Divide-and-Conquer

Dr. Yingwu Zhu
Divide-and-Conquer

The most-well known algorithm design technique:

1. Divide instance of problem into two or more smaller instances

2. Solve smaller instances *independently* and recursively
   - When to stop?

3. Obtain solution to original (larger) instance by *combining* these solutions
Divide-and-Conquer Technique (cont.)

- Divide
  - subproblem 1 of size $n/2$
    - a solution to subproblem 1
  - subproblem 2 of size $n/2$
    - a solution to subproblem 2

- a problem of size $n$
  - a solution to the original problem
A General Template

//S is a large problem with input size of n

Algorithm divide_and_conquer(S)
    if (S is small enough to handle)
        solve it //conquer
    else
        split S into two (equally-sized) subproblems S₁ and S₂
        divide_and_conquer(S₁)
        divide_and_conquer(S₂)
        combine solutions to S₁ and S₂
    endif
End
General Divide-and-Conquer Recurrence

• Recursive algorithms are a natural fit for divide-and-conquer
  – Distinguish from Dynamic Programming
• Recall in Chapter 2, algorithm efficiency analysis for recursive algorithms
  – Key: Recurrence Relation
  – Solve: backward substitution, often cumbersome!
General Divide-and-Conquer Recurrence

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

where \( f(n) \in \Theta(n^d), \ d \geq 0, f(n) \) accounts for the time spent on dividing the problem into smaller ones and combining their solutions.

**Master Theorem:**
- If \( a < b^d, \ T(n) \in \Theta(n^d) \)
- If \( a = b^d, \ T(n) \in \Theta(n^d \log n) \)
- If \( a > b^d, \ T(n) \in \Theta(n^{\log_b a}) \)

Note: The same results hold with \( O \) instead of \( \Theta \).

Examples:
- \( T(n) = 4T(n/2) + n \Rightarrow T(n) \in ? \)
- \( T(n) = 4T(n/2) + n^2 \Rightarrow T(n) \in ? \)
- \( T(n) = 4T(n/2) + n^3 \Rightarrow T(n) \in ? \)
Divide-and-Conquer Examples

- Sorting: mergesort and quicksort
- Binary tree traversals
- Binary search (degenerated)
- Multiplication of large integers
- Matrix multiplication: Strassen’s algorithm
- Closest-pair and convex-hull algorithms
Divide-and-Conquer Examples

• Sorting: mergesort and quicksort

• Binary tree traversals

• Binary search (degenerated)

• Multiplication of large integers

• Matrix multiplication: Strassen’s algorithm

• Closest-pair and convex-hull algorithms
Mergesort

• Question #1: what makes mergesort distinct from many other sorting algorithms?

  internal and external algorithm

• Question #2: How to design mergesort using divide-and-conquer technique?
Mergesort

• Split array \( A[0..n-1] \) in two about equal halves and make copies of each half in arrays \( B \) and \( C \)

• Sort arrays \( B \) and \( C \) recursively
  • Q: when to stop?

• Merge sorted arrays \( B \) and \( C \) into array \( A \) as follows:
  – Repeat the following until no elements remain in one of the arrays:
    • compare the first elements in the remaining unprocessed portions of the arrays
    • copy the smaller of the two into \( A \), while incrementing the index indicating the unprocessed portion of that array
  – Once all elements in one of the arrays are processed, copy the remaining unprocessed elements from the other array into \( A \).
Pseudocode of Mergesort

**ALGORITHM** \( Mergesort(A[0..n-1]) \)

// Sorts array \( A[0..n-1] \) by recursive mergesort
// Input: An array \( A[0..n-1] \) of orderable elements
// Output: Array \( A[0..n-1] \) sorted in nondecreasing order

\[ \textbf{if} \ n > 1 \]

- \( \text{copy} \ A[0..\lfloor n/2 \rfloor - 1] \) to \( B[0..\lfloor n/2 \rfloor - 1] \)
- \( \text{copy} \ A[\lfloor n/2 \rfloor..n-1] \) to \( C[0..\lfloor n/2 \rfloor - 1] \)
- \( Mergesort(B[0..\lfloor n/2 \rfloor - 1]) \)
- \( Mergesort(C[0..\lfloor n/2 \rfloor - 1]) \)
- \( \text{Merge}(B, C, A) \)
ALGORITHM  
\text{Merge}(B[0..p - 1], C[0..q - 1], A[0..p + q - 1])

//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p - 1]$ and $C[0..q - 1]$ both sorted
//Output: Sorted array $A[0..p + q - 1]$ of the elements of $B$ and $C$

$i \leftarrow 0$; $j \leftarrow 0$; $k \leftarrow 0$

\textbf{while} $i < p$ \textbf{and} $j < q$ \textbf{do}

\hspace{1em} \textbf{if} $B[i] \leq C[j]$

\hspace{2em} $A[k] \leftarrow B[i]$; $i \leftarrow i + 1$

\hspace{2em} \textbf{else} $A[k] \leftarrow C[j]$; $j \leftarrow j + 1$

\hspace{1em} $k \leftarrow k + 1$

\hspace{1em} \textbf{if} $i = p$

\hspace{2em} \text{copy $C[j..q - 1]$ to $A[k..p + q - 1]$}

\hspace{1em} \textbf{else} \text{copy $B[i..p - 1]$ to $A[k..p + q - 1]$}
Mergesort Example
Analysis of Mergesort

• Time efficiency by recurrence relation:
  \[ T(n) = 2T(n/2) + f(n) \]
  \( n-1 \) comparisons in merge operation for worst case!
  \[ T(n) = \Theta(n \log n) \]

• Number of comparisons in the worst case is close to theoretical minimum for comparison-based sorting:
  \[ \lceil \log_2 n! \rceil \approx n \log_2 n - 1.44n \]  (Section 11.2)

• Space requirement: \( \Theta(n) \) (not in-place)

• Can be implemented without recursion (bottom-up)
Mergsort: A big picture

• **Problem**: Assume you want to sort $X$ terabytes (even perabytes) of data using a cluster of $M$ (even thousands) computers.

• **Solution**?
Similar to Search Engines in some aspects: data partitioning!
Applications in My Recent Research Work

"Measurement and Analysis of An Online Content Voting Network: A Case Study of Digg",

- In 19th International World Wide Web Conference (WWW2010)
- More interesting to you: I applied divide-and-conquer techniques
  - Use PageRank algorithm (112 lines of Python code) to rank users and thus weigh their votes on content (Matrix manipulations) to defend against vote spam
a. Write a pseudocode for a divide-and-conquer algorithm for finding the position of the largest element in an array of n numbers.

b. Set up and solve a recurrence relation for the number of key comparisons made by your algorithm.

c. How does this algorithm compare with the brute-force algorithm for this problem?
Quicksort

• Select a *pivot* (partitioning element) – here, the first element for simplicity!

• Rearrange the list so that all the elements in the first $s$ positions are smaller than or equal to the pivot and all the elements in the remaining $n-s$ positions are larger than the pivot (see next slide for an algorithm)

• Exchange the pivot with the last element in the first (i.e., $\leq$) subarray — *the pivot is now in its final position*

• Sort the two subarrays recursively
Quicksort

• Basic operation: split/divide
  – Differ from the divide operation in mergesort
  – What is the major difference?
Quicksort

• Basic operation: split/divide
  – Differ from the divide operation in mergesort
  – What is the major difference?
    • Each split will place the pivot in the right position, and the left sublist < the right sublist
    • No explicit merge
Partitioning Algorithm

Algorithm Partition($A[l..r]$)

// Partitions a subarray by using its first element as a pivot
// Input: A subarray $A[l..r]$ of $A[0..n-1]$, defined by its left and right
// indices $l$ and $r$ ($l < r$)
// Output: A partition of $A[l..r]$, with the split position returned as
// this function's value

$p \leftarrow A[l]$
$i \leftarrow l; \quad j \leftarrow r + 1$

repeat
    repeat $i \leftarrow i + 1$ until $A[i] > p$
    repeat $j \leftarrow j - 1$ until $A[j] \leq p$
    swap($A[i], A[j]$)
until $i \geq j$

swap($A[l], A[j]$)
return $j$
Quicksort Example

8, 2, 13, 5, 14, 3, 7
Analysis of Quicksort

• Best case: $T(n) =$?
• Worst case: $T(n) =$?
Analysis of Quicksort

• Best case: split in the middle — $\Theta(n \log n)$
• Worst case: sorted array! — $\Theta(n^2)$
• Average case: random arrays — $\Theta(n \log n)$
  — Assume the split can happen in each position with equal probability! See textbook for details!

• Improvements:
  – better pivot selection: median-of-three partitioning
  – switch to insertion sort on small sublists
  – elimination of recursion
  These combination makes 20-25% improvement

• Considered the method of choice for internal sorting of large files ($n \geq 10000$)
Questions for Quicksort

• Q1: How to implement the median-of-three rule by reusing the previous implementation of the split operation?
• Q2: How to implement a non-recursive quicksort?
Divide-and-Conquer Examples

- Sorting: mergesort and quicksort
- Binary search (degenerated)
- Binary tree traversals
- Multiplication of large integers
- Matrix multiplication: Strassen’s algorithm
- Closest-pair and convex-hull algorithms
Binary Search

Very efficient algorithm for searching in sorted array:

\[ K \]

vs

\[ A[0] \ldots A[m] \ldots A[n-1] \]

If \( K = A[m] \), stop (successful search); otherwise, continue searching by the same method in \( A[0..m-1] \) if \( K < A[m] \) and in \( A[m+1..n-1] \) if \( K > A[m] \)

\( l \leftarrow 0; \quad r \leftarrow n-1 \)

**while** \( l \leq r \) **do**

\[ m \leftarrow \lceil (l+r)/2 \rceil \]

if \( K = A[m] \) return \( m \)

else if \( K < A[m] \) \( r \leftarrow m-1 \)

else \( l \leftarrow m+1 \)

return -1
Analysis of Binary Search

• Time efficiency
  – worst-case recurrence: \( T_w(n) = 1 + T_w(\lfloor n/2 \rfloor) \), \( T_w(1) = 1 \)
  solution: \( T_w(n) = \lceil \log_2(n+1) \rceil \)

  This is VERY fast: e.g., \( T_w(10^6) = 20 \)

• Optimal for searching a sorted array

• Limitations
  • must be a sorted array
  • direct access to each element (not linked list)

• Bad (degenerate) example of divide-and-conquer
Divide-and-Conquer Examples

- Sorting: mergesort and quicksort
- Binary search (degenerated)
- Binary tree traversals
- Multiplication of large integers
- Matrix multiplication: Strassen’s algorithm
- Closest-pair and convex-hull algorithms
Binary Tree Algorithms

Binary tree is a divide-and-conquer ready structure!

Ex. 1: Classic traversals (preorder, inorder, postorder)

Algorithm Inorder(T)

if T ≠ ∅

   Inorder(T_{left})

   print(root of T)

   Inorder(T_{right})

Efficiency: Θ(n)

![Binary Tree Example](image)

external node: □

# of □: x = n+1
Ex. 2: Computing the height of a binary tree
Ex. 2: Computing the height of a binary tree

\[ h(T) = \max\{h(T_L), h(T_R)\} + 1 \text{ if } T \neq \emptyset \text{ and } h(\emptyset) = 0 \]

Efficiency: \( \Theta(n) \)
Divide-and-Conquer Examples

- Sorting: mergesort and quicksort
- Binary search (degenerated)
- Binary tree traversals
- Multiplication of large integers
- Matrix multiplication: Strassen’s algorithm
- Closest-pair and convex-hull algorithms
Multiplication of Large Integers

Consider the problem of multiplying two (large) \( n \)-digit integers represented by arrays of their digits such as:

\[ A = 12345678901357986429 \]
\[ B = 87654321284820912836 \]
Multiplication of Large Integers

Consider the problem of multiplying two (large) \( n \)-digit integers represented by arrays of their digits such as:

\[ A = 12345678901357986429 \quad B = 87654321284820912836 \]

The grade-school algorithm:

\[
\begin{align*}
& a_1 \ a_2 \ \ldots \ a_n \\
& b_1 \ b_2 \ \ldots \ b_n \\
& (d_{10}) \ d_{11} d_{12} \ \ldots \ d_{1n} \\
& (d_{20}) \ d_{21} d_{22} \ \ldots \ d_{2n} \\
& \ldots \ \ldots \ \ldots \ \ldots \ \ldots \\
& (d_{n0}) \ d_{n1} d_{n2} \ \ldots \ d_{nn}
\end{align*}
\]

Efficiency: \( n^2 \) one-digit multiplications
Multiplication of Large Integers

Consider the problem of multiplying two (large) \( n \)-digit integers represented by arrays of their digits such as:

\[
A = 12345678901357986429 \\
B = 87654321284820912836
\]

Discussion: How to apply “divide-and-conquer” to this problem?
A small example: \( A \times B \) where \( A = 2135 \) and \( B = 4014 \)

\[
A = (21 \cdot 10^2 + 35), \quad B = (40 \cdot 10^2 + 14)
\]

So, \( A \times B = (21 \cdot 10^2 + 35) \times (40 \cdot 10^2 + 14) \)

\[
= 21 \times 40 \cdot 10^4 + (21 \times 14 + 35 \times 40) \cdot 10^2 + 35 \times 14
\]

In general, if \( A = A_1A_2 \) and \( B = B_1B_2 \) (where \( A \) and \( B \) are \( n \)-digit, \( A_1, A_2, B_1, B_2 \) are \( n/2 \)-digit numbers),

\[
A \times B = A_1 \times B_1 \cdot 10^n + (A_1 \times B_2 + A_2 \times B_1) \cdot 10^{n/2} + A_2 \times B_2
\]

Recurrence for the number of one-digit multiplications \( T(n) \):

\[
T(n) = 4T(n/2), \quad T(1) = 1
\]

Solution: \( T(n) = n^2 \)
A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2

The idea is to decrease the number of multiplications from 4 to 3:

\[(A_1 + A_2) \cdot (B_1 + B_2) = A_1 \cdot B_1 + (A_1 \cdot B_2 + A_2 \cdot B_1) + A_2 \cdot B_2,\]

i.e., \((A_1 \cdot B_2 + A_2 \cdot B_1) = (A_1 + A_2) \cdot (B_1 + B_2) - A_1 \cdot B_1 - A_2 \cdot B_2,

which requires only 3 multiplications at the expense of (4-1) extra add/sub.

Recurrence for the number of multiplications T(n):

\[T(n) = 3T(n/2), \quad T(1) = 1\]

Solution: \(T(n) = 3^{\log 2^n} = n^{\log 2^3} \approx n^{1.585}\)
Large-Integer Multiplication

• Questions:
  • What if two large numbers have different number of digits?
  • What if $n$ is an odd number?
Divide-and-Conquer Examples

- Sorting: mergesort and quicksort
- Binary search (degenerated)
- Binary tree traversals
- Multiplication of large integers
- Matrix multiplication: Strassen’s algorithm
- Closest-pair and convex-hull algorithms
Strassen’s Matrix Multiplication

Strassen observed [1969] that the product of two matrices can be computed as follows:

\[
\begin{pmatrix}
C_{00} & C_{01} \\
C_{10} & C_{11}
\end{pmatrix}
= 
\begin{pmatrix}
A_{00} & A_{01} \\
A_{10} & A_{11}
\end{pmatrix}
\times 
\begin{pmatrix}
B_{00} & B_{01} \\
B_{10} & B_{11}
\end{pmatrix}
\]

\[
= 
\begin{pmatrix}
M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\
M_2 + M_4 & M_1 + M_3 - M_2 + M_6
\end{pmatrix}
\]

\[
M_1 = (A_{00} + A_{11}) \times (B_{00} + B_{11}) \\
M_2 = (A_{10} + A_{11}) \times B_{00} \\
M_3 = A_{00} \times (B_{01} - B_{11}) \\
M_4 = A_{11} \times (B_{10} - B_{00}) \\
M_5 = (A_{00} + A_{01}) \times B_{11} \\
M_6 = (A_{10} - A_{00}) \times (B_{00} + B_{01}) \\
M_7 = (A_{01} - A_{11}) \times (B_{10} + B_{11})
\]
Analysis of Strassen’s Algorithm

If \( n \) is not a power of 2, matrices can be padded with zeros.

Number of multiplications:

\[
T(n) = 7T(n/2), \quad T(1) = 1
\]

Solution: \( T(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807} \) vs. \( n^3 \) of brute-force alg.

Algorithms with better asymptotic efficiency are known but they are even more complex.
Divide-and-Conquer Examples

• Sorting: mergesort and quicksort
• Binary search (degenerated)
• Binary tree traversals
• Multiplication of large integers
• Matrix multiplication: Strassen’s algorithm
• Closest-pair and convex-hull algorithms
Closest-Pair Problem

- $S$ is a set of $n$ points $P_i=(x_i, y_i)$ in the plane
- For simplicity, $n$ is a power of two
- Without loss of generality, we assume points are ordered by their $x$ coordinates
- Discussion: How to apply divide-and-conquer?
Closest-Pair Problem by Divide-and-Conquer

Step 1  Divide the points given into two subsets $S_1$ and $S_2$ by a vertical line $x = c$ so that half the points lie to the left or on the line and half the points lie to the right or on the line.
Step 2  Find recursively the closest pairs for the left and right subsets.

Step 3  Set $d = \min\{d_1, d_2\}$

We can limit our attention to the points in the symmetric vertical strip of width $2d$ as possible closest pair. Let $C_1$ and $C_2$ be the subsets of points in the left subset $S_1$ and of the right subset $S_2$, respectively, that lie in this vertical strip. The points in $C_1$ and $C_2$ are stored in increasing order of their $y$ coordinates, which is maintained by merging during the execution of the next step.

Step 4  For every point $P(x,y)$ in $C_1$, we inspect points in $C_2$ that may be closer to $P$ than $d$. There can be no more than 6 such points (because $d \leq d_2$)!
Closest Pair by Divide-and-Conquer: Worst Case

The worst case scenario is depicted below:
Efficiency of the Closest-Pair Algorithm

Running time of the algorithm is described by

\[ T(n) = 2T(n/2) + f(n), \text{ where } f(n) \in O(n) \]

By the Master Theorem (with \( a = 2, b = 2, d = 1 \))

\[ T(n) \in O(n \log n) \]