

Algorithm - P1

算法

(Algorithm) 非形式地指任何良定义的计算过程

该过程取某个值或值的集合作为输入，并产生某个值或值的集合作为输出

算法即是把输入转换成输出的计算步骤的一个序列

如果有良说明的计算问题是陈述说明了期望的输入/输出关系

则算法可看成求解的工具，描述一个特定的计算过程来实现该输入/输出关系

如关于排序问题的形式定义

输入：包含 n 个数的一个序列 $\langle a_1, a_2, \dots, a_n \rangle$

输出：输入序列的一个排列 $\langle a'_1, a'_2, \dots, a'_n \rangle$ ，满足 $a'_1 \leq a'_2 \leq \dots \leq a'_n$

实例 (Instance) 指计算问题的一个特定输入，如排序问题的输入 $\langle 13, 21, 5, 8, 2, 1, 1, 3 \rangle$

问题是实例由计算该问题的解所必须的（满足问题陈述中强加的各种约束的）输入组成

正确的

一个算法是正确的，若对每个输入实例，算法都以正确的输出停机

并称正确的算法解决了给定的计算问题

不正确的算法或者对某些输入实例可能根本不停机，或者可能以不正确的回答而停机

注意：不正确的算法只要其错误率可控，有时也可能是有用的

如：对于正整数 n, b ，且 $1 < b < n$

如果 n 是合数且 n 通过以 b 为底的米勒测试 (Miller-Rabin primality test)

则称 n 为以 b 为底的强伪素数

注意一个合数 n 最多只能通过至多 $(n-1)/4$ 个以 b 为底的米勒测试

数据结构是一种存储和组织数据的方式，旨在便于访问和修改

注意：没有一种单一的数据结构对所有用途均有效

有效算法

等同于离散数学中所描述的多项式 (polynomial complexity) 最坏情形复杂度的算法

而能用有效算法求解的问题称为易解的 (tractable)

NP 类 (nondeterministic polynomial) 问题没有有效算法求解

但对于一个解可以有效算法来验证

如果计算机无限快，并且计算机存储器是免费的，那么用于求解某个问题的任何正确的方法都可用

即将算法看成像计算机硬件一样的技术

Algorithm - P2

伪代码 (pseudocode) 用最清晰、最简洁的表达方法来说明给定的算法
通常不关心软件工程的问题，常常忽略数据结构、模块性和错误处理问题

约定 缩进表示结构 即采用缩进来代替常规的块结构(block)标志

如 Python 即采用这种以缩进表示 block 结构的方式

其他如 begin...end 语句，VBA 中使用 end 语句表示 block 结构结束

C++ 采用 {} (braces), Racket 采用 () (parentheses) 和 [] (square bracket)

循环结构 while, for, repeat-until 与编程语言中使用的逻辑一致 具有等价的结果

注意：伪代码中规定在退出循环后，循环计数器将保持其值

因此紧接在一个 for 循环后，循环计数器值为第一个超出 for 循环界限的那个值

如在 C++ 中 int i; for(i=1; i<11; ++i){}

std::cout << "i = " << i << '\n'; → i = 11

在 for 循环中：如果每次循环增加其循环计数器时，使用关键词 to，逻辑同 C++ 中的 ++i

如果每次循环减少其循环计数器时，使用关键词 downto，逻辑同 --i, i += step

如果循环计数器以大于 1 的一个量改变时，改变量要在可选关键词 by 之后，逻辑同

注意 for 循环与 Python 中的 for 循环的实现略有不同

伪代码 for 循环采用循环计数器 - 循环界限 - 更新计数器的形式

Python for 循环采用从迭代器 (Iterator) 中依次获得下一计数器的形式

条件结构 if-else：与 Python 中的 if-else 语句最相似，并用 elseif (elif) 来进行多路测试

注意：与 Haskell 相比省略了 if-then-else 中的 then，与 Racket 相比则保留了 else

多重赋值 i=j=e 表示将表达式 e 的值绑定给变量名 i, j，其处理等价于 j=e; i=j;

注意：在 C++ 和 Python 中支持如 i=j=e 的赋值语法，并且其运算是右结合的，即 P(i=(j=e))

复合数据通常组织成对象 (Object)，对象又由属性 (property/member) 组成

通过形如 对象名.属性名 的句法来访问 特定对象的特定属性，如 list.length (list.size())

把表示数组或对象的变量看做指向表示数组或对象的数据的一个指针

对于某个对象 x 的所有属性 f，赋值 y=x 导致 y.f 等于 x.f，完成后 x, y 指向相同对象

属性记号可以“串联”，且是左结合的，即有 x.f.g 等价于 (x.f).g

且对赋值 y=x.f，则有 y.g 等于 x.f.g

当指针根本不指向任何对象，赋值给其特殊值 NIL，与 C++ 中的 nullptr 等价

注意：伪代码对象的处理与 Python 的处理方式类似

Algorithm - P3

伪代码约定

按值传递参数到过程：被调用过程接收其参数自身的副本

在被调用过程中对参数赋值，对调用过程不可见

■ 传递对象到过程：指向对象数据的指针被复制，而对象属性未被复制

对于指针的赋值，对调用过程不可见，如 $x = y$

而对于对象属性的赋值，对调用过程是可见的，如 $x.f = e$

注意这种处理方式类似于 Python 中的参数传递，即函数对可变对象参数的改变会影响原对象

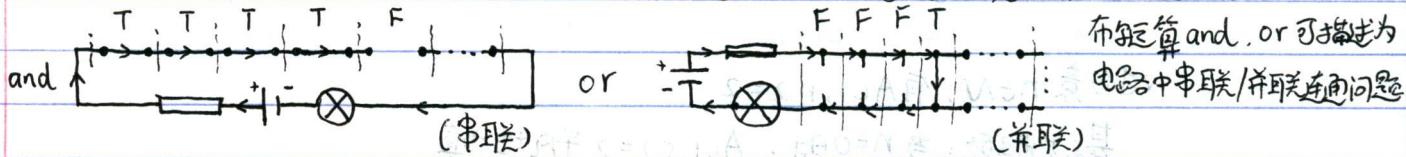
return i语句：伪代码中 return i语句立即控制返回到调用过程的调用点

另外与 Python 类似，伪代码允许一个 return 语句返回多个值，如 return x, y

布尔运算符短路：伪代码中布尔运算符 and 和 or 都是短路的 (short circuit)

即 and 运算在遇到第一个 False 的表达式即返回 False，否则继续计算下一个表达式

or 运算在遇到第一个 True 的表达式即返回 True，否则继续计算下一个表达式



好处是允许使用如 $x \neq \text{NIL} \text{ and } x.f = y$ 的表达式而无需担心 $x = \text{NIL}$ 时后一表达式错误

关键词 error 表示被调用过程遇到错误情况而报告一个错误

伪代码假定调用者可以正确处理错误，于是不对错误说明采取什么行动

插入排序

(insertion sort) 对于少量元素的排序，是一个有效的算法

输入：n个数的一个序列 $\langle a_1, a_2, \dots, a_n \rangle$

输出：输入序列的一个排列 $\langle a'_1, a'_2, \dots, a'_n \rangle$ ，满足 $a'_1 \leq a'_2 \leq \dots \leq a'_n$

期望排序的数据称为关键词，虽然概念上是在排序一个序列，但输入以 n 元素数组的形式出现

■ ■ ■ ■ ■ INSERTION-SORT(A) 如下：

for j=2 to A.length

key = A[j]

i=j-1

while i>0 and A[i]>key

A[i+1] = A[i]

i=i-1

A[i+1] = key

将 A[j] 插入

已排序好的

A[1..j-1] 中的

插入器

正确位置

| | | | | |
|---|---|---|---|---|
| 5 | 2 | 2 | 2 | 1 |
| 2 | 5 | 4 | 4 | 2 |
| 4 | 4 | 5 | 5 | 4 |
| 5 | 5 | 6 | 6 | 5 |
| 6 | 6 | 1 | 1 | 6 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | 3 |
| 3 | 3 | 3 | 3 | 3 |
| 4 | 4 | 4 | 4 | 4 |
| 5 | 5 | 5 | 5 | 5 |
| 6 | 6 | 6 | 6 | 6 |

注意：插入排序算法原址排序输入的数据，即任何时候最多只有常数个数字存储在数组外

Algorithm - P4

CS 331

Homework 4

循环不变式 (loop invariant) 或称循环不变量, a property of a program loop that is true before and after each iteration
即一个断言，在循环任何一个 iteration 开始前 (及结束) 均为真，
进而可知，只要确定循环在某一次循环前停止，则此断言为真
再进一步，由于断言在整个循环过程均为真，则可知循环的输出结果为真
注意，可以统一到某一次循环开始前为真，因为可以认为这是最后一次循环结束后
这样循环停止时的断言状态即为循环计数器首次超出界限的该次循环开始前
进一步扩展为在循环条件首次为假的循环开始前

基本证明 初始化：循环的第一次迭代 (iteration) 开始前，断言为真

保持：如果循环的某次迭代开始前断言为真，则循环的下次迭代开始前断言仍为真

终止：在循环终止时，断言为真，并且作为一个有用的性质用于证明算法是正确的

与递归的关系 注意循环不变式可视为一种向后的推导过程

数学归纳法

循环不变量

基础步骤： $n=1$ 时， $P(1)$ 为真

初始化：第 1 次迭代开始前，断言为真

归纳步骤： $\forall n \in \mathbb{Z}^+ (P(n) \rightarrow P(n+1))$

保持：如果第 n 次迭代前为真，则第 $n+1$ 次迭代前为真 ($n \in \mathbb{Z}^+$)

全局实例：对于确定的 $C \in \mathbb{Z}^+$, $P(C)$ 为真

终止：如果循环在第 C 次迭代开始前结束，此时断言为真

正确性：对于待证的命题 P , 或者 $P(C) \equiv P$

正确性：对于算法的正确性，或者断言本身即是算法正确

或者 $P(C) \wedge (C \geq q_i) \rightarrow P$

或者断言可用于证明算法是正确的

插入排序的
循环不变量

断言：在循环不计数器为 j 的迭代开始前 $A[1..j-1]$ 是已经排序的子数组

初始化：在首次迭代 (循环计数器 $j=2$) 开始前 $A[1..1]$ 是已经 (平凡地) 排序的子数组

保持：在第 $j-1$ 次迭代 (循环计数器 j) 开始前 $A[1..j-1]$ 是已经排序的数组

在第 j 次迭代 (循环计数器 $j+1$) 开始前，此时子数组为 $A[1..j]$ ，假设原来的 $A[j..j]$ 在第 j 个位置

可知 $\forall i \leq k < j (A[i] \leq A[j]) \wedge A[j] = A_0[k-1]$ (即 $A[j..j]$ 与原子数组对应位置相同)，

而且 $\forall i \leq k < j (A[j] < A[k] \wedge A[k] = A_0[k-1])$ ，可是有 $A[1..j]$ 是已经排序的子数组

终止：在第 $length$ 次迭代 (循环计数器 $length+1$) 开始前，循环终止，此时 $A[1..length]$ 是已经排序的数组

用于正确性证明，而非直接得出算法正确结论的情况：如 Racket 中 k -item 函数，

(define (k-item stream n) ...)

终止第 $n+1$ 次迭代后。

(第 $n+1$ 次调用返回前)

(letrec ([f (lambda (st acc n)

保持：在第 k 次迭代前，
acc 为前 $k-1$ 个元素的逆序

if (= 0 n) (reverse acc)]
[即第一次调用返回时，
acc 为 'c a n d n-1 ... a 1)

[即第一次调用返回时，
acc 为 'c a n d n-1 ... a 1)

(let ([pr (cdr st)])
(f (cdr pr) (cons (car pr) acc) (- n 1))))], [第 $n+1$ 次迭代后]

(f stream '() n))) [即第一次调用返回后，
acc 为前 0 个元素的逆序，即 'c)]

] 初始化：在第 1 次迭代前，acc 为前 0 个元素的逆序，即 'c)

Algorithm - P5

线性查找

LINEAR SEARCH C 数组 A, $A = \langle a_1, a_2, \dots, a_n \rangle$, 值 v

$i = N\text{IL}$ ($\text{NIL} = \text{int} = 0 = \text{nullptr}$) 初始化为空指针 NIL, 即 C++ 中的 nullptr

for $j = 1$ to $A.\text{length}(n)$ 顺序检查每个数组元素

if $A[j] = v$ then $i = j$ 如果 $A[j] = \text{查找值 } v$, 则更新 i 为当前下标 j

return i; $i = \text{NIL}$

循环不变量: $i = \text{NIL}$ 或 $i \leq j$ 在第 j 次迭代前, $1 \leq i \leq j-1$ 且 $A[i] = v$

初始化: 在第 1 次迭代前, i 被初始化为 NIL

保持: 在第 j 次迭代前, $i = \text{NIL}$ 或者 $1 \leq i \leq j-1$ 且 $A[i] = v$. 在循环中, 如果 $A[j] = v$, 则 i 更新为 j

在第 $j+1$ 次迭代前, i 或者保持第 j 次迭代前的状态, 或者更新为 j, 于是 $A[i] = v$

终止: 在第 $n+1$ 次迭代前, $i = \text{NIL}$ 或者 $1 \leq i \leq n$ 且 $A[i] = v$

二进制整数加法 输入: 两个 n 位二进制整数 a , 保存于两个 n 元数组 $A = \langle a_1, a_2, \dots, a_n \rangle$, $B = \langle b_1, b_2, \dots, b_n \rangle$

输出: 一个 $n+1$ 位二进制整数, 保存于一个 $n+1$ 元数组 $C = \langle c_1, c_2, \dots, c_n, c_{n+1} \rangle$

分析算法 目的是预测该算法需要的资源, 特别是计算时间, 也包括内存、通信带宽或计算机硬件

随机访问机 (random-access machine, RAM), 通用的单处理器计算模型, 用于描述算法所有资源及其代价

RAM 模型中, 指令一多接一多执行, 没有并发操作. 包含了真实计算机中的常见指令

算术指令 (加法, 减法, 乘法, 除法, 取余, 向上取整, 向下取整)

数据移动指令 (装入, 存储, 复制) (如 8086 的 mrmovl, rmmovl, rrmovl)

控制指令 (条件与无条件转移, 子程序调用与返回) (如 8086 的 jmp, jxx, call, ret)

通常认为每条指令所需时间为常量.

RAM 模型中的数据类型有整数型 (integer) 和浮点型 (float)

并对每个数据字的规模假定一个范围

如当处理规模为 n 的输入时, 假定一个常量 $C \geq 1$, 由 $\lceil C \lg n \rceil$ 位表示整数

$C \geq 1$, 则每个字都可以保存 n 的值, 从而可以索引到单个输入元素

(为常量, 则字长不受无限增长)

RAM 模型中存在应尽量避免的灰色区域, 如指数运算指令

在受限情况下, 指数运算可以是常量时间指令, 如以左移 k 位计算与 2^k 的乘积

RAM 模型中不试图对当代计算机中常见的内存层次进行建模, 通常已能很好地预测性能

RAM 模型中需要一种简单、易于理解的公式的形式总结算法的行为

也需要用到数学工具, 如组合学, 概率论, 代数技巧.

以及识别一个公式中最有意义的项的能力

Algorithm - P6

时间复杂度

8.7 - Time Analysis

通常把一程序的运行时间描述成其输入规模的函数

$f(n)$ n 为输入规模，即对不同的问题有不同的输入规模定义

输入规模 其最佳概念依赖于研究的问题，即对不同的问题是是否有不同的输入规模定义

如排序或计算离散傅立叶变换，其度量为输入中的项数 n

如两个二进制整数相乘，其输入规模度量为表示输入所需的总位数 k

如求一个图的最大生成树，其度量为图中的顶点数和边数 (v, e)

对于研究的每个问题，应指出所使用的输入规模量度

运行时间 指算法在特定输入上的执行的基本操作数或步数

“步”的概念的定义是为尽量独立于机器

通常假设执行每行伪代码需要常量时间，即分析时假定第 i 行每次执行需要时间常量 c_i

插入排序分析 INSERTION-SORT(A) 假设输入项数为 n 代价 执行次数

for $j=2$ to $A.length$ $\left\{ \begin{array}{l} C_1 = 1 \\ C_2 = n-1 \end{array} \right.$ 执行循环测试判定的次数比循环体多 1 次

$key = A[j]$ $\left\{ \begin{array}{l} C_3 = 0 \\ C_4 = 1 \end{array} \right.$ $n-1$ 包括 1 次退出判定

$i=j-1$ $\left\{ \begin{array}{l} C_5 = 0 \\ C_6 = 1 \end{array} \right.$ $n-1$ t_j 为在第 j 次迭代中

 while $i > 0$ and $A[i] > key$ $\left\{ \begin{array}{l} C_7 = 1 \\ C_8 = 1 \end{array} \right.$ $\sum_{j=2}^n (t_j-1)$ while 循环所执行

$A[i+1] = A[i]$ $\left\{ \begin{array}{l} C_9 = 1 \\ C_{10} = 1 \end{array} \right.$ $\sum_{j=2}^n (t_j-1)$ 循环判定测试的次数

$i = i-1$ $\left\{ \begin{array}{l} C_{11} = 1 \\ C_{12} = 1 \end{array} \right.$ $n-1$

$A[i+1] = key$ $\left\{ \begin{array}{l} C_{13} = 1 \\ C_{14} = 1 \end{array} \right.$ $n-1$

注意：以语言说明的计算步往往是一个过程的度量，该过程需要的时间大于一个常量

所以需区分调用子程序的过程（找到函数的定义，并计算参数的值传给函数运算环境）

执行子程序的过程（在函数的运算环境中运行函数/子程序）

现有 $T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_4 \sum_{j=2}^n t_j + C_5 \sum_{j=2}^n (t_j-1) + C_6 \sum_{j=2}^n (t_j-1) + C_7(n-1)$

最坏情况（线性函数 $\Theta(n)$ ） 最坏情况（二次函数 $\Theta(n^2)$ ）

输入数组已排序，则每次迭代中 $A[j-1] \leq A[j]$ 输入数组是逆序排列的，即在每次迭代中

于是有 $t_j = 1$ $\forall 1 \leq i \leq j-1 A[i] > key$ ，于是有 $t_j = j$ ($i=j-1$ to 1)

即 $T(n) = (C_1 + C_2 + C_3 + C_4 + C_7)n$ 即 $T(n) = (C_1 + C_2 + C_3 + C_4 + C_7)n$

最好情况（所有 $t_j = 1$ ） $T(n) = (C_1 + C_2 + C_3 + C_4 + C_7)n$

注意：对于内存等资源，执行 C_i 步且执行 n 次的语句贡献 $C_i n$ 不必成立，

即访问 m 个存储字且执行 n 次的一个语句不必访问 mn 个不同的存储字

注意：通常算法的运行时间对给定的输入是固定的，也存在不固定的“随机化”算法

Algorithm - P7

通常集中于求最坏情况运行时间，即对规模为几的任何输入，算法的最大运行时间

最坏情况运行时间给出任何输入的运行时间的一个上界

对某些算法，最坏情况经常出现，如在数据库中检索一条特定信息

“平均情况”往往与最坏情况大致一样差

常常假定给定规模的所有输入具有相同的可能划分，但通常不成立

有时可使用“随机化算法”，以允许进行概率分析，并产生某个期望的运行时间

增长率

或称增长量，一种更简化的抽象。不但忽略实际的语句代价，也忽略抽象的代价 C_i

只考虑公式中最重要的项，即忽略低阶项和最重要的项的常系数

如果一个算法的最坏情况运行时间具有比另一个算法更低的增长量级，则称更有效

分治法

(divide and conquer algorithm)，遵循分治法的算法在结构上是递归的(recursive)

即把原问题分解为几个规模较小但类似于原问题的子问题

递归地求解这些子问题，再合并这些子问题的解来建立原问题的解

对于给定的问题，算法一次或多次地递归地调用自己以解决紧密相关的若干子问题

分治模式

分解：分解原问题为若干子问题，这些问题是原问题的规模较小的实例

角解决：递归地求解各子问题，如果子问题规模足够小，则直接求解(对应基础步骤)

合并：合并这些子问题的解以得到原问题的解

归并排序的合并步骤：对于已排序的子数组 $A[p..q]$ 和 $A[q+1..r]$ ，以 $\Theta(n)$ 的时间合并为 $A[p..r]$ (已排序的)

MERGE(A, p, q, r)

$$n_1 = q - p + 1$$

$$n_2 = r - q$$

for $i = 1$ to n_1

$$L[i] = A[p+i]$$

for $j = 1$ to n_2

$$R[j] = A[q+j]$$

$$L[n_1+1] = \infty$$

$$R[n_2+1] = \infty$$

$$j = 1$$

for $k = p$ to r

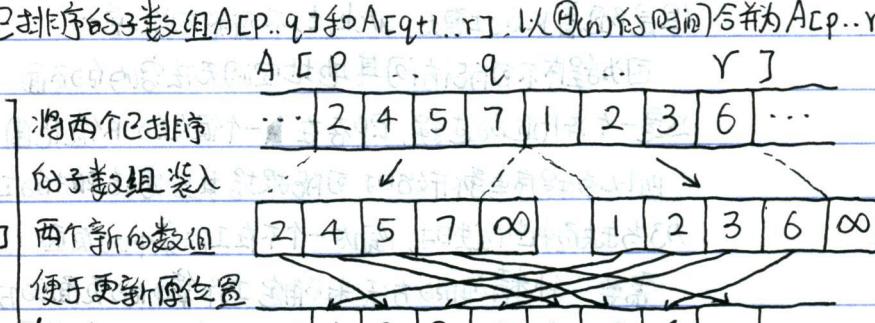
if $L[i] < R[j]$

$$A[k] = L[i]$$

$$i = i + 1$$

$$\text{else } A[k] = R[j]$$

$$j = j + 1$$



】设置哨兵牌，在这个算法中用于避免在每个步骤必须检查是否有堆为空

】重置循环计数器(堆的下标指针)，也可以重新定义

】每次迭代选取两个堆顶的数字中较小的一个

】并将其装入原数组的相应位置，第 m 次迭代更新 $A[p-1+m]$

】循环计数器(数组下标为 $[p, r]$)，第 $r-p+2$ 次迭代前终止

】由于哨兵牌为 ∞ ，所以当一个堆到达哨兵时，则只取另一堆的值

Algorithm - P8

归并排序的循环不变量：(合并步骤)

第 $k-p+1$ 次迭代前，子数组 $A[p..k-1]$ 为已排序的 $L[i..n_1+1]$ 和 $R[i..n_2+1]$ 中的 $k-p$ 个最小元素
 $L[i..n_1+1]$ 和 $R[i..n_2+1]$ 为各自数组未被复制回 A 的最小元素

初始化：在第一次迭代前，子数组 $A[p..k-1]$ 为空数组，由于 $k=p$ ，其包含 $k-p=0$ 个 L 和 R 的最小元素
 $L[i..n_1+1]$ 和 $R[i..n_2+1]$ 此时 $i=j=1$ ，而 L 和 R 是已排序的数组，于是有 $L[i..j]$ 和 $R[i..j]$ 为未复制回 A 的最小元素

保持：假设在第 $k-p+1$ 次迭代前，循环不变量为真，则在循环迭代中比较 $L[i..j]$ 与 $R[i..j]$
 如果 $L[i..j] \leq R[i..j]$ 则将 $L[i..j]$ 复制回 $A[k..j]$ 的位置上， $i=i+1$, $R[i..j]$ 保持不变
 于是有 $A[p..k]$ 为 L 和 R 中的 $k-p+1$ 个最小元素

而 $L[i..j]$ 和 $R[i..j]$ 为未复制回 A 的各自数组最小元素

如果 $L[i..j] > R[i..j]$ ，则将 $R[i..j]$ 复制回 $A[k..j]$ 的位置上， $j=j+1$, $L[i..j]$ 保持不变
 于是有 $A[p..k]$ 为 L 和 R 中的 $k-p+1$ 个最小元素

终止：终止时 $k=r+1$ ，则此时 $A[p..r]$ 为 L 和 R 中的 $r-p+1$ 个最小元素

而 L 和 R 初始只有 $r-p+3$ 个元素，于是此时 $L[i..j]$ 和 $R[i..j]$ 为各自数组的哨兵牌
 可知此时所有元素都已复制回数组 A 并完成排序

归并排序

MERGE-SORT(A, p, r)

if $p < r$

$q = \lfloor (p+r)/2 \rfloor$

MERGE-SORT(A, p, q)

MERGE-SORT($A, q+1, r$)

MERGE(A, p, q, r)

】如果 $p=r$ ，即 $A[p..r]$ 只有 1 个元素，则直接返回

】计算分界点 q 的下标

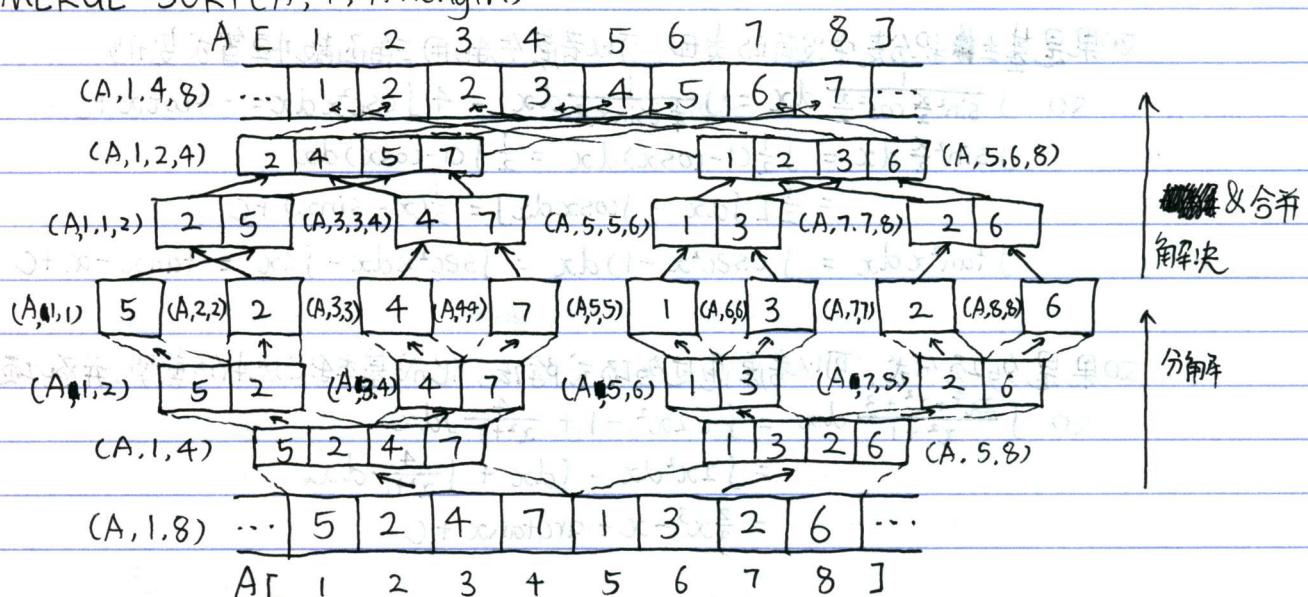
$A[p..q]$ 中有 $(r-p+1)/2$ 个元素

$A[q+1..r]$ 中有 $(r-p+1)/2$ 个元素

】合并已排序的 $A[p..q]$ 和 $A[q+1..r]$

调用

MERGE-SORT($A, 1, A.length$)



Algorithm - Pg

Study

递归方程

(或递归式), 用于当算法包含对其自身的递归调用时, 描述其运行时间

方程根据较小输入规模上的运行时间来描述规模为 n 的运行时间

通常对于一个规模足够小的问题, 直接求解只需要常量时间

即有常量 C , $\forall n (n \leq C \rightarrow T(n) = \Theta(1))$

通常这种情况对应于归纳法中的基础步骤

而对于规模足够大的问题, 可以把原问题分解为 a 个子问题, 每个问题是原问题规模的 $1/b$

注意, 在分解过程中, a, b 并不必然相等, 即存在 $a \neq b$ 的分治算法

又由每个规模为 n/b 的子问题是的运行时间为 $T(n/b)$,

而分解原问题是需要时间 $D(n)$, 合并子问题是的合并需要时间 $C(n)$

于是有对于规模 n 足够大的情况, $T(n) = aT(n/b) + D(n) + C(n)$

即有 $T(n) = \begin{cases} \Theta(1) & n \leq C \\ aT(n/b) + D(n) + C(n) & n > C \end{cases}$, 其中 C 为常量

如对于归并排序算法的分析, 且当 $n=1$ 时, $T(n) = \Theta(1)$, 则当 $n>1$ 时

分解: 分解步 马聚仅计算数组中间位置的下标, 需要常量时间, 即 $D(n) = \Theta(1)$

解决: 递归地求解两个规模为 $T(n/2)$ 的问题, 需要 $2T(n/2)$ 的时间

合并: 用 MERGE 过程合并两个合计 n 个元素的子数组, 需要 $\Theta(n)$ 的时间, 即 $C(n) = \Theta(n)$

于是有 $T(n) = \begin{cases} \Theta(1) & n=1 \\ 2T(n/2) + \Theta(n) & n>1 \end{cases}$, 其中 $D(n) + C(n) = \Theta(1) + \Theta(n) = \Theta(n)$

进一步地写为递归方程有 $T(n) = \begin{cases} C & n=1 \\ 2T(n/2) + cn & n>1 \end{cases}$

递归树

通过将树分解成由递归式所确定的其组成部分来扩展树中的每个信息, 直到问题规模下降到 1

对于归并排序, 可以假定规模 n 为 2 的幂, 即有 $n=2^k$, 其中 $k \in \mathbb{N}$

树的信点表示 $T(n) = cn \lg n + cn$

在当前层引起的代价 $T(n) = cn(c2^{k-1}) + cn(c2^{k-1}) \rightarrow cn(c2^k)$ 进一步有 $T(n) = \Theta(cn \lg n)$

代价 $T(n) = cn(c2^{k-1}) + cn(c2^{k-1}) \rightarrow cn(c2^k)$

树有 $\lg n + 1$ 层, 有 n 个叶信点 基础步聚: $n=2^0$ 时, 有 0+1 层, 1 个叶信点

递归步聚: 假设 $\forall i \in \mathbb{N} P(2^i)$ 成立

则对于 $P(2^{i+1})$, 可知树有 2^{i+1} 个叶信点 又树是二叉树, 所以比 2^i 的树多一层, 即层数为 $i+2$

$T(n) = cn(c2^k) + cn(c2^k) \rightarrow cn(c2^k)$

$= (i+1)+1$

Algorithm - P10

二分查找

BINARY-SEARCH (已排序的数组 A, 查找值 v)

```
i = 1  
j = A.length  
while i < j  
    m = L(i+j)/2  
    if v > A[m]  
        i = m+1  
    else  
        j = m  
    if v = A[i]  
        return i  
    else  
        return NIL
```

初始化上界与下界为 1 和 A.length
此处假设数组 A 是非空的
当 $i < j$, 即范围尚未缩小到 1 个元素
计算中间位置 $m = \lfloor (i+j)/2 \rfloor$
如果 v 的值大于中间位置的元素
则更新搜索下界 i 为 $m+1$
否则有 $v \leq A[m]$
则更新搜索上界 j 为 m
当 $i = j$ 时, 循环终止。
如果此时 $A[i] = v$
则返回下标 i
否则返回特殊值 NIL

查找值 $v > A[i-1] \wedge v < A[j+1]$
循环不变量: 对于下标 $1 \leq i \leq j \leq A.length$,

初始化: 在第 1 次迭代前, 可检查是否有 $v < A[1] \vee v > A[A.length]$, 为真则直接返回 NIL

否则对 $i=1, j=A.length$, 结论平凡地为真

保持: 假设在第 k 次迭代前, 对于当前的下标下界与上界 i, j , 有 $v > A[i-1] \wedge v < A[j+1]$

在第 k 次迭代中, 比较 v 与 $A[m]$ 的大小, 其中 $m = \lfloor (i+j)/2 \rfloor$

如果 $v > A[m]$, 则下界 $i = m+1$, $A[i-1] = A[m+1]-1 = A[m]$, $A[j+1]$ 不变

在第 $k+1$ 次迭代前 $v > A[i-1] = A[m+1]-1 = A[m]$, $A[j+1]$ 不变

如果 $v \leq A[m]$, 则上界 $j = m$, $v \leq A[m] < A[m+1] = A[j+1]$

在第 $k+1$ 次迭代前, $A[i-1]$ 不变,

停止: 当下标上界下界重合, 即 $i=j$ 时终止, 此时有 $v > A[i-1] \wedge v < A[j+1]$

即或者 $v = A[i]$, 即下标 i 为所在位置, 或者 $v \neq A[i]$, 则 v 不在数组 A 中

在数组 S 中查找一对元素, 使其和为给定值 x, S 中有 n 个元素

先对数组 S 应用归并排序, 需要时间为 $\Theta(n \lg n)$

ADD-SEARCH (已排序数组 S, 查找值 x). 注意这个过程也是 $\Theta(n \lg n)$ 的

```
for i = 1 to S.length-1  
    k = BINARY-SEARCH(S[i+1..], x - S[i])  
    if k != NIL  
        return (i, k)  
return NIL
```

对于每个 $S[i]$, 在子数组 $S[i+1..]$ 中查找 $x - S[i]$
如果返回值 $k \neq NIL$, 则有 $x = S[i] + S[k]$
未找到则返回 NIL

