中国科学院大学
University of Chinese Academy of Sciences

CS101

Algorithmic Thinking

Divide and Conquer

zxu@ict.ac.cn
zhangjialin@ict.ac.cn

Outline

- What is algorithmic thinking
- Divide-and-conquer paradigm
  - Insertion Sort
  - Merge Sort Algorithm
  - Single Factor Optimization
  - Integer Multiplication
  - Matrix Multiplication
- Other interesting paradigms
- P vs. NP

These slides acknowledge sources for additional data not cited in the textbook

2. Divide-and-conquer

- Idea: recursively break down a problem into two or more subproblems of the similar type, until these subproblems become simple enough to be solved directly
- Three phases
  - Divide: divide problem into subproblems
  - Conquer: obtain solution to each of the subproblems
  - Combine: combine subsolutions into solution

2.1 Insertion sort algorithm

- **Input**: An array A of length n to be sorted
  - e.g., A=[6, 2, 4, 1, 5, 9], n = 6
- **Output**: A sorted array A
  - e.g., A=[1, 2, 4, 5, 6, 9]
- **Steps**:

```
for i = 1 to n-1
        j=i+1;
        while (j>1) and (A[j-1]>A[j])
                exchange A[j] with A[j-1];
                j=j-1;
```

Idea: for each round, A[1,…,i] sorted, insert A[i+1] into proper position, to get a new sorted array A[1,…,i+1]

- $T(n) = O(n^2)$

Red: sorted
Blue: to be inserted

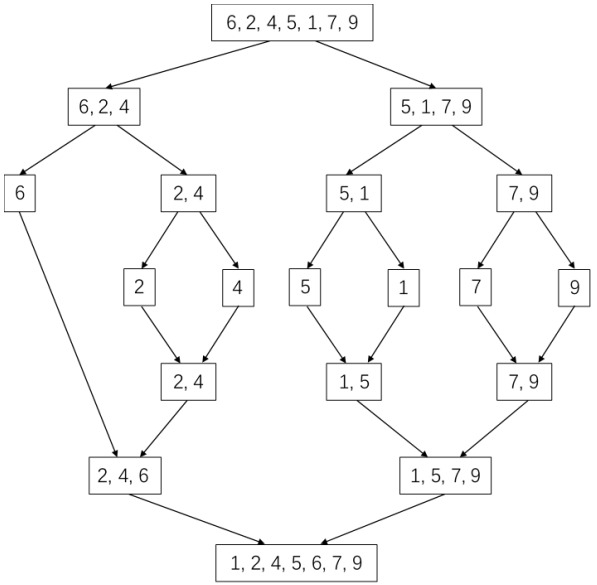
Initial: [6, 2, 4, 1, 5, 9]
i=1 [6, **2**, 4, 1, 5, 9]
j=i+1=2; A[1]>A[2]
exchange 6 and 2
[2, 6, 4, 1, 5, 9]
j=j-1=1; exit while loop

i=2 [2, 6, **4**, 1, 5, 9]
j=i+1=3; A[2]>A[3]
exchange 6 and 4
[2, **4**, 6, 1, 5, 9]
j=j-1=2 A[1]<A[2]
no exchange
[2, 4, 6, 1, 5, 9]
j=j-1=1; exit while loop

…
Please continue to finish

2.2 Merge sort algorithm

- MergeSort([A[1],…,A[n]])
- **if** (n==1) **then** return [A[1]];
- B=MergeSort([A[1],…,A[n/2]]);
- C=MergeSort([A[n/2+1],…,A[n]]);
- return merge(B, C);

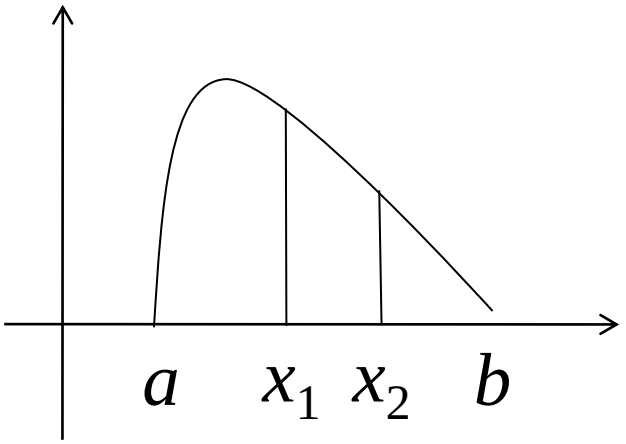


- $T(n) = O(n \log n)$
  - Compare to $O(n^2)$ for insertion sort

- It is often smart to divide a problem into subproblems of (almost) equal sizes
- We use three more problems to show nuances of divide-and-conquer

2.3 Single factor optimization

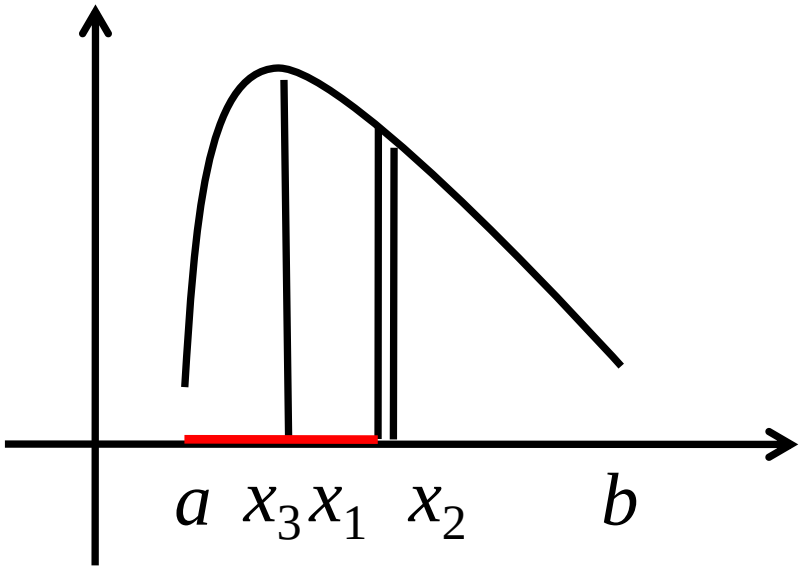
- Given a function f defined in the interval $[a, b]$
  - single-peak condition: f firstly (strictly) monotonically increases and then (strictly) monotonically decreases
- Goal: find $x = \arg\max f(x)$
- Idea:
  - Choose x_1, x_2
  - If $f(x_1) > f(x_2)$, delete $[x_2, b]$
  - If $f(x_1) < f(x_2)$, delete $[a, x_1]$
  - If $f(x_1) = f(x_2)$, delete $[a, x_1]$ and $[x_2, b]$



- Smarter idea:
  - Divide a problem not in the middle (0.5), but at 0.618

Divide the problem at the middle

- Method 1：
- $x_1 \approx x_2 \approx (a + b)/2$



- Discretize interval $[a, b]$ into n points
- $T(n) = T(n/2) + 2$
- $T(2) = 2$

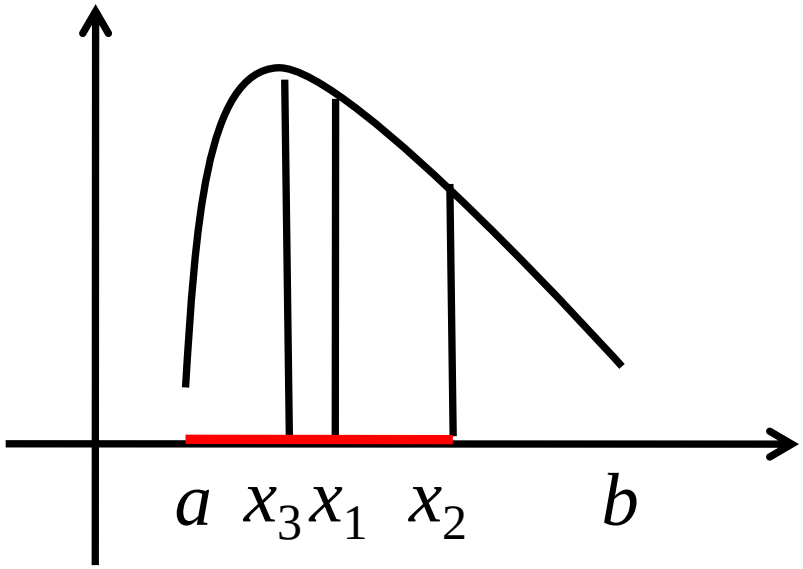
- Need to compute $2\log_2 n$ function values

Divide the problem at 0.618

- Method 2: $x_1 = ta + (1-t)b, x_2 = (1-t)a + tb$, where

$$t = \frac{\sqrt{5}-1}{2} \approx 0.618$$

- Key observation: one of the two points we need to query is already known!



Need to compute $\log_{1.618} n$ function values
$\approx 1.44 \log_2 n$

Versus
$2\log_2 n$ in Method 1

2.4 Integer multiplication

- Input: $X=x_n x_{n-1}\ldots x_1$, $Y=y_n y_{n-1}\ldots y_1$
- Output: $Z = XY$
- Idea:

$$X = X_1 \times 10^{n/2} + X_2,$$

$$Y = Y_1 \times 10^{n/2} + Y_2$$

$$Z = XY =$$

$$X_1Y_1 \times 10^{n} + (X_1Y_2 + X_2Y_1) \times 10^{n/2} + X_2Y_2$$

$$\begin{array}{r} 123 \\ \times 321 \\ \hline 123 \\ 246 \\ 369 \\ \hline 39483 \end{array}$$

- Needs to compute n^2 multiplication operations

Integer multiplication

- Compute X_1Y_1, X_1Y_2, X_2Y_1, X_2Y_2
- #Multiplication operation: $T(n) = 4\,T(n/2)$, $T(1) = 1$
- $T(n) = n^2$…
- $X_1Y_1 \times 10^n + (X_1Y_2 + X_2Y_1) \times 10^{n/2} + X_2Y_2$
- Compute: X_1Y_1, X_2Y_2, $\mathbf{(X_1+X_2)(Y_1+Y_2)}$

$$X_1Y_2 + X_2Y_1 = \mathbf{(X_1+X_2)(Y_1+Y_2)} - X_1Y_1 - X_2Y_2$$

Integer multiplication

- #multiplication operations: $T(n) = 3\ T(n/2),\ T(1) = 1$
- $T(n) = n^{\log_2 3} \approx n^{1.59}$

Thinking problem***

- Can we compute integer multiplication faster?

- YES！Fast Fourier Transformation (FFT）

$$X = X_2 \times 10^{2n/3} + X_1 \times 10^{n/3} + X_0,$$

$$Y = Y_2 \times 10^{2n/3} + Y_1 \times 10^{n/3} + Y_0,$$

$$Z = XY =$$

$$X_2Y_2 \times 10^{4n/3} + (X_1Y_2 + X_2Y_1) \times 10^{n}$$

$$+ (X_0Y_2 + X_1Y_1 + X_2Y_0) \times 10^{2n/3}$$

$$+ (X_1Y_0 + X_0Y_1) \times 10^{n/3} + X_0Y_0$$

- Method 1
  - Compute $X_0Y_0,\ X_1Y_1,\ X_2Y_2,\ (X_0+X_1)(Y_0+Y_1)$ $(X_0+X_2)(Y_0+Y_2),\ (X_1+X_2)(Y_1+Y_2)$
  - $T(n) = 6T(n/3),\ T(1) = 1$
  - $T(n) = n^{\log_3 6} \approx n^{1.631}$
- It runs slower than the previous method…

$$\omega = e^{i\frac{2\pi}{3}}$$

$$A_0 = X_2 + X_1 + X_0, \qquad B_0 = Y_2 + Y_1 + Y_0,$$

$$A_1 = X_2 + \omega X_1 + \omega^2 X_0, \quad B_1 = Y_2 + \omega Y_1 + \omega^2 Y_0,$$

$$A_2 = X_2 + \omega^2 X_1 + \omega X_0, \quad B_2 = Y_2 + \omega^2 Y_1 + \omega Y_0.$$

$$A_0B_0 + A_1B_1 + A_2B_2 = 3(X_2Y_2 + X_1Y_0 + X_0Y_1)$$

$$A_0B_0 + \omega A_1B_1 + \omega^2 A_2B_2 = 3(X_0Y_2 + X_1Y_1 + X_2Y_0)$$

$$A_0B_0 + \omega^2 A_1B_1 + \omega A_2B_2 = 3(X_2Y_1 + X_1Y_2 + X_0Y_0)$$

- T(n): the number of multiplication operations of two n-bit integers
  - $T(n) = 5\,T(n/3), T(1) = 1$
  - $T(n) = n^{\log_3 5} \approx n^{1.465}$

- Can we do better?
- YES: Fast Fourier Transform (FFT)

Integer Multiplication

- Based on FFT
- $O(n \log n \log\log n)$ (Schönhage and Strassen, 1971)
- $O(n \log n \, 2^{O(\log^* n)})$ (Fürer, 2007)
- $O(n \log n \, 8^{\log^* n})$ (Harvey and Hoeven, 2016)
- $O(n \log n \, 4^{\log^* n})$ (Harvey and Hoeven, 2019)
- $O(n \log n)$ (Harvey and Hoeven)
  - $\log^* n$：https://en.wikipedia.org/wiki/Iterated_logarithm

2.5 Matrix multiplication

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

- $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$
- # multiplication operations: $O(n^3)$
- # addition operations: $O(n^3)$

Matrix multiplication

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$m_1 = (a_{12} - a_{22})(b_{21} + b_{22})$$
$$m_2 = (a_{11} + a_{22})(b_{11} + b_{22})$$
$$m_3 = (a_{11} - a_{21})(b_{11} + b_{12})$$
$$m_4 = (a_{11} + a_{12})b_{22}$$
$$m_5 = a_{11}(b_{12} - b_{22})$$
$$m_6 = a_{22}(b_{21} - b_{11})$$
$$m_7 = (a_{21} + a_{22})b_{11}$$

$$c_{11} = m_1 + m_2 - m_4 + m_6$$
$$c_{12} = m_4 + m_5$$
$$c_{21} = m_6 + m_7$$
$$c_{22} = m_2 - m_3 + m_5 - m_7$$

multiplication operations: $O(n^{\log 7 \approx 2.81})$

Matrix multiplication

- Strassen algorithm'69 $O(n^{2.81})$
- $O(n^{2.79})$, $O(n^{2.55})$, $O(n^{2.48})$ …
- Coppersmith–Winograd algorithm'89 $O(n^{2.376})$
- Stothers'10 $O(n^{2.374})$
- Williams'11 $O(n^{2.373})$
- Le Gall'14 $O(n^{2.3729})$
- New breakthrough in 2021

Algorithmic paradigms

- Divide-and-conquer
  - Single factor optimization
  - Integer multiplication
  - Matrix multiplication
- Other paradigms
  - Greedy, recursive, exhaustive search, backtracking method, dynamic programming …
  - Randomized algorithms, approximation algorithms, online algorithms, streaming algorithms …