

数据结构

第 2 周 – 算法的效率，顺序查找与二分查找

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

复习

- **算法**：有限的、定义明确的步骤序列
- **数据类型**：值的集合 + 运算 + 表示
- **数据结构**：元素 + 关系 + 操作
- 访问方式：**直接**（数组）· **顺序**（链表）

今天的问题

- 怎样判断一个算法是**快**还是**慢**？
- 怎样在数组中找到一个元素 – 能不能找得**更快**？



算法的效率

为什么只用秒表不够？

运行时间取决于

- 机器（处理器、内存），
- 编程语言和编译器，
- 同时运行的其他程序，
- **输入的规模和内容。**

因此我们不以秒为单位来衡量，而是考察**步数**如何随**输入规模（n）**增长。

步（基本操作）：赋值、比较、算术运算、访问数组元素 – 各算作一步。

计算步数：一个例子

伪代码 – 求和

```
Function Sum(A, n)
    s ← 0                // 1
    For i ← 1 to n        // n 次
        s ← s + A[i]      // n
    End for
    Return s              // 1
End function
```

- 赋值和返回： **常数步**
- 循环体执行 **n** 次
- 总共约 **$c_1 \cdot n + c_2$** 步

常数 (c_1 、 c_2) 取决于机器 – 重要的是步数随 n **线性**增长。

最好、最坏和平均情况

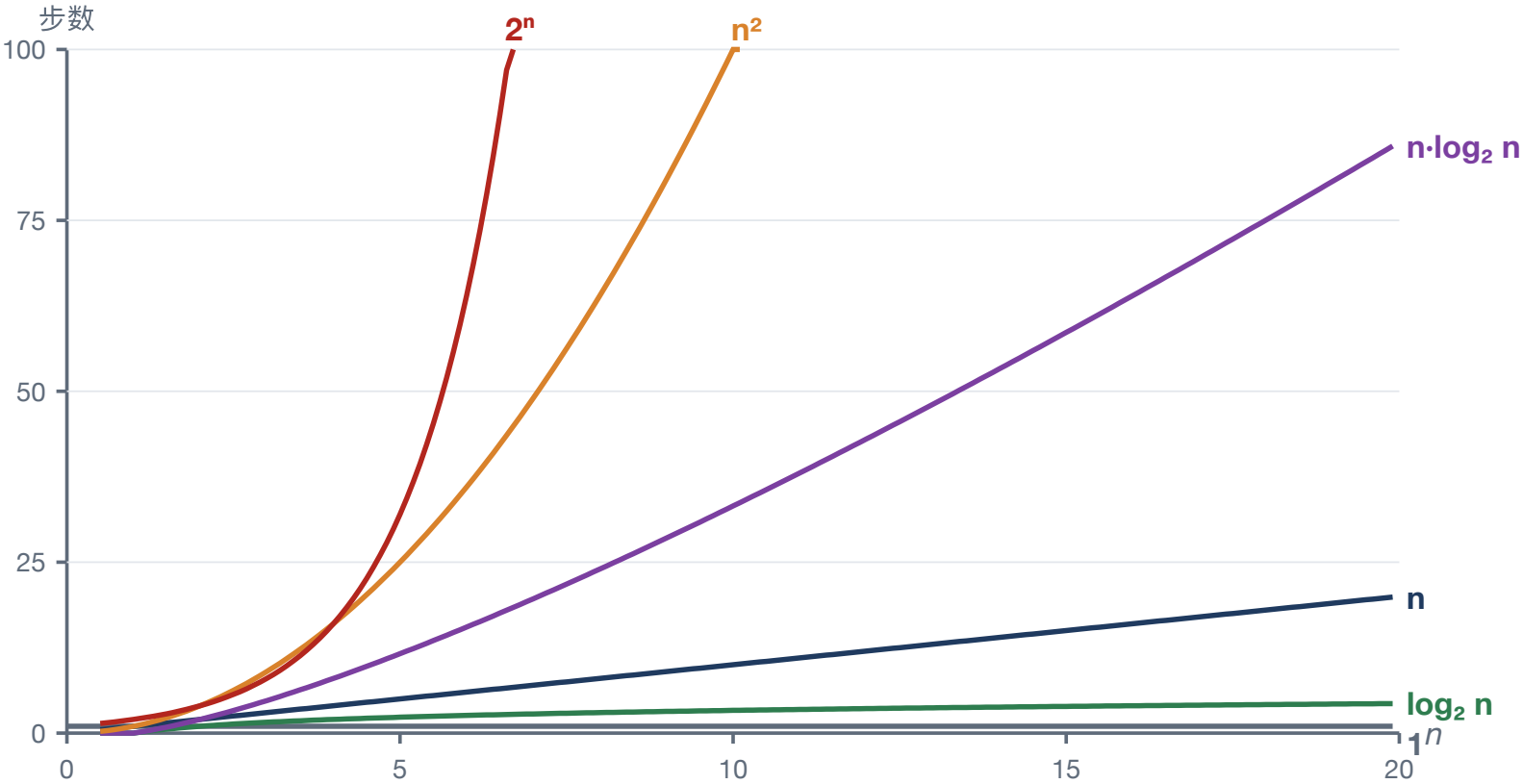
即使输入规模相同，步数也可能不同！

例：在 n 个元素的数组中从前往后查找值 x 。

情况	何时？	步数
最好	x 是第一个元素	1
最坏	x 是最后一个，或者不在数组中	n
平均	x 出现在各位置的可能性相同	约 $n / 2$

通常我们考察**最坏情况**：它是一种保证 – 肯定不会比这更慢。

增长速度



实际需要多长时间？

假设机器每秒执行 10^9 步。

n	$\log_2 n$	n	$n \cdot \log_2 n$	n^2	2^n
10	3 ns	10 ns	33 ns	100 ns	1 μ s
1 000	10 ns	1 μ s	10 μ s	1 ms	∞ ($\approx 10^{284}$ 年)
10^6	20 ns	1 ms	20 ms	17 分钟	∞
10^9	30 ns	1 s	30 s	32 年	∞

更快的机器几乎没有帮助：**快 10 倍**的机器只是把 32 年变成 3.2 年。出路在于**更好的算法**。

大O记号

直观地说：如果 **f** 的增长至多和 **g** 一样快（忽略常数因子），就写作 $f(n) = O(g(n))$ 。

精确地说： $f(n) = O(g(n))$ ，如果存在常数 **c** > 0 和 **n₀**，使得

对所有 $n \geq n_0$ ，有 $f(n) \leq c \cdot g(n)$ 。

规则：

- 常数因子可以省略： $5n = O(n)$
- 只有最高阶项起作用： $3n^2 + 10n + 7 = O(n^2)$
- 对数的底无关紧要： $\log_2 n = O(\log n)$

相关记号：Ω（下界）、Θ（精确阶）。

常见的增长阶

记号	名称	例子
$O(1)$	常数阶	按下标访问数组元素、栈的 <code>push</code>
$O(\log n)$	对数阶	二分查找
$O(n)$	线性阶	顺序查找、求和、求最大值
$O(n \log n)$	“ $n \log n$ ”阶	归并排序、快速排序（平均）
$O(n^2)$	平方阶	冒泡排序、考察所有元素对
$O(2^n)$	指数阶	考察所有子集

经验法则：当 $n = 10^6$ 时， $O(n \log n)$ 完全可以接受， $O(n^2)$ 则不行。

从代码确定阶

```
// O(n)
for (int i = 0; i < n; i++)
    sum += a[i];

// O(n2)
for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
        count++;
```

```
// O(log n) - 每次减半
while (n > 1)
    n = n / 2;

// O(n) + O(n2) = O(n2)
for (int i = 0; i < n; i++) ...
for (int i = 0; i < n; i++)
    for (int j = i; j < n; j++) ...
```

- **顺序执行**：代价相加 → 取较大者
- **嵌套**：代价相乘
- **减半**：如果每一步都把问题规模减半 → $\log n$

空间复杂度

不仅时间，**内存**也是一种资源。

- **空间复杂度**：除输入之外所需的内存，表示为 n 的函数
- 顺序查找和二分查找： **$O(1)$** – 只需几个辅助变量
- 复制一个数组： **$O(n)$**

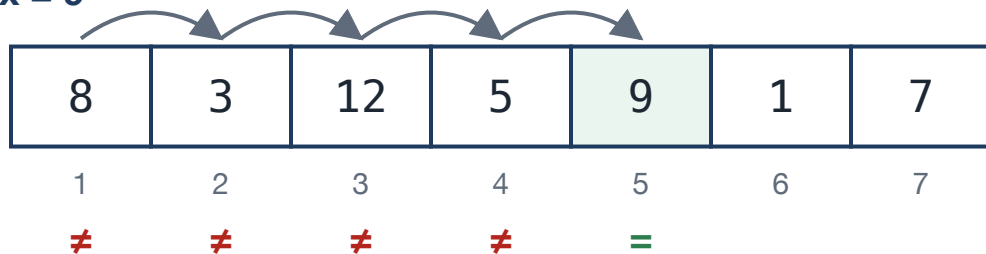
我们常常需要在**时间和空间之间权衡**：多用一些内存（例如哈希表，第 11 周），查找就可以更快。



在数组中查找

顺序（线性）查找

查找 $x = 9$



5 次比较

最坏情况: n

- 从前往后，把每个元素与要找的值比较
- 数组**不需要**有序
- 在链表上同样适用（顺序访问）

顺序查找 – 算法

伪代码

```
Function LinearSearch(A, n, x)
  i ← 1
  While i ≤ n and A[i] ≠ x
    i ← i + 1
  End while
  If i ≤ n then
    Return i
  else
    Return 0      // 未找到
  End if
End function
```

步数：最坏情况 $O(n)$

Java

```
static int linearSearch(int[] a, int x) {
    int i = 0;
    while (i < a.length && a[i] != x)
        i++;
    return i < a.length ? i : -1;
}
```

Go

```
func linearSearch(a []int, x int) int {
    for i, v := range a {
        if v == x {
            return i
        }
    }
    return -1
}
```

注意：条件的顺序！

```
While  $i \leq n$  and  $A[i] \neq x$ 
```

- 当 $i = n + 1$ 时， $A[i]$ 是一个**不存在**的元素
- 得益于**短路求值**，这仍然是正确的：如果 $i \leq n$ 为假，就不会计算 $A[i] \neq x$

如果顺序反过来（ $A[i] \neq x$ and $i \leq n$ ），Java 会抛出 `ArrayIndexOutOfBoundsException`，Go 会报 `index out of range` 错误。

哨兵查找

思路：把要找的值放在数组末尾（第 $n + 1$ 个位置） – 这样一定能找到，也就不必检查下标边界。

伪代码

```
Function SentinelSearch(A, n, x)
  A[n + 1] ← x           // 哨兵
  i ← 1
  While A[i] ≠ x
    i ← i + 1
  End while
  If i ≤ n then Return i
  else Return 0
End function
```

- 每一步**一次**比较，而不是两次
- 仍然是 $O(n)$ ，但比较次数约减少一半
- 数组末尾需要一个**额外的位置**

方法	x = 4 不存在, n = 7
普通	15 次比较
哨兵	8 次比较

第 10 周的带哨兵链表会再次用到哨兵的思想。

在有序数组中顺序查找

如果数组**有序**，一旦遇到**大于** x 的元素就可以停止。

```
Function SortedLinear(A, n, x)
  i ← 1
  While i ≤ n and A[i] < x
    i ← i + 1
  End while
  If i ≤ n and A[i] = x then Return i
  else Return 0
End function
```

- 查找失败时平均更快
- 但最坏情况仍然是 $O(n)$ – 能不能更好?

二分查找 – 思路

前提：数组有序。

1. 看**中间**的元素
2. 如果正是要找的：完成
3. 如果要找的值**更小**：在左半部分继续
4. 如果要找的值**更大**：在右半部分继续

每一步查找范围都**减半**。

就像在字典里查一个词（或在电话簿里查一个名字）：我们不会一页一页地翻，而是从中间翻开，再决定往哪边走。

二分查找 – 示例

查找 $x = 23$

第 1 步

1	2	3	4	5	6	7	8	9	10
2	5	8	12	16	23	38	56	72	91
lo				mid		hi			

$A[5] = 16 < 23 \rightarrow lo \leftarrow 6$

第 2 步

2	5	8	12	16	23	38	56	72	91
lo					mid		hi		

$A[8] = 56 > 23 \rightarrow hi \leftarrow 7$

第 3 步

2	5	8	12	16	23	38	56	72	91
lo=mid					hi				

$A[6] = 23 = x \checkmark$

二分查找 – 算法

伪代码

```
Function BinarySearch(A, n, x)
  lo ← 1
  hi ← n
  While lo ≤ hi
    k ← (lo + hi) div 2
    If A[k] = x then
      Return k
    else if A[k] < x then
      lo ← k + 1
    else
      hi ← k - 1
    End if
  End while
  Return 0
End function
```

Java

```
static int binarySearch(int[] a, int x) {
    int lo = 0, hi = a.length - 1;
    while (lo <= hi) {
        int mid = lo + (hi - lo) / 2;
        if (a[mid] == x) return mid;
        if (a[mid] < x) lo = mid + 1;
        else hi = mid - 1;
    }
    return -1;
}
```

Go

```
for lo <= hi {
    mid := lo + (hi-lo)/2
    switch {
    case a[mid] == x: return mid
    case a[mid] < x:  lo = mid + 1
    default:         hi = mid - 1
    }
}
```

二分查找 – 效率

每一步之后，范围的大小至多为原来的一半： $n \rightarrow n/2 \rightarrow n/4 \rightarrow \dots \rightarrow 1$

至多 $\lfloor \log_2 n \rfloor + 1$ 步 $\rightarrow O(\log n)$

n	顺序查找（最坏）	二分查找（最坏）
10	10	4
1 000	1 000	10
1 000 000	1 000 000	20
1 000 000 000	1 000 000 000	30

在全中国人口（14 亿）的有序名单中找一个名字：至多 31 步。

二分查找 – 常见错误

- 数组**无序** → 结果错误（而且没有错误提示！）
- 用 `lo < hi` 代替 `lo ≤ hi` → 漏掉只有一个元素的范围
- 用 `lo ← k` 代替 `lo ← k + 1` → 可能**死循环**
- `(lo + hi) / 2` → 下标很大时**溢出**
应改为: `lo + (hi - lo) / 2`

Java 类库的 `Arrays.binarySearch` 函数中这个错误存在了很多年（2006 年才修复）。

内置版本: Java `Arrays.binarySearch(a, x)` · Go `sort.SearchInts(a, x)`

顺序查找还是二分查找？

	顺序	二分
前提	无	有序数组
访问	顺序访问即可	需要 直接 访问
步数	$O(n)$	$O(\log n)$
在链表上	✓	✗（无法直接访问）

如果要在同一组数据上多次查找，值得**先排序一次**（ $O(n \log n)$ ，第 13 周）；之后每次查找都是 $O(\log n)$ 。

小结

- 效率用**步数随输入规模的增长**来衡量
- 通常考察**最坏情况**
- **大O**：省略常数和低阶项
- 阶的大小： $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$
- **顺序查找**： $O(n)$ ，适用于任何数组 · 用哨兵可减少比较次数
- **二分查找**： $O(\log n)$ ，只适用于可直接访问的**有序**数组

下周：数组 – 一维和 multidimensional 数组、存储方式、地址计算、动态数组。

复习题

1. $3n^2 + 100n + 5$ 的阶（大O）是什么？ $n + \log n$ 呢？
2. 判断对错：二分查找是 $O(n)$ 的。
3. 判断对错：二分查找在链表上也是 $O(\log n)$ 步。
4. 在 500 个元素的数组中，二分查找至多需要多少步？
5. 在无序数组上进行二分查找会怎样？
6. 下面代码的阶是什么？

```
for (i = 1; i < n; i = i * 2) for (j = 0; j < n; j++) ...
```



实践课

示例程序

程序	Java	Go	演示内容
顺序查找	<code>LinearSearch.java</code>	<code>w02/linearsearch</code>	普通、哨兵和有序三种写法；比较次数
二分查找	<code>BinarySearch.java</code>	<code>w02/binarysearch</code>	打印每一步，最坏情况表
计时	<code>Timing.java</code>	<code>w02/timing</code>	$O(n)$ 、 $O(\log n)$ 、 $O(n^2)$ 的实际表现

运行 `Timing` 程序并观察：n 增大 10 倍时，顺序查找和二分查找的时间怎样变化？n 加倍时， $O(n^2)$ 算法的时间怎样变化？

练习

1. **最大值**: 返回数组中最大元素的下标! 需要多少步?
2. **计数**: 数组中有多少个元素等于 `x` ?
3. **第一次出现**: 在有序数组中某个值可能出现**多次**, 用 $O(\log n)$ 步找到 `x` **第一次**出现的位置!
(`[1,2,2,2,5]` , `x=2` $\rightarrow$ `1`)
4. **插入位置**: 返回 `x` 可以插入的下标, 使数组仍然有序!
5. **整数平方根**: 用二分查找求满足 $r^2 \leq n$ 的最大 `r`!
6. **猜数字**: 计算机想一个 1 到 1000 之间的数, 我们得到“小了/大了”的回答。至多需要问几次?

参考答案: `Exercises02.java` 、 `w02/exercises`



课后作业

1. 写出阶 (大O) !

a) `for i ← 1 to n: for j ← 1 to 10: ...`

b) `for i ← 1 to n: for j ← 1 to i: ...`

c) `i ← n; While i > 0: i ← i div 3`

2. 在纸上对数组 `[3, 8, 15, 21, 30, 42, 57, 66, 81]` 分别查找 `x = 57` 和 `x = 10` ! 写出每一步的 `lo`、`hi` 和 `k` !

3. 编写函数, 用 $O(\log n)$ 步统计 `x` 在**有序**数组中出现的次数!