数据结构

夏天

xiat@ruc.edu.cn

中国人民大学信息资源管理学院

概论

- 计算机程序设计和数据处理的理论与技术基础
- 核心内容：
  - 线性表、栈和队列、字符串、树与森林、图、排序、查找
- 目标
  - 掌握数据结构的特点、存储方法和基本运算
  - 初步掌握算法的时间和空间分析技术
  - 能够针对不同数据对象的特性，选择适当的数据结构和存储结构以及相应的算法

课程信息

- 教学方式
  - 课堂教学与演示（听讲、看书、练习）
  - 练习
- 平时成绩：
  - 课程作业：40%
  - 课程现场展示：40%
  - 课堂提问与发言：10%
  - 考勤：10%
- 最终成绩：
  - 随堂闭卷考试 (50%) + 平时成绩 (50%)

学习建议

- 教材
  - 严蔚敏，吴伟民. 《数据结构 (C 语言版)》，清华大学出版社. (国内经典教材，写作严谨，广泛使用。)
  - 算法导论
  - 大话数据结构
- 在线课程
- 编程语言
  - 选择一种作为实践语言：Java、Python（根据同学们反馈选择）
  - 能够看懂不同编程语言的代码
  - 伪代码（Pseudocode）
    - 一种非正式，类似于英语结构，用于描述模块结构图的语言。

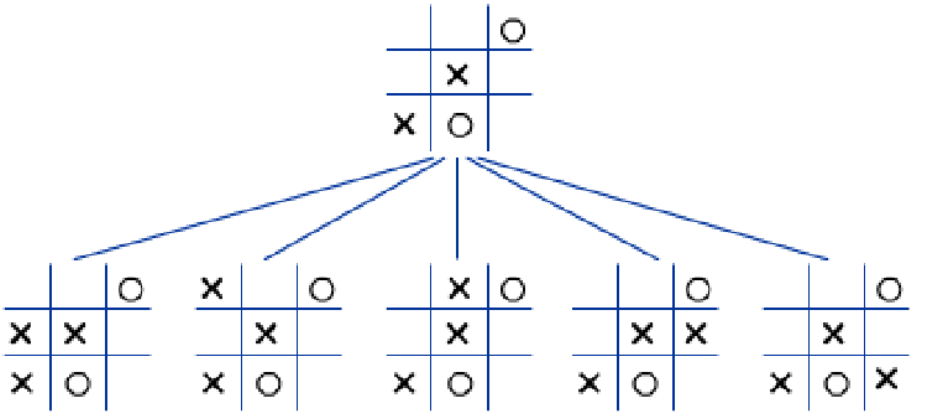
1997 年的人机大战

1997 年 5 月 11 日，国际象棋世界冠军卡斯帕罗夫与 IBM 公司的超级电脑深蓝（Deep Blue）对弈。

棋迷在纽约通过电视观战。当日，卡斯帕罗夫在纽约再次负于深蓝，从而在当年的“人机大战”中以一胜二负三和的战绩败北。

1. 深蓝重量达 1.4 吨，有 32 个 CPU，每个 CPU 有 8 块专门为进行国际象棋对弈设计的处理器，平均运算速度为每秒 200 万步。
2. 总计 256 块处理器集成在 IBM 研制的 RS6000/SP 并行计算系统中，从而拥有每秒超过 2 亿步的惊人速度。
3. IBM 研制小组向深蓝输入了 100 年来所有国际特级大师开局和残局的下法。美国特级大师本杰明将他对象棋的理解编成程序教给深蓝。虽不会思考，但它无穷无尽的计算能力在很大程度上弥补了这一缺陷。

1. 模型：棋盘、棋子的表示
2. 算法：对弈的规则和策略



3. 对弈的本质即在该空间里进行有效的搜索

2016 年 3 月 9 日，Google 旗下的 AlphaGo 电脑击败韩国九段棋手李世石。

2017 年 5 月 27 日，世界排名第一的人类棋手柯洁负于 AlphaGo，人机大战 2.0 定格在了 0:3。

- 对于擅长于计算的计算机来说，围棋的难度在很大程度上来自于19*19 路棋盘背后所蕴含的巨大的无法穷尽的变化（3361 种下法），这是基于“穷尽法”的“深蓝”无法在围棋上战胜人类的原因。
- AlphaGo 取得如此成绩，关键是深度学习和类神经网络技术。
- AlphaGo 将棋盘看作是一个 19*19 像素构成的图片，利用类似于卷积神经网络的技术预测下一步走法。

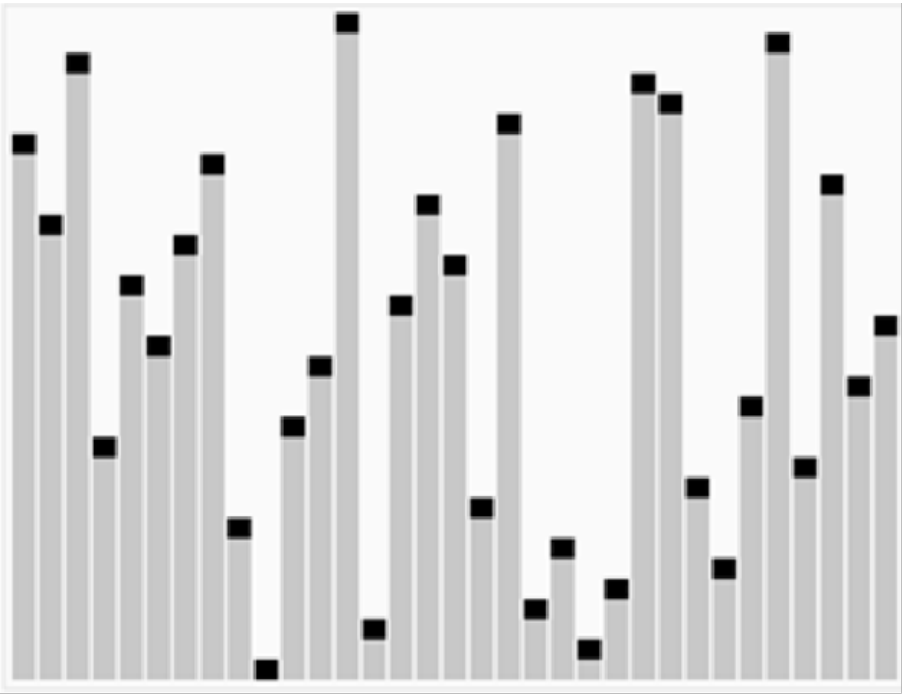
大语言模型加速技术

- Page Attention
- Flash Attention

数据结构与算法的优化，可以解决 LLM 落地应用的速度问题。

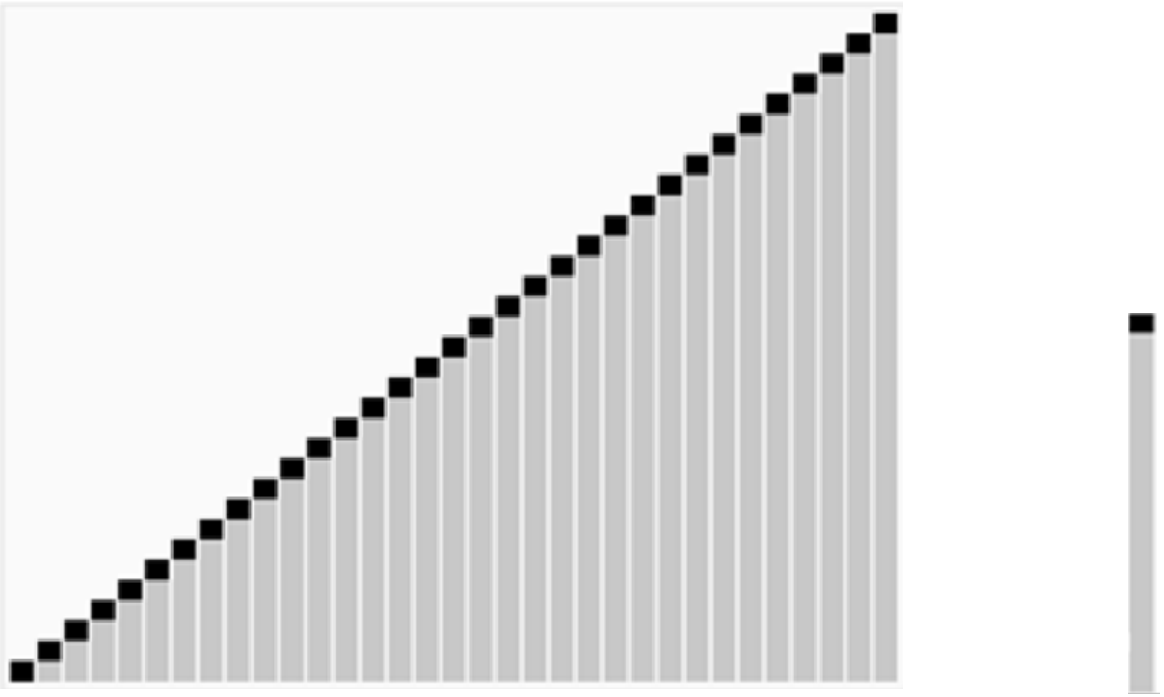
依赖于高效的搜索

- 如何快速的搜索到右方的图例?



依赖于高效的搜索

- 如何快速的搜索到右方的图例?



数据有序有利于查找。需要对数据进行高效的组织/排序。

Algorithms + Data Structures = Programs

- 算法
  - 处理问题的策略/操作步骤
- 数据结构
  - 静态数据表示的数学模型 (以及必须的操作)
- 程序
  - 为计算机处理问题编制的一组指令集

数据结构：研究描述现实世界实体的数学模型及其上的操作在计算机中的表示和实现。

FAQ

1. 数据结构是又一门编程课吗?
2. 不会 C/Java/Python 语言怎么办?
3. 用别的语言学习数据结构可以吗?
4. 怎样学好数据结构?

基本概念和术语

- 基本概念和术语
- 抽象数据类型的表示与实现
- 算法和算法分析

基本概念

- 1、数据 (Data)
  - 对客观事物的符号表示，在计算机科学中特指所有能被输入到计算机中并能被计算机程序处理的符号的总称。数据是程序加工的“原料”。
  - 例：一个文字处理程序的处理对象是字符串。
- 2、数据元素 (Data Element)
  - 数据的基本单位，在计算机程序中通常作为一个整体被考虑和处理。
  - 例：一本书，一条学生记录。
  - 数据元素可由若干数据项 (Data Item) 组成。

基本概念

- 3、数据对象 (Data Object)：性质相同的数据元素的集合，是数据的一个子集。
  - 例 1: 整数数据对象, $\mathrm{N} = \{0, 1, 2, \cdots\}$
  - 例 2: 字符数据对象, $\mathrm{C} = \{'\mathrm{A}', '\mathrm{B}', \cdots\}$
- 4、数据类型 (Data Type)：一个值的集合和定义在这个值集上的一组操作的总称。
  - 原子类型
    - 不可分解的数据类型
    - 如：整型、字符型、指针型、空类型...
  - 结构类型
    - 可分解为若干成分
    - 如：数组由分量组成，分量可以是整型，也可以是数组.

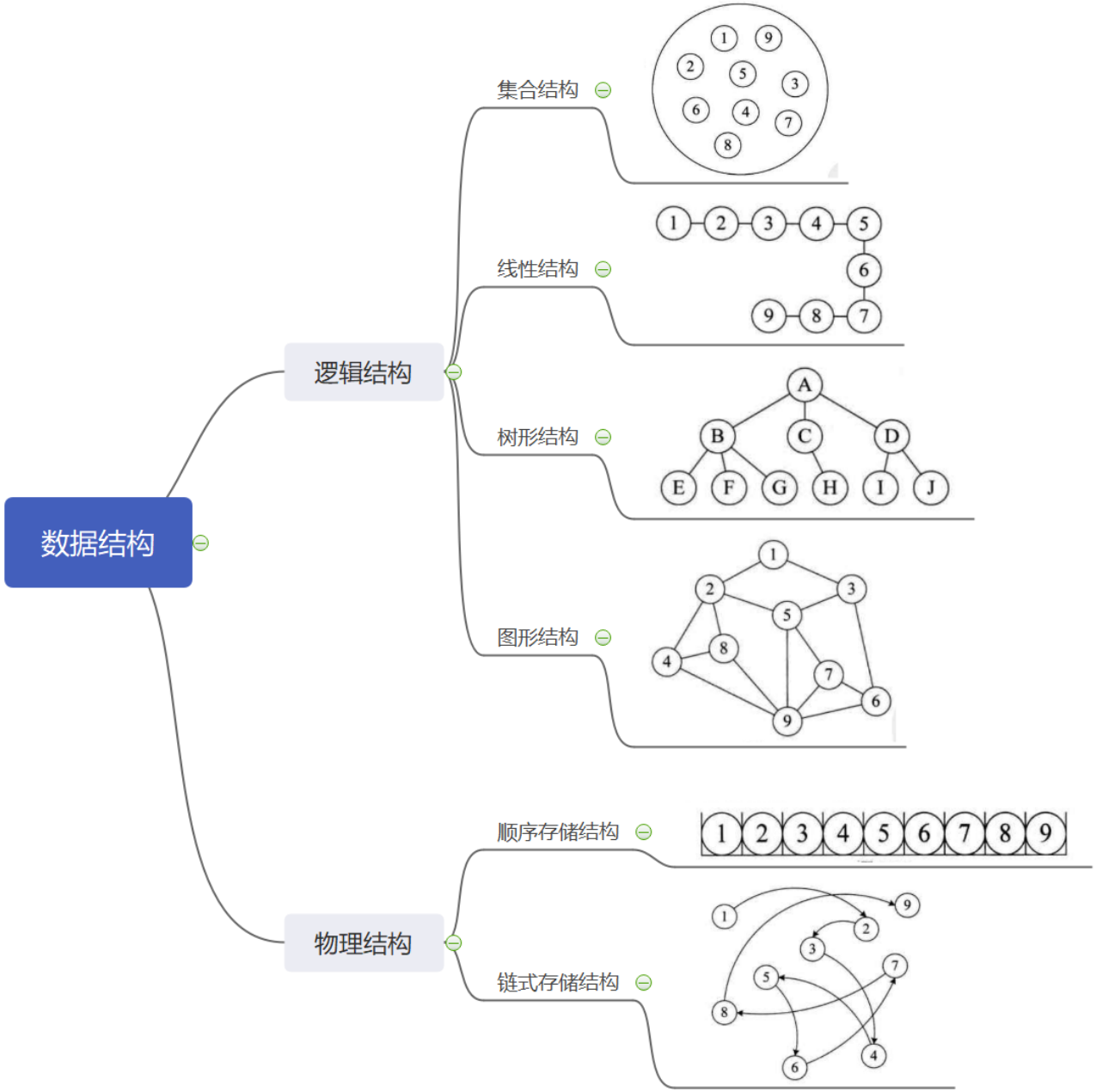
数据结构

数据结构 (狭义上) 是相互之间存在一种或多种特定关系的数据元素的集合。

- 逻辑结构
  - 集合、线性结构、树形结构、图状或网状结构
- 物理结构：数据结构在计算机中的表示/映像
  - 数据元素的表示是结点 (node)，即在计算机内用若干位组合起来表示一个数据元素。
  - 数据关系的表示有顺序映像和非顺序映像两种，由此得到顺序存储结构和链式存储结构。

物理结构

- 顺序映像 (顺序存储结构)
  - 逻辑上相邻的数据元素存储在物理位置上相邻的存储单元中比如在高级语言中的“一维数组”。
- 非顺序映像 (链式存储结构)
  - 数据元素可以存储在计算机内任意位置上，它们的逻辑关系用指针来链接。比如在高级语言中的“指针”。


数据结构
逻辑结构
集合结构
线性结构
树形结构
图形结构
物理结构
顺序存储结构
链式存储结构

抽象数据类型的表示和实现

- 抽象数据类型 (Abstract Data Type) 是指一个数学模型以及定义在该模型上的一组操作。
  - 例如，矩阵 +（求转置、加、乘、求逆、求特征值）构成一个矩阵的抽象数据类型

    “抽象”在于数据类型的数学抽象特性，与具体表示和实现无关。

抽象数据类型的示例

```
ADT Complex {
    数据对象: D = {e1,e2 | e1,e2 in RealSet  }
    数据关系: R1 = {<e1,e2> | e1 是复数的实数部分,  e2 是复数的虚数部分 }
    基本操作:
        InitComplex( &Z, v1, v2 )
            操作结果: 构造复数 Z,实部和虚部分别被赋以参数 v1 和 v2 的值.
        DestroyComplex( &Z)
            操作结果: 复数 Z 被销毁.
        GetReal( Z, &realPart )
            初始条件: 复数已存在。
            操作结果: 用 realPart 返回复数 Z 的实部值.
      GetImag( Z, &ImagPart )
            初始条件: 复数已存在。
            操作结果: 用 ImagPart 返回复数 Z 的虚部值.
      Add( z1,z2, &sum )
            初始条件: z1,z2 是复数。
            操作结果: 用 sum 返回两个复数 z1,z2 的和值.
} ADT Complex
```

算法和算法分析

- 基本概念和术语
- 抽象数据类型的表示与实现
- 算法和算法分析

Alkhwarizmi 与 Algorithm

- Alkhwarizmi（约 780～约 850），数学家，代数与算术的整理者。阿拉伯文 Alkhwarizmi 原意是来自 (al-) 花剌子模 (Khwarizmi) 。
- Alkhwarizmi 在当时的学问中心巴格达，服务于宫廷。他写了一本有关代数的书，这本书转成欧文，书名逐渐简化为 algebra（代数）。
- 后来 Alkhwarizmi 又引进了印度数字发展算术，后经 Fibonacci（1170～1250 年）引入欧洲。欧洲人把 Alkhwarizmi 这个字拉丁化，称用十进位印度阿拉伯数字来进行有规则可寻的计算的算术为 Algorithm。后来算术转用其它的字（如 arithmetic）, 而 algorithm 则成为计算机科学的行话。

算法 (Algorithm)

- 算法是对特定问题求解步骤的一种描述，是指令的有限序列，其中每一条指令表示一个或多个操作。算法可以看做是从输入到输出的一个映射。
  - Input $\rightarrow$ Ouput
- 算法的描述
  - 自然语言
  - 程序流程图
  - 伪码: 它忽略高级程序设计语言中一些严格的语法规则与描述细节，因此比程序设计语言更容易描述和被人理解，而比自然语言更接近程序设计语言。它虽然不能直接执行但很容易被转换成高级语言。
  - 编程语言

试写出 [9,2,5,1,6,7,10] 经过下列程序处理后的序列。

```
//冒泡法
void bubble (int a[], int n) {
  for (i=n-1; i>=1; i - -) {
    for (j=0; j<i; j++)
      if (a[j]>a[j+1]) {
        swap(&a[j],&a[j+1]);
      }
  }
}
```

- $i = n - 1 \cdots$
- $i = n - 2 \cdots$

算法设计的要求

- 无语法错误
- 对于选择的、典型、苛刻甚至带有刁难性的几组输入能够得出满足要求的结果
- 对于一切合法输入都给出满意结果

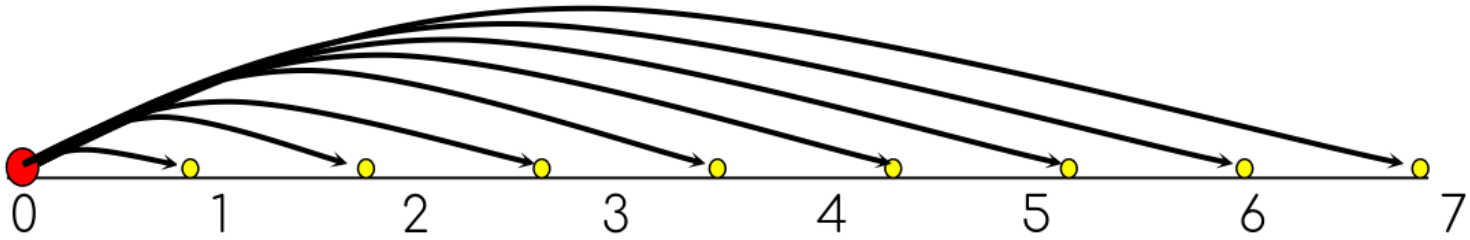
- 算法应易于（人的）理解、易于发现错误及进行调试。
- 对非法输入有恰当的反映，而不是中断程序、或给出错误的输出
- 效率指的是算法执行时间；存储量指的是算法执行过程中所需的最大存储空间。两者都与问题的规模有关。

算法效率

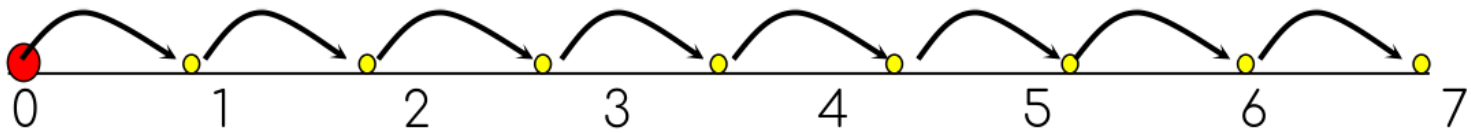
- 算法执行时间需通过依据该算法编制的程序在计算机上运行时所消耗的时间来度量。
- 相关影响因素包括：
  - 机器硬件配置直接影响计算机执行指令的速度
  - 编写程序的语言：越高级，效率越低
  - 算法选用的策略
  - 问题的规模：规模越大，耗时越久
  - 编译程序产生的机器代码的质量

快递员配送包裹

- 假定快递员以恒定速度行驶，比较下面的方案
  - 方案 1：配送 n 个包裹的总时间为 $n^2 + n$



  - 方案 2：配送 n 个包裹的总时间为 2n



各函数的增长率

在问题规模增长时，算法执行时间必定也会增长。我们关心的是这个执行次数以什么样的数量级（增长率）增长。

n	$\log_2 n$	$n \cdot \log_2 n$	n^2	2^n
1	0	0	1	2
2	1	2	2	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1,024	4,294,967,296

观察 $g(n)$ 和 $f(n)$ 的相对增长趋势

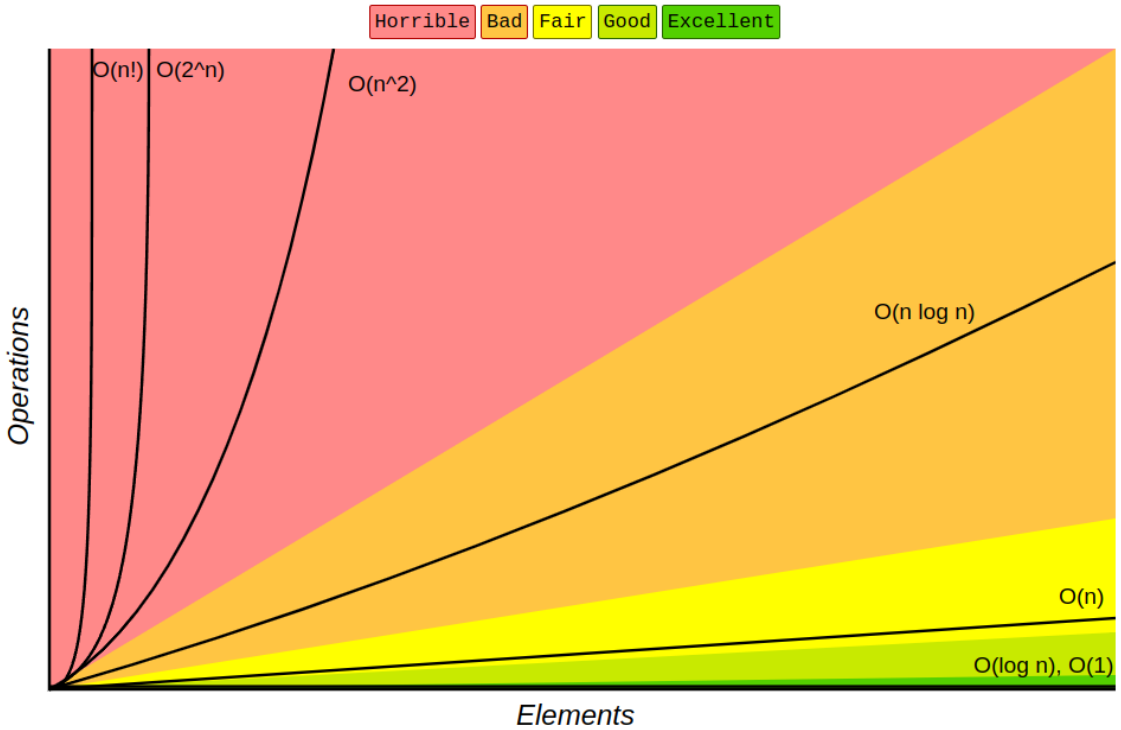
n	$g(n) = n^2$	$f(n) = n^2 + 4n + 20$
10	300	360
50	2,500	2,720
100	10,000	10,420
1,000	1,000,000	1,004,020
10,000	100,000,000	100,040,020

- 当 n 变得很大时，n^2 成为主导项，通过 $g(n)$ 的变化情况就可以预测 $f(n)$ 的变化。

$$O(c_1 n^k + c_2 n^{k-1} + \cdots + c_k) = O(n^k)$$

- 例如，$T(n) = 3n^2 + 4n + 20$，记为 $O(n^2)$

Big-O Complexity Chart



From better to worse:

$\mathrm{O}(1)$	constant time
$\mathrm{O}(\log \mathrm{n})$	log time
$\mathrm{O}(\mathrm{n})$	linear time
$\mathrm{O}(\mathrm{n} \log \mathrm{n})$	log linear time
$\mathrm{O}(\mathrm{n}^2)$	quadratic time
$\mathrm{O}(\mathrm{n}^3)$	cubic time
$\mathrm{O}(2^{\mathrm{n}})$	exponential time

From:

`https://www.bigocheatsheet.com`

大 O 表示法

- 算法的 (渐近) 时间复杂度记作：$\mathrm{T(n) = O(f(n))}$
  - 表示随问题规模 n 的增大，算法执行时间的增长率和 $\mathrm{f(n)}$ 的增长率相同。
- 通常是从算法中选取一种最基本的元操作，以该操作重复执行的次数 (频度) 来度量算法效率

例：矩阵相乘 (C 代码示例)

```
for (i=1; i<=n; i++)
  for (j=1; j<=n; j++) {
    c[i][j] = 0;
    for (k=1; k<=n; k++)
      c[i][j] =c[i][j]+ a[i][k] * b[k][j]; // 该语
  }
```

整个算法的执行时间与元操作重复执行的次数 n^3 成正比，记作 $T(n) = O(n^3)$

例：矩阵相乘 (Python 代码示例)

```
n = 3
a = [[1,2,3],[4,5,6], [7,8,9]]
b = [[1,2,3],[4,5,6], [7,8,9]]

c = [ [0]*n for _ in range(n) ]
for i in range(n):
    for j in range(n):
        c[i][j] = 0
        for k in range(n):
            c[i][j] = c[i][j] + a[i][k] * b[k][j]
```

整个算法的执行时间与元操作重复执行的次数 n^3 成正比，记作 $T(n) = O(n^3)$

请观察 [10,9,7,6] 和 [6,7,9,10] 在处理上的不同。

```
void bubble(int a[], int n) {
  for (i=n-1; i>=1; i - -) {
    for (j=0; j<i; j++)
      if (a[j]>a[j+1]) {
          swap(&a[j],&a[j+1]);
      }
  }
}
```

请观察 [10,9,7,6] 和 [6,7,9,10] 在处理上的不同。

```
void bubble(int a[], int n) {
  for (i=n-1; i>=1; i - -) {
    for (j=0; j<i; j++)
      if (a[j]>a[j+1]) {
          swap(&a[j],&a[j+1]);
      }
  }
}
```

- 若初始文件是正序的，一趟扫描即可完成排序。
- 若初始文件是反序的，需要进行 $n-1$ 趟排序。
- 计算平均复杂度，或最坏情况下的时间复杂度.

- 最好情况的时间复杂度（初始文件是正序的）

  [6,7,9,10] — 一趟扫描即可完成排序，共计 $\mathrm{n}-1$ 次比较（0 次交换）。

  冒泡排序最好情况的时间复杂度为 $\mathrm{O}(\mathrm{n})$

- 最坏情况的时间复杂度（初始文件是反序的）

  [10,9,7,6] — 需要进行 n-1 趟扫描，每趟进行 i 次比较 + 交换 $(1 \leq \mathrm{i} \leq \mathrm{n}-1)$，共计 $\frac{\mathrm{n}(\mathrm{n}-1)}{2}$ 次比较 + 交换。

  冒泡排序的最坏情况的时间复杂度为 $\mathrm{O}(\mathrm{n}^2)$

算法的存储空间需求 (内存)

- 算法需要为: 输入数据、程序、辅助变量提供存储空间。
- 算法的空间复杂度：$\mathrm{S(n)} = \mathrm{O(g(n))}$
- 随着问题规模增大，算法运行所需存储量的增长率与 $\mathrm{g(n)}$ 的增长率相同。
- 若所需存储量依赖于特定的输入，则通常按最坏情况考虑。
- 相对于时间复杂度而言，空间复杂度在很多时候不需进行分析。

小结

1. 数据结构研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作。
2. 数据、数据元素、数据对象、数据结构的概念
3. 常见的基本数据结构
4. 以大 O 表示法分析算法时间复杂度