CS3230 – Design and Analysis of Algorithms (S2 AY2024/25)

Lecture 3b: Divide and Conquer

Divide and conquer

1. Divide the problem into smaller subproblems.
2. Solve the subproblems recursively.
3. Combine the subproblem solutions to get the solution of the full problem.

Divide and conquer

1. Divide the problem into smaller subproblems.
2. Solve the subproblems recursively.
3. Combine the subproblem solutions to get the solution of the full problem.

MergeSort$(A[1..n])$
- If $n \geq 2$, do the following steps.
  - **MergeSort**$(A[1..\lceil n/2 \rceil])$.
  - **MergeSort**$(A[\lceil n/2 \rceil + 1..n])$.
  - "Merge" the two sorted arrays.

VisuAlgo (Merge sort): https://visualgo.net/en/sorting?mode=Merge

Divide and conquer

$\Theta(n)$ = the cost for **splitting/combining:**
- Split a problem into subproblems.
- Combine the solutions of subproblems.

1. Divide the problem into smaller subproblems.
2. Solve the subproblems recursively.
3. Combine the subproblem solutions to get the solution of the full problem.

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

MergeSort$(A[1..n])$
- If $n \ge 2$, do the following steps.
  - **MergeSort**$(A[1..\lfloor n/2 \rfloor])$.
  - **MergeSort**$(A[\lfloor n/2 \rfloor + 1..n])$.
  - "Merge" the two sorted arrays.

The size of each subproblem is $n/2$.

There are 2 subproblems.

Divide and conquer

$f(n)$ = the cost for **splitting/combining:**

- Split a problem into subproblems.
- Combine the solutions of subproblems.

1. Divide the problem into smaller subproblems.
2. Solve the subproblems recursively.
3. Combine the subproblem solutions to get the solution of the full problem.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \le 1 \\ aT\left(\frac{n}{b}\right) + f(n) & \text{if } n > 1 \end{cases}$$

The size of each subproblem is n/b.

There are a subproblems.

Question

- The recurrence for the running time of a divide-and-conquer algorithm:
  - $T(n) = 8T\left(\frac{n}{2}\right) + n^3$
- Two improvements to the algorithm are found:
  - **Improvement 1**: The cost for splitting/combining is reduced from n^3 to n^2.
  - **Improvement 2**: The number of subproblems is reduced from 8 to 7.

Question

- The recurrence for the running time of a divide-and-conquer algorithm:
  - $T(n) = 8T\left(\frac{n}{2}\right) + n^3$

- Two improvements to the algorithm are found:
  - **Improvement 1**: The cost for splitting/combining is reduced from n^3 to n^2.
  - **Improvement 2**: The number of subproblems is reduced from 8 to 7.

- Which of the improvements is asymptotically better?
  - Improvement 1
  - Improvement 2
  - Both improvements yield the same improved asymptotic running time.
  - Both improvements do not improve the asymptotic running time.

Answer

Both improvements yield the same improved asymptotic running time.

	Recurrence	n^d	$f(n)$	Master theorem	$T(n)$
Original algorithm	$T(n) = 8T\left(\frac{n}{2}\right) + n^3$	$n^{\log_2 8} = n^3$	n^3	Case 2	$\Theta(n^3 \log n)$
Improvement 1	$T(n) = 8T\left(\frac{n}{2}\right) + n^2$	$n^{\log_2 8} = n^3$	n^2	Case 1	$\Theta(n^3)$
Improvement 2	$T(n) = 7T\left(\frac{n}{2}\right) + n^3$	$n^{\log_2 7} = n^{2.807\ldots}$	n^3	Case 3	$\Theta(n^3)$

$\triangle$

Remember to check the regularity condition.

Exponentiation

- **Input:** two positive integers a and n.
- **Output:** a^n

Exponentiation

- **Input:** two positive integers a and n.
- **Output:** a^n

Note: To ensure that the output fits into one word, we consider **modular arithmetic**.

- The output is $a^n \pmod{m}$.
- m is some integer that fits into one word.

For the sake of simplicity, we omit explicitly stating $(\mathrm{mod}\ m)$ in the subsequent discussion.

Exponentiation

- **Input:** two positive integers a and n.
- **Output:** a^n

First approach:

- $a^n = a^{n-1} \cdot a$
- Recurrence: $T(n) = T(n-1) + \Theta(1)$
  - Computing a^{n-1} recursively: $T(n-1)$ time
  - Computing a^n from a^{n-1}: $\Theta(1)$ time
- $T(n) \in \Theta(n)$

Exponentiation

- **Input:** two positive integers a and n.
- **Output:** a^n

First approach:

- $a^n = a^{n-1} \cdot a$
- Recurrence: $T(n) = T(n-1) + \Theta(1)$
  - Computing a^{n-1} recursively: $T(n-1)$ time
  - Computing a^n from a^{n-1}: $\Theta(1)$ time
- $T(n) \in \Theta(n)$

Second approach:

- If (n is even), $a^n = a^{\lfloor \frac{n}{2} \rfloor} \cdot a^{\lfloor \frac{n}{2} \rfloor}$
- If (n is odd), $a^n = a^{\lfloor \frac{n}{2} \rfloor} \cdot a^{\lfloor \frac{n}{2} \rfloor} \cdot a$
- Recurrence: $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + \Theta(1)$
  - Computing $a^{\lfloor \frac{n}{2} \rfloor}$ recursively: $T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$ time
  - Computing a^n from $a^{\lfloor \frac{n}{2} \rfloor}$: $\Theta(1)$ time
- $T(n) \in \Theta(\log n)$

Exponential improvement!

Fibonacci numbers

- $F_0 = 0$
- $F_1 = 1$
- For $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$
- $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$
- **Recall:** F_n can be computed in $O(n)$ time.

Fibonacci numbers

Question: Can we do better by divide and conquer?

- $F_0 = 0$
- $F_1 = 1$
- For $n \geq 2$, $F_n = F_{n-1} + F_{n-2}$
- $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$
- **Recall:** F_n can be computed in $O(n)$ time.

$\mathbf{IFib}(n)$
- If $n \leq 1$
  - return n
- Else,
  - prev2 $=0$
  - prev1 $=1$
  - for $i = 2$ to n
    - temp $=$ prev1
    - prev1 $=$ prev1+prev2
    - prev2 $=$ temp
  - return prev1

Fibonacci numbers

- $\phi = \frac{1+\sqrt{5}}{2}$
- $\psi = \frac{1-\sqrt{5}}{2}$
- It can be shown that $F_n = \frac{1}{\sqrt{5}}(\phi^n - \psi^n)$.
- Can we use the exponentiation algorithm to compute F_n in $O(\log n)$ time?

Fibonacci numbers

- $\phi = \frac{1+\sqrt{5}}{2}$
- $\psi = \frac{1-\sqrt{5}}{2}$
- It can be shown that $F_n = \frac{1}{\sqrt{5}}(\phi^n - \psi^n)$.
- Can we use the exponentiation algorithm to compute F_n in $O(\log n)$ time?

- **Potential issues:**
  - Even if we intend to do modulo arithmetic, handling real numbers can be tricky.
  - How many bits of precision do we need to ensure that the output is correct?

Fibonacci numbers

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_n + F_{n-1} & F_n \\ F_{n-1} + F_{n-2} & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

Fibonacci numbers

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_n + F_{n-1} & F_n \\ F_{n-1} + F_{n-2} & F_{n-1} \end{pmatrix} = \begin{pmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

The exponentiation algorithm can compute $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ in $O(\log n)$ time.

F_n can be computed in $O(\log n)$ time.

Exponential improvement!

Midterm exam information

- 13/03/2025 (THU)
- MPSH 1B
- Start time: 14:00
- Mode of assessment: Hardcopy (Pen and Paper), Open book, No calculators or electronic devices
- Bring 2B pencils
- Scratch paper

Matrix multiplication

- **Input:** Two $(n \times n)$ matrices $A = [a_{i,j}]$ and $B = [b_{i,j}]$
- **Output:** $C = A \cdot B$

$$\begin{bmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,n} \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,n} \end{bmatrix}$$

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}$$

Matrix multiplication

- **Input:** Two $(n \times n)$ matrices $A = [a_{i,j}]$ and $B = [b_{i,j}]$
- **Output:** $C = A \cdot B$ ← $\Theta(n^3)$ time

$$\underbrace{\begin{bmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,n} \end{bmatrix}}_{n^2 \text{ entries}} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,n} \end{bmatrix}$$

$$c_{i,j} = \underbrace{\sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}}_{\Theta(n) \text{ time}}$$

Matrix multiplication

Question: Can we do better by divide and conquer?

- **Input:** Two $(n \times n)$ matrices $A = [a_{i,j}]$ and $B = [b_{i,j}]$
- **Output:** $C = A \cdot B$ ← $\Theta(n^3)$ time

$$\underbrace{\begin{bmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,n} \end{bmatrix}}_{n^2 \text{ entries}} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{bmatrix} \cdot \begin{bmatrix} b_{1,1} & \cdots & b_{1,n} \\ \vdots & \ddots & \vdots \\ b_{n,1} & \cdots & b_{n,n} \end{bmatrix}$$

$$c_{i,j} = \underbrace{\sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}}_{\Theta(n) \text{ time}}$$

Divide and conquer

$$\overset{\color{red}{C}}{\begin{bmatrix} r & s \\ t & u \end{bmatrix}} = \overset{\color{red}{A}}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \cdot \overset{\color{red}{B}}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}$$

- $r = ae + bg$
- $s = af + bh$
- $t = ce + dg$
- $u = cf + dh$

$$A = \begin{bmatrix} 2 & 3 & 1 & 7 \\ 9 & 4 & 5 & 0 \\ 6 & 3 & 6 & 7 \\ 8 & 6 & 3 & 4 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 2 & 3 \\ 9 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 7 \\ 5 & 0 \end{bmatrix} \\ \begin{bmatrix} 6 & 3 \\ 8 & 6 \end{bmatrix} & \begin{bmatrix} 6 & 7 \\ 3 & 4 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- $a = \begin{bmatrix} 2 & 3 \\ 9 & 4 \end{bmatrix}$
- $b = \begin{bmatrix} 1 & 7 \\ 5 & 0 \end{bmatrix}$
- $c = \begin{bmatrix} 6 & 3 \\ 8 & 6 \end{bmatrix}$
- $d = \begin{bmatrix} 6 & 7 \\ 3 & 4 \end{bmatrix}$

$r, s, t, u, a, b, c, d, e, f, g, h$ are $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices.

Divide and conquer

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

- $r = ae + bg$
- $s = af + bh$
- $t = ce + dg$
- $u = cf + dh$

8 multiplications of $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices: $8T\left(\frac{n}{2}\right)$ time.

4 additions of $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices: $\Theta(n^2)$ time.

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$T(n) \in \Theta\left(n^{\log_2 8}\right) = \Theta(n^3)$$

$r, s, t, u, a, b, c, d, e, f, g, h$ are $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices.

Divide and conquer

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

- $r = ae + bg$
- $s = af + bh$
- $t = ce + dg$
- $u = cf + dh$

8 multiplications of $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices: $8T\left(\frac{n}{2}\right)$ time.

4 additions of $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices: $\Theta(n^2)$ time.

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$T(n) \in \Theta\left(n^{\log_2 8}\right) = \Theta(n^3)$$

$r, s, t, u, a, b, c, d, e, f, g, h$ are $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices.

Observation: The asymptotic running time can be improved if the number of subproblems is reduced.

Strassen’s algorithm

$$\overset{\color{red}{C}}{\begin{bmatrix} r & s \\ t & u \end{bmatrix}} = \overset{\color{red}{A}}{\begin{bmatrix} a & b \\ c & d \end{bmatrix}} \cdot \overset{\color{red}{B}}{\begin{bmatrix} e & f \\ g & h \end{bmatrix}}$$

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

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

Strassen's algorithm

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

- $r = P_5 + P_4 - P_2 + P_6$ ◁ Checking its correctness.
- $s = P_1 + P_2$
- $t = P_3 + P_4$
- $u = P_5 + P_1 - P_3 - P_7$

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

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

Strassen's algorithm

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

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

7 multiplications of $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices: $7T\left(\frac{n}{2}\right)$ time.

18 additions of $\left(\frac{n}{2} \times \frac{n}{2}\right)$ matrices: $\Theta(n^2)$ time.

$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$

$T(n) \in \Theta\left(n^{\log_2 7}\right) = \Theta(n^{2.807\ldots})$

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

State of the art

Strassen's algorithm

Coppersmith–Winograd algorithm

Timeline of matrix multiplication exponent		
Year	Matrix multiplication exponent	Authors
1969	2.8074	Strassen
1978	2.796	Pan
1979	2.780	Bini, Capovani, Romani
1981	2.522	Schönhage
1981	2.517	Romani
1981	2.496	Coppersmith, Winograd
1986	2.479	Strassen
1990	2.3755	Coppersmith, Winograd
2010	2.3737	Stothers
2012	2.3729	Williams
2014	2.3728639	Le Gall
2020	2.3728596	Alman, Williams
2022	2.371866	Duan, Wu, Zhou
2024	2.371552	Williams, Xu, Xu, and Zhou

https://en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication

Acknowledgement

- The slides are modified from previous editions of this course and similar course elsewhere.
- **List of credits:**
  - Diptarka Chakraborty
  - Yi-Jun Chang
  - Erik Demaine
  - Steven Halim
  - Sanjay Jain
  - Wee Sun Lee
  - Charles Leiserson
  - Hon Wai Leong
  - Warut Suksompong
  - Wing-Kin Sung