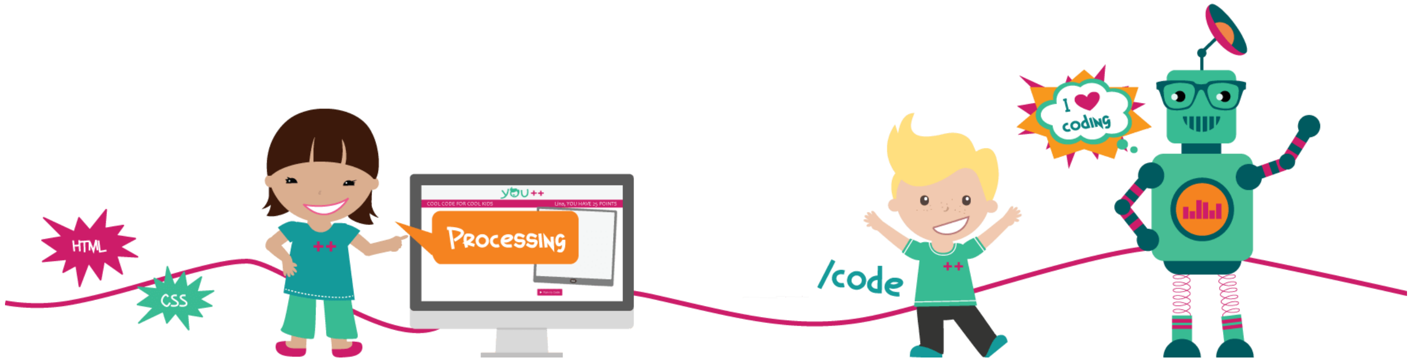
亲子算法课（4）——查找算法与时间复杂度

作者：叶蒙蒙

查找算法

顺序查找和二分查找

- 认识程序
  - 把控制流程和数据结构结合起来
  - 用编程语言表示算法
- 查找算法
  - 算法的衡量标准
  - 顺序查找
  - 二分查找



查找——找东西

找东西的目的

1. 找到“这个东西”
2. 确定是否真的存在“这个东西”
3. 发现“这个东西”的位置

在“一堆数字”中查找一个数字

- 为了发现
  - 确定这个数字到底是否存在于这堆数字中
  - 如果存在，它的位置是什么?

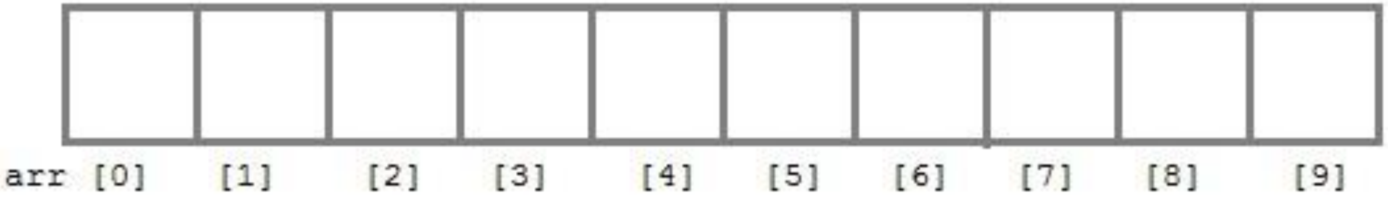
- 那“一堆数字”怎么组织起来?
  - 数据结构！！！

用数组来存储要查找的数字

- 数组是什么样的？
- 数组适合用来存储什么样的数字？ *（顺序结构）*
- 数组和链表相比，有什么优缺点？

- 数字的写法：
  - 变量名——数组的名字
  - 下标——数组中每一个“盒子”的标号

下图的这个数组的名字叫做 arr，它一共有十个单元（“盒子”）
第一个盒子的标号是0， 第一个盒子里的数字是：arr[0],
第二个盒子的标号是1， 第一个盒子里的数字是：arr[1],
……
第十个盒子的标号是9， 第一个盒子里的数字是：arr[9],



在一串数字里查找一个数字

- 如果这一串数字是混乱的（没有按大小排列过的），应该如何查找?
- 类比：在一堆随便揉在一起的衣服里找一件衣服——一件件拿起来

顺序查找

在数组中，从头到尾依次查看每一个“盒子”，确认“盒子”里有没有自己要找的数字

先找第一个盒子，如果里面的数字就是要找的，则结束，查找结果是：找到了；否则继续找第二个盒子……

如果所有盒子都找完了也没有找到，则查找结果是：没找到。

顺序查找算法

1. 打开一个盒子，看看里面的数字是不是要找的数字？如果是，就找到啦！

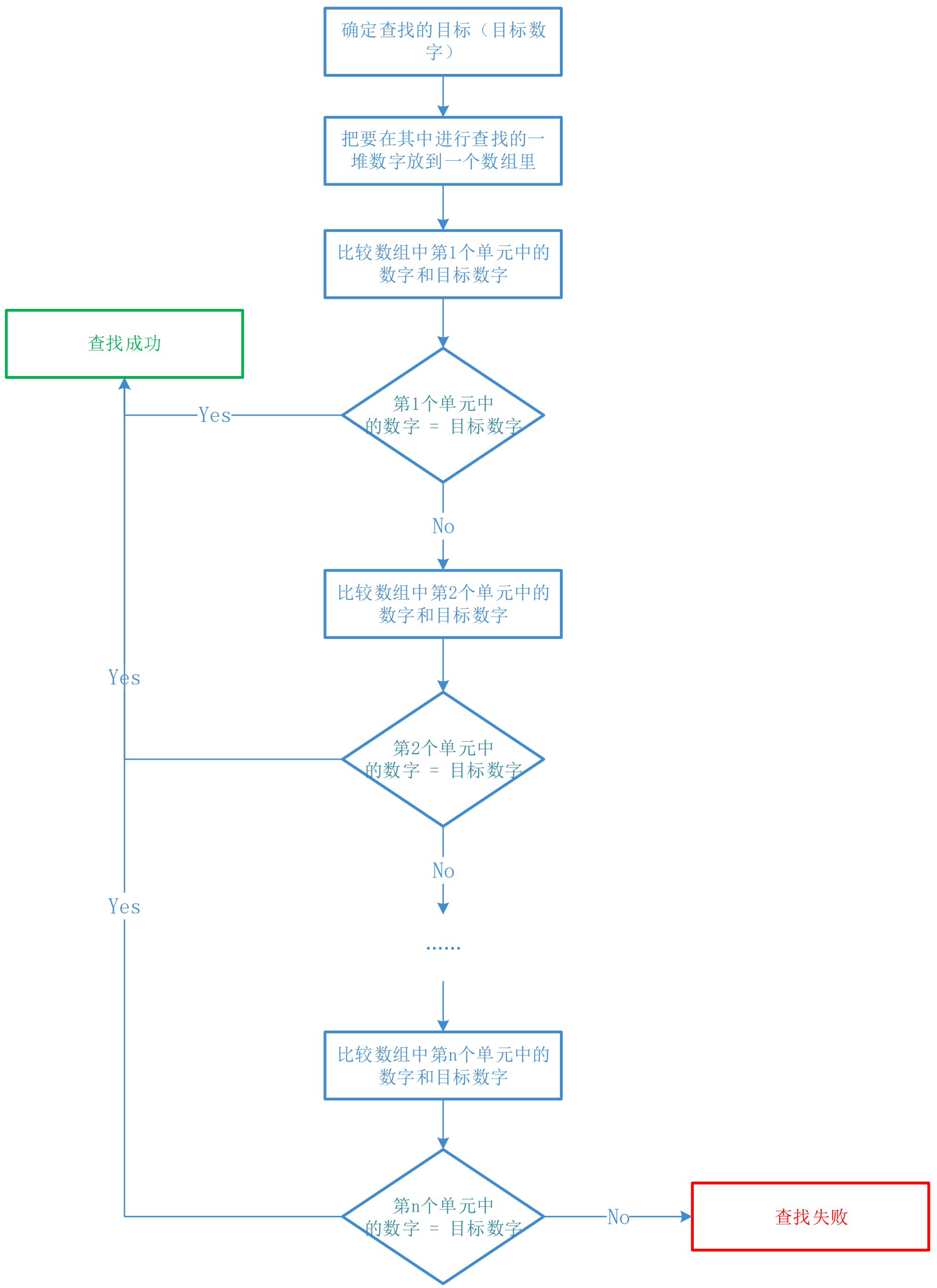
想想该用什么样的控制结构来做这一步？

2. 找完一个盒子，再找下一个盒子，再找下下个盒子……

想想用什么样的控制结构来做这一步？

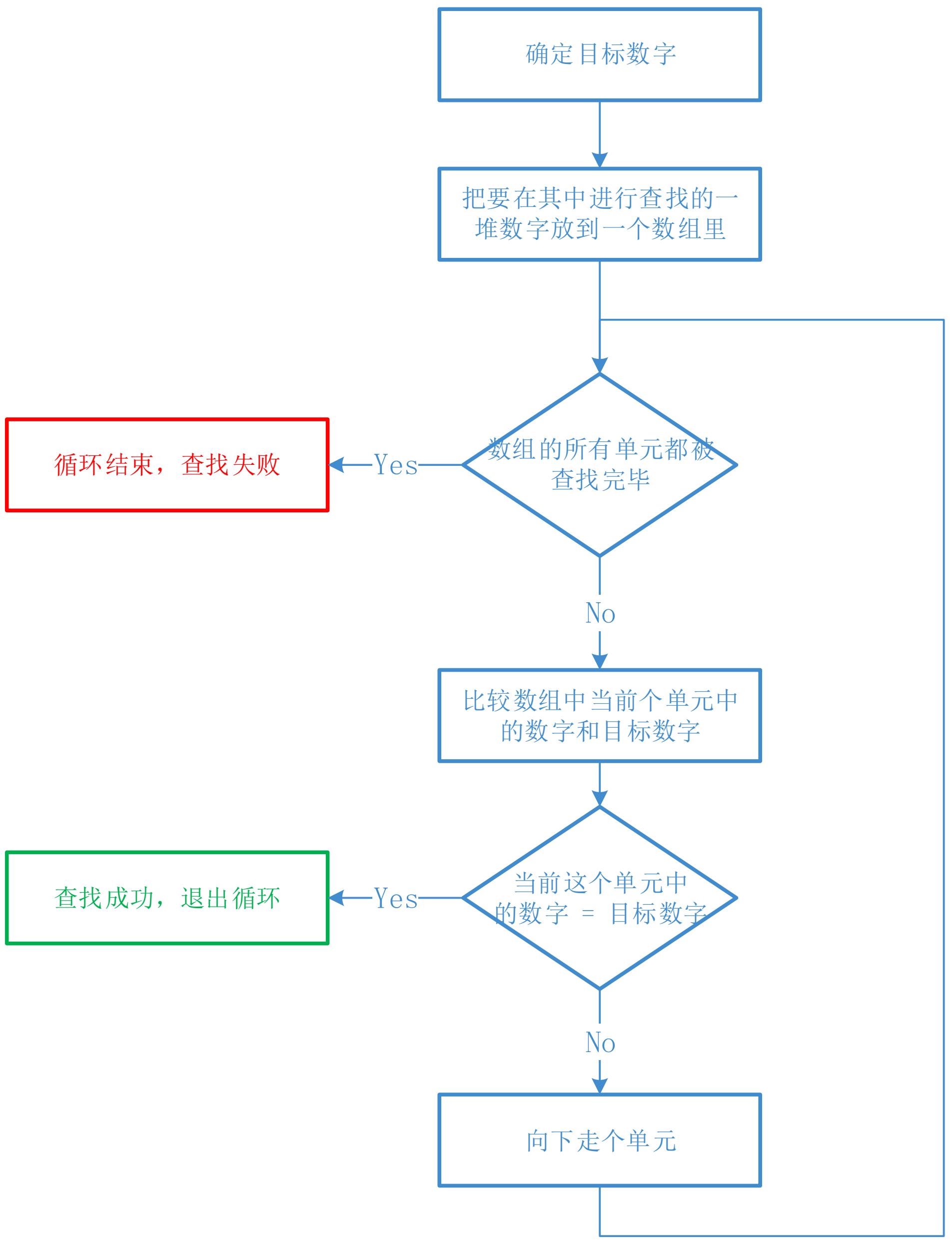
顺序查找算法流程图（1）

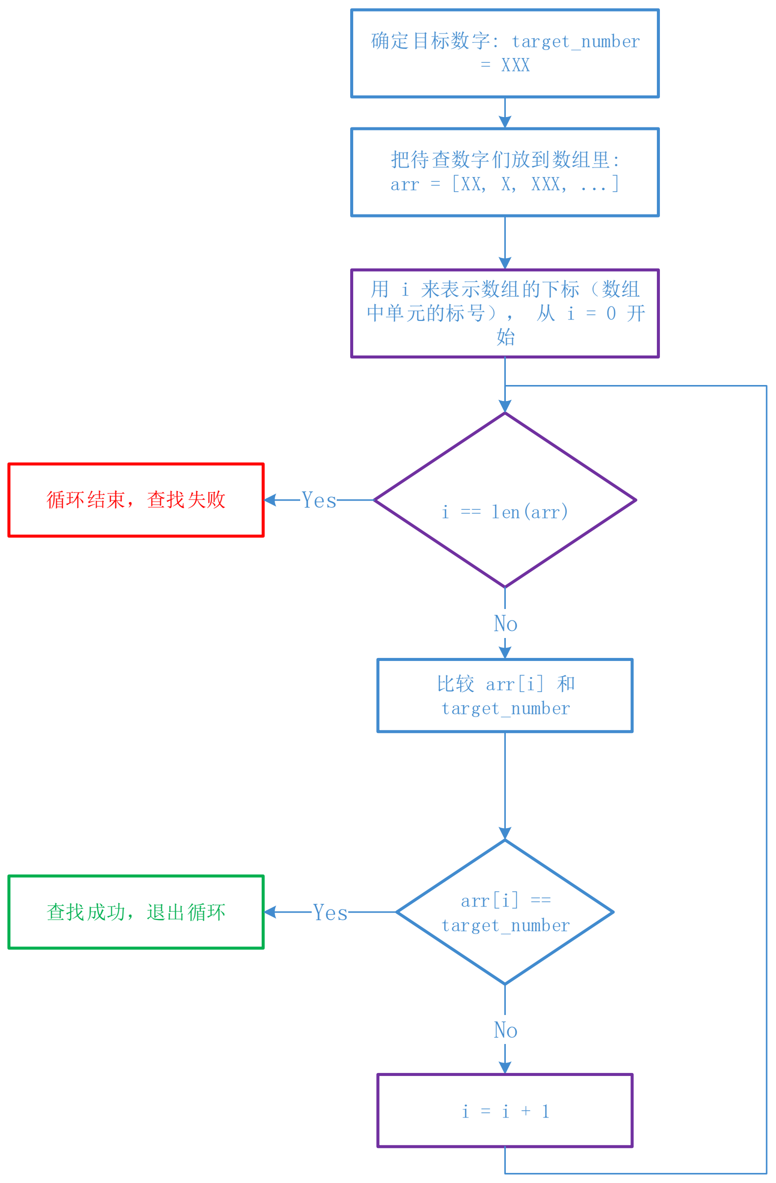
一步步的完成



顺序查找算法流程图（2）

引入循环





顺序查找算法流程图（3）

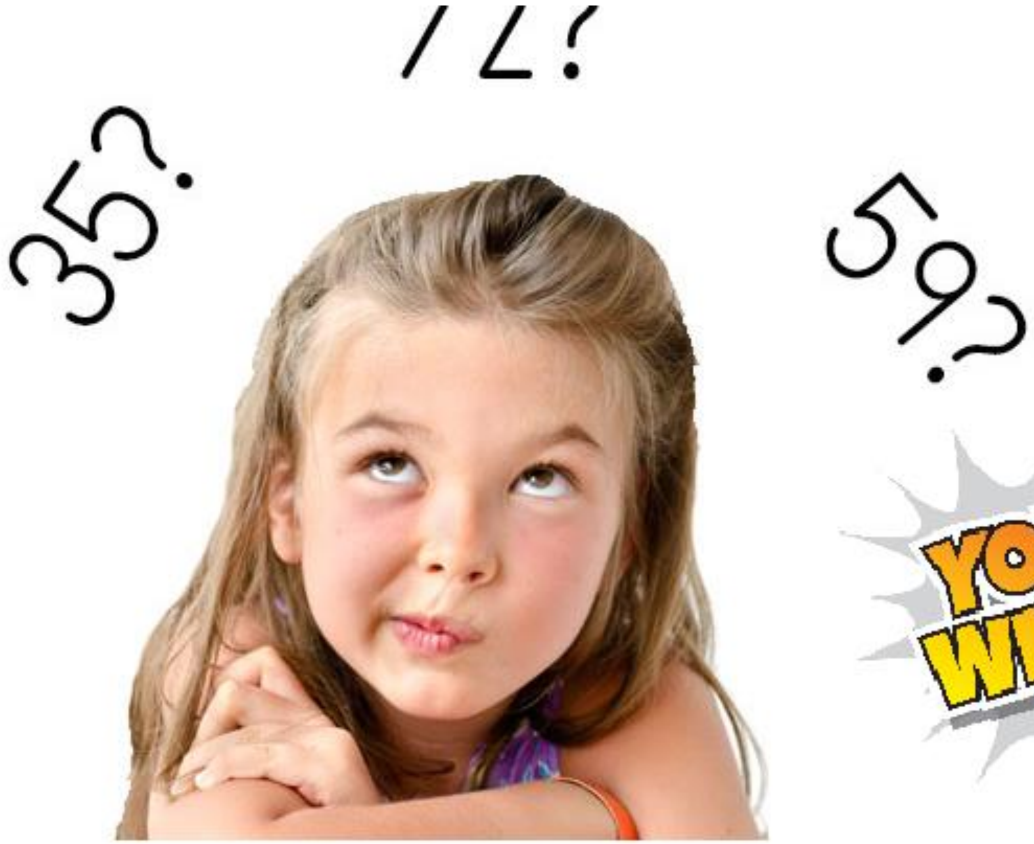
将数组的代码表示引入到流程图中

【小游戏】猜数字

- 游戏双方：甲方，乙方
- 游戏过程：
  - 甲方在1到1000之间任选一个自然数，作为神秘数（secret number）记住，然后开始游戏。
  - 每一轮：
    - 乙方猜一个数，问甲方：是这个数吗？
    - 甲方要根据事实，给出下面三个答案之一：i）比神秘数小；ii）比神秘数大；iii）就是这个数。
- 游戏结果：10轮（含）之内乙方猜中算乙方赢，否则是甲方赢。

猜数字游戏

有什么办法能让乙方每次都赢吗?



有!!!

在排好序的数组里进行查找

- 排好序的数组，其中的数字
  - 小->大 （升序）
  - 大->小 （降序）

- 在排好序的数组里还可以进行顺序查找吗？
- 在排好序的数组里必须进行顺序查找吗？

2	9	13	16	17	25	28	38	40	41
0	1	2	3	4	5	6	7	8	9

1	2	2	3	3	4	5	6	6	6
0	1	2	3	4	5	6	7	8	9

1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9

“跳着找”的查找算法

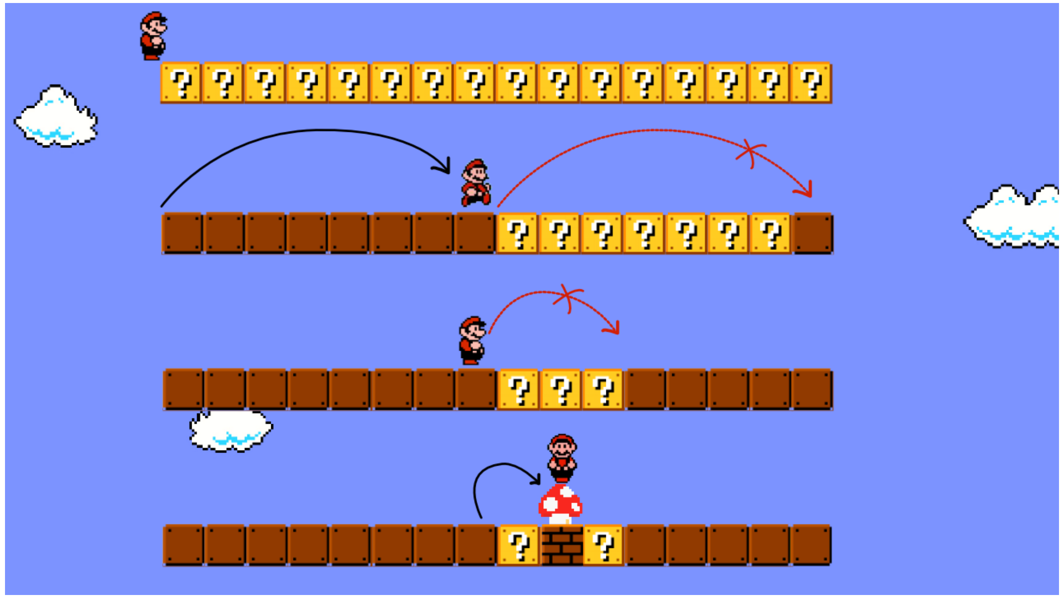
- 在排好序的数字序列中进行
  - 数字序列——数组、链表
  - 我们还是选用数组作为数据结构

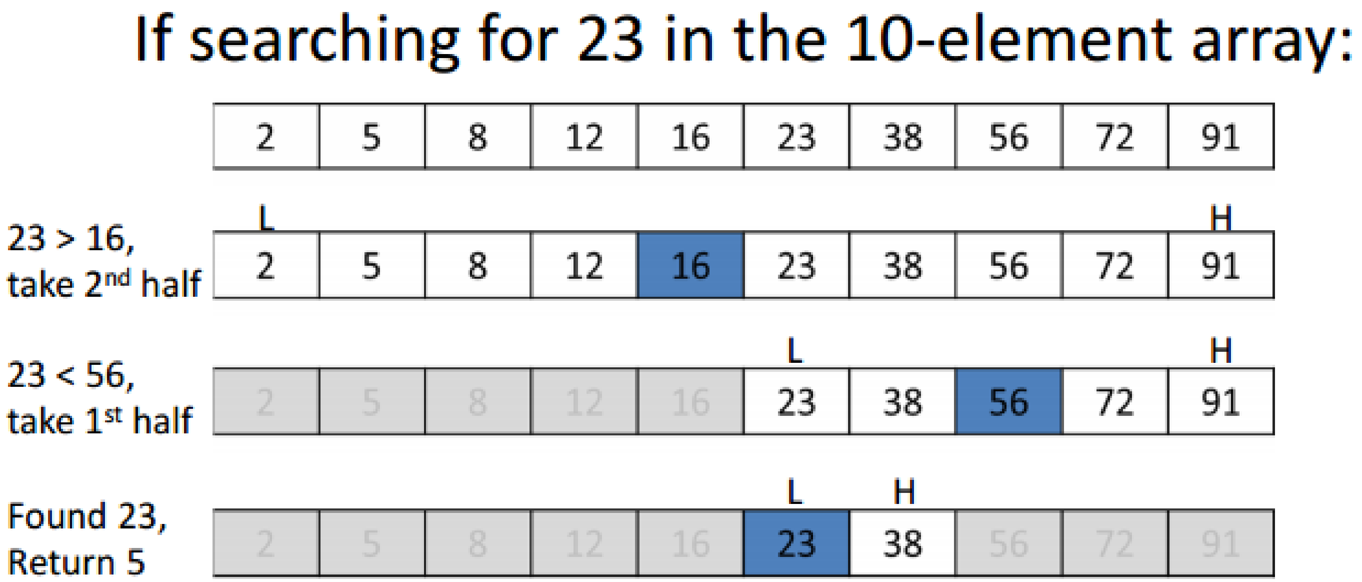
- 不再一个接一个的顺序访问了，而是要“跳着”访问

- *怎么“跳”？*

二分查找算法 (Binary Search)

- 循环算法
- 每次循环
  - 将剩下数字平分成两份（两“串”数字）—— 二分
  - 排除掉其中的一份
- 结束循环
  - 中间正好找到等于神秘数的数字
  - 或者最后一次循环，一份（一“串”）只包含一个数字



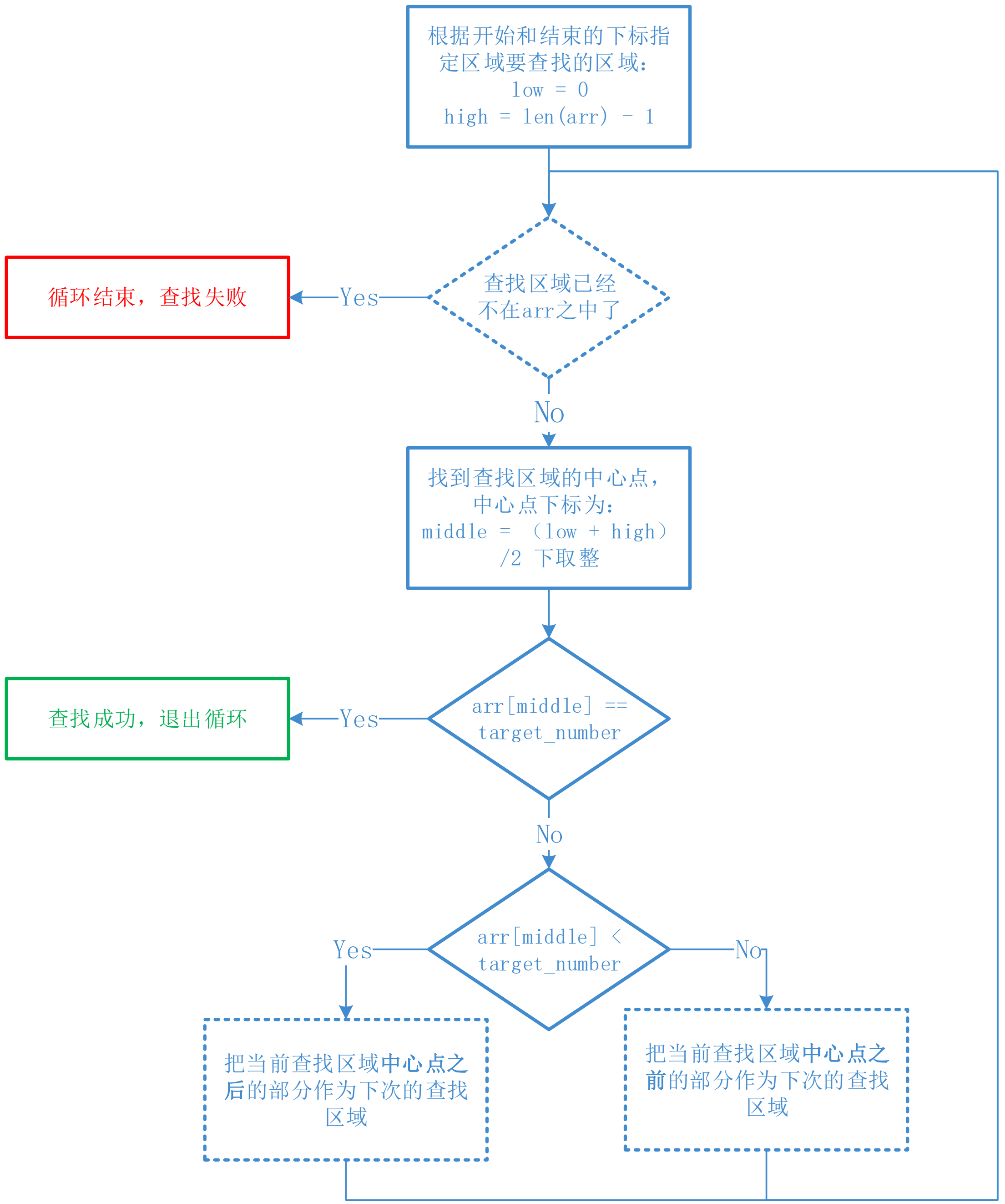


一个例子

- 从含有10个单元的数组里找到23这个数字
- 查找目标：23
- 被查找数字：10个，升序排列

思考：如果要在这个数组中查找1，会怎么样？

二分查找算法

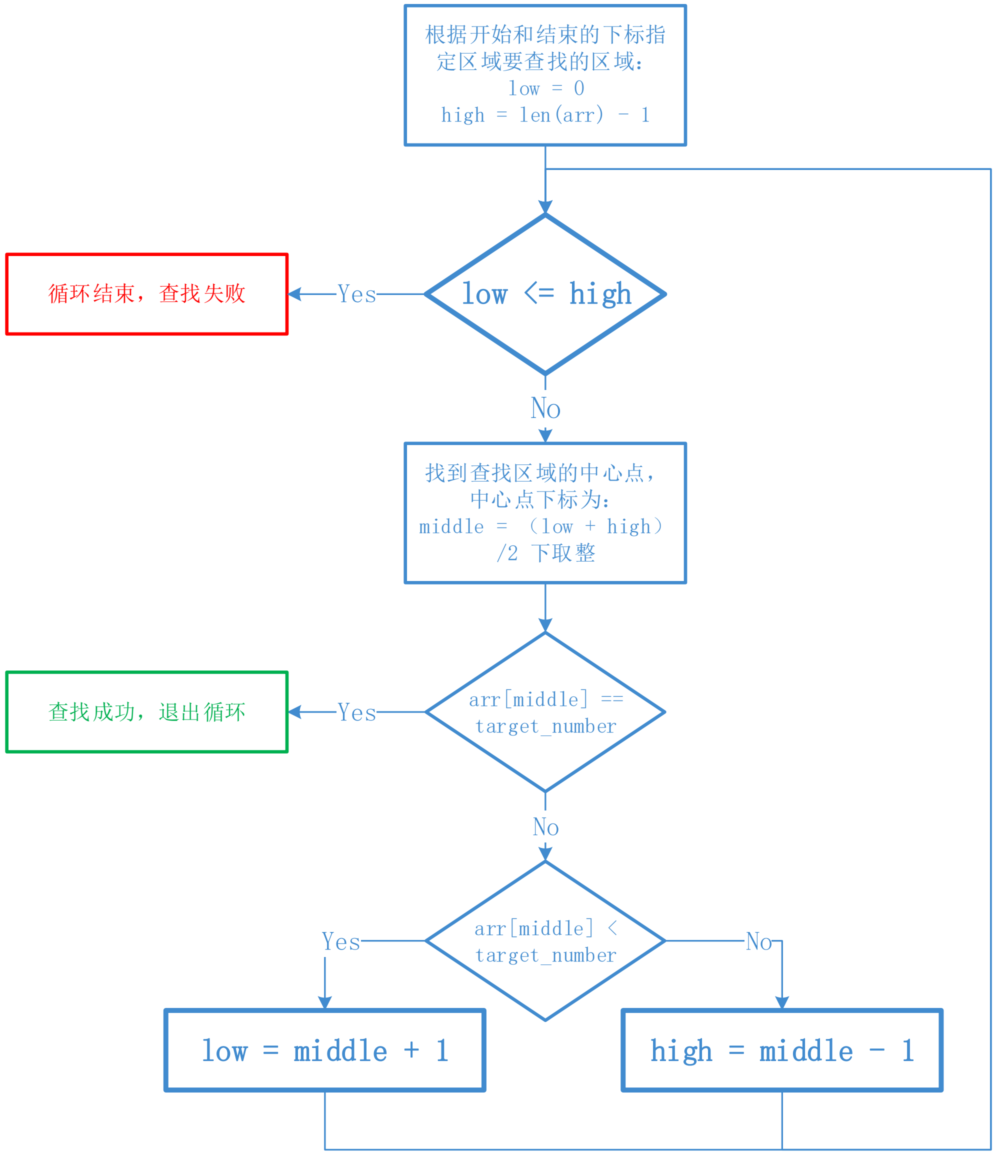


根据开始和结束的下标指定区域要查找的区域：
low = 0
high = len(arr) - 1
查找区域已经不在arr之中了
Yes
循环结束，查找失败
No
找到查找区域的中心点，中心点下标为：
middle = （low + high）/2 下取整
arr[middle] == target_number
Yes
查找成功，退出循环
No
arr[middle] < target_number
Yes
把当前查找区域**中心点之**后的部分作为下次的查找区域
No
把当前查找区域**中心点之**前的部分作为下次的查找区域

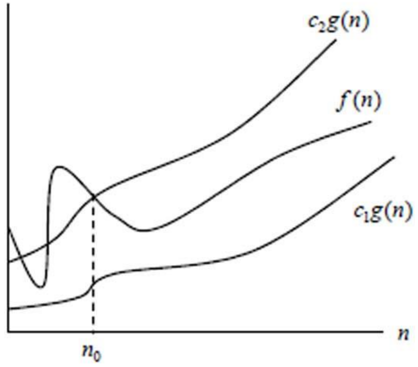
二分查找算法

思考:

- 为什么用low <= high就能判断查找区域是不是还在arr中?
- 如果最下面两个框改为low = middle 和high = middle, 会怎么样?



The O Notation



$O(g(n)) = \{f(n)$: there exist positive constants c_1, c_2, and n_0 such that $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ for all $n \ge n_0\}$.

$g(n)$ is an asymptotically tight bound for $f(n)$.

Example:

$\frac{n^2}{} - 2n = \Theta(n^2)$, with $c_1 = \frac{1}{}$, $c_2 = \frac{1}{}$, and $n_0 = 8$.

时间复杂度

- 时间复杂度 -> 操作的次数
- 查找算法的操作：访问数组单元（访问的同时就进行了比较）
- 如果数组中有n个数字，顺序查找算法的平均访问次数 = （1 + 2 + 3 + … + n) / n

  当 n == 10时，平均访问次数 = 5.5

  当 n == 100时，平均访问次数 = 55

- 顺序查找算法的时间复杂度：O(n)

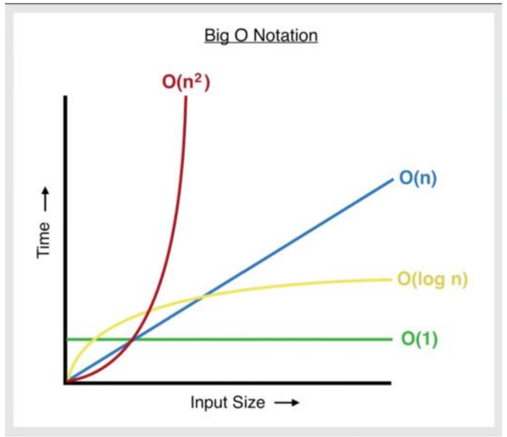
算法好坏的衡量标准

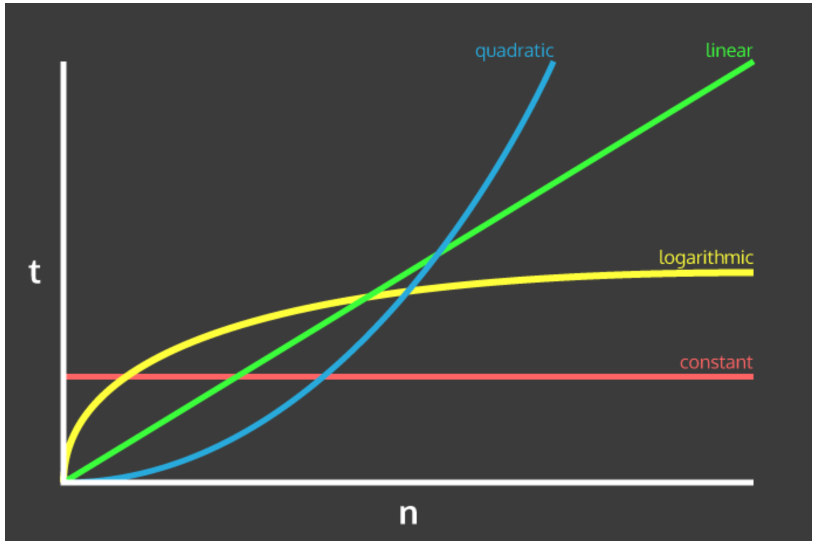
- 功能
- 性能
    - 时间复杂度 —— 计算时间
    - 空间复杂度 —— 存储大小

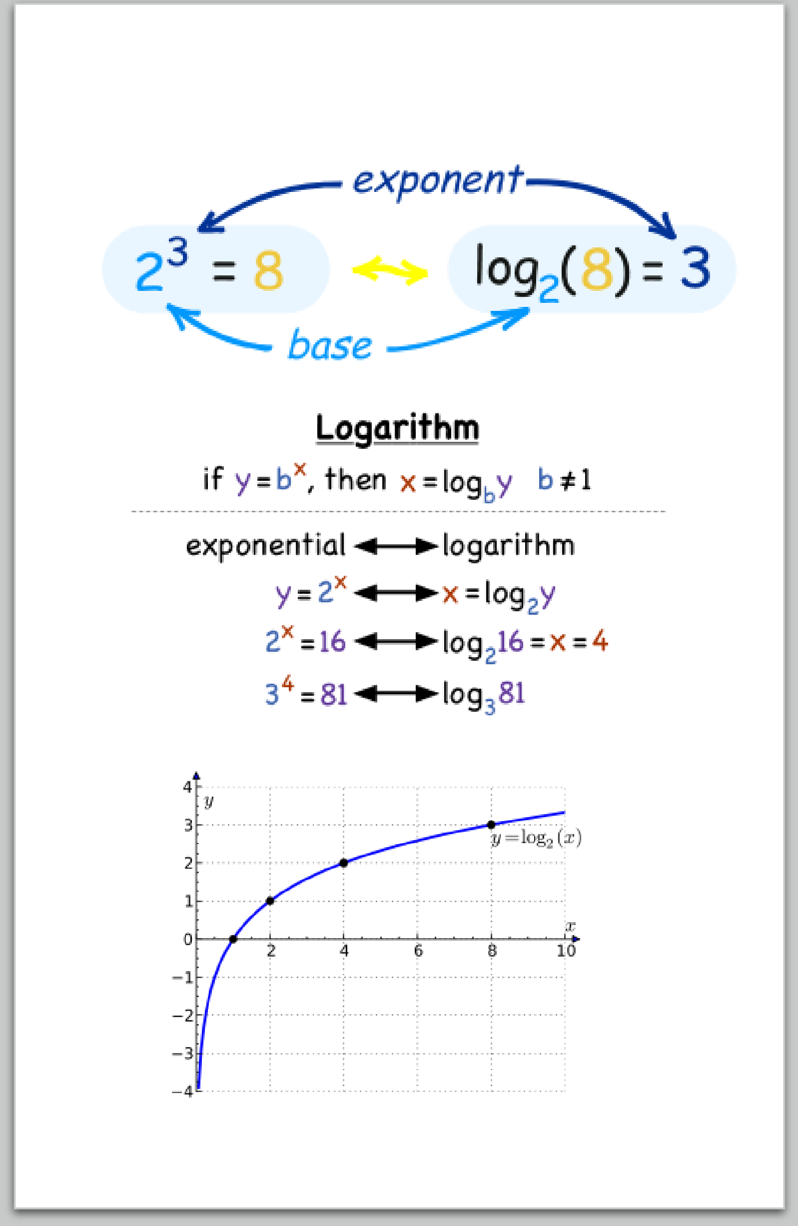
一般情况下，我们最关心：一个算法的时间复杂度最大是多少

- Big O Notation

Notation	Common Name
O(1)	Constant time
O(log n)	Logarithmic time
O(n)	Linear time
O(n log n)	Log-linear time
O(n^2)	Quadratic
O(n^c)	Polynomial
O(c^n)	Exponential
O(n!)	Factorial
O(n^n)	N to the N



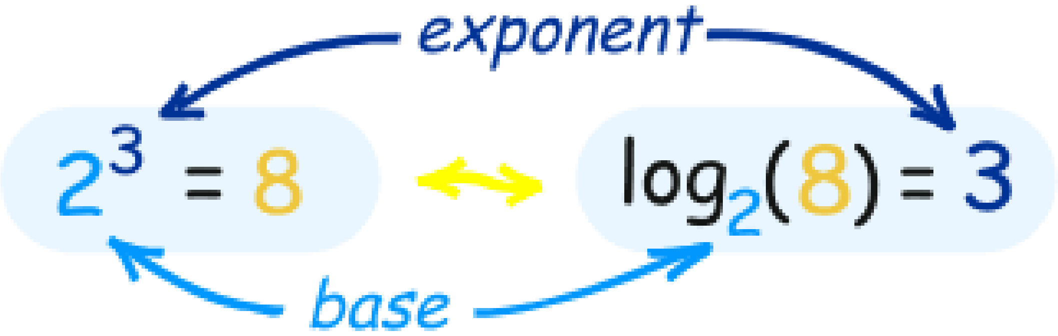


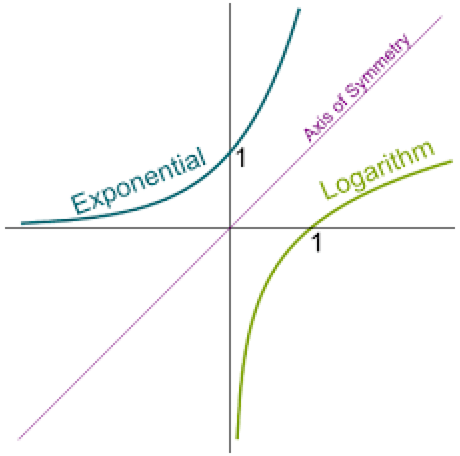


二分查找的时间复杂度

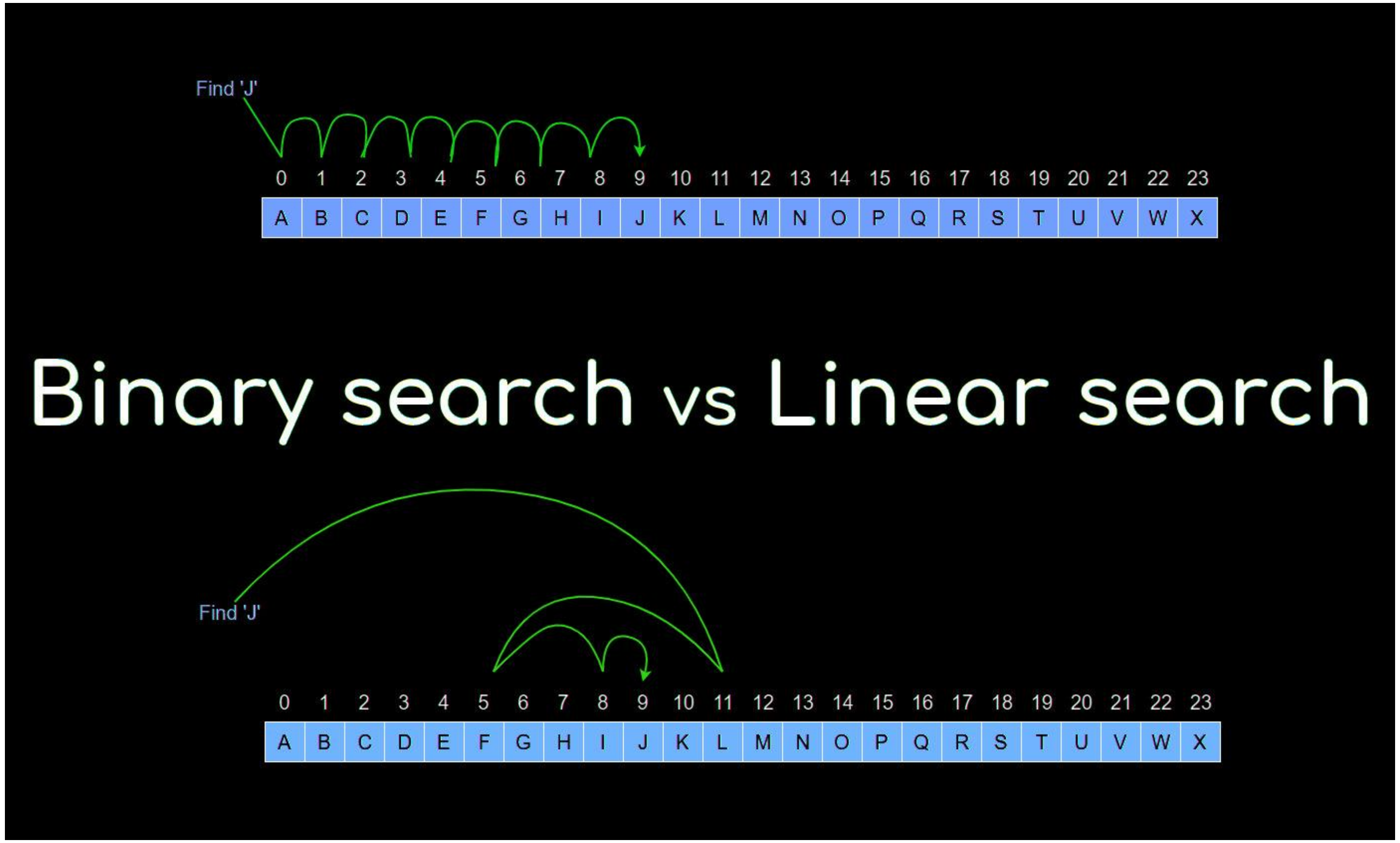
- 想想需要多少次访问？
  - 每次排除掉一般，也就是说每次循环后需要查找数字个数的就变成了上次的1/2——每次循环：被查找数字个数 / 2
  - 在总共n个数字中查找：(... (((n / 2) / 2) / 2)...) /2 —— 对数

- 最佳时间复杂度：O(1)
- 最差时间复杂度：O(log(n))
- **平均时间复杂度**：O(log(n))

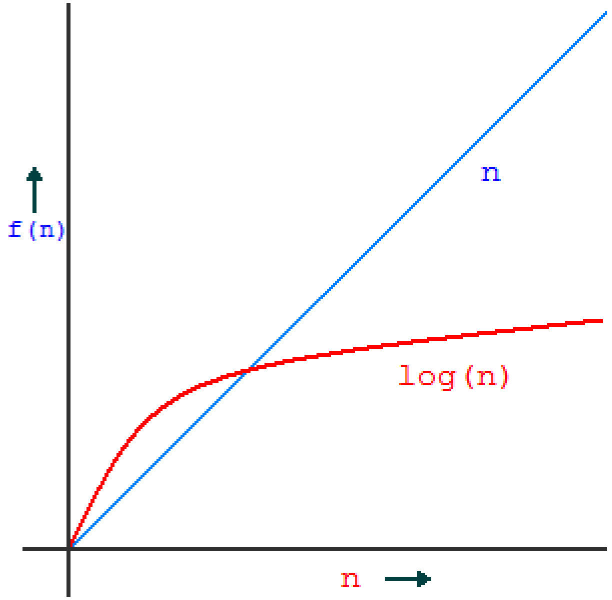




指数（exponent）和对数（logarithm）



对数时间复杂度 vs 线性时间复杂度



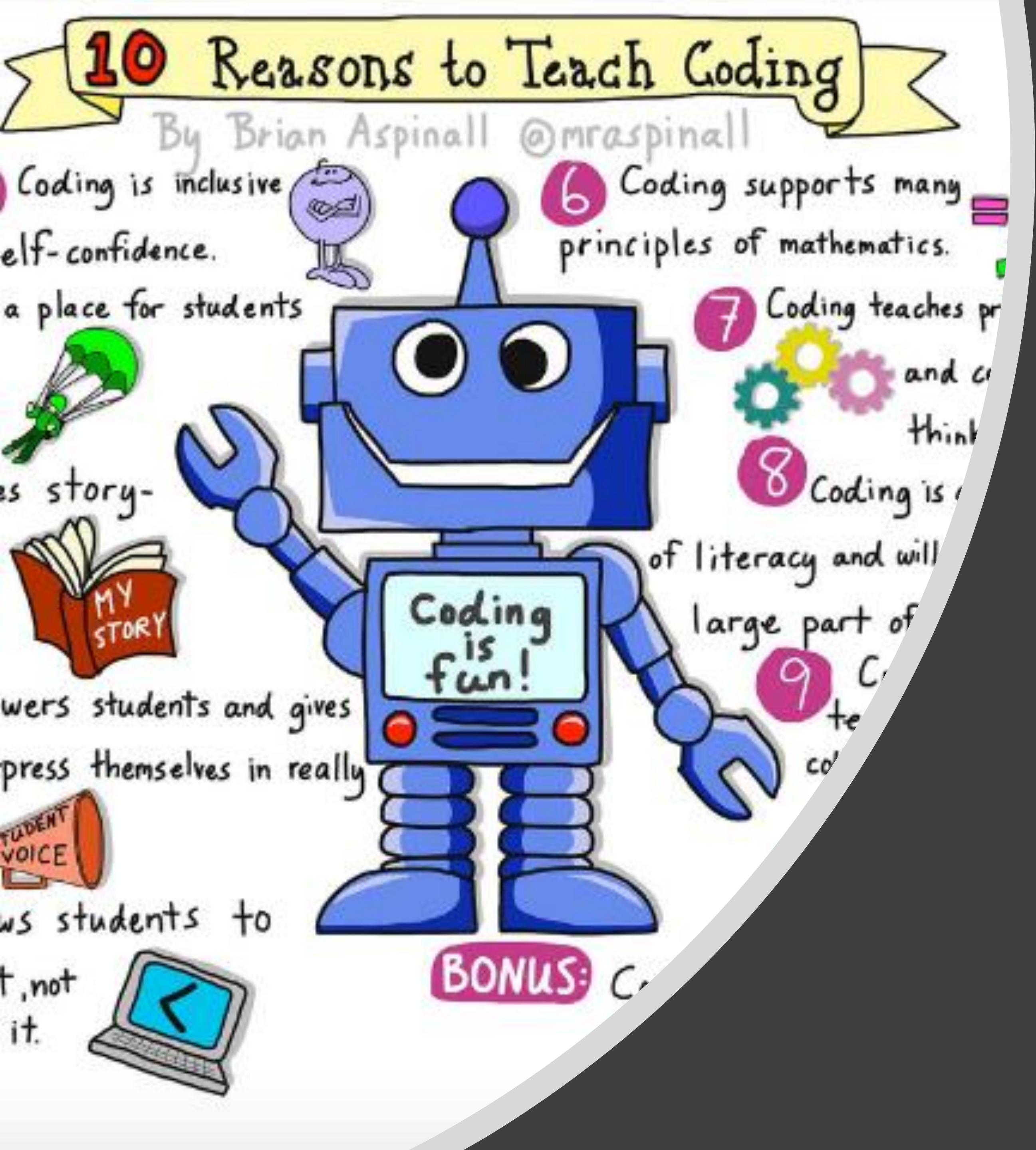
思考题

很多实际问题可以用查找算法来解决

【思考题】写一个算法，用来计算距离一个整数的平方根最近的整数

- 平方根，如果 s x s = a，那么s 就是a的平方根
- 大多数整数的平方根都不是整数
- 我们的算法要求：给定一个数a，如果它的平方根s是整数，这个整数就是所求结果；如果s不是整数，那么设s上取整为s_u, 下取整为s_d，如果 | s_d x s_d – a| < |s_u * s_u – a| 则s_d为所求结果，否则s_u为所求结果

这道题应该如何求解？最小时间复杂度可以是多少？



10 Reasons to Teach Coding
By Brian Aspinall @mraspinall
Coding is inclusive
elf-confidence.
a place for students
es story-
MY STORY
wers students and gives
press themselves in really
STUDENT VOICE
ws students to
t, not
it.
Coding is fun!
6 Coding supports many
principles of mathematics.
7 Coding teaches pr
and c
think
8 Coding is
of literacy and will
large part of
9
BONUS:

谢谢！