CSCE 411 Design and Analysis of Algorithms

Divide and Conquer
(based on Dr. Klappenecker’s lecture slides)
The divide and conquer paradigm is an important general technique for designing algorithms. In general, it follows the steps:

1. divide the problem into subproblems
2. recursively solve the subproblems
3. combine solutions to subproblems to get solution to original problem
Example - Mergesort
Example - Mergesort

Example courtesy of Wikipedia
Example - Mergesort

1. DIVIDE an input sequence of length n into two parts:
   - the initial $\lceil n/2 \rceil$ elements and
   - the final $\lfloor n/2 \rfloor$ elements

2. RECURSIVELY sort the two halves, using sequences with 1 key as a basis of the recursion

3. COMBINE the two sorted subsequences by merging them
Let $T(n)$ be the worst case running time of mergesort on a sequence of $n$ keys

If $n = 1$, then $T(n) = \Theta(1)$ (constant)

If $n > 1$, then

$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$
Let $T(n)$ be the worst case running time of mergesort on a sequence of $n$ keys

If $n = 1$, then $T(n) = \Theta(1)$ (constant)

If $n > 1$ and $n$ even, then

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

Indeed, we recursively sort the two subproblems of size $n/2$, and we need $\Theta(n)$ time to do the merge
Recurrence Relations
Consider a recurrence of the form

\[ T(n) = a \cdot T(n/b) + f(n) \]

with \( a \geq 1 \), \( b > 1 \), and \( f(n) \) eventually positive.

1. If \( f(n) = O(n^{\log_b(a)-\varepsilon}) \), then \( T(n) = \Theta(n^{\log_b(a)}) \)
2. If \( f(n) = \Theta(n^{\log_b(a)}) \), then \( T(n) = \Theta(n^{\log_b(a)} \log(n)) \)
3. If \( f(n) = \Omega(n^{\log_b(a)+\varepsilon}) \) and \( f(n) \) is regular, then \( T(n) = \Theta(f(n)) \)

Note: \( f(n) \) is regular if and only if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \)
Essentially, the Master theorem compares the function $f(n)$ with the function $g(n) = n^{\log_a(b)}$.

Roughly, the theorem says:

1. If $f(n) \ll g(n)$, then $T(n) = \Theta(g(n))$
2. If $f(n) \approx g(n)$, then $T(n) = \Theta(g(n)\log(n))$
3. If $f(n) \gg g(n)$, then $T(n) = \Theta(f(n))$

Now it is the time to memorize the theorem!
The Master theorem does not cover all possible cases. For example, if
\[ f(n) = \Theta(n^{\log_b(a)}\log(n)), \]
then we lie between case 2 and case 3, but the theorem does not apply.

There exist better versions of the Master theorem that cover more cases, but these are harder to memorize.
1. \[ T(n) = 9T(n/3) + n \]
1. $T(n) = 9T(n/3) + n$

   $a = 9$, $b = 3$, and $f(n) = n$

Thus, $n^{\log_b(a)} = n^{\log_3(9)} = \Theta(n^2)$

Since $f(n) = O(n^{\log_3(9) - \epsilon})$, where $\epsilon = 1$,

   case 1 applies that $T(n) = \Theta(n^2)$
1. \( T(n) = 9T(n/3) + n \)
   \[ a = 9, \ b = 3, \text{ and } f(n) = n \]
   Thus, \( n^{\log_b(a)} = n^{\log_3(9)} = \Theta(n^2) \)
   Since \( f(n) = O(n^{\log_3(9)-\varepsilon}) \), where \( \varepsilon=1 \),
   case 1 applies that \( T(n) = \Theta(n^2) \)

2. \( T(n) = T(2n/3) + 1 \)
1. $T(n) = 9T(n/3) + n$
   
a = 9, b = 3, and $f(n) = n$
   
   Thus, $n^{\log_b(a)} = n^{\log_3(9)} = \Theta(n^2)$
   
   Since $f(n) = O(n^{\log_3(9)-\varepsilon})$, where $\varepsilon=1$,
   
   case 1 applies that $T(n) = \Theta(n^2)$

2. $T(n) = T(2n/3) + 1$
   
a = 1, b = 3/2, and $f(n) = 1$
   
   Thus, $n^{\log_b(a)} = n^{\log_{3/2}(1)} = n^0 = 1$
   
   Since $f(n) = \Theta(1)$, 
   
   case 2 applies that $T(n) = \Theta(\log n)$
3. $T(n) = 3T(n/4) + n \log n$
3. \( T(n) = 3T(n/4) + n \log n \)
   
   \( a = 3, \ b = 4, \) and \( f(n) = n \log n \)
   
   Thus, \( n^{\log_b(a)} = n^{\log_4(3)} = O(n^{0.793}) \)

   \( f(n) = \Omega(n^{\log_4(3)+\varepsilon}), \) where \( \varepsilon \approx 0.2 \)

   regularity: for sufficiently large \( n, \)
   
   \[ af(n/b) = 3(n/4) \log(n/4) \leq (3/4)n \log n = cf(n) \text{ for } c = 3/4 \]

   case 3 applies that \( T(n) = \Theta(n \log n) \)
3. \( T(n) = 3T(n/4) + n \log n \)
   
   \( a = 3, \ b = 4, \) and \( f(n) = n \log n \)
   
   Thus, \( n^{\log_{b}(a)} = n^{\log_{4}(3)} = O(n^{0.793}) \)
   
   \( f(n) = \Omega(n^{\log_{4}(3)+\varepsilon}), \) where \( \varepsilon \approx 0.2 \)
   
   regularity : for sufficiently large \( n, \)
   
   \[ af(n/b) = 3(n/4) \log(n/4) \leq (3/4)n \log n = cf(n) \] for \( c = 3/4 \)
   
   case 3 applies that \( T(n) = \Theta(n \log n) \)

4. \( T(n) = 2T(n/2) + n \log n \)
3. \( T(n) = 3T(n/4) + n \log n \)
   
   \( a = 3, \ b = 4, \) and \( f(n) = n \log n \)
   
   Thus, \( n^{\log_b(a)} = n^{\log_4(3)} = O(n^{0.793}) \)
   
   \( f(n) = \Omega(n^{\log_4(3)+\epsilon}), \text{ where } \epsilon \approx 0.2 \)
   
   regularity: for sufficiently large \( n, \)
   
   \[ af(n/b) = 3(n/4) \log(n/4) \leq (3/4)n \log n = cf(n) \text{ for } c = 3/4 \]
   
   case 3 applies that \( T(n) = \Theta(n \log n) \)

4. \( T(n) = 2T(n/2) + n \log n \)
   
   \( a = 2, \ b = 2, \) and \( f(n) = n \log n \)
   
   Thus, \( n^{\log_b(a)} = n^{\log_2(2)} = n \)
   
   But, unfortunately, there is no \( \epsilon \) such that \( n \log n \) is polynomially larger than \( n \)
   
   The Master theorem does not work!
Let us iteratively substitute the recurrence:

\[ T(n) = aT(n/b) + f(n) \]
\[ = a(aT(n/b^2) + f(n/b)) + f(n) \]
\[ = a^2T(n/b^2) + af(n/b) + f(n) \]
\[ = a^3T(n/b^3) + a^2f(n/b^2) + af(n/b) + f(n) \]
\[ \vdots \]
\[ = a^{\log_b(n)-1}T(1) + \sum_{i=0}^{\log_b(n)-1} a^i f(n/b^i) \]
\[ = n^{\log_b(a)}T(1) + \sum_{i=0}^{\log_b(n)-1} a^i f(n/b^i) \]
Thus, we obtained

\[ T(n) = n^{\log_b(a)}T(1) + \sum_{i=0}^{\log_b(n)-1} a^i f(n/b^i) \]

The proof proceeds by distinguishing three cases:

1. The first term in dominant: \( f(n) = O(n^{\log_b(a) - \epsilon}) \)
2. Each part of the summation is equally dominant: \( f(n) = \Theta(n^{\log_b(a)}) \)
3. The summation can be bounded by a geometric series:
   \[ f(n) = \Omega(n^{\log_b(a)+\epsilon}) \] and the regularity of \( f(n) \) is key to make the argument work
Fast Integer Multiplication
Elementary school algorithm (in binary)

101001 = 41
x 101010 = 42

---------------------------------
1010010
1010010
1010010

---------------------------------
11010111010 = 1722
Elementary school algorithm (in binary)

101001 = 41
x 101010 = 42

-----------------------------
1010010
1010010
1010010

-----------------------------
11010111010 = 1722

Scan second number from right to left
Whenever you see a 1, add the first number to the result shifted by the appropriate number of bits
The multiplication of two $n$ bits numbers takes $\Omega(n^2)$ time using the elementary school algorithm.

Can we do better?

Andrey Kolmogorov conjectured in one of his seminars that one cannot, but was proved wrong by Anatoly Karatsuba.
Let's split the two integers X and Y into two parts: their most significant part and their least significant part

- \( X = 2^{n/2}A + B \) (where A and B are \( n/2 \) bit integers)
- \( Y = 2^{n/2}C + D \) (where C and D are \( n/2 \) bit integers)

Then,

\[
XY = 2^nAC + 2^{n/2}BC + 2^{n/2}AD + BD
\]
Multiplication by $2^x$ can be done in hardware with very low cost (just a shift)

We can apply this algorithm recursively
We replaced one multiplication of n bits numbers by four multiplications of n/2 bits numbers and 3 shifts and 3 additions

Thus, $T(n) = 4T(n/2) + cn$
Solve the Recurrence

\[ T(n) = 4T(n/2) + cn \]

By the Master theorem, \( g(n) = n^{\log_2(4)} = n^2 \)

Since \( cn = O(n^{2-\varepsilon}) \), we can conclude that \( T(n) = \Theta(n^2) \)
Suppose that we are able to reduce the number of multiplications from 4 to 3, allowing for more additions and shifts (but still a constant number)

Then, $T(n) = 3T(n/2) + dn$

Master theorem: $dn = O(n^{\log_2(3)-\epsilon})$, so we get $T(n) = \Theta(n^{\log_2(3)}) = O(n^{1.585})$
Let’s rewrite
\[ XY = 2^nAC + 2^{n/2}BC + 2^{n/2}AD + BD \]
in the form
\[ XY = (2^n - 2^{n/2})AC + 2^{n/2}(A + B)(C + D) + (1 - 2^{n/2})BD \]

Now, we have three multiplications with several shifts and additions

Then, \( T(n) = 3T(n/2) + dn \) with \( T(n) = \Theta(n^{\log_2(3)}) = O(n^{1.585}) \)
Split input $X$ into two parts $A$ and $B$ such that

$$X = 2^{n/2}A + B$$

Split input $Y$ into two parts $C$ and $D$ such that

$$Y = 2^{n/2}C + D$$

Then, calculate $AC$, $(A+B)(C+D)$, $BD$

Copy and shift the results, and add/subtract:

$$XY = (2^n - 2^{n/2})AC + 2^{n/2}(A + B)(C + D) + (1 - 2^{n/2})BD$$
Strassen’s Matrix Multiplication
Consider two $n \times n$ matrices $A$ and $B$

Recall that the matrix product $C = AB$ of two $n \times n$ matrices is defined as the $n \times n$ matrix that the coefficient

$$c_{kl} = \sum_m a_{km} b_{ml}$$

in row $k$ and column $l$, where the sum ranges over the integers from 1 to $n$; the scalar product of the $k^{th}$ row of $A$ with the $l^{th}$ column of $B$

The straightforward algorithm uses $O(n^3)$ scalar operations

Can we do better?
Let us write the product $AB = C$ as follows

\[
\begin{array}{ccc}
A_0 & A_1 & \times & B_0 & B_1 \\
A_2 & A_3 & & B_2 & B_3 \\
\end{array}
\]

\[
= \begin{array}{cccc}
A_0x B_0 & + & A_1x B_2 & A_0x B_1 & + & A_1x B_3 \\
A_2x B_0 & + & A_3x B_2 & A_2x B_1 & + & A_3x B_3 \\
\end{array}
\]

Divide $n \times n$ matrices $A$ and $B$ into four $n/2 \times n/2$ submatrices each

We have 8 smaller matrix multiplications and 4 additions

Is it faster?
Let us investigate this recursive version of the matrix multiplication

Since we divide A, B and C into 4 submatrices each, we can compute the resulting matrix C by

- 8 matrix multiplications on the submatrices of A and B
- plus $\Theta(n^2)$ scalar operations
Running time of recursive version of straightforward algorithm is

\[ T(n) = 8T(n/2) + \Theta(n^2) \text{ and } T(2) = \Theta(1) \]

where \( T(n) \) is running time on an \( n \times n \) matrix

Case 1 of the Master theorem gives us

\[ T(n) = \Theta(n^3) \]

Can we do fewer recursive calls (fewer multiplications of the \( n/2 \times n/2 \) submatrices)?
Strassen found a way to get all the required information with only 7 matrix multiplications, instead of 8

Recurrence for new algorithm is

\[ T(n) = 7T(n/2) + \Theta(n^2) \]
### Strassen’s Matrix Multiplication

\[
\begin{array}{cc}
A_{11} & A_{12} \\
A_{21} & A_{22} \\
\end{array}
\times
\begin{array}{cc}
B_{11} & B_{12} \\
B_{21} & B_{22} \\
\end{array}
=
\begin{array}{cc}
C_{11} & C_{12} \\
C_{21} & C_{22} \\
\end{array}
\]

\[
P_1 = A_{11} \times (B_{12} - B_{22})
\]
\[
P_2 = (A_{11} + A_{12}) \times B_{22}
\]
\[
P_3 = (A_{21} + A_{22}) \times B_{11}
\]
\[
P_4 = A_{22} \times (B_{21} - B_{11})
\]
\[
P_5 = (A_{11} + A_{22}) \times (B_{11} + B_{22})
\]
\[
P_6 = (A_{12} - A_{22}) \times (B_{21} + B_{22})
\]
\[
P_7 = (A_{11} - A_{21}) \times (B_{11} + B_{12})
\]
\[
C_{11} = P_5 + P_4 - P_2 + P_6
\]
\[
C_{12} = P_1 + P_2
\]
\[
C_{21} = P_3 + P_4
\]
\[
C_{22} = P_5 + P_1 - P_3 - P_7
\]
Applying the Master Theorem to

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

Since \( f(n) = O(n^{\log_b(a)-\varepsilon}) = O(n^{\log_2(7)-\varepsilon}) \),

case 1 of the Master theorem applies and we get

\[ T(n) = \Theta(n^{\log_b(a)}) = \Theta(n^{\log_2(7)}) = O(n^{2.81}) \]
Discussion of Strassen’s Algorithm

Not always practical

- constant factor is larger than for naive method
- specially designed methods are better on sparse matrices
- issues of numerical (in)stability
- recursion uses lots of space

Not the fastest known method

- Fastest known is $O(n^{2.3727})$
  (Winograd-Coppersmith algorithm improved by V. Williams)
- Best known lower bound is $\Omega(n^2)$