第二章 算法基础

1. **插入排序**（第六章）

2. **分析算法**（概念）

1） 随机访问机（Random-Access Machine, RAM）：一种单处理器计算模型，没有并发操作；

2） RAM 模型指令：算术指令（加、减、乘、除、取余、向上/下取整）；数据移动指令（装入、存储、复制）；控制指令（条件/无条件转移、子程序调用与返回）；未列出指令（RAM 模型灰色区域，视情况而定）；

3） **输入规模**：依赖于研究问题；排序或计算离散傅里叶变换规模是输入的项数；两个整数相乘规模是表示数字的二进制位数；算法的输入是图则规模需要两个参数（顶点数和边数）；

4） **运行时间**：执行的基本操作数或步数，假定执行每行伪代码需要常量时间；给定输入规模，一个算法的运行时间也可能依赖于输入，如最佳情况和最坏情况；运行时间对给定的输入是固定的（随机算法除外）；

5） **最坏情况与平均情况**：我们往往只研究最坏情况运行时间；算法最坏情况运行时间给出任何输入的运行时间的上界（，最好情况的运行时间给出任何输入运行时间的下界）；平均情况一般与最坏情况相同；

6） **增长量级**：只对增长率或增长量级感兴趣，所以①只考虑公式中最重要的项，因为 n 很大时低阶项不重要；②忽略最重要项的系数，因为对大输入常量因子不如增长率重要；

3. **设计算法——分治法**（第四章）和**归并排序**（第六章）

第三章 函数的增长

1. 渐进记号

1) 渐进记号，如$\Theta(g(n))$，表示一个集合,函数$f(n) \in \Theta(g(n))$，通常记为$f(n) = \Theta(g(n))$

①渐近紧确界记号： Θ（big-theta）

对所有 $n \geq n_0$时，函数$f(n)$乘一个常量因子可等于$g(n)$，我们称$g(n)$是$f(n)$的一个**渐近紧确界**（asymptotically tight bound）；$\Theta(g(n))$的定义要求每个成员$f(n) \in \Theta(g(n))$均**渐近非负**，即当n足够大时，$f(n)$非负;**渐近正函数**就是对所有足够大的 n 均为正的函数;

方式一：设$f(n)$和$g(n)$是定义域为自然数集合的函数。如果$\lim_{n\to\infty}\frac{f(n)}{g(n)}$存在，并且等于某个常数$c(c>0)$，那么$f(n)=\Theta(g(n))$。通俗理解为$f(n)$和$g(n)$同阶，Θ用来表示算法的精确阶。

方式二：$\Theta(g(n))= f(n)$:存在正常量 c_1、c_2和n_0，使得对所有$n \geq n_0$，有$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ 若存在正常量c_1、c_2，使得对于足够大的n，函数$f(n)$能“夹入”$c_1 g(n)$与$c_2 g(n)$之间，则$f(n)$属于集合$\Theta(g(n))$，记作$f(n) \in \Theta(g(n))$。作为代替，我们通常记“$f(n) = \Theta(g(n))$”。

②渐近上界记号：O(big-oh)

根据符号 O 的定义，用它评估算法的复杂度得到的只是问题规模充分大时的一个上界;这个上界的阶越低，评估越精确，越有价值;**渐近上界**包括**渐进紧确的上界**和**非渐进紧确的上界**；

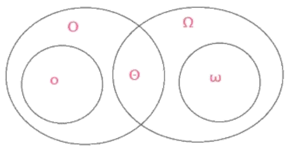
定义：设$f(n)$和$g(n)$是定义域为自然数集N上的函数。若存在正数c和n_0，使得对一切$n \geq n_0$都有$0 \leq f(n) \leq cg(n)$成立，则称$f(n)$的渐进的上界是$g(n)$，记作$f(n) = O(g(n))$。通俗的说n满足一定条件范围内，函数$f(n)$的阶不高于函数$g(n)$。

$O(1) < O(\log(n)) < O(n) < O(n \log n) < O(n^2) < O(2^n) < O(n!) < O(n^n)$
需要注意的是：对数函数在没有底数时，默认底数为2；如$\lg n = \log n = \log_2 n$因为计算机中很多程序是用二分法实现的。

③渐近下界记号：Ω(big-omege)

当我们说**运行时间为 $O(n^2)$**时，指存在一个属于 $O(n^2)$的函数$f(n)$，使得对 n 的任意值（不论什么规模，哪一种输入），其运行时间的上界都是$f(n)$；插入排序的**运行时间介于$\Omega(n)$和 $O(n^2)$**；插入排序的**最坏情况运行时间为$\Theta(n^2)$或$\Omega(n^2)$**；

定义：设$f(n)$和$g(n)$是定义域为自然数集N上的函数。若存在正数c和n_0，使得对一切$n \geq n_0$都有$0 \leq cg(n) \leq f(n)$成立，则称$f(n)$的渐进的下界是$g(n)$，记作$f(n) = \Omega(g(n))$。通俗的说n满足一定条件范围内，函数$f(n)$的阶不低于函数$g(n)$。



④非渐近紧确上界：o(小-oh)

$2n^2 = O(n^2)$是渐近紧确的，而 $2n = O(n^2)$是非紧确上界;

定义1：设 $f(n)$ 和 $g(n)$ 是定义域为自然数集N上的函数。若对于任意正数c，都存在n_0，使得对一切$n \geq n_0$都有$0 \leq f(n) < cg(n)$成立，则称$f(n)$的渐进的非紧确上界是$g(n)$，记作$f(n) = o(g(n))$。通俗的说n满足一定条件范围内，函数$f(n)$的阶低于函数$g(n)$。

定义2：设$f(n)$和$g(n)$是定义域为自然数集合的函数。如果$\lim_{n\to\infty}\frac{f(n)}{g(n)} = 0$，那么$f(n) = o(g(n))$。通俗理解为$f(n)$低于$g(n)$的阶。

⑤非渐近紧确下界：ω(小-omege)

定义1：设$f(n)$和$g(n)$是定义域为自然数集N上的函数。若对于任意正数c，都存在n_0，使得对一切$n \geq n_0$都有$0 \leq cg(n) < f(n)$成立，则称$f(n)$的渐进的非紧确下界是$g(n)$，记作$f(n) = \omega(g(n))$。通俗的说n满足一定条件范围内，函数$f(n)$的阶高于函数$g(n)$。

定义2：设$f(n)$和$g(n)$是定义域为自然数集合的函数。如果$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \infty$，那么$f(n) = o(g(n))$。通俗理解为$f(n)$高于$g(n)$的阶。

2) 等式和不等式中的渐进记号

渐进记号在公式中表示不关注名称的**匿名函数**；渐进记号出现在等式左边表示**任意**，出现在等式右边表**存在**；对于任意等号左边的匿名函数，总存在一个等号右边的匿名函数使得等式成立；

3) 各种性质

①**传递性**；②**自反性**；③**对称性**；④**转置对称性**；⑤**三分性**（不是所有函数都可渐进比较）

2. 标准记号与常用函数

第四章 分治策略

0. 概述

1）**层递归步骤**：①**分解 Divide**（将问题划分为一些子问题）；②**解决 Conquer**（递归求解子问题）；③**合并 Combine**（将子问题的解组合成原问题的解）

2）**子问题种类**：①递归情况（Recursive Case）：子问题足够大需要递归求解时；②基本情况（Base Case）：子问题小到直接求解；③合并步骤：除与原问题形式相同规模更小的子问题外，需要求解的与原问题不同的子问题，这类子问题求解视作合并的一部分；

3）**递归式（Recurrence）**：一个等式或不等式，通过更小输入上的函数值来描述一个函数；通常忽略边界条件（向上/下取整）；

4）**求解递归式Θ或 O 渐进界的方法**：①代入法；②递归树法；③主方法；

5）**不等式递归式**：如$T(n) \le 2T(n) + \Theta(n)$，仅描述了$T(n)$的上界，因此可以用大 O 符号来表示解；$T(n) \le 2T(n) + \Theta(n)$，仅描述了$T(n)$的下界，因此可以用大Ω符号来表示解；

1. 最大子数组问题

1） **问题描述**：某时刻可以买入一支股票并在之后某时刻卖出，目标时最大化收益；

2） **暴力解法**：尝试每对买进卖出日期组合，只要卖出在买入之后；则 n 天共有$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \Theta(n^2)$种日期组合；处理每对日期的开销为常量，因此运行时间为$\Omega(n^2)$；

3） **问题变换**：我们的目的是寻找一段日期，使得第一天到最后一天的价格差最大；我们将输入数据从每日价格变为每日价格变化；第 i 天的价格变化定义为第 i 天和第 i-1 天的价格差，称为数组 A；此时问题转换为寻找 A 的和最大的非空连续子数组，称为最大子数组（Maximum Subarray）；只求一个最大子数组；只有当数组中包含负数时，最大子数组问题才有意义；

4） **分治法**： A$[low..high]$的任何连续子数组 A[i .. j]所处的情况是以下三种情况之一；A$[low..high]$的最大子数组必然是 A$[low..mid]$、A$[mid+1..high]$、跨越中点所有子数组中和最大者；A$[low..mid]$ 和 A$[mid+1..high]$仍是最大子数组问题，可以递归求解；跨越中点所有子数组并非原问题更小规模实例，因为加入了子数组必须跨越中点的限制，我们需要找出 A$[i..mid]$和 A$[mid+1..j]$然后将其合并；如果 A$[low..high]$包含 n 个元素，则调用花费$\Theta(n)$时间，for 循环每次迭代$\Theta(1)$，3-7 行$mid - low + 1$ 次，10-14 行$high - mid$次，共计$high - low$+1 次；递归过程中，6 行子问题并非原问题更小规模实例，因此视为合并部分。

完全位于子数组 $A[low..mid]$中，因此 $low \leqslant i \leqslant j \leqslant mid$。
完全位于子数组 $A[mid+1..high]$中，因此 $mid < i \leqslant j \leqslant high$。
跨越了中点，因此 $low \leqslant i \leqslant mid < j \leqslant high$。

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
1   left-sum = -∞
2   sum = 0
3   for i = mid downto low
4       sum = sum + A[i]
5       if sum > left-sum
6           left-sum = sum
7           max-left = i
8   right-sum = -∞
9   sum = 0
10  for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)
```

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
1   if high == low
2       return (low, high, A[low])          // base case: only one element
3   else mid = ⌊(low + high)/2⌋
4       (left-low, left-high, left-sum) =
            FIND-MAXIMUM-SUBARRAY(A, low, mid)
5       (right-low, right-high, right-sum) =
            FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
6       (cross-low, cross-high, cross-sum) =
            FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7       if left-sum ≥ right-sum and left-sum ≥ cross-sum
8           return (left-low, left-high, left-sum)
9       elseif rightr-sum ≥ left-sum and right-sum ≥ cross-sum
10          return (right-low, right-high, right-sum)
11      else return (cross-low, cross-high, cross-sum)
```

合并

5） **算法分析**：假设原问题规模为 2 的幂，这样所有子问题规模均为整数；用$T(n)$表示 FIND-MAXIMUM-SUBARRAY 求解 n 个元素的最大子数组的运行时间；n=1 时$T(1) = \Theta(1)$；当 n>1 时为递归情况，第 1、3 行为常量时间，第 4、5 行为规模更小实例用时$T(n/2)$，第 6 行调用 FIND-MAX-CROSSING-SUBARRAY 花费$\Theta(n)$时间，第 7-11 行仅花费$\Theta(1)$时间，因此有$T(n) = \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) = 2T(n/2) + \Theta(n)$；最大子数组问题实际存在一个线性时间算法。

2. 归并排序

1） **问题描述**：见第 6 章

2） **分治法**：归并算法的关键是合并，调用 MERGE(A,p,q,r)来完成合并；其中 A 是一个数组 p、q、r 是数组下标；认为 A[p..q]和 A[q+1..r]都已经排好序；需要$\Theta(n)$时间；每个堆底部放一张哨兵牌；

```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1..n1 + 1] and R[1..n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p+r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q+1, r)
    MERGE(A, p, q, r)
```

3. 矩阵乘法的 Strassen 算法

4. 用代入法求解递归式

5. 用递归树方法求解递归式

6. 用主方法求解递归式

1）主定理：

定理 4.1（主定理） 令 $a \geqslant 1$ 和 $b > 1$ 是常数，$f(n)$是一个函数，$T(n)$是定义在非负整数上的递归式：

$$T(n) = aT(n/b) + f(n)$$

其中我们将 n/b 解释为$\lfloor n/b \rfloor$或$\lceil n/b \rceil$。那么 $T(n)$有如下渐近界：

1. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = O(n^{\log_b a - \varepsilon})$，则 $T(n) = \Theta(n^{\log_b a})$。
2. 若 $f(n) = \Theta(n^{\log_b a})$，则 $T(n) = \Theta(n^{\log_b a} \lg n)$。
3. 若对某个常数 $\varepsilon > 0$ 有 $f(n) = \Omega(n^{\log_b a + \varepsilon})$，且对某个常数 $c < 1$ 和所有足够大的 n 有 $af(n/b) \leqslant cf(n)$，则 $T(n) = \Theta(f(n))$。 ■

2）主方法：①$T(n) = 9T(n/3) + n \Rightarrow T(n) = \Theta(n^2)$；②$T(n) = T(2n/3) + 1 \Rightarrow T(n) = \Theta(\lg n)$其中 b=3/2；③$T(n) = 3T(2n/4) + n\lg n \Rightarrow T(n) = \Theta(n\lg n)$由于 $f(n) = \Omega(n^{\log_4 3+\varepsilon})$，其中ε约等于 0.2；④$T(n) = 2T(n/2) + n\lg n \Rightarrow$ None，其中ε不存在；

7. 证明主定理