Divide and Conquer

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
In this lecture, we will discuss the divide and conquer technique for designing algorithms with strong performance guarantees. Our discussion will be based on the following problems:

1. Sorting (a review of merge sort)
2. Counting inversions
3. Dominance counting
4. Matrix multiplication
Principle of divide and conquer:

Divide a problem into sub-problems, solve the sub-problems by recursion, and derive the final answer from the sub-problems’ outputs.
Sorting
**Problem:** Given an array $A$ of $n$ distinct integers, produce another array where the same integers have been arranged in ascending order.

- **Divide:** Let $A_1$ the array containing the first $\lceil n/2 \rceil$ elements of $A$, and $A_2$ be the array containing the other elements of $A$. Sort $A_1$ and $A_2$ recursively.

- **Conquer:** Merge the two sorted arrays $A_1$ and $A_2$ in ascending order. This can be done in $O(n)$ time.

This is the merge sort algorithm.
**Sorting**

**Running Time:** Let \( f(n) \) denote the worst-case cost of the algorithm on an array of size \( n \). Then:

\[
f(n) \leq 2 \cdot f(\lceil n/2 \rceil) + O(n)
\]

which gives \( f(n) = O(n \log n) \).
Counting Inversions
Counting Inversions

Let: \( A \) = an array of \( n \) distinct integers.

An inversion is a pair of \((i, j)\) such that

- \( 1 \leq i < j \leq n \), and

**Example:** Consider \( A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6) \).
Then \((1, 2)\) is an inversion because \( A[1] = 10 > A[2] = 3 \). So are \((1, 3), (3, 4), (4, 5), \) and so on.
There are in total 29 inversions.

**Think:** How many inversions can there be in the worst case?

**Answer:** \( \binom{n}{2} = \Theta(n^2) \).
Counting Inversions

Problem: Given an array $A$ of $n$ distinct integers, count the number of inversions.

We will do in the class: $O(n \log^2 n)$ time.
You will do as an exercise: $O(n \log n)$ time.
Counting Inversions

- **Divide:** Let $A_1$ be the array containing the first $\lceil n/2 \rceil$ elements of $A$, and $A_2$ be the array containing the other elements of $A$. Solve the “counting inversions” problem recursively on $A_1$ and $A_2$, respectively. By doing so, we have already obtained the number $m_1$ of inversions in $A_1$, and similarly, the number $m_2$ for $A_2$.

- **Conquer:** It remains to count the number of crossing inversions $(i, j)$ where $i \in A_1$ and $j \in A_2$. 

Counting Inversions

$A_1 = \text{the array containing the first } \lceil n/2 \rceil \text{ elements of } A$

$A_2 = \text{the array containing the other elements of } A.$

Sort $A_1$ and $A_2$.

For each element $e \in A_1$, count how many crossing inversions $e$ produces using binary search.

**Example (cont.):** $A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6)$.

$A_1 = (2, 3, 8, 9, 10), \ A_2 = (1, 4, 5, 6, 7)$

Element 2 produces 1 crossing inversion
Element 3 produces 1, too.
Elements 8, 9, and 10 each produce 5 crossing inversions.

**Think:** How to obtain each count with binary search?

In total, $n/2$ binary searches are performed, which takes $O(n \log n)$ time.
Counting Inversions

**Running Time:** Let $f(n)$ denote the worst-case cost of the algorithm on an array of size $n$. Then:

$$f(n) \leq 2 \cdot f(\lceil n/2 \rceil) + O(n \log n)$$

which gives $f(n) = O(n \log^2 n)$. 
Dominance Counting
Denote by $\mathbb{Z}$ the set of integers. Given a point $p$ in two-dimensional space $\mathbb{Z}^2$, denote by $p[1]$ and $p[2]$ its x- and y-coordinate, respectively.

Given two distinct points $p$ and $q$, we say that $q$ dominates $p$ if $p[1] \leq q[1]$ and $p[2] \leq q[2]$; see the figure below:

\begin{center}
$\bullet$ $q$
\end{center}

\begin{center}
$\bullet$ $p$
\end{center}
Let $P$ be a set of $n$ points in $\mathbb{Z}^2$ with distinct $x$-coordinates. Find, for each point $p \in P$, the number of points in $P$ that are dominated by $p$.

**Example:**

We should output: $(p_1, 0), (p_2, 1), (p_3, 0), (p_4, 2), (p_5, 2), (p_6, 5), (p_7, 2), (p_8, 0)$. 
Dominance Counting

Let $P$ be a set of $n$ points in $\mathbb{Z}^2$ with distinct $x$-coordinates. Find, for each point $p \in P$, the number of points in $P$ that are dominated by $p$.

We will do in the class: $O(n \log^2 n)$ time.
You will do as an exercise: $O(n \log n)$ time.
Dominance Counting

**Divide:** Find a vertical line $\ell$ such that $P$ has $\lceil n/2 \rceil$ points on each side of the line.

**Example:**

![Diagram showing points $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8$ with a vertical line $\ell$.]

**Think:** How to find such $\ell$ in $O(n \log n)$ time? How about $O(n)$ time?
Dominance Counting

**Divide:**

\[ P_1 = \text{the set of points of } P \text{ on the left of } \ell \]
\[ P_2 = \text{the set of points of } P \text{ on the right of } \ell \]

**Example:**

\[ P_1 = \{ p_1, p_2, p_3, p_4 \} \]
\[ P_2 = \{ p_5, p_6, p_7, p_8 \} . \]
Dominance Counting

**Divide:**
Solve the dominance counting problem on $P_1$ and $P_2$ separately.

**Example:**

On $P_1$, we have obtained: $(p_1, 0), (p_2, 1), (p_3, 0), (p_4, 2)$.

On $P_2$, we have obtained: $(p_5, 0), (p_6, 1), (p_7, 0), (p_8, 0)$.

The counts obtained for the points in $P_1$ are final (think: why?).
Dominance Counting

Conquer:
It remains to count, for each point $p_2 \in P_2$, how many points in $P_1$ it dominates.

Example:

On $P_2$, we have obtained: $(p_5, 0), (p_6, 1), (p_7, 0), (p_8, 0)$.

Regarding $p_5$, for example, we still need to find out that it dominates 2 points from $P_1$.

The x-coordinates do not matter any more!
Dominance Counting

Conquer:

Sort $P_1$ by $y$-coordinate.
Then, for each point $p_2 \in P_2$, we can obtain the number points in $P_1$ dominated by $p_2$ using binary search.

**Example:**

$P_1$ in ascending of $y$-coordinate: $p_3, p_1, p_4, p_2$.

How to perform binary search to obtain the fact that $p_5$ dominates 2 points in $P_1$?

- Search using the $y$-coordinate of $p_5$. 
Dominance Counting

Analysis:

Let $f(n)$ be the worst-case running time of the algorithm on $n$ points. Then:

$$f(n) \leq 2f(\lceil n/2 \rceil) + O(n \log n)$$

which solves to $f(n) = O(n \log^2 n)$. 
Matrix Multiplication
**Problem:** Given two $n \times n$ matrices $A$ and $B$, compute their product $AB$.

We store an $n \times n$ matrix with an array of length $n^2$ in “row-major” order.

**Example:** 
\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\] is stored as $(1, 2, 3, 4)$.

Note that any $A[i,j]$ — the element of $A$ at the $i$-th row and $j$-th column — can be accessed in $O(1)$ time.

**Trivial:** $O(n^3)$ time

We will do in the class: $O(n^{2.81})$ time for $n$ being a power of 2

You will do as an exercise: $O(n^{2.81})$ time for any $n$. ```
**Matrix Multiplication**

**Warm Up:** Suppose we want to compute \[
\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}
\begin{bmatrix}
e & f \\
g & h
\end{bmatrix}.
\] How many multiplication operations do we need to perform?

**Trivial:** 8.

**Non-trivial:** 7.

\[
\begin{bmatrix}
a & b \\
c & d
\end{bmatrix}
\begin{bmatrix}
e & f \\
g & h
\end{bmatrix} = \begin{bmatrix}
p_5 + p_4 - p_2 + p_6 & p_1 + p_2 \\
p_3 + p_4 & p_1 + p_5 - p_3 - p_7
\end{bmatrix}
\]

where

\[
\begin{align*}
p_1 &= a(f - h) \\
p_2 &= (a + b)h \\
p_3 &= (c + d)e \\
p_4 &= d(g - e) \\
p_5 &= (a + d)(e + h) \\
p_6 &= (b - d)(g + h) \\
p_7 &= (a - c)(e + f)
\end{align*}
\]
Matrix Multiplication (Strassen’s Algorithm)

Recall that the input $A$ and $B$ are order-$n$ (i.e., $n \times n$) matrices. Assume for simplicity that $n$ is a power of 2. Divide each of $A$ and $B$ into 4 submatrices of order $n/2$:

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

It is easy to verify:

$$
AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}
$$

How many order-$(n/2)$ matrix multiplications do we need?

**Trivial:** 8.

**Non-trivial:** 7 — see the next slide.
Matrix Multiplication

\[
\begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
  B_{11} & B_{12} \\
  B_{21} & B_{22}
\end{bmatrix}
= \begin{bmatrix}
  p_5 + p_4 - p_2 + p_6 & p_1 + p_2 \\
  p_3 + p_4 & p_1 + p_5 - p_3 - p_7
\end{bmatrix}
\]

\begin{align*}
p_1 &= A_{11}(B_{12} - B_{22}) \\
p_2 &= (A_{11} + A_{12})B_{22} \\
p_3 &= (A_{21} + A_{22})B_{11} \\
p_4 &= A_{22}(B_{21} - B_{11}) \\
p_5 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\
p_6 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\
p_7 &= (A_{11} - A_{21})(B_{11} + B_{12})
\end{align*}

If \( f(n) \) is the worst-case time of computing the product of two order-\( n \) matrices, then each of \( p_i \) (\( 1 \leq i \leq 7 \)) can be computed in \( f(n/2) + O(n^2) \) time.
Matrix Multiplication

Therefore:

\[ f(n) = 7f(n/2) + O(n^2) \]

which solves to \( f(n) = O(n^{\log_2 7}) = O(n^{2.81}) \).