计算机问题求解 – 论题2-3
- 分治法与递归

2019年03月11日

Part I
Divide-and-conquer

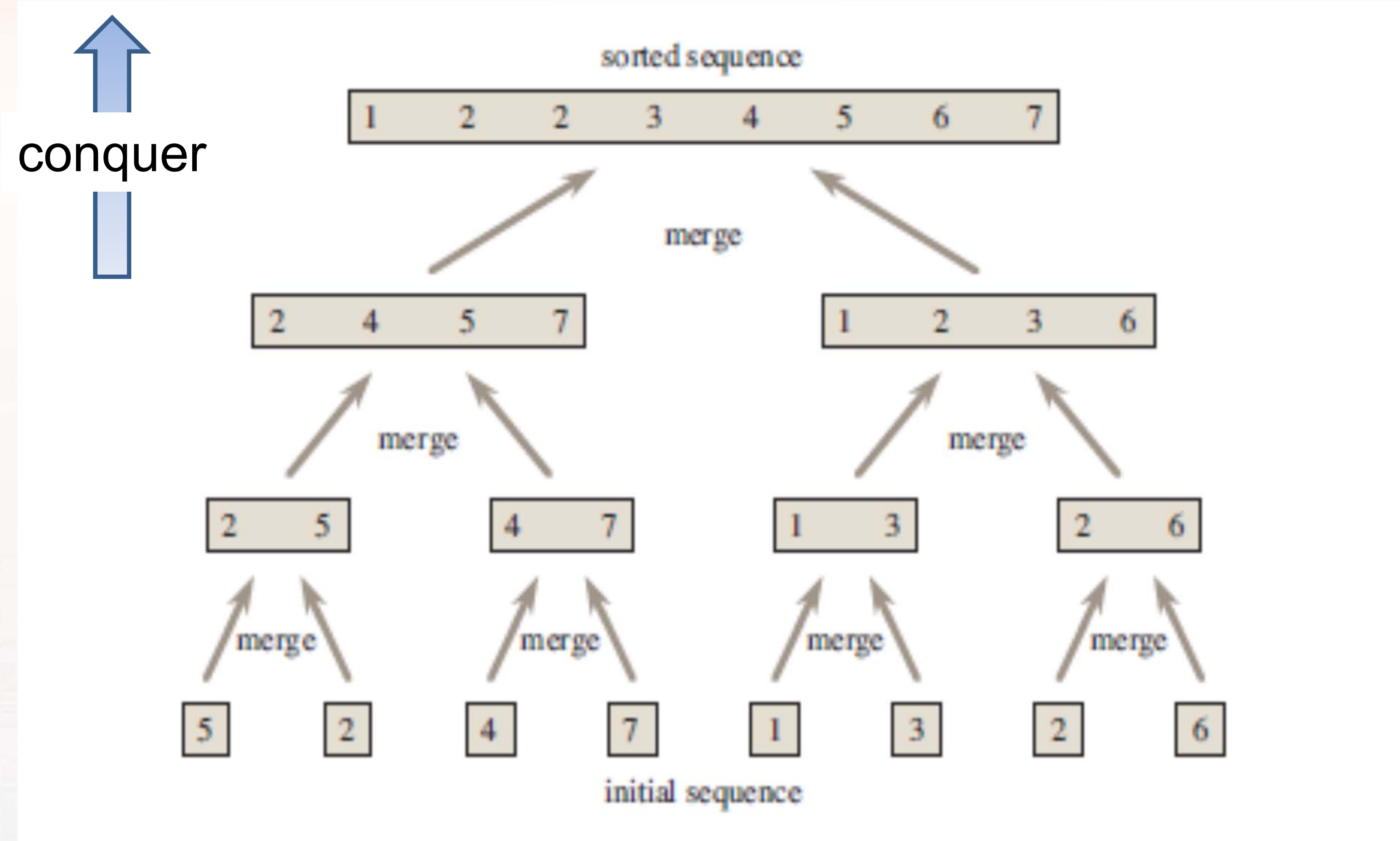
问题1：“Divide-and-conquer”，这是什么？策略，技术，算法？

Mergesort Revisited

```
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)
```

问题2:

这个算法究竟是如何“排”序的？


conquer
sorted sequence
1 2 2 3 4 5 6 7
merge
2 4 5 7
1 2 3 6
merge
merge
2 5
4 7
1 3
2 6
merge
merge
merge
merge
5
2
4
7
1
3
2
6
initial sequence

问题3：

人的思维“分而治之”如何变为算法策略的呢？

递归在这里起了什么作用？

问题4：

如何考虑分治算法的代价？

递归代价与非递归代价

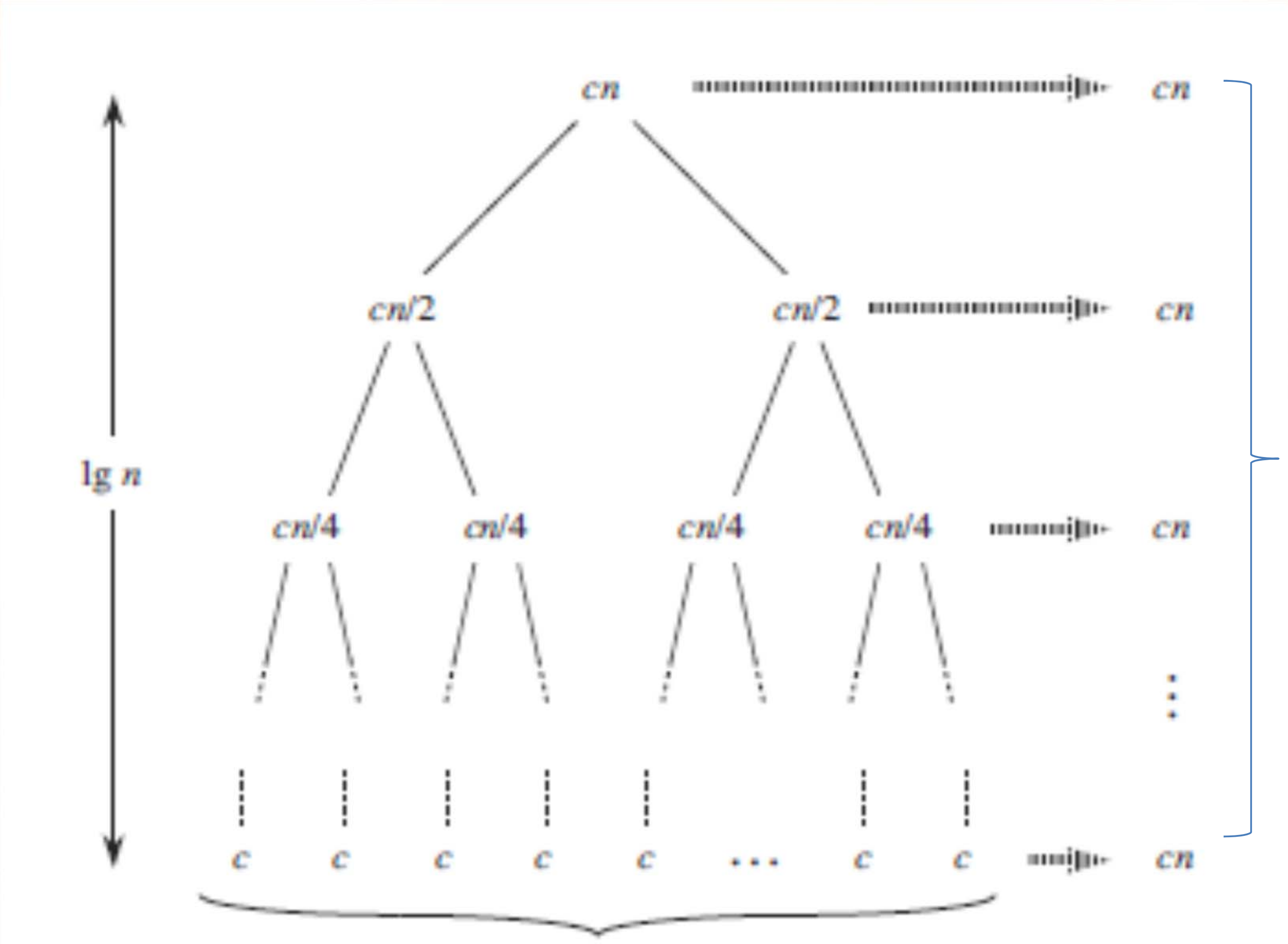
导出递归式

```
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)
```

两次递归，理想情况下每次问题规模是原来的一半。

非递归开销

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$



n

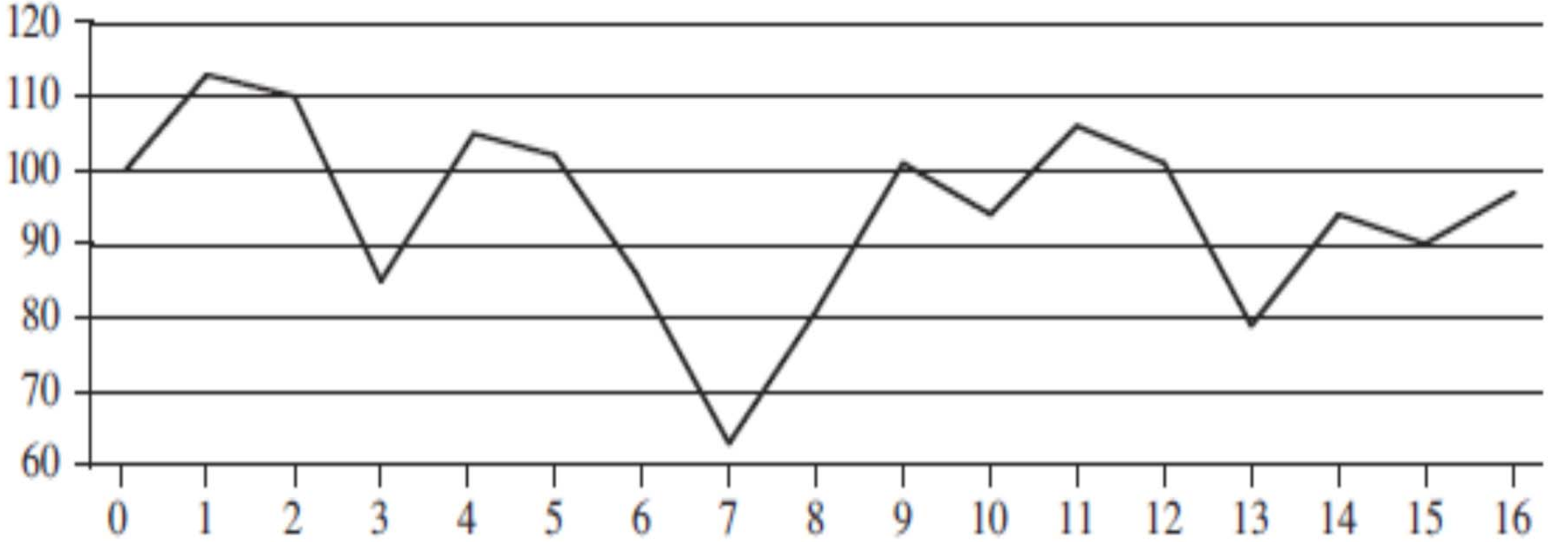
cn* log *n

确实比插入排序效率高。

这里似乎假设*n*是2的整次幂，在我们涉及的大多数情况下，这不影响结果。

问题5：

书上的投资回报问题是怎样被转化为最大子数组问题的？



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

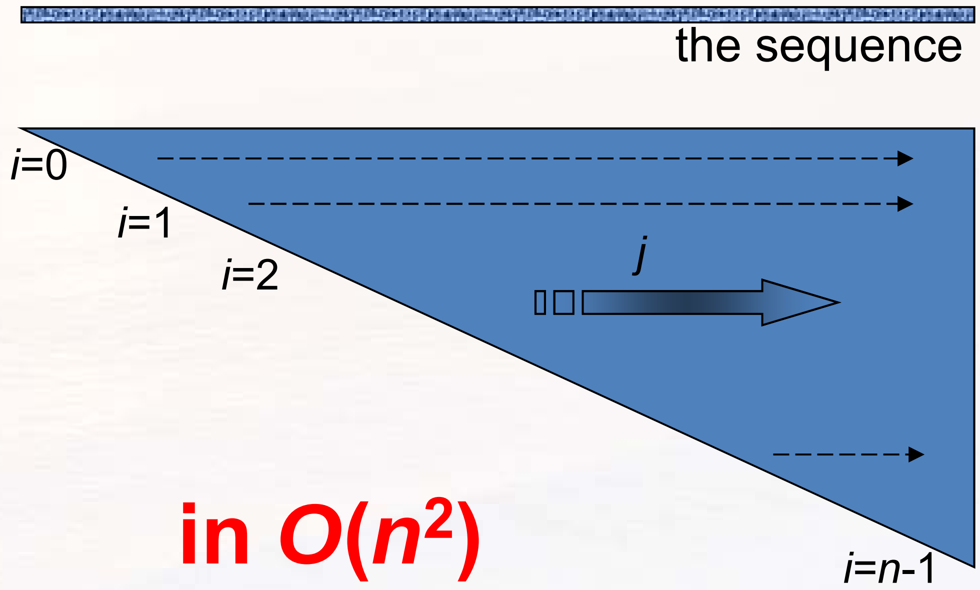
Maximum subarray

简单的遍历所有可能的子序列

下面的过程遍历的顺序为：

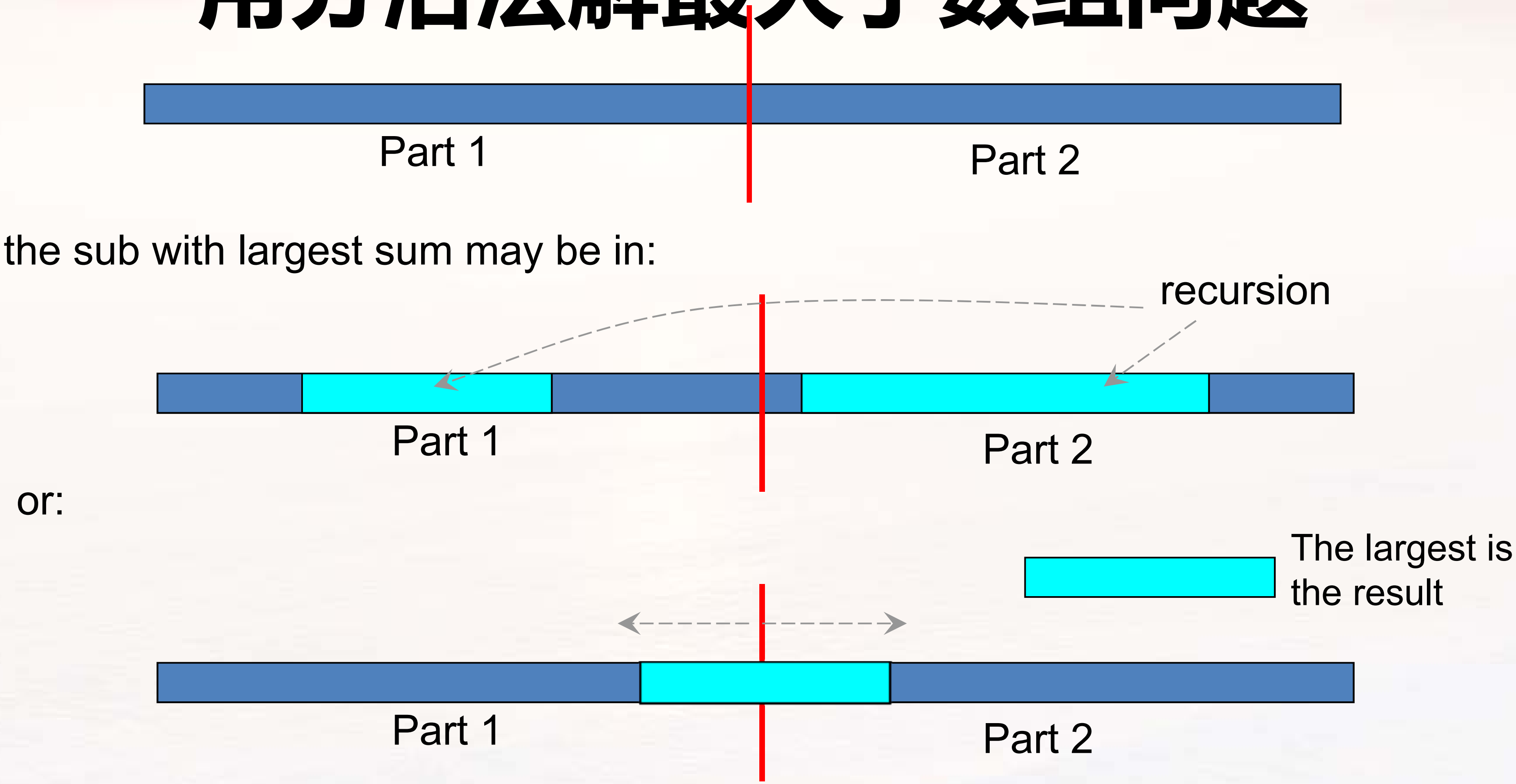
(0,0), (0,1), …, (0,*n*-1); (1,1), (1,2), …, (1,*n*-1), ……
(*n*-2,*n*-2), (*n*-2, *n*-1), (*n*-1,*n*-1)

```
MaxSum = 0;
 for (i = 0; i < N; i++)
 {
   ThisSum = 0;
   for (j = i; j < N; j++)
   {
     ThisSum += A[j];
     if (ThisSum > MaxSum)
       MaxSum = ThisSum;
   }
 }
 return MaxSum;
```



in $O(n^2)$

用分治法解最大子数组问题



问题5：跨中点的部分如何计算？

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

Part I

Part II

Part III

问题6：

为什么这个算法代价是线性的？

顺便问一句，三个子问题有什么不同？

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
  if high == low
      return (low, high, A[low])          // base case: only one element
  else mid = ⌊(low + high)/2⌋
      (left-low, left-high, left-sum) =
          FIND-MAXIMUM-SUBARRAY(A, low, mid)
      (right-low, right-high, right-sum) =
          FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
      (cross-low, cross-high, cross-sum) =
          FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
      if left-sum ≥ right-sum and left-sum ≥ cross-sum
          return (left-low, left-high, left-sum)
      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
          return (right-low, right-high, right-sum)
      else return (cross-low, cross-high, cross-sum)
```

非递归代价：常量

递归，理想状况下问题规模是原来的一半。

非递归代价：线性

非递归代价：常量

$\boldsymbol{O(n\log n)}$

Part II
对效率的追求

问题 7：

你觉得求最大子串的算法还能改进吗？

线性代价！

线性算法

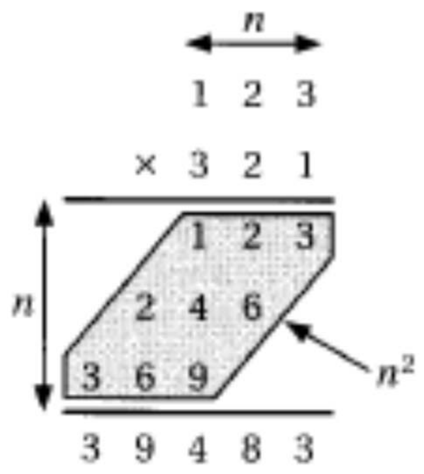
```
ThisSum = MaxSum = 0;
  for (j = 0; j < N; j++)
  {
    ThisSum += A[j];
    if (ThisSum > MaxSum)
      MaxSum = ThisSum;
    else if (ThisSum < 0)
      ThisSum = 0;
  }
  return MaxSum;
```

This is an example of
“online algorithm”

你能解释一下吗?

in $O(n)$

假如我们将“1位数乘”作为基本“关键”操作



你是否会认为两个n位整数相乘是平分复杂度的呢？

Divide-and-Conquer:

$$x=10^{n/2}a+b \quad \text{and} \quad y=10^{n/2}c+d\,.$$

$$xy=10^{n}ac+10^{n/2}(ad+bc)+bd\,.$$

$$(a+b)(c+d)-ac-bd=ad+bc\,.$$

先估计，再验证

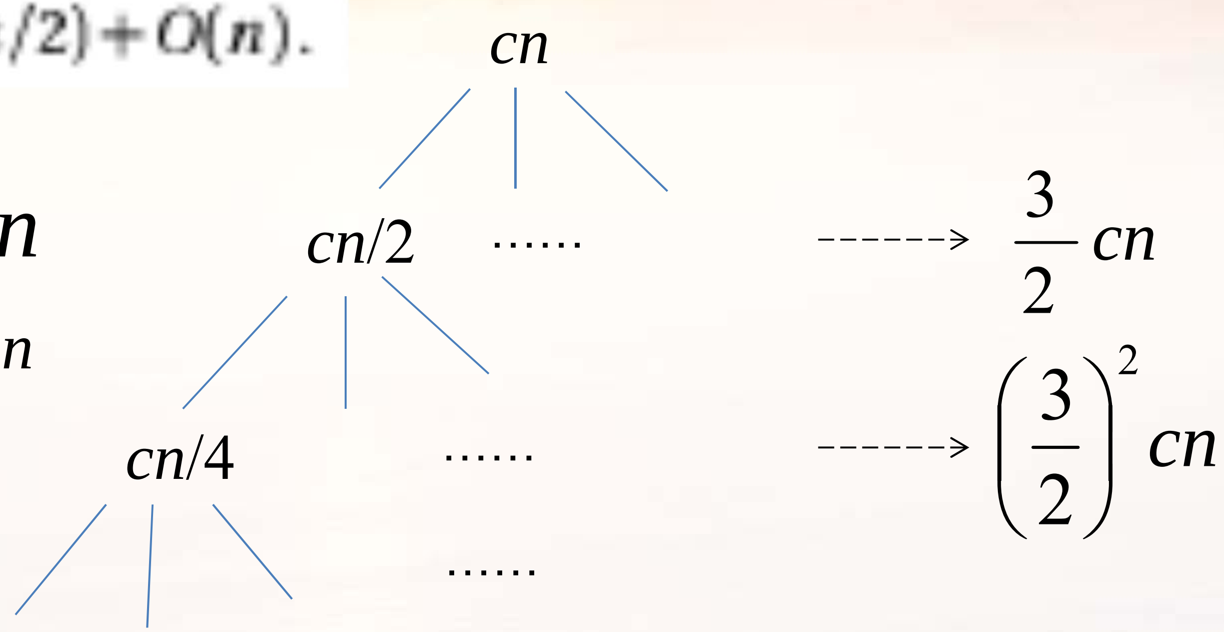
$$T(n)=3T(n/2)+O(n)\,.$$

$$T(n)=\Theta(n^{\alpha}) \text{ where } \alpha=\log_2 3\approx 1.585\,.$$

$T(n)=3T(n/2)+O(n).$

树的层数：$\log_2 n$

叶结点数：$3^{\log_2 n}$



$n^{\log_2 3}$

$$T(n)=3T\left(\frac{n}{2}\right)+O(n)\le 3\left(\frac{n}{2}\right)^{\log_2 3}+O(n)$$

$$=n^{\log_2 3}+cn$$

$$\boxed{T(n)\in O(n^{\log_2 3})}$$

矩阵乘法：似乎非得 $\Omega(n^3)$

If $A = (a_{ij})$ and $B = (b_{ij})$ are square $n \times n$ matrices, then in the product $C = A \cdot B$, we define the entry c_{ij}, for $i, j = 1, 2, \ldots, n$, by

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

Suppose that we partition each of A, B, and C into four $n/2 \times n/2$ matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}.$$

Equation (4.10) corresponds to the four equations

$$\left.\begin{aligned} C_{11} &= A_{11} \cdot B_{11} + A_{12} \cdot B_{21}, \\ C_{12} &= A_{11} \cdot B_{12} + A_{12} \cdot B_{22}, \\ C_{21} &= A_{21} \cdot B_{11} + A_{22} \cdot B_{21}, \\ C_{22} &= A_{21} \cdot B_{12} + A_{22} \cdot B_{22}. \end{aligned}\right\}$$

1个n阶方阵相乘的问题可以分解为8个$n/2$阶方阵相乘的子问题。

仍然是立方复杂度

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1\,. \end{cases}$$

问题8：

在讨论上述算法的代价时，有些量被包含在其它表示中，不单独计了，有些却不能。你能举出其中不同的地方吗？是为什么呢？

问题9：

你能否描述Strassen方法的基本思想？

The key to Strassen's method is to make the recursion tree slightly less bushy.

复杂的组合为了减少一次乘法

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 \\
P_2 &= S_2 \cdot B_{22} \\
P_3 &= S_3 \cdot B_{11} \\
P_4 &= A_{22} \cdot S_4 \\
P_5 &= S_5 \cdot S_6 \\
P_6 &= S_7 \cdot S_8 \\
P_7 &= S_9 \cdot S_{10}
\end{aligned}
$$

$$
\boxed{
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
}
$$

诸S_i只需通过加减法计算

这个算法曾经引起轰动

Strassen's algorithm runs in $O(n^{2.81})$ time, which is asymptotically better than the simple SQUARE-MATRIX-MULTIPLY procedure.

Strassen's method is not at all obvious. (This might be the biggest understatement in this book.)

问题10：

你对于这个结果是否有感性认识？

问题11：

为什么降低子问题个数会导致复杂度的阶下降？

问题12:

在这里的几个用分治法的例子中，算法复杂度的阶均比用“brute-force”的方法减低了，究竟是什么原因呢？

课外作业

- TC p75-: ex.4.1-5;
- TC p.87-: 4.3-3, 4.3-7;
- TC p.92-: 4.4-2, 4.4-8;
- TC p.107-: 4.1, 4.2, 4.4