CSE 317: Design and Analysis of Algorithms

Shahid Hussain

Week 4: September 9, 11: Fall 2024

Divide and Conquer Algorithms

Master Theorem

Theorem

Let $L(n)$ be a function depending on natural n. Let c be a natural number, $c \geq 2$, a, b, γ be real constants such that, $a \geq 1$, $b > 0$, $\gamma \geq 0$, and for any $n = c^k$, where k is an arbitrary natural number, the following inequality holds:

$$L(n) \leq aL\left(\frac{n}{c}\right) + bn^{\gamma}.$$

Suppose for any natural k for any $n \in \{c^k + 1, c^k + 2, \ldots, c^{k+1}\}$ the inequality $L(n) \leq L(c^{k+1})$ holds. Then:

$$L(n) = \begin{cases} O(n^{\gamma}) & \textit{if } \gamma > \log_c a, \\ O(n^{\log_c a}) & \textit{if } \gamma < \log_c a, \\ O(n^{\gamma} \log n) & \textit{if } \gamma = \log_c a. \end{cases}$$

Proof of Master Theorem

- Let $n = c^k$ where, we obtain:

$$
\begin{aligned}
L(n) &\leq aL\left(\frac{n}{c}\right) + bn^\gamma \leq a\left(aL\left(\frac{n}{c^2}\right) + b\left(\frac{n}{c}\right)^\gamma\right) + bn^\gamma \\
&= bn^\gamma + ab\left(\frac{n}{c}\right)^\gamma + a^2L\left(\frac{n}{c^2}\right) \\
&\leq bn^\gamma + b\left(\frac{a}{c^\gamma}\right)n^\gamma + a^2\left(aL\left(\frac{n}{c^3}\right) + b\left(\frac{n}{c^2}\right)^\gamma\right) \\
&= bn^\gamma + bn^\gamma\left(\frac{a}{c^\gamma}\right) + bn^\gamma\left(\frac{a}{c^\gamma}\right)^2 + a^3L\left(\frac{n}{c^3}\right) \leq \cdots \\
&\leq bn^\gamma + bn^\gamma\left(\frac{a}{c^\gamma}\right) + \cdots + bn^\gamma\left(\frac{a}{c^\gamma}\right)^{k-1} + a^kL\left(\frac{n}{c^k}\right)
\end{aligned}
$$

Proof of Master Theorem (cont.)

- Let $d = \max\{b, L(1)\}$. Since $n/c^k = 1$, we have:

$$\begin{aligned} L(n) &\leq dn^\gamma \left(1 + \frac{a}{c^\gamma} + \left(\frac{a}{c^\gamma}\right)^2 + \cdots + \left(\frac{a}{c^\gamma}\right)^{k-1}\right) + da^k \\ &= dn^\gamma \left(1 + \frac{a}{c^\gamma} + \left(\frac{a}{c^\gamma}\right)^2 + \cdots + \left(\frac{a}{c^\gamma}\right)^{k}\right). \end{aligned}$$

- We can now use the following fact about geometric series. Let α be a real number and $0 \leq \alpha \leq 1$. Then, for any n,

$$\sum_{i=0}^{n} \alpha^j = 1 + \alpha + \alpha^2 + \cdots + \alpha^n = \frac{1 - \alpha^n}{1 - \alpha} < \frac{1}{1 - \alpha}.$$

Proof of Master Theorem (cont.)

- Let us consider three cases:

  (1) $\gamma > \log_c a$ (2) $\gamma < \log_c a$ (3) $\gamma = \log_c a$
- If $\gamma > \log_c a$. Then $a/c^\gamma < 1$.

  In this case $L(n) \le dn^\gamma \cdot const_1 = p_1 n^\gamma$ for some positive constant p_1
- If $\gamma < \log_c a$. Then $a/c^\gamma > 1$, and

$$L(n) \le dn^\gamma \left(\frac{a}{c^\gamma}\right)^k \left(1 + \frac{c^\gamma}{a} + \left(\frac{c^\gamma}{a}\right)^2 + \cdots + \left(\frac{c^\gamma}{a}\right)^k\right)$$

  Since $n = c^k$, we have $L(n) \le dn^\gamma \cdot const_2 = p_2 a^\gamma$.

  Therefore,

$$L(n) \le p_2 a^k = p_2 a^{\log_c n} = p_2 n^{\log_c a}$$

- If $\gamma = \log_c a$. Then $a/c^\gamma = 1$ and

$$L(n) \le dn^\gamma (k+1) = dn^\gamma (1 + \log_c n) \le 2dn^\gamma \log_c n \qquad \text{for } n \ge c$$

Proof of Master Theorem (cont.)

- For an arbitrary $n \in \mathbb{N} > c$, $\exists k \in \mathbb{N}$ s.t $c^k < n \leq c^{k+1}$
- Let us consider three cases for which $L(n) \leq L(c^{k+})$ holds
- If $\gamma > \log_c a$. Then

$$L(n) \leq L(c^{k+1}) \leq p_1 \left(c^{k+1}\right)^{\gamma} = p_1 c^{\gamma} \left(c^k\right)^{\gamma} \leq p_1 c^{\gamma} n^{\gamma}.$$

  Thus $L(n) = O(n^{\gamma})$.
- If $\gamma < \log_c a$. Then

$$L(n) \leq L(c^{k+1}) \leq p_2 \left(c^{k+1}\right)^{\log_c a} = p_2 c^{\log_c a} \left(c^k\right)^{\log_c a} \leq p_2 a n^{\log_c a}.$$

  Thus, $L(n) = O(n^{\log_c a})$.
- If $\gamma = \log_c a$. Then

$$L(n) \leq L(c^{k+1}) \leq p_3 c^{(k+1)\gamma} \log_c \left(c^{k+1}\right)$$

$$\leq p_3 c^{\gamma} \left(c^k\right)^{\gamma} (k+1) \leq p_3 c^{\gamma} n^{\gamma} (1+\log_c n) \leq 2 p_3 c^{\gamma} n^{\gamma} \log_c n.$$

  Thus, $L(n) = O(n^{\gamma} \log n)$.

Divide and Conquer Recurrences

If in above theorem the inequality $L(n) \leq aL\left(\frac{n}{c}\right) + bn^{\gamma}$ is replaced with $L(n) \leq aL\left(\frac{n}{c}\right) + O(n^{\gamma})$ then the statement of the theorem will still be true.

- $A(n) \leq A(n/2) + n$ for any $n = 2^k$, $k = 1, 2, 3, \ldots$. So $a = c = 2$, $b = 1$ and $\gamma = 1$. We have $\gamma = \log_c a$. We assume $A(n)$ is a nondecreasing function so $A(n) = O(n \log n)$
- $B(n) \leq 3B(n/2) + 1$ for any $n = 2^k$, $k = 1, 2, 3, \ldots$. So $a = 3$, $c = 2$, and $\gamma = 0$. This means $\gamma < \log_3 2$. We assume $B(n)$ is a nondecreasing function so $B(n) = O(n^{\log_3 2}) = O(n^{0.6309})$

Merge Sort

- Let us consider the MERGESORT
- MERGESORT is a recursive algorithm
- Let $\langle a_1, \ldots, a_n \rangle$ be input sequence to be sorted
- Merge sort divides the array into two (almost) equal parts as $\langle a_1, \ldots, a_{\lfloor n/2 \rfloor} \rangle$ and $\langle a_{\lfloor n/2 \rfloor + 1}, \ldots, a_n \rangle$
- Use MERGESORT to sort these two subproblems
- Let α and β be two the sorted sequences we receive after recursive calls
- We combine (merge) these lists to form a new list
- We compare first element of α with first element of β and transfer the smaller element to the new sequence and move the pointer where we take element from. If at any point if one of the sequences α or β becomes empty we concatenate the other sequence to the new sequence

Merge Sort

Algorithm: MERGESORT

Input: $A = \langle a_1, \ldots, a_n \rangle$: a sequence of n numbers

Output: A sorted permutation of A

1. **if** $n > 1$ **then**
2. $\quad \alpha =$ MERGESORT$(\langle a_1, a_2, \ldots, a_{\lfloor n/2 \rfloor} \rangle)$
3. $\quad \beta =$ MERGESORT$(\langle a_{\lfloor n/2 \rfloor + 1}, a_{\lfloor n/2 \rfloor + 2}, \ldots, a_n \rangle)$
4. $\quad$ **return** MERGE(α, β)
5. **else return** A

Merge Sort. Merging Two Sorted Lists

Algorithm: MERGE

Input: Two sorted lists A and B

Output: Merged sorted list of A and B

1. **if** $k = 0$ **then return** $\langle b_1, \ldots, b_l \rangle$
2. **if** $l = 0$ **then return** $\langle a_1, \ldots, a_k \rangle$
3. **if** $a_1 \leq b_1$ **then**
4.   **return** $\langle a_1 \rangle \circ$ MERGE$(\langle a_2, \ldots, a_k \rangle, \langle b_1, \ldots, b_k \rangle)$
5. **else**
6.   **return** $\langle b_1 \rangle \circ$ MERGE$(\langle a_1, \ldots, a_k \rangle, \langle b_2, \ldots, b_l \rangle)$

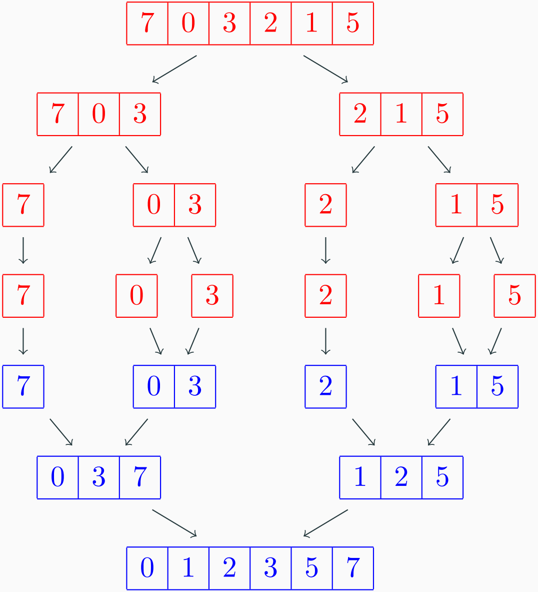
- Here $\circ$ denotes concatenation

Analysis of Merge Sort

- The running time for MERGEing is $O(k + l)$ i.e., it is linear in sizes of both arrays. Therefore, overall running time $T(n)$ of MERGESORT is (using Master Theorem):

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$$

Example: Sorting the sequence $\langle 7, 0, 3, 2, 1, 5 \rangle$



Finding Maximum

- We can apply divide and conquer design technique to solve a variety of problems including some trivial ones
- Suppose we need to find the maximum element from a sequence (array) $\langle a_1, a_2, \ldots, a_n \rangle$ of n unordered elements. Clearly the lower bound is $\Omega(n)$ as we need to check each and every element of the array
- Following is a divide and conquer algorithm that finds the maximum element from then sequence (array) $\langle a_1, a_2, \ldots, a_n \rangle$ of n unordered elements

Finding Maximum

Algorithm: DC-MAX

Input: A sequence $\langle a_1, a_2, \ldots, a_n \rangle$ of n unordered elements

Output: a_k such that $\forall i, a_i < a_k$ or $-\infty$ if $n = 0$

1. **if** $n = 1$ **then return** a_1
2. **else if** $n < 1$ **then return** $-\infty$
3. **else**
4. $\quad m_1 =$DC-MAX$(\langle a_1, \ldots, a_{\lfloor n/2 \rfloor} \rangle)$
5. $\quad m_1 =$DC-MAX$(\langle a_{\lfloor n/2 \rfloor + 1}, \ldots, a_n \rangle)$
6. **return** $\max\{m_1, m_2\}$

Matrix Multiplcation

- Let A and B be two matrices of size 2×2 each

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- Let $C = A \times B$, then

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

- To multiply two matrices of size 2×2 we need to perform 8 multiplications and 4 additions
- Strassen proposed an algorithm to multiply two matrices of size 2×2 using only 7 multiplications

Matrix Multiplcation

- Let us define following:

$$
\begin{aligned}
m_1 &= (a_{11} + a_{22}) \cdot (b_{11} + b_{22}) \\
m_2 &= (a_{21} + a_{22}) \cdot b_{11} \\
m_3 &= a_{11} \cdot (b_{12} - b_{22}) \\
m_4 &= a_{22} \cdot (b_{21} - b_{11}) \\
m_5 &= (a_{11} + a_{12}) \cdot b_{22} \\
m_6 &= (a_{21} - a_{11}) \cdot (b_{11} + b_{12}) \\
m_7 &= (a_{12} - a_{22}) \cdot (b_{21} + b_{22})
\end{aligned}
$$

- Now we can calculate $C = A \times B$ as follows:

$$
\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 - m_2 + m_3 + m_6 \end{bmatrix}
$$

Matrix Multiplcation: Strassen's Algorithm

- Let A and B be two $n \times n$ matrices each (for $n = 2^k$)
- The product $C = A \times B$ can be calculated as follows:
- Divide A and B into four $n/2 \times n/2$ matrices each as follows:

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

- Here A_{ij} and B_{ij} are $n/2 \times n/2$ matrices
- Calculate 7 matrices $M_1, M_2, \ldots, M_7$ and then calculate C_{ij}
- The running time $T(n)$ of Strassen's algorithm is:

$$T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) = O(n^{\log_2 7}) \approx O(n^{2.8074})$$

Integer Multiplcation

- Let us consider the problem of multiplying two integers x and y of n bits each
- The product $z = xy$ requires $O(n^2)$ bit-multiplications
- Karatsuba proposed an algorithm to multiply two integers of n digits each using only $O(n^{\log_2 3})$ bit-multiplications
- Let $x = \langle x_0, \dots x_{n-1} \rangle_2$ and $y = \langle y_0, \dots y_{n-1} \rangle_2$
- We can say that: $x = x_L x_R$ and $y = y_L y_R$ where $x_L = \langle x_0, \dots, x_{n/2-1} \rangle_2$, $x_R = \langle x_{n/2}, \dots, x_{n-1} \rangle_2$, $y_L = \langle y_0, \dots, y_{n/2-1} \rangle_2$, and $y_R = \langle y_{n/2}, \dots, y_{n-1} \rangle_2$
- We can write $x = x_L \cdot 2^{n/2} + x_R$ and $y = y_L \cdot 2^{n/2} + y_R$
- The product $z = xy$ can be calculated as follows:

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

Integer Multiplcation

- The product
  $z = xy = x_L y_L \cdot 2^n + (x_L y_R + x_R y_L) \cdot 2^{n/2} + x_R y_R$
- Requires 4 multiplications of $n/2$-bit numbers
- We can reduce the number of multiplications to 3
- As following:

$$\begin{aligned} x_L y_R + x_R y_L &= (x_L + x_R) \cdot (y_L + y_R) - x_L y_L - x_R y_R \\ &= x_L y_L + x_L y_R + x_R y_L + x_R y_R - x_L y_L - x_R y_R \\ &= x_L y_L + x_R y_R \end{aligned}$$

- Now:

$$z = x_L y_L \cdot 2^n + ((x_L + x_R) \cdot (y_L + y_R) - x_L y_L - x_R y_R) \cdot 2^{n/2} + x_R y_R$$

- The running time $T(n)$ of Karatsuba's algorithm is:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O(n^{\log_2 3}) \approx O(n^{1.585})$$