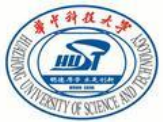
数据结构与算法设计

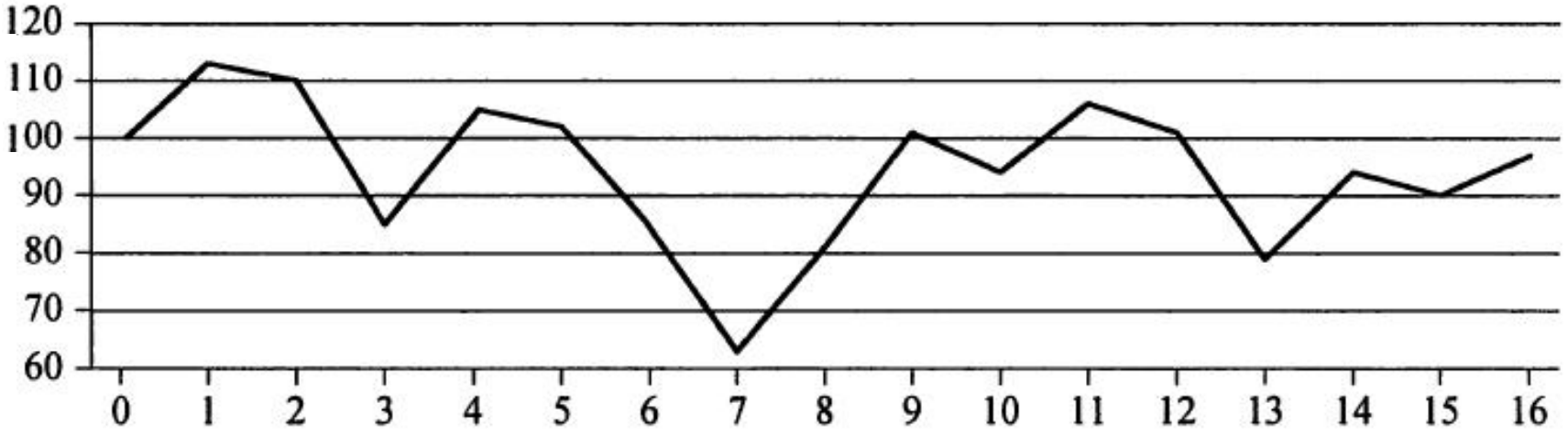


4.1 最大子数组

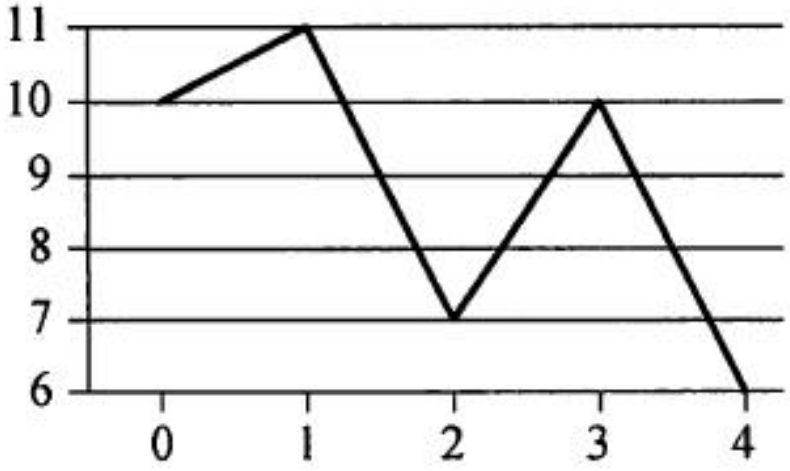
4.2 Strassen算法

The maximum-subarray problem

- A story of the stock.



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



天	0	1	2	3	4
价格	10	11	7	10	6
变化		1	-4	3	-4

Q:

（1）最大收益一定是最低价格买进，最高价格卖出?

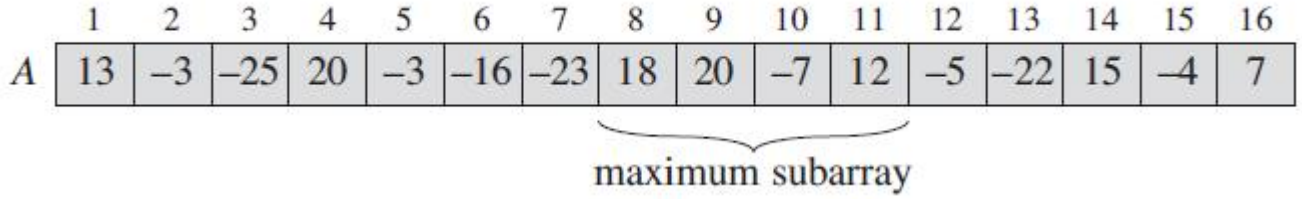
（2）上图中的最大收益是多少?

The maximum-subarray problem

- the ***maximum subarray problem:***

  for an array A, we want to find the nonempty, contiguous subarray of A whose values have the largest sum.

  ——We call this contiguous subarray the ***maximum subarray***.



How to solve the maximum subarray problem?

A brute-force solution

- **Stock case:** **Simply try each pair of possible date combinations**

（buying date, selling date) & (buying date < selling date）

⇩

Whole combinations = $\binom{n}{2}$ = n(n-1)/2 = Θ(n²)

⇩

Running time : Ω(n²)

⇧

- **Largest sub-array:** **$\binom{n}{2}$ + n = n(n+1)/2 = Θ(n²)**

How to solve the maximum subarray problem?

- A brute-force solution

```
Public static int MaxSubsequenceSum (int[] a) {
    int maxSum = 0;
    for (int i=0; i<a. length; i++) {            // i为子序列的左边界
        for (int j=i; j<a.length; j++) {         // j为子序列的右边界
            int thisSum = 0;
            for (int k=i; k<=j; k++) {
                thisSum += a[k] ;
                if  (thisSum > maxSum)
                maxSum = thisSum;
            }
        }
    }
    return maxSum;
}
```

Running time : $O(n^3)$

How to solve the maximum subarray problem?

- A brute-force solution

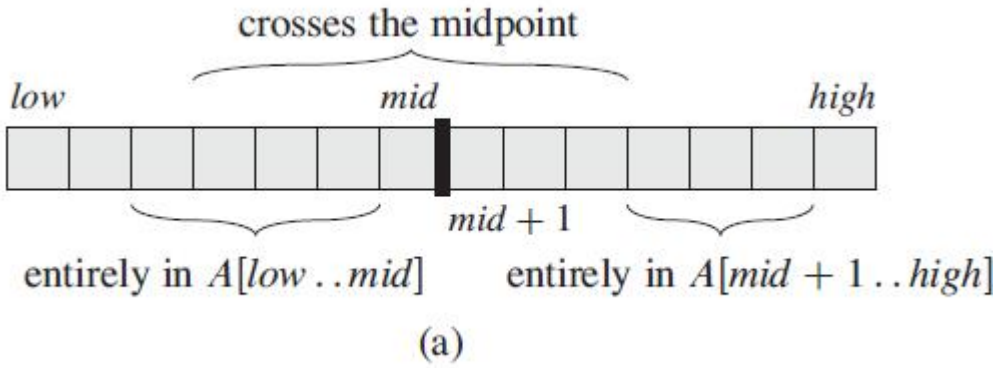
Algorithm 1 Brute Force Algorithm to Solve Maximum Subarray Problem

```
left = 1
right = 1
max = A[1]
curSum = 0
for i = 1 to n do // Increment left end of subarray
    curSum = 0
    for j = i to n do // Increment right end of subarray
        curSum = curSum + A[j]
        if curSum > max then
            max = curSum
            left = i
            right = j
        end if
    end for
end for
```

Running time : $O(n^2)$

How to solve the maximum subarray problem?

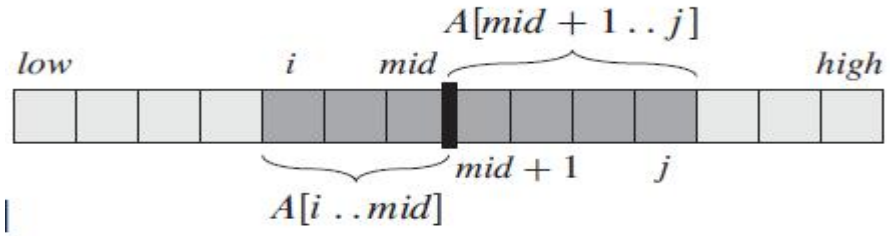
- A brute-force solution
- A solution using divide-and-conquer



(a)

- subarray A[*i*..*j*] of A[*low* ..*high]* must lie in exactly one of the following
  - entirely in the subarray $A[low\,..\,mid]$, so that $low \le i \le j \le mid$,
  - entirely in the subarray $A[mid + 1\,..\,high]$, so that $mid < i \le j \le high$, or
  - crossing the midpoint, so that $low \le i \le mid < j \le high$.

- a maximum subarray of A[*low* .. *high*] must have the greatest sum over all subarrays entirely in A[*low* .. *mid*], entirely in A[*mid* .. *high*], or crossing the midpoint.



(b)

```
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)
```

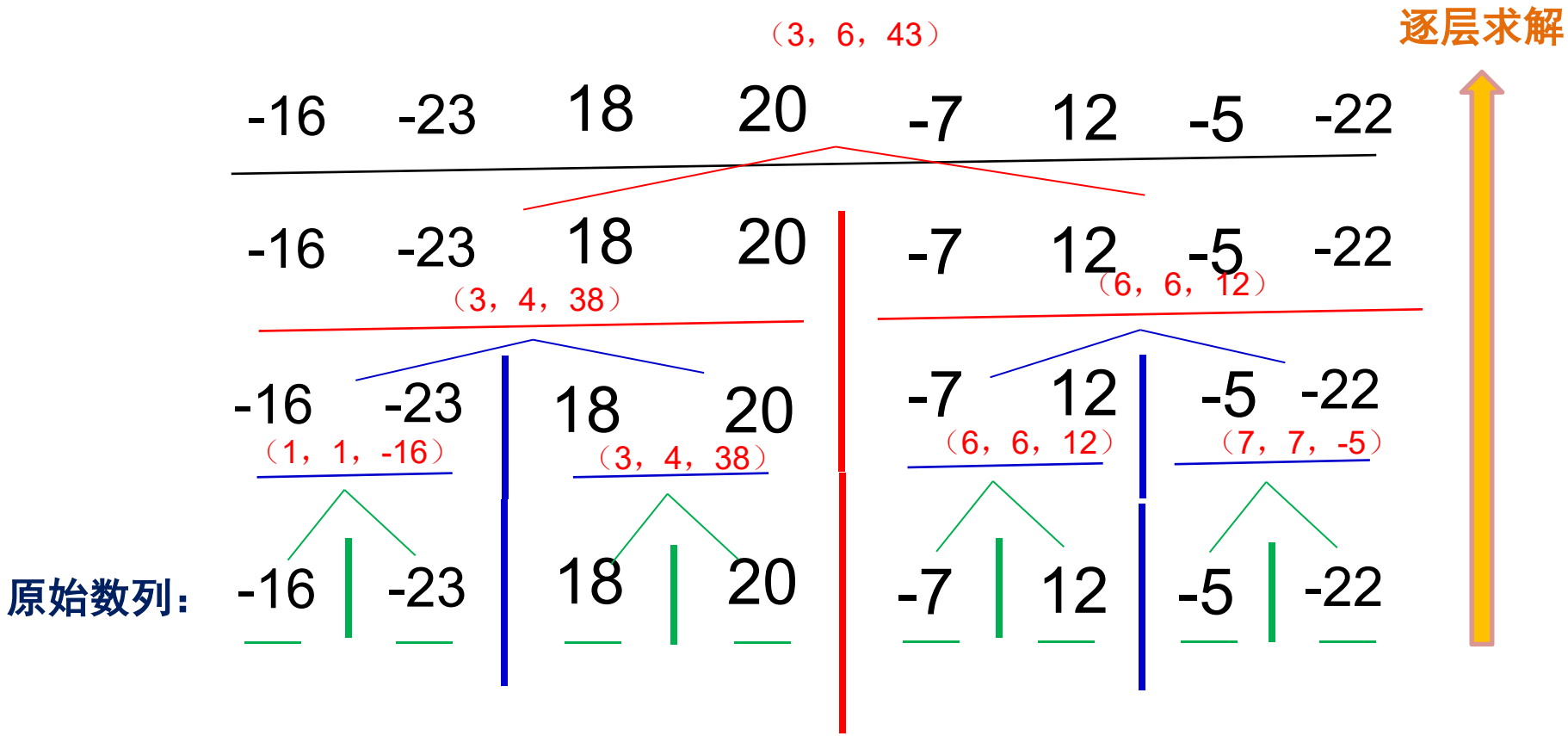
- FIND-MAX-CROSSING-SUBARRAY(A, *low, mid, high*) takes $\Theta(n)$ time.

```
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 right-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)
```

- the running time $T(n)$ of FIND-MAXIMUM-SUBARRAY

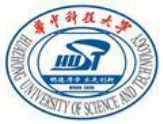
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases}$$

原始数列A=（-16, -23, 18, 20, -7, 12, -5, -22）。



思考：

当A的所有元素均为负数时，FIND-MAXIMUM-SUBARRAY返回什么？



4.1 最大子数组

4.2 Strassen算法

matrix multiplication

Input: $A = [a_{ij}], B = [b_{ij}].$
Output: $C = [c_{ij}] = A \cdot B.$

$\Big\}\ i, j = 1, 2, \ldots, n.$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

matrix multiplication

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

Standard algorithm

for $i \leftarrow 1$ **to** n

  do for $j \leftarrow 1$ **to** n

    do $c_{ij} \leftarrow 0$

      for $k \leftarrow 1$ **to** n

        do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$

Running time $= \Theta(n^3)$

Divide-and-conquer algorithm

IDEA:

$n \times n$ matrix $= 2 \times 2$ matrix of $(n/2) \times (n/2)$ submatrices:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$$\left.\begin{aligned} r &= ae + bg \\ s &= af + bh \\ t &= ce + dg \\ u &= cf + dh \end{aligned}\right\} \begin{array}{l} 8 \text{ mults of } (n/2) \times (n/2) \text{ submatrices} \\ 4 \text{ adds of } (n/2) \times (n/2) \text{ submatrices} \end{array}$$

1. Master Theorem

主定理适用于求解如下递归式算法的时间复杂度：

$$T(n) = aT(\frac{n}{b}) + f(n)$$

其中：

- n 是问题规模大小；
- a 是原问题的子问题个数；
- n/b 是每个子问题的大小，这里假设每个子问题有相同的规模大小；
- $f(n)$ 是将原问题分解成子问题和将子问题的解合并成原问题的解的时间。

对上面的式子进行分析，得到三种情况：

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a} \log^k n)$ with[1] $k \geq 0$, then $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ with $\epsilon > 0$, and $f(n)$ satisfies the regularity condition, then $T(n) = \Theta(f(n))$.
   Regularity condition: $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

Analysis of D&C algorithm

$$T(n) = 8\,T(n/2) + \Theta(n^2)$$

 # submatrices

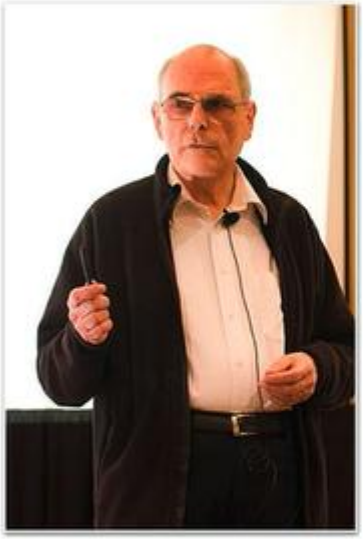
submatrix size

work adding submatrices

$$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow \text{CASE 1} \Rightarrow T(n) = \Theta(n^3).$$

No better than the ordinary algorithm.

Volker Strassen



1936，德国

- 开创性贡献
  - 随机游走，国际数学家大会
  - 矩阵乘法
  - 大整数乘法 $O(n \log n \log \log n)$
  - 素数测试
- 矩阵乘法的理论研究
  - 下界$\Omega(n^2)$?
  - 从3到2.x
  - 理论界持续改进
- 如何理解Strassen's Algo.
  - 研究课题

Strassen's idea 1969

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = C$$

- Multiply 2×2 matrices with only 7 recursive mults.

$$P_1 = a \cdot (f - h)$$
$$P_2 = (a + b) \cdot h$$
$$P_3 = (c + d) \cdot e$$
$$P_4 = d \cdot (g - e)$$
$$P_5 = (a + d) \cdot (e + h)$$
$$P_6 = (b - d) \cdot (g + h)$$
$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$
$$s = P_1 + P_2$$
$$t = P_3 + P_4$$
$$u = P_5 + P_1 - P_3 - P_7$$

7 mults, 18 adds/subs.
Note: No reliance on commutativity of mult!

Strassen's idea

$$
\begin{aligned}
r &= ae + bg \\
s &= af + bh \\
t &= ce + dg \\
u &= cf + dh
\end{aligned}
$$

- Multiply 2×2 matrices with only 7 recursive mults.

$$
\begin{aligned}
P_1 &= a \cdot (f - h) \\
P_2 &= (a + b) \cdot h \\
P_3 &= (c + d) \cdot e \\
P_4 &= d \cdot (g - e) \\
P_5 &= (a + d) \cdot (e + h) \\
P_6 &= (b - d) \cdot (g + h) \\
P_7 &= (a - c) \cdot (e + f)
\end{aligned}
$$

$$
\begin{aligned}
r &= P_5 + P_4 - P_2 + P_6 \\
&= (a + d)(e + h) \\
&\quad + d(g - e) - (a + b)h \\
&\quad + (b - d)(g + h) \\
&= ae + ah + de + dh \\
&\quad + dg - de - ah - bh \\
&\quad + bg + bh - dg - dh \\
&= ae + bg
\end{aligned}
$$

Strassen's algorithm

(1) 矩阵分解，包括输入、输出； $O(1)$

(2) 创建10个n/2×n/2的矩阵，$S_1, S_2, \ldots, S_{10}$ $\Theta(n^2)$

(3) 用(1)(2)生成的矩阵，计算7个矩阵积 $7T(n/2)$

$P_1, P_2, \ldots, P_7$

(4) $P_1 \sim P_7$ ⟹ $C_{11}, C_{12}, C_{21}, C_{22}$ $\Theta(n^2)$

r,s,t,u

$T(n)=7T(n/2)+\Theta(n^2)$

Strassen's algorithm

1. ***Divide:*** Partition A and B into $(n/2)\times(n/2)$ submatrices. Form terms to be multiplied using $+$ and $-$.
2. ***Conquer:*** Perform 7 multiplications of $(n/2)\times(n/2)$ submatrices recursively.
3. ***Combine:*** Form C using $+$ and $-$ on $(n/2)\times(n/2)$ submatrices.

$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

Analysis of Strassen

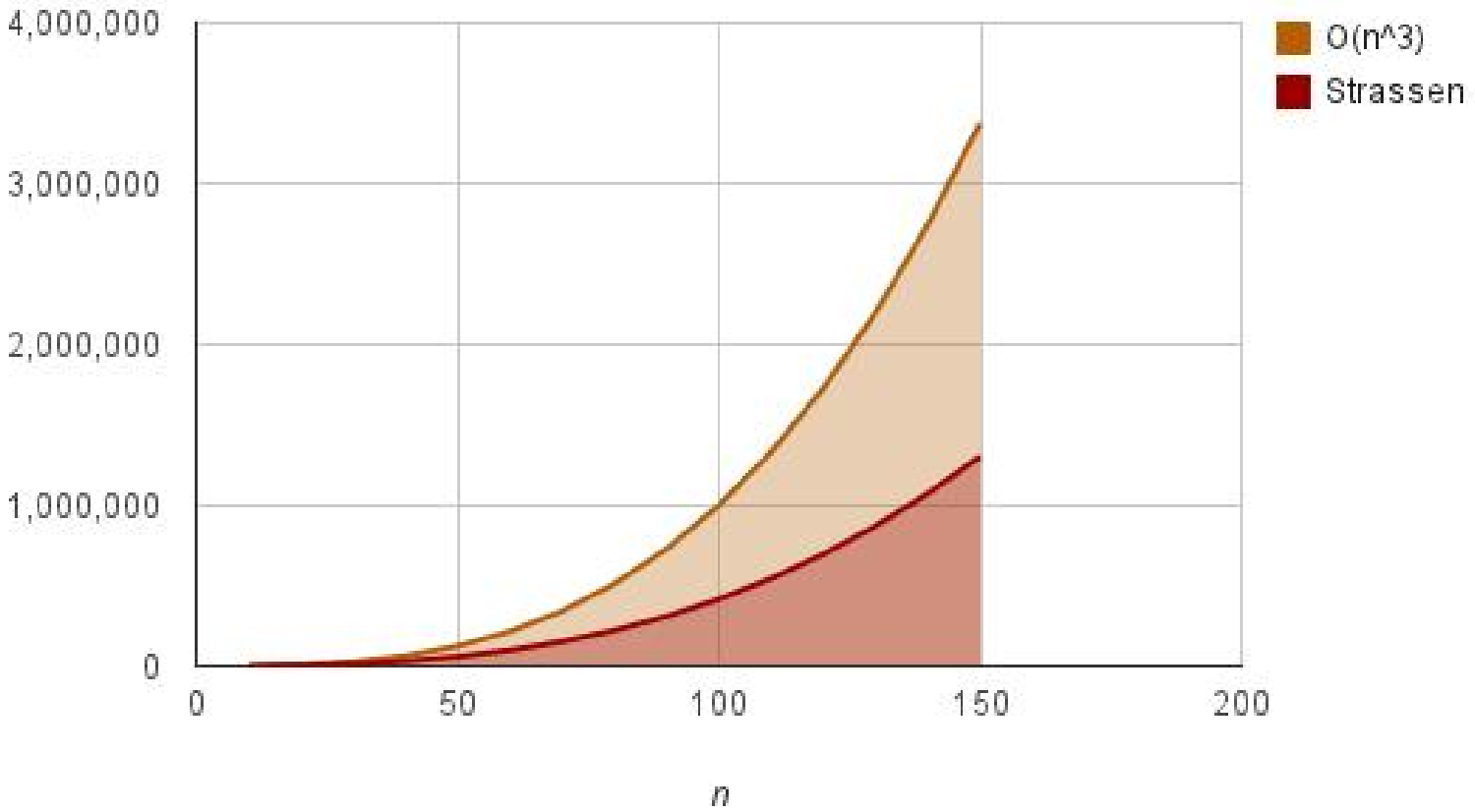
$$T(n) = 7\,T(n/2) + \Theta(n^2)$$

$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow$ CASE 1 $\Rightarrow T(n) = \Theta(n^{\lg 7})$.

The number 2.81 may not seem much smaller than 3, but because the difference is in the exponent, the impact on running time is significant. In fact, Strassen's algorithm beats the ordinary algorithm on today's machines for $n \geq 30$ or so.

Best to date (of theoretical interest only): $\Theta(n^{2.376\cdots})$.

Algorithm analysis



哈佛、MIT学者联手，创下矩阵乘法运算最快纪录

原创 机器之心Pro 2021-03-24 13:23:30

编辑：小舟、维度

作为一种基本数学运算，矩阵乘法的运算速度一直是一个重要的研究课题。哈佛大学和 MIT 研究者联合进行的一项研究创下了矩阵相乘的最快纪录。

矩阵乘法作为一种基本的数学运算，在计算机科学领域有着非常广泛的应用，矩阵乘法的快速算法对科学计算有着极为重要的意义。自 1969 年 Strassen 算法开始，人们意识到了快速算法的存在，开始了长达数十年的探索研究。

当你拥有两个大小一致的矩阵时，则可以将它们相乘得到第三个矩阵。例如，一对 2×2 矩阵的乘积也将是 2×2 矩阵，包含 4 个元素。即一对 n×n 矩阵的乘积是具有 n^2 个元素的另一个 n×n 矩阵。

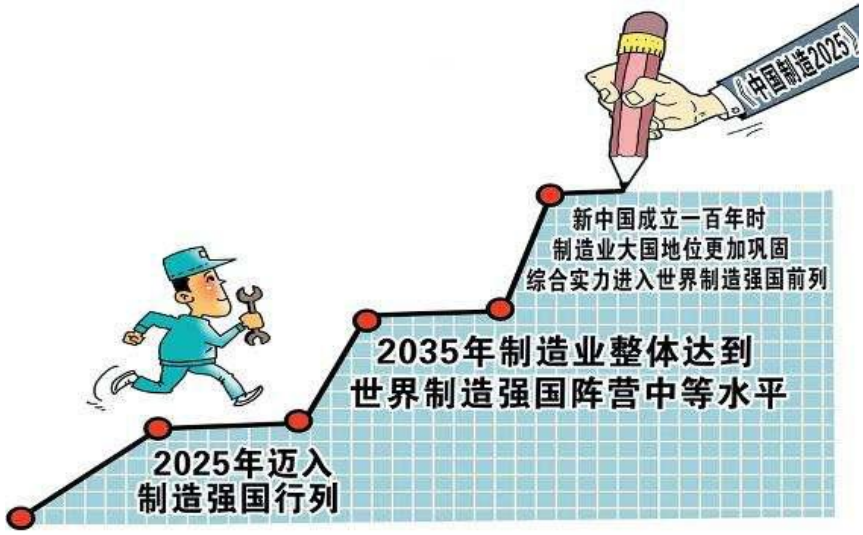
因此，矩阵乘法至少需要 n^2 步，人们理想中的计算复杂度也就是 O(n^2)。

2020 年 10 月，来自哈佛大学与 MIT 的两位研究者发表了一篇论文，他们创建了有史以来矩阵相乘的最快算法，相比于之前最快算法，计算复杂度下降了 10 万分之一。其中，论文一作 Josh Alman 是哈佛大学的博士后研究生，主要研究算法设计与复杂度理论。二作 Vassilevska Williams 是 MIT 计算机科学与人工智能实验室（CSAIL）副教授，致力于将组合和图论工具应用于计算领域。

$$\omega < 2.37286$$

有没有
更快的算法?

“互联网+” 产业升级



矩阵乘算法为例：

- 1969年，**Strassen算法， $O(n^{2.807355})$**
- 1981年，Pan算法，$O(n^{2.494})$)
- 1987年，Coppersmith–Winograd算法，$O(n^{2.376})$，1990改进为$O(n^{2.375477})$
- 2010年，Andrew Stothers，$O(n^{2.374})$
- 2011年底，斯坦福大学的Virginia Vassilevska Williams在Andrew Stothers基础上，改进为$O(n^{2.3728642})$
- 2014年，François Le Gall简化斯坦福方法，达到$O(n^{2.3728639})$
- 2020年，哈佛、MIT学者创造了目前最快算法, $\omega<2.37286$

人类的科技进步伴随着算法的进步！

The latest research

nature

Explore content ⌄ About the journal ⌄ Publish with us ⌄

nature › articles › article

Article | Open Access | Published: 05 October 2022

Discovering faster matrix multiplication algorithms with reinforcement learning

Alhussein Fawzi ✉, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis & Pushmeet Kohli

Nature **610**, 47–53 (2022) | Cite this article

392k Accesses | **3649** Altmetric | Metrics

AlphaTensor discovered algorithms that outperform the state of-the-art complexity for many matrix sizes. Particularly relevant is the case of 4×4 matrices in a finite field, where AlphaTensor's algorithm improves on Strassen's two level algorithm for the first time, to our knowledge.

Thank You!

Q&A