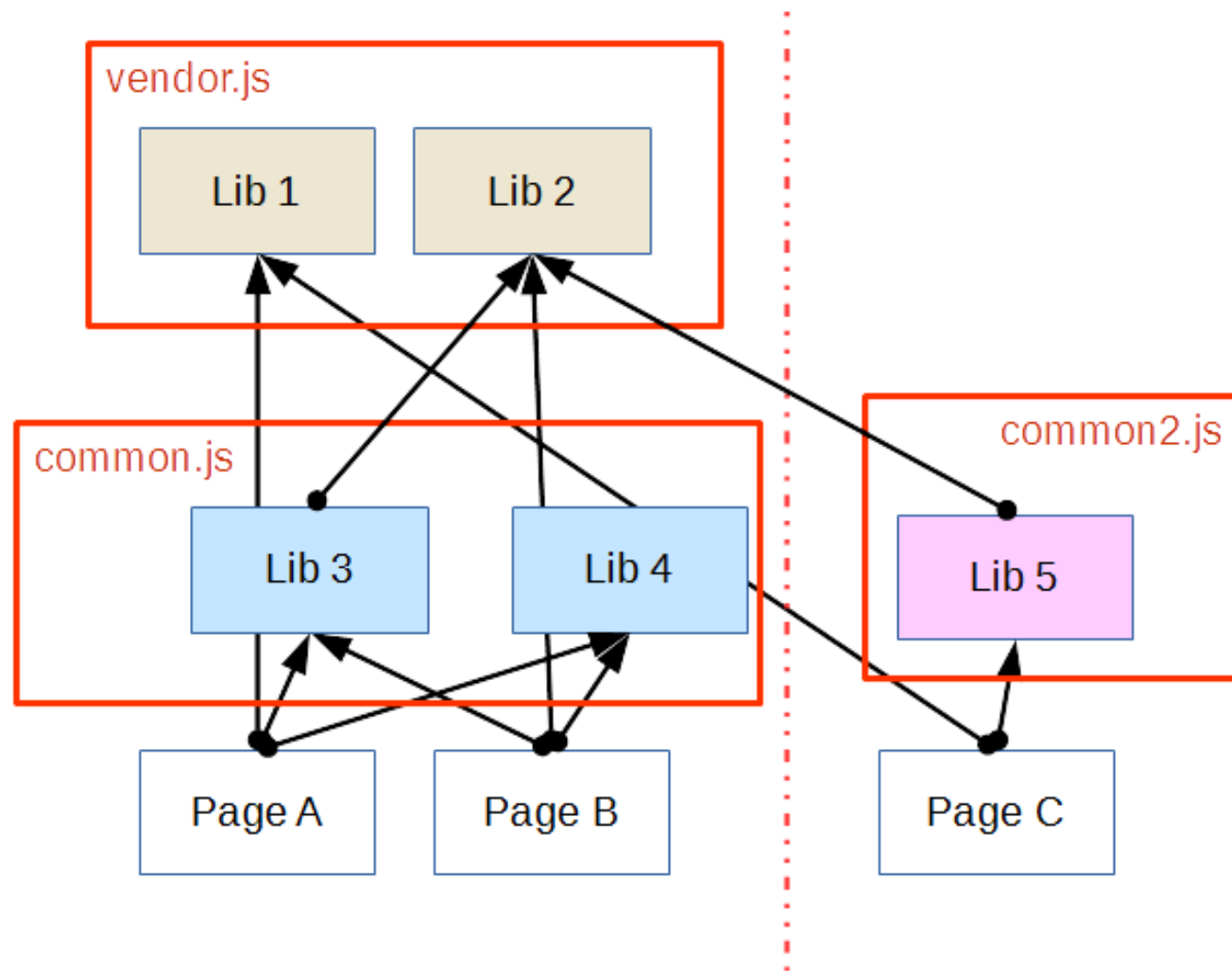
React使用基于树的调和算法计算需要变化的节点

# 队列和调度算法(React Fiber)

## Fiber Example

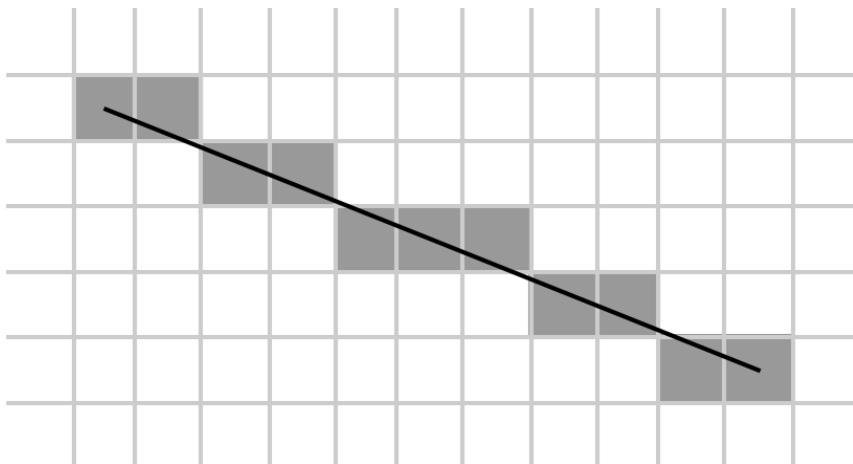


## 图论(Webpack split chunk plugin的计算)

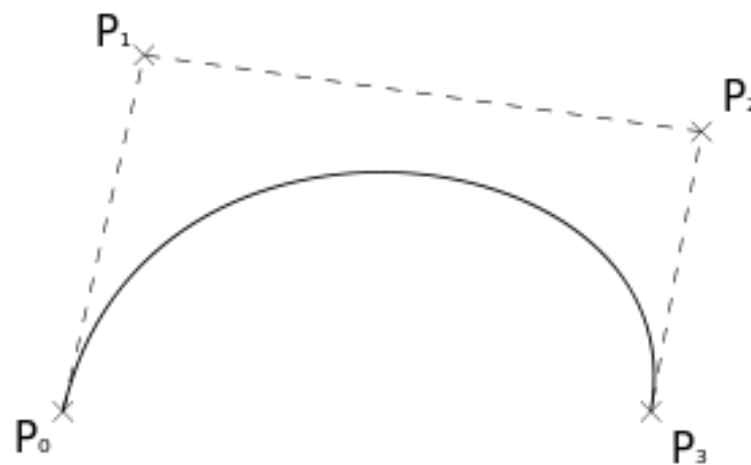


# 图形算法

SVG和Canvas绘图底层的算法，衍生出d3.js,highcharts,echarts,canvas.js等等一系列的图标库；以及构成html中渲染的基础

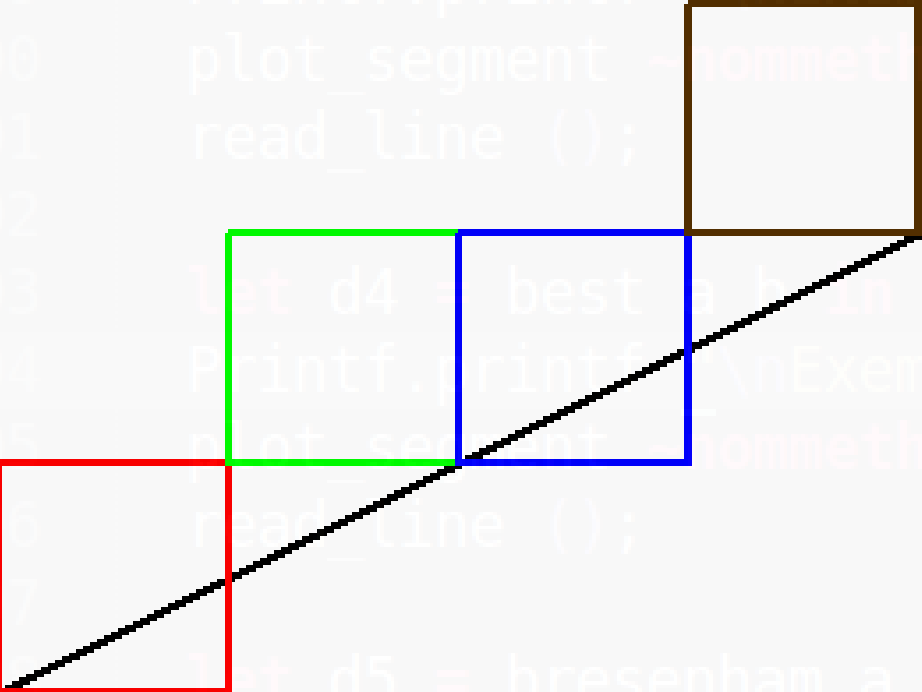


Bresenham绘线算法



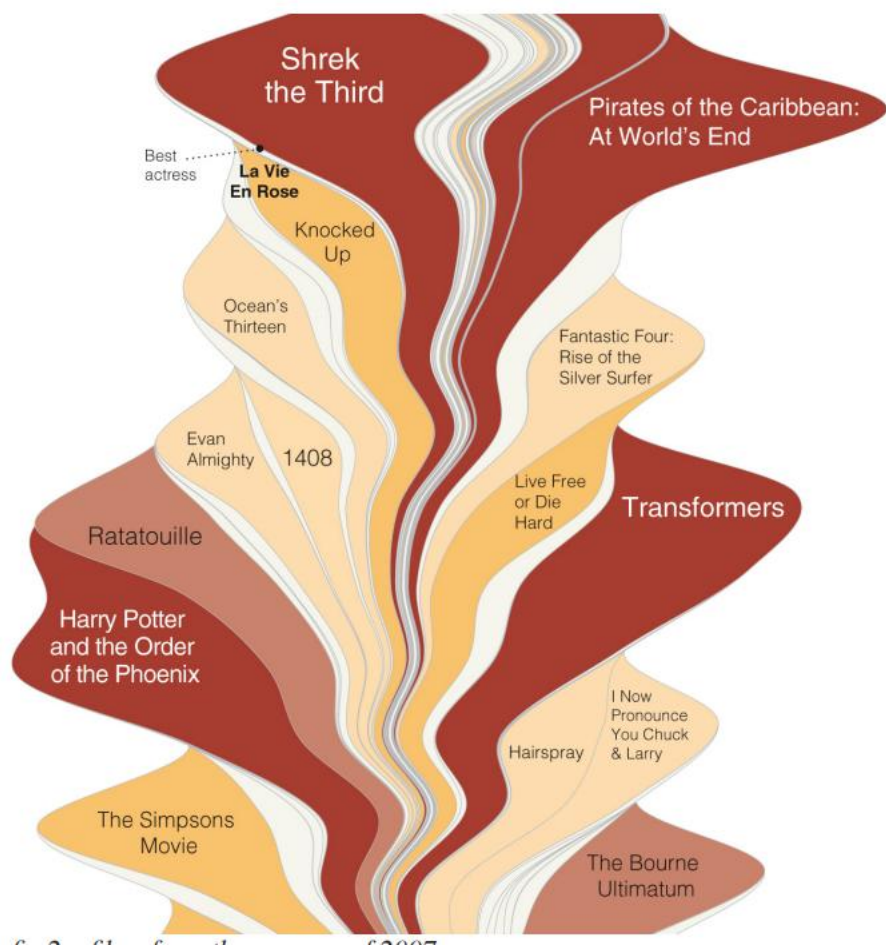
贝塞尔曲线

```
+186 read_line ();
+187
+188 let d3 = upper a b in
+189 Printf.printf "\nExemple par la méthode de
+190 plot_segment ~frommethode:"sélection supérieure";
+191 read_line ();
+192
+193 let d4 = best a b in
+194 Printf.printf "\nExemple par la méthode de
+195 plot_segment ~frommethode:"sélection optimale";
+196 read_line ();
+197
+198 let d5 = bresenham a b in
+199 Printf.printf "\nExemple par la méthode de
```





# 数据可视化算法



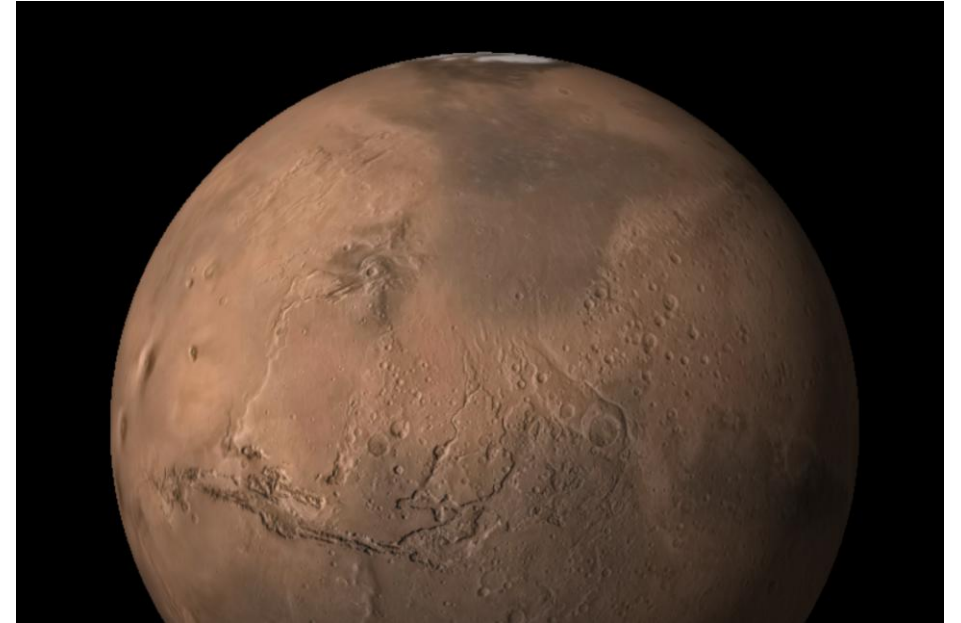
2017年夏天电影的欢迎程度

利用数据可视化算法，绘制图表，发现电影、音乐、服装等等的流行趋势。一些名词的受欢迎程度，作者的影响力等等。

## 3D相关算法



normanvr: 用vr控制web 3D动画



tree.js绘制的地球3d模型

<https://www.nationalgeographic.com/science/2016/11/exploring-mars-map-panorama-pictures/>

# 算法执行的环境

CPU+内存的模型

# 随机访问存储器

内存地址	数据
00000000	00000000
00000001	00000001
00000002	00000002
00000003	00000003
00000004	00000004
00000005	00000005
...	...
FFFFFFFF	FFFFFFFF

你可以把计算机的内存想象成一个非常大的数组，你可以像读取数组中元素一样读取内存中的值。读取内存中任意位置的值，消耗的时间是相同的。

变量赋值

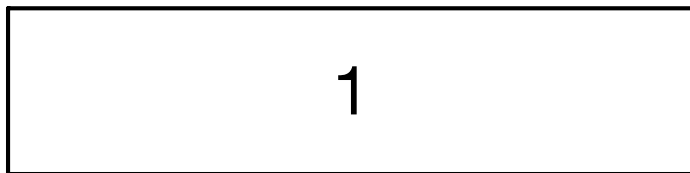
var **x** = 1

内存地址

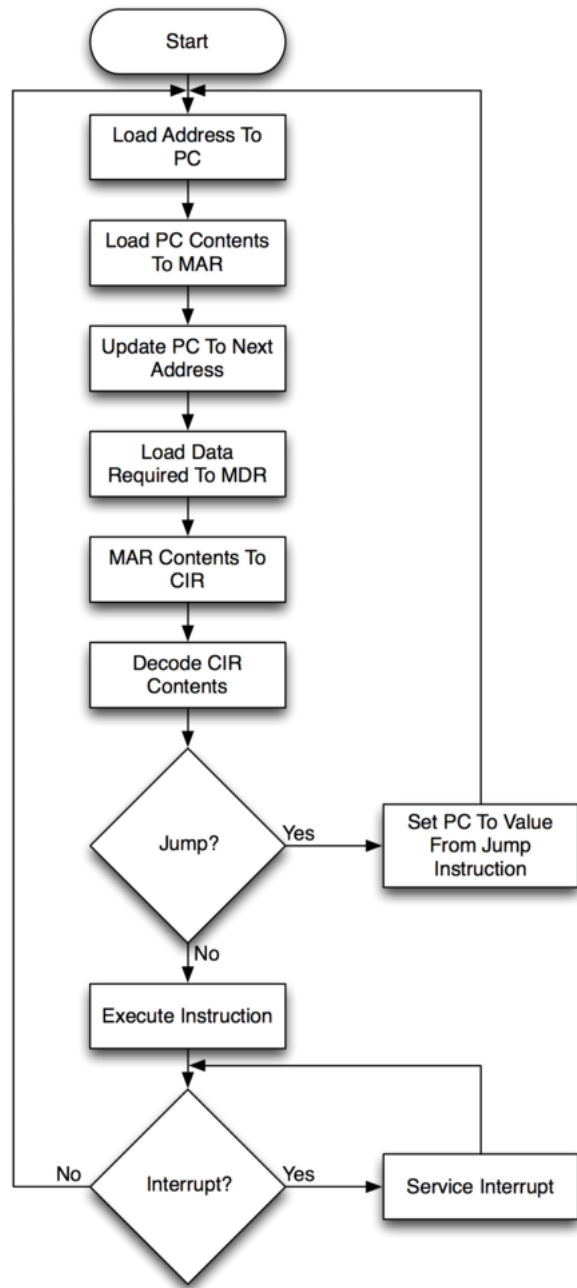
数据

00003456

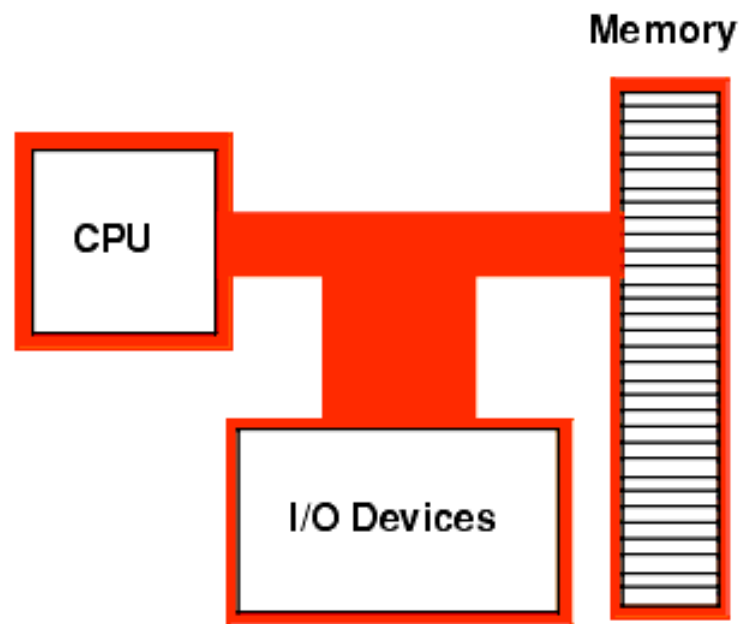
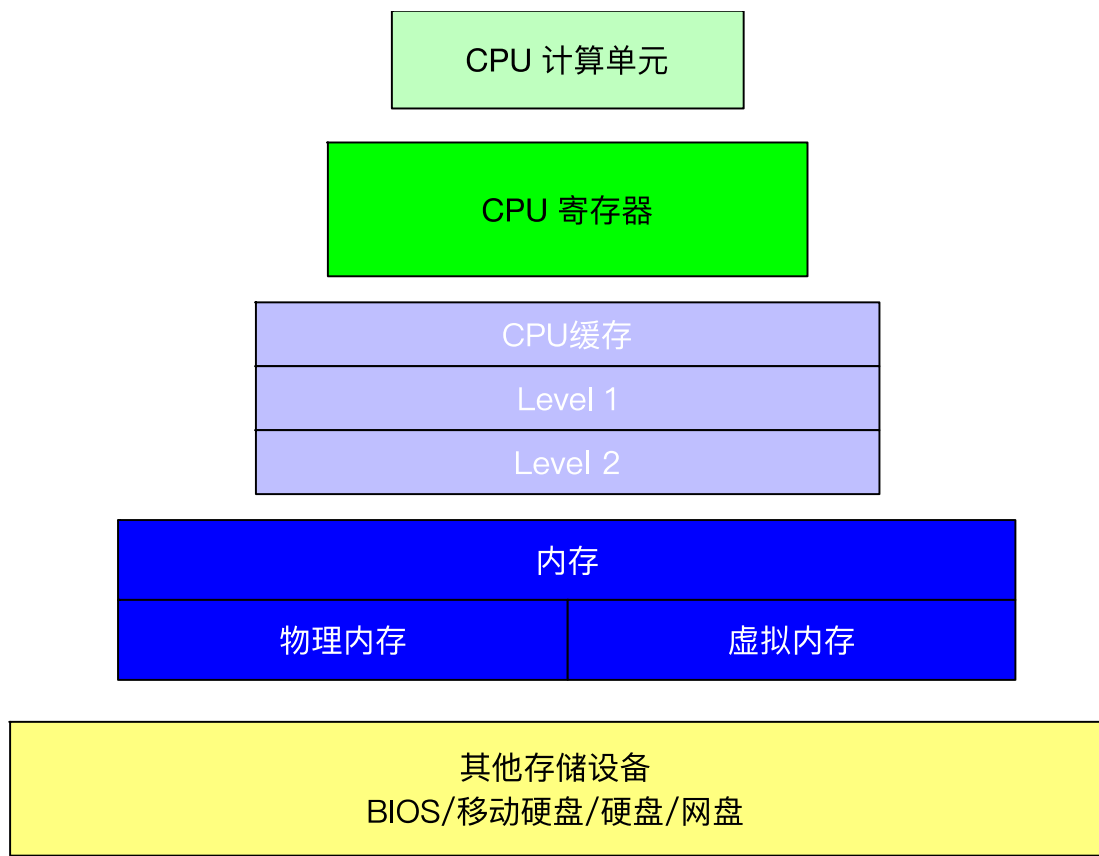
x



变量名实际代表了内存中的一个值



# 算法执行的环境



# 算法的复杂度

衡量算法好坏的标准



# O(n)的算法-寻找数组最大值

i=0 max=负无穷	9	5	-3	-9	20	1	6
i=1 max=9	9	5	-3	-9	20	1	6
i=2 max=9	9	5	-3	-9	20	1	6
i=3 max=9	9	5	-3	-9	20	1	6
i=4 max=9	9	5	-3	-9	20	1	6
i=5 max=20	9	5	-3	-9	20	1	6
i=6 max=20	9	5	-3	-9	20	1	6
i=7 max=20	9	5	-3	-9	20	1	6

$$T(100) = ? * 100$$

$$T(200) = ? * 200$$

$$T(300) = ? * 300$$

```
1  function find_max(arr){  
2      let max = Number.NEGATIVE_INFINITY  
3      for(let i = 0; i < arr.length; i++){  
4          max = (arr[i] > max ? arr[i] : max)  
5      }  
6      return max  
7  }
```

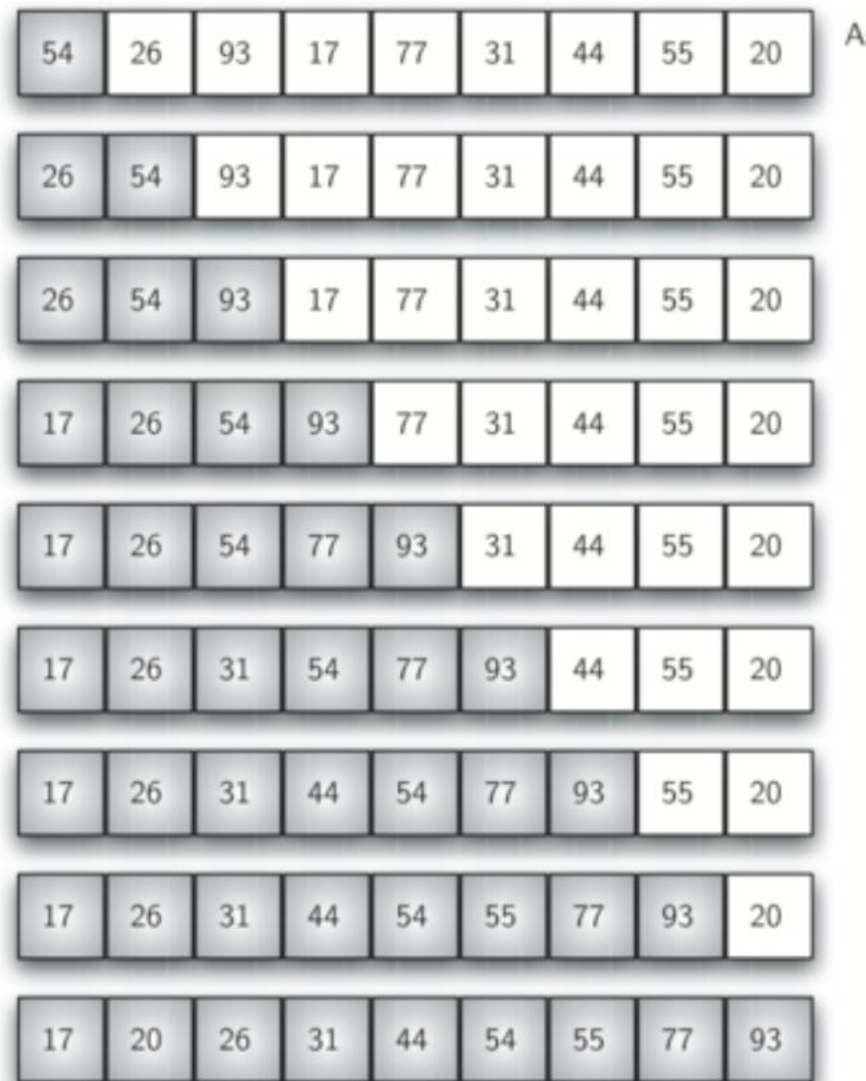
插入排序 ( $O^2$ )

6 5 3 1 8 7 2 4

# $O(n^2)$ 的算法——插入排序



**循环不变式：**每次循环结束，存在一个已经排序的列表和一个未排序的列表， $j$ 指向下一个未排序的数字



```

4  function insertion_sort(A) {
5      for (
6          let j = 1;                // 1
7          j < A.length ;           // N
8          j++) {                   // N - 1
9          const key = A[j]         // N - 1
10         let i = j - 1            // N - 1
11
12         // 这个循环将抓到的牌插入合适的位置
13         while (i >= 0 && A[i] > key) { // Mk
14             A[i + 1] = A[i]       // (Mk-1)
15             i--                   // (Mk-1)
16         }
17         A[i + 1] = key            // N - 1
18
19         //                      j
20         // | --- 已排序 --- | ---- 未排序 ---- |
21         // 每次循环结束的时候j的位置代表下一张需要排序的牌
22     }
23 }

```

- 第6行执行1次
- 第7行执行N次
- 第8行执行N-1次
- 第9、10、17行执行N-1次
- 第13行-第15行执行时间不固定（有最坏和最好情况）  
最好情况下执行N-1次；最坏情况下执行  
 $1+2+3+4+\dots+N-1$ 次

## 等差数列求和公式

- 形如 $1+2+3+4+\dots+N$ 的数列，求和公式：

$$S_n = \frac{(a_1 + a_n)n}{2}$$

$a_1$ 是数列的第一项， $a_n$ 是数列的第 $n$ 项

$$1 + 2 + 3 + \dots + N - 1 = (1 + N - 1) \frac{N - 1}{2} = \frac{N(N - 1)}{2}$$

## 插入排序的执行时间

**最坏情况:**

$$\begin{aligned}T(n) &= 1 + N + 4(N - 1) + \frac{N(N - 1)}{2} = 1 + 5N - 4 + \frac{N^2}{2} - \frac{N}{2} \\&= \frac{N^2}{2} + \frac{9}{2}N - 3 = O(N^2)\end{aligned}$$

**最好情况:**

$$T(n) = 1 + N + 4(N - 1) = 1 + 5N - 4 = 5N - 3$$

最坏情况决定性的量为 $N^2$ ，最好情况下决定性的量为 $N$

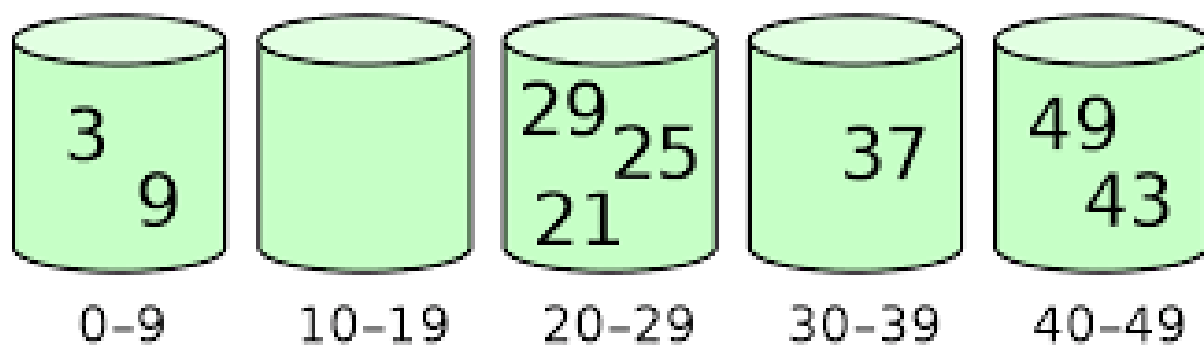
另一个 $O(n^2)$ 的算法——冒泡排序

6 5 3 1 8 7 2 4



## $O(n)$ 的算法——桶排序

29 25 3 49 9 37 21 43

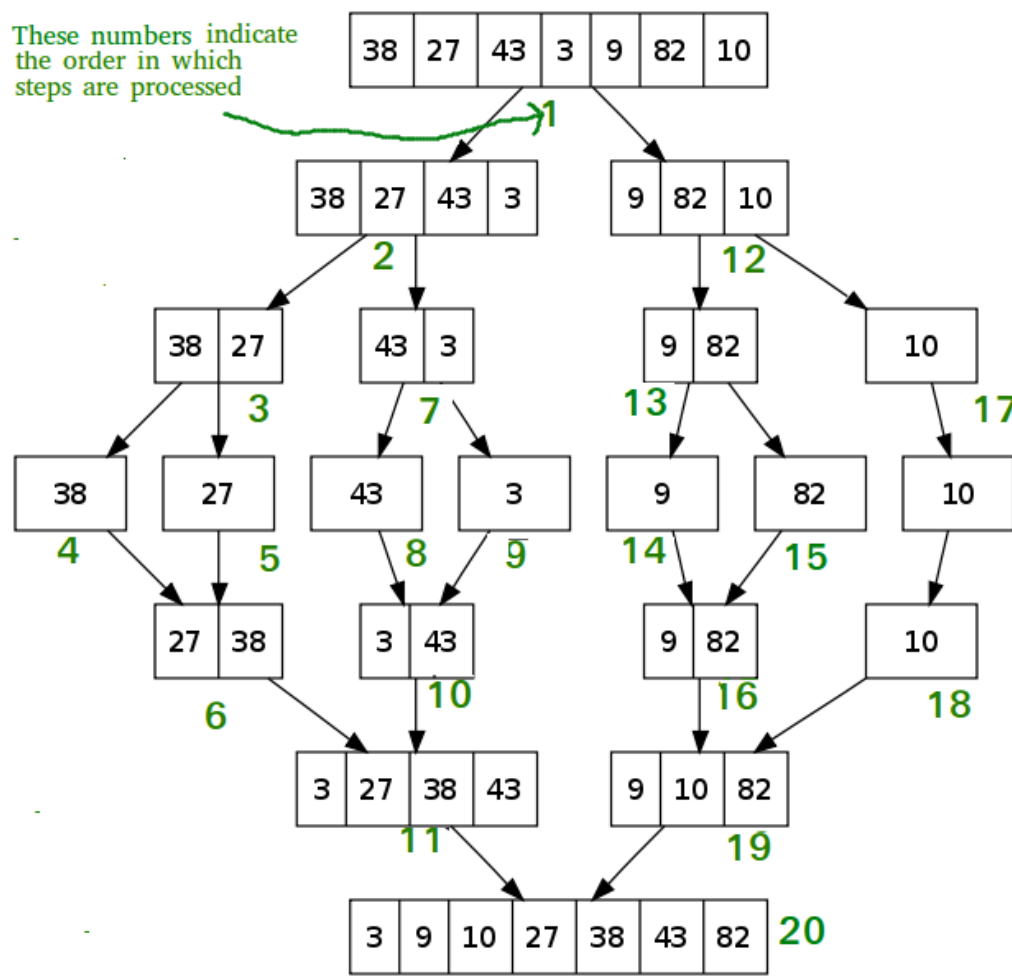


```
1  function bucket_sort(A){
2      const buckets = [...Array(1000)].map(x => [])
3      for(let i = 0; i < A.length; i++){
4          buckets[A[i]].push(A[i])
5      }
6      let result = []
7      for(let i = 0; i < buckets.length; i++){
8          if(buckets[i].length > 0){
9              buckets[i].forEach(x => {
10                 result.push(x)
11             })
12         }
13     }
14     return result
15 }
```

## 归并排序

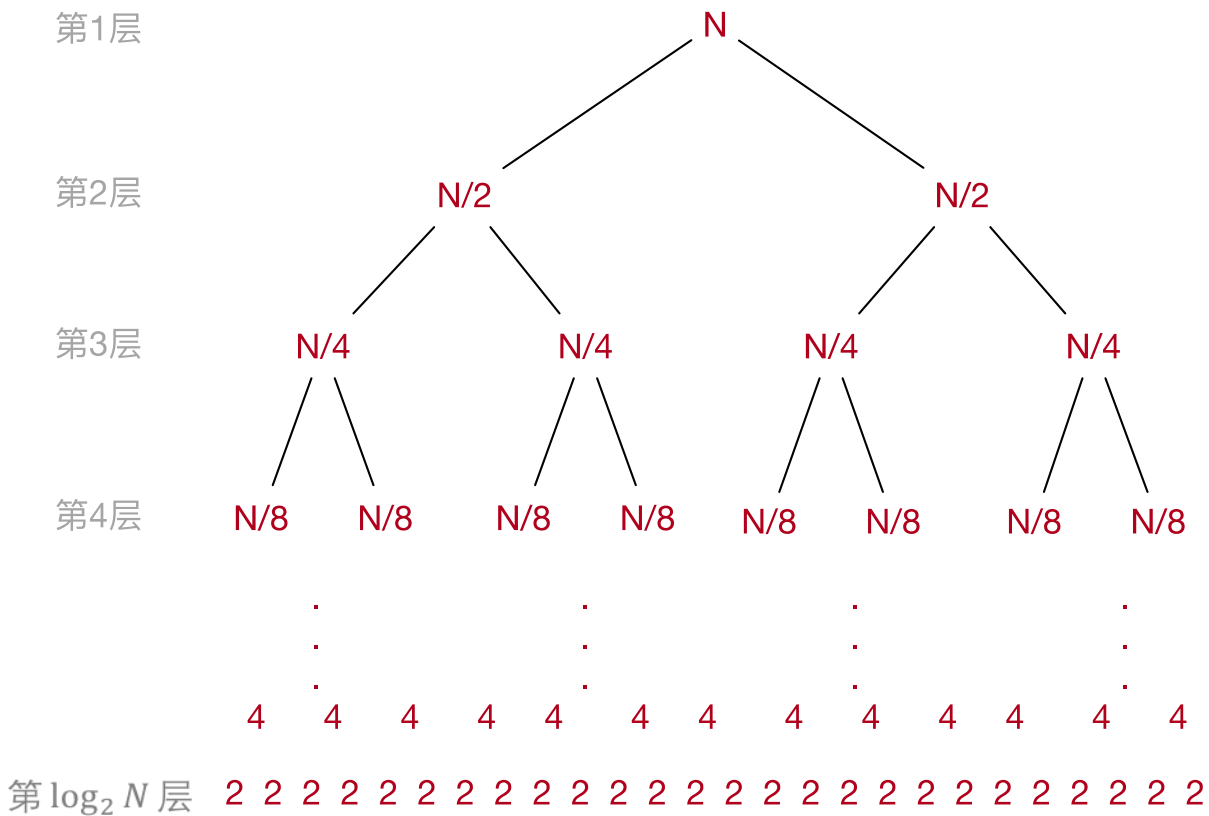
6 5 3 1 8 7 2 4

## $O(n\log n)$ 的算法——归并排序



首先递归不断将数组拆分成更小的数组，  
然后再将这些子数组两两组合排序

## 归并排序复杂度分析(拆分过程)



拆分用时: 1

在此处键入公式。拆分可以在常数时间内执行完成。

拆分用时: 2

拆分用时: 4

拆分用时: 8

拆分用时： $2^{\log_2 N/4} = \frac{N}{4}$

**$2^y = x$  等价于  $\log_2 x = y$**

$$2^{\log_2 x} = 2^y = \mathbf{x}$$

## 拆分用时计算(等比数列求和)

- 形如 $1+2+4+8+\dots+2^N$ 的数列如何求和呢？

$$S_n = \frac{a(1-r^n)}{1-r}$$

$a$  第一项,  $r$  公比,  $n$  项数

证明：

$$S_n = a + ar + ar^2 + \dots + ar^{n-1}$$

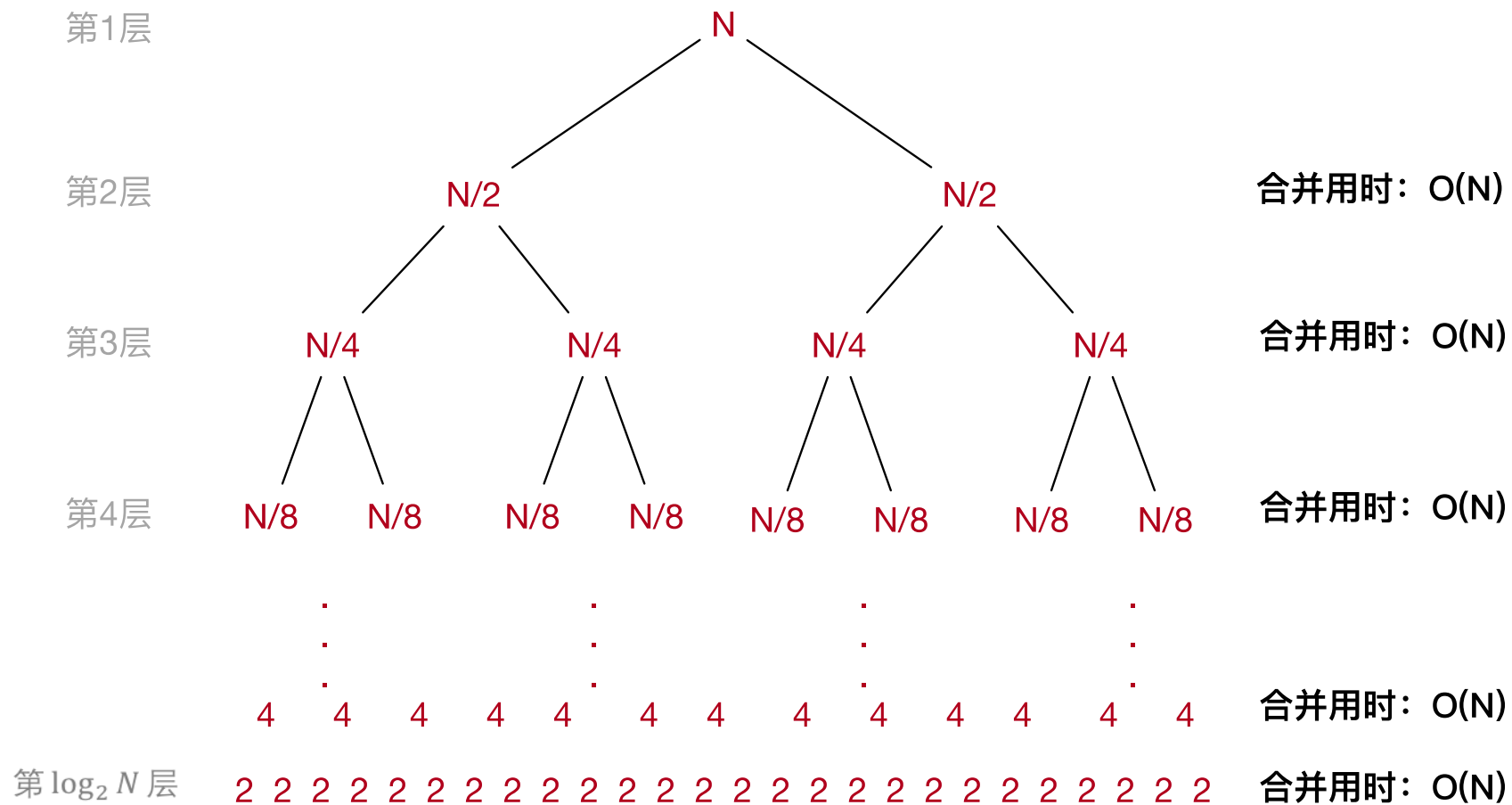
$$rS_n = ar + ar^2 + ar^3 + \dots + ar^n$$

$$S_n - rS_n = a - ar^n$$

$$S_n = \frac{a(1-r^n)}{1-r}$$

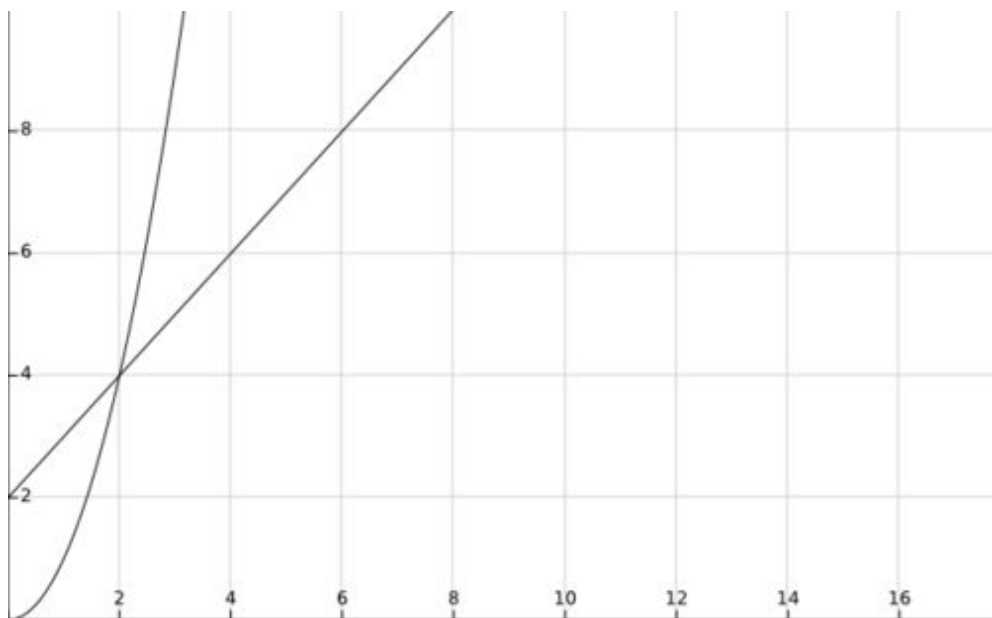
$$\text{拆分用时：} \quad 1 + 2 + 4 + \dots + \left(\frac{N}{2}\right) = 1 \frac{1 - 2^{\log_2 \frac{N}{2}}}{1 - 2} = 2^{\log_2 \frac{N}{2}} - 1 = \frac{N}{2} - 1 = O(N)$$

# 合并用时



$$\text{合并用时} = O(N) * \log_2 N = O(N \log N)$$

# O(N)和O(N^2)算法对比



	O(n)	O(n^2)
10	10	100
100	100	10^4
1000	1000	10^6
10000	10000	10^8
100000	100000	10^10



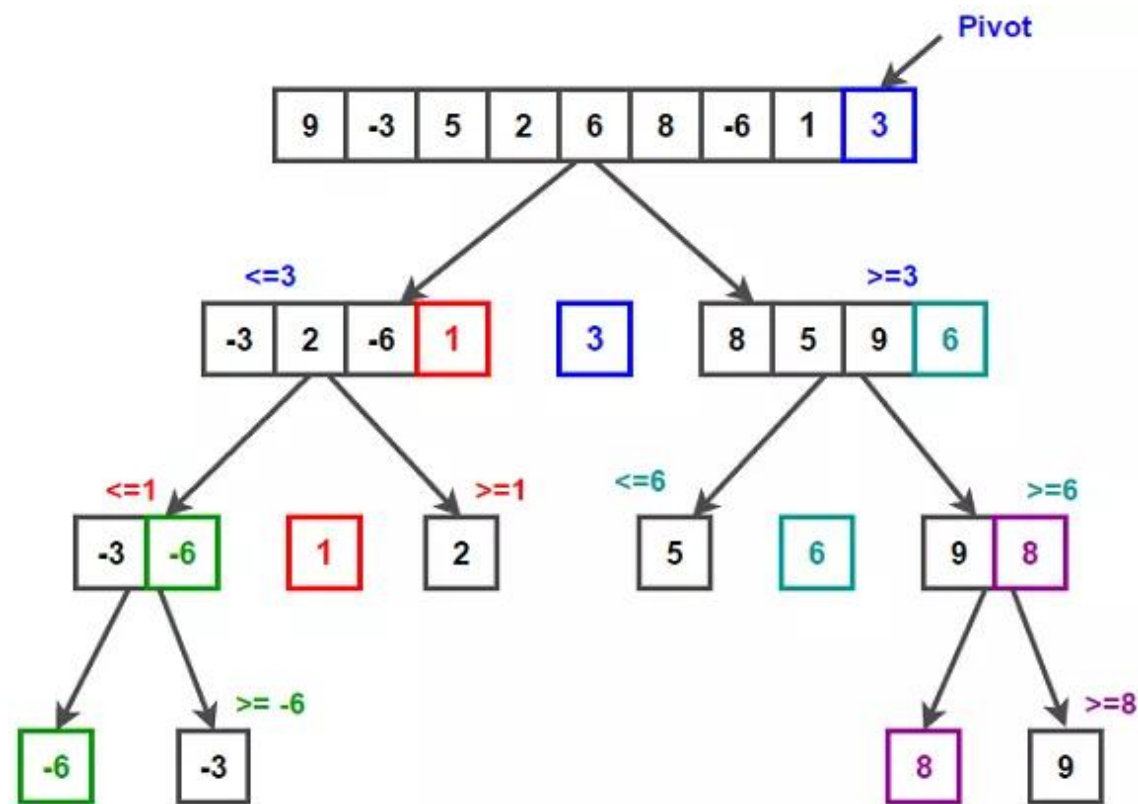
## $O(N)$ $O(N^2)$ $O(N\log N)$ 算法对比

	$O(n)$	$O(n\lg n)$	$O(n^2)$
10	10	23	100
100	100	460	$10^4$
1000	1000	6907	$10^6$
10000	10000	92103	$10^8$
100000	100000	1151292	$10^{10}$

# 快速排序

## 快速排序的循环不变式

	i	j
小于支点	大于支点	未确认



取最后一个数为支点，将比最后一个数小的放左边，比它大的放右边的过程

9	-3	5	2	6	8	-6	1	3
9	-3	5	2	6	8	-6	1	3
-3	9	5	2	6	8	-6	1	3
-3	9	5	2	6	8	-6	1	3
-3	2	5	9	6	8	-6	1	3
-3	2	5	9	6	8	-6	1	3
-3	2	5	9	6	8	-6	1	3
-3	2	-6	9	6	8	5	1	3
-3	2	-6	9	6	8	5	9	3
-3	2	-6	1	3	8	5	9	6

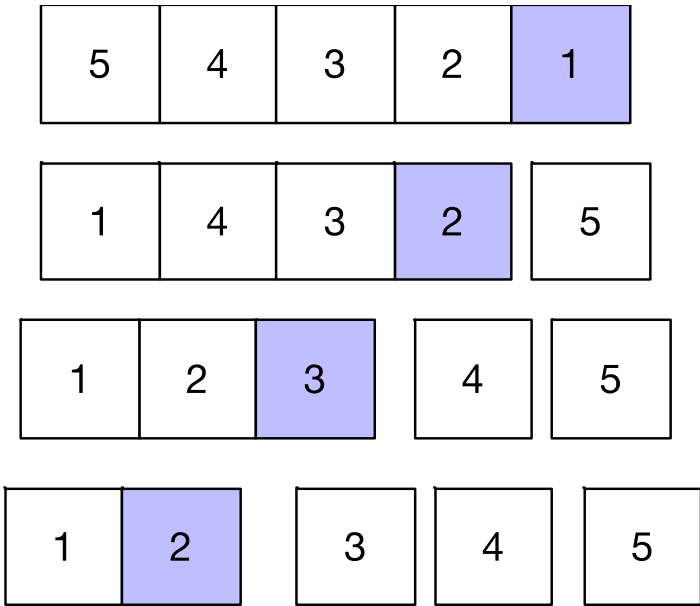
# 快速排序的正确性

归纳证明法：

1. 只有两个数的时候，选最后一个数是支点，最终结果比支点小的在左边，比支点大的在右边
2. 有N个数的时候，上述过程仍然成立(由循环不变式得出)

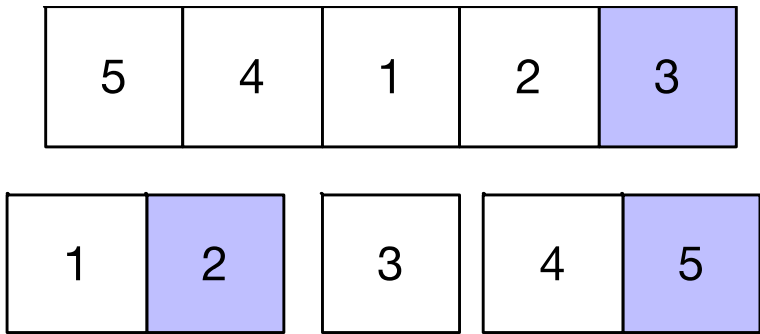
# 归并排序的复杂度分析(最坏情况 )

最坏情况：递归层级4



类似插入排序最差情况

最好情况：递归层级2



类似归并排序

# 数组和链表

最简单的数据结构

# 数组

内存地址

数据

00000000

00000000

00000001

00000001

00000002

00000002

00000003

00000003

00000004

00000004

00000005

00000005

...

...

FFFFFFFF

FFFFFFFF