# Algorithm Design, Analysis, and Lower Bound

— well, they are just "divide and conquer", "amortized analysis", and "adversary argument"

Hengfeng Wei

hengxin0912@gmail.com

November 7, 2014

# Outline

Divide and Conquer

Amortized Analysis

Adversary Argument

# Outline

Divide and Conquer

Amortized Analysis

Adversary Argument

# Recurrences

$$T(n) = aT(n/b) + f(n), \text{ assuming } n = b^x$$

$$T(n) = \underbrace{\Theta(n^{\log_b a})}_{\text{solving base cases}} + \underbrace{\sum_{j=0}^{\log_b n-1} a^j f(\frac{n}{b^j})}_{g(n)\text{=dividing and combining}}$$

Case 1: $f(n) = O(n^{\log_b a-\epsilon})$ :
$$g(n) = O(n^{\log_b a}), T(n) = \Theta(n^{\log_b a})$$

Case 2: $f(n) = \Theta(n^{\log_b a})$ :
$$g(n) = n^{\log_b a} \log_b n, T(n) = \Theta(n^{\log_b a}) \log_b n$$

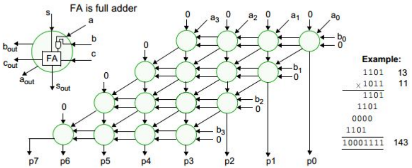
Case 3: $f(n) = \Omega(n^{\log_b a+\epsilon})$ :
$$g(n) = \Theta(f(n)), T(n) = \Theta(f(n))$$

# Integer Multiplication

## Problem (Integer Multiplication)

Multiplying two $n$-bit integers in $o(n^2)$ time. (Assuming $n = 2^i$.)

"Column multiplication in $\Theta(n^2)$"




### Elementray operations:

- $n$-bit + $n$-bit: $O(n)$
- 1-bit $\times$ $n$-bit : $O(1)$
- $n$-bit shifted by 1-bit: $O(1)$

# Integer Multiplication

## Simple Divide and Conquer:

$$x = x_L : x_R = 2^{n/2}x_L + x_R$$
$$y = y_L : y_R = 2^{n/2}y_L + y_R$$

$$\begin{aligned} xy &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\ &= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

$$T(n) = 4T(n/2) + \Theta(n) = \Theta(n^2)$$

# Integer Multiplication

## A Little History:

- Kolmogorov (1952) conjecture: $\Omega(n^2)$
- Kolmogorov (1960) seminar
- Karatsuba (23 Y/O., *within a week*): $\Theta(n^{1.59})$
- "The Complexity of Computations" (1995)

# Integer Multiplication

## Karatsuba Algorithm:

$$T(n) = 3T(n/2) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59})$$

$$\underbrace{(x_L + x_R)(y_L + y_R)}_{P_0} = \underbrace{x_L y_L}_{P_1} + (x_L y_R + x_R y_L) + \underbrace{x_R y_R}_{P_2}$$

$$xy = 2^n P_1 + 2^{n/2}(P_0 - P_1 - P_2) + P_2$$

# Matrix Multiplication

**Problem (Matrix Multiplication)**

Multiplying two $n \times n$ matrices in $o(n^3)$ time. (Assuming $n = 2^i$.)

$$Z = X \times Y$$

$$Z_{ij}$$

$$T(n) = \Theta(n^2 \cdot n) = \Theta(n^3)$$

Elementrary operations:

- integer addition: $O(1)$
- integer multiplication: $O(1)$

# Matrix Multiplication

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \quad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \qquad (A \dots H \in \mathbb{R}^{n/2} \times \mathbb{R}^{n/2})$$

$$XY = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$$T(n) = 8T(n/2) + \Theta(n^2) = \Theta(n^3)$$

# Matrix Multiplication

## Strassen Algorithm:

$$T(n) = 7T(n/2) + \Theta(n^2) = \Theta(n^{\lg 7}) = \Theta(n^{2.808})$$

- Strassen (1969): $\Theta(n^{2.808})$
- (2014): $\Theta(n^{2.373})$
- Known lower bound: $\Omega(n^2)$

# Computing $\lceil\sqrt{N}\rceil$

**Problem (Computing $\lceil\sqrt{N}\rceil$)**

Given an $n$-bit natural number $N$, how to compute $\lceil\sqrt{N}\rceil$ using only $O(n)$ additions and shifts?

Elementrary operations:

- $n$-bit + $n$-bit: $O(1)$
- $n$-bit shifted by 1-bit: $O(1)$
- $\Rightarrow x^2 : O(n)$

- Naïve search: $O(2^n \cdot n)$
- Binary search: $O(n \cdot n)$
- Binary search in range:

$$2^{\lfloor \frac{n-1}{2} \rfloor} \leq \lceil\sqrt{N}\rceil \leq 2^{\lceil \frac{n}{2} \rceil}$$

$$\lg\left(2^{\lceil \frac{n}{2} \rceil} - 2^{\lfloor \frac{n-1}{2} \rfloor}\right) = n$$

$$O(n \cdot n)$$

# Computing $\lceil \sqrt{N} \rceil$

## A Little History:

- Mid-term problem ($\sim$ 2008)
- ($\sim$ 2013): $O(n^2)$
- (2014): $O(n)$

# Computing $\lceil \sqrt{N} \rceil$

Given

$$M = \lfloor N/4 \rfloor, x = \lceil \sqrt{M} \rceil, \text{ and } (x, x^2),$$

what is

$$y = \lceil \sqrt{N} \rceil \text{ and } (y, y^2)?$$

An Example:

$$\begin{array}{lll}
N = 280 & y = \lceil \sqrt{280} \rceil = 17 & y^2 = 289 \\
M = \lfloor 280/4 \rfloor = 70 & x = \lceil \sqrt{70} \rceil = 9 & x^2 = 81 \\
M = \lfloor 70/4 \rfloor = 17 & x = \lceil \sqrt{17} \rceil = 5 & x^2 = 25 \\
M = \lfloor 17/4 \rfloor = 4 & x = \lceil \sqrt{4} \rceil = 2 & x^2 = 4 \\
M = \lfloor 4/4 \rfloor = 1 & x = \lceil \sqrt{1} \rceil = 1 & x^2 = 1
\end{array}$$

# Computing $\lceil \sqrt{N} \rceil$

**Algorithm 1** Computing $\lceil \sqrt{N} \rceil$.

**procedure** $\textsc{Sqrt-Root}(N)$
  **if** $N < 3$ **then**
    **return** $1 \Rightarrow (1,1); 2 \Rightarrow (2,4); 3 \Rightarrow (2,4)$
  $M \leftarrow \lfloor N/4 \rfloor$
  $(x, x^2) \leftarrow \textsc{Sqrt-Root}(M)$
  **return** the $(y, y^2)$ with $y^2 \sim N$:

$$(y, y^2) = \begin{cases} y = 2x & y^2 = 4x^2 \\ y = 2x + 1 & y^2 = 4x^2 + 4x + 1 \\ y = 2x - 1 & y^2 = 4x^2 - 4x + 1 \end{cases}$$

# Computing $\lceil \sqrt{N} \rceil$

$$\cancel{T(n) = T(n/4) + O(1) = \Theta(\lg n)}$$

$$T(n) = T(n-2) + O(1) = \Theta(n)$$

# VLSI Layout

## Problem (Area-Efficient VLSI Layout)

Embedding a complete binary tree with $n$ leaves into a grid with minimum area.

- VLSI: Very Large Scale Integration
- complete binary tree circuit of $\#layer = 3, 5, 7, \ldots$
- vertex on grid; no crossing edges
- area = width $\times$ height

# VLSI Layout

- Naïve embedding

$$H(n) = H(\frac{n}{2}) + \Theta(1) = \Theta(\lg n)$$
$$W(n) = 2W(\frac{n}{2}) + \Theta(1) = \Theta(n)$$
$$A(n) = \Theta(n \lg n)$$

- Smart (H-Layout) embedding

$$\square \times \square = n?\ 1 \times n;\ \frac{n}{\lg n} \times \lg n;\ \sqrt{n} \times \sqrt{n}$$

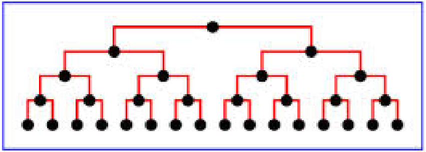
Goal: $H(n) = \Theta(\sqrt{n}); W(n) = \Theta(\sqrt{n}); A(n) = \Theta(n).$

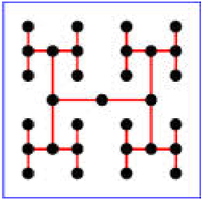
$$H(n) = \square H(\frac{n}{\square}) + O(\square); H(n) = 2H(\frac{n}{4}) + O(n^{\frac{1}{2}-\epsilon})$$

$$H(n) = 2H(\frac{n}{4}) + \Theta(1)$$

Here it is: H-Layout

# VLSI Layout





# Local Minimum in Tree (Optional)

**Problem (Local Minimum in Tree)**

Consider an $n$ node complete binary tree $T$. Each node $v$ is labeled with a (distinct) number $x_v$, how to find a *local minimum* in $O(\log n)$ time?

# Local Minimum in Grid (Optional)

**Problem (Local Minimum in Grid)**

Consider an $n \times n$ grid. Each cell is labeled with a (distinct) number and has (at most) four neighbors. How to find a *local minimum* in $O(n)$ time?

# Outline

Divide and Conquer

Amortized Analysis

Adversary Argument

# Amortized Analysis

*Amortized analysis is a strategy for analyzing a sequence of operations irrespective of the input to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.*

Key points:

(Array doubling; Multipop stack; Binary counter; Stack $\rightarrow$ Queue)

- cheap ops (often) vs. expensive ops (rare)
- on op sequence (where?); not on separate ops
- $\neq$ average-case analysis; no probability here (Quick-Sort, HashTable)
- worst-case analysis; upper-bound

# Methods for Amortized Analysis

1. Summation Method
   - $\sum_{i=1}^{n} c_i / n$
   - the op. sequence is known and easy to analyze (pattern)
2. Accounting Method
   - impose an extra charge on inexpense ops and use it to pay for expensive ops later on
   - $\hat{c_i} = c_i + a_i$ $(a_i >=< 0)$
     (amortized cost = actual cost + accounting cost)
   - $\forall n, \sum_{i=1}^{n} \hat{c_i} = \sum_{i=1}^{n} c_i + \sum_{i=1}^{n} a_i$
   - $\forall n, \sum_{i=1}^{n} \hat{c_i} \geq \sum_{i=1}^{n} c_i \Rightarrow \forall n, \sum_{i=1}^{n} a_i \geq 0$
   - put the accounting cost on specific objects
3. Potential Method
   - see Section 17.3 of CLRS (3rd edition)

# Array Doubling Revisited

## Summation Method:

*on any sequence of $n$* INSERT *ops on an initially empty array*

Q: What is the cost of $c_i$ of the $i$-th op?

$$\begin{array}{lcccccccccccc} i: & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ c_i: & & 1 & 2 & 3 & 1 & 5 & 1 & 1 & 1 & 8 & 1 \end{array}$$

$$c_i = \begin{cases} (i-1)+1 = i & \text{if } i-1 \text{ is an exact power of 2} \\ 1 & \text{o.w.} \end{cases}$$

$$\sum_{i=1}^{n} c_i = n + \sum_{j=0}^{\lceil \lg n \rceil - 1} 2^j = n + (2^{\lceil \lg n \rceil} - 1) \leq n + 2n = 3n$$

The *amortized* cost of a single operation is 3.

# Array Doubling Revisited

## Accounting method:

$$\hat{c}_i = 3$$

Why not $\hat{c}_i = 2$?

An example:

$$0; 0; 0; 0; 1 \Rightarrow 0; 0; 0; 0; 1; 1; 1; 1 \Rightarrow ☹$$

$$\hat{c}_i = 3 = \underbrace{1}_{\text{insert}} + \underbrace{1}_{\text{move itself}} + \underbrace{1}_{\text{move another}}$$

| | $\hat{c}_i$ | $c_i$ (actual cost) | $a_i$ (accounting cost) |
|---|---|---|---|
| INSERT(normal) | 3 | 1 | 2 |
| INSERT(expensive) | 3 | 1 + t | -t + 2 |

# Two Stacks, One Queue

## Problem (Two Stacks, One Queue)

Correctness proof

Amortized analysis

---

**Algorithm 2** Simulating a queue using two stacks $S_1, S_2$.

---

**procedure** $\textsc{Enq}(x)$
    $Push(S_1, x)$

**procedure** $\textsc{Deq}()$
    **if** $S_2 = \emptyset$ **then**
        **while** $S_1 \neq \emptyset$ **do**
            $Push(S_2,\ Pop(S_1))$
    $Pop(S_2)$

---

Ex: $\textsc{Enq}(1, 2, 3)$, $\textsc{Deq}()$, $\textsc{Deq}()$, $\textsc{Enq}(4)$, $\textsc{Deq}()$

# Two Stacks, One Queue

Simple observation: $\boxed{S_1 \text{ to push; } S_2 \text{ to pop.}}$

## Correctness proof:

$$\text{FIFO:} \quad \textsc{Deq}() = x \prec \textsc{Deq}() = y \quad \Rightarrow \quad Enq(x) \prec \textsc{Enq}(y)$$

- Summation method: the sequence is NOT known
- Accounting method: $\hat{c}_i = c_i + a_i$

$$\forall n, \sum_{i}^{n} a_i \geq 0$$

# Two Stacks, One Queue

$$\text{item:}\quad \underbrace{\textsc{Push}\text{ into } S_1}_{\textsc{Enq}:\ 1}\quad \textsc{Pop}\text{ from } S_1:\ 1\quad \textsc{Push}\text{ into } S_2:\ 1\quad \underbrace{\textsc{Pop}\text{ from } S_2:\ 1+1}_{\textsc{Deq}(normal)}$$

$\#S_1 = t$ :

| | $\hat{c}_i$ | $c_i$ (actual cost) | $a_i$ (accounting cost) |
|---|---|---|---|
| Enqueue | 3 | 1 | 2 |
| Dequeue(normal) | 2 | 2 | 0 |
| Dequeue(expensive) | 2 | 2+2t | -2t |

$$\sum_{i=1}^{n} a_i = \#S_1 \times 2 \geq 0$$

# Outline

Divide and Conquer

Amortized Analysis

Adversary Argument

# Lower Bound

$$T(A) = \max_I T(A, I)$$

$$T(P) = \min_A \max_I T(A, I)$$

$$UB(P) = t \iff \exists_A T(A) \leq t \iff \exists_A \forall_I T(A, I) \leq t$$

$$LB(P) = t \iff \forall_A T(A) \geq t \iff \forall_A \exists_I T(A, I) \geq t$$

# Lower Bound

$n! \Rightarrow n^2$ *(playing card)*

$\Rightarrow n \lg n$ *(MergeSort by John von Neumann@1948)*

SORTING

$\Leftarrow n \lg n$ *(decision tree)*

$\Leftarrow n$

$n^2 \Rightarrow n \lg n \Rightarrow n$

MEDIAN($k^{th}$)

$\Leftarrow n$

$\Rightarrow 16n$ *(median-of-median)*

$\Rightarrow 2.95n$ *()*

MEDIAN($k^{th}$)

$$\Leftarrow \frac{3n}{2} - \frac{3}{2}$$

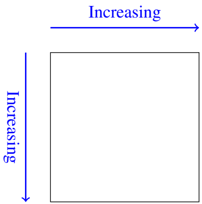
"Time Bounds for Selection" **BFPRT**@1973

# Matrix Search

**Problem (Matrix Search)**

Given an $n \times n$ integer matrix with both rows and columns in increasing order, how to find an element $x$ in $O(n)$ time?

Give an adversary argument to establish its lower bound.



## Matrix Search

Finding $x = 28$:

| | | | | |
|---|---|---|---|---|
| 1 | 3 | 5 | 7 | 9 |
| 6 | 8 | 12 | 14 | 15 |
| 10 | 13 | 18 | 22 | 33 |
| 20 | 24 | 29 | 30 | 35 |
| 26 | 31 | 32 | 40 | 45 |

Algorithm:

1. check the lower left element
2. delete a row or a column; goto 1.

$$T(n) = 2(n-1) + 1 = 2n - 1 = \Theta(n)$$

# Matrix Search

Adversary argument:

$$LB(P) = t \iff \forall_A T(A) \geq t \iff \forall_A \exists_I T(A, I) \geq t$$

Adversary is constructing an input:

| | | | | |
|---|---|---|---|---|
| 1 | 3 | 5 | 7 | $*$ |
| 6 | 8 | 12 | $*(14)$ | $*$ |
| 10 | 13 | $*$ | $*$ | 33 |
| 20 | $*$ | $*(29)$ | 30 | 35 |
| $*$ | $*(31)$ | 32 | 40 | 45 |

$i + j \leq n : \; M[i][j] < n$

$i + j > n : \; M[i][j] > n$

To prove: all the elements on the two diagonals must be checked


THANK YOU