CS 341: Algorithms
Lec 03: Divide and Conquer

Armin Jamshidpey   Collin Roberts

Based on lecture notes by Éric Schost and many previous CS 341 instructors

David R. Cheriton School of Computer Science, University of Waterloo

Winter 2025

Divide-and-Conquer

A general algorithmic paradigm (strategy):

- Divide: Split a problem into several subproblems.

Divide-and-Conquer

A general algorithmic paradigm (strategy):

- Divide: Split a problem into several subproblems.
- Conquer: Solve the subproblems (recursively) applying the same algorithm.

Divide-and-Conquer

A general algorithmic paradigm (strategy):

- Divide: Split a problem into several subproblems.
- Conquer: Solve the subproblems (recursively) applying the same algorithm.
- Combine: Use subproblem results to derive a final result for the original problem.

Divide-and-Conquer

A general algorithmic paradigm (strategy):

- Divide: Split a problem into several subproblems.
- Conquer: Solve the subproblems (recursively) applying the same algorithm.
- Combine: Use subproblem results to derive a final result for the original problem.

When can we use Divide and Conquer?

- Original problem is easily decomposable into subproblems (we do not want to see "overlap" in the subproblems).

When can we use Divide and Conquer?

- Original problem is easily decomposable into subproblems (we do not want to see "overlap" in the subproblems).
- Combining solutions is not too costly.

When can we use Divide and Conquer?

- Original problem is easily decomposable into subproblems (we do not want to see "overlap" in the subproblems).
- Combining solutions is not too costly.
- Subproblems are not overly unbalanced.

Counting inversions

Collaborative filtering:

- matches users preference (movies, music, ...)
- determine users with *similar* tastes
- recommends new things to users based on preferences of similar users

Padlet

The basis of collaborative filtering is ...

`https://padlet.com/arminjamshidpey/CS341`

Counting inversions

Goal: given an unsorted array $A[1..n]$, find the number of inversions in it.

Def: (i, j) is an inversion if $i < j$ and $A[i] > A[j]$

Example: with $A = [1, 5, 2, 6, 3, 8, 7, 4]$, we get

$$(2, 3), (2, 5), (2, 8), (4, 5), (4, 8), (6, 7), (6, 8), (7, 8)$$

Counting inversions

Goal: given an unsorted array $A[1..n]$, find the number of inversions in it.

Def: (i, j) is an inversion if $i < j$ and $A[i] > A[j]$

Example: with $A = [1, 5, 2, 6, 3, 8, 7, 4]$, we get

$$(2,3),(2,5),(2,8),(4,5),(4,8),(6,7),(6,8),(7,8)$$

Remark: we show the indices where inversions occur

Remark: easy algorithm (two nested loops) in $\Theta(n^2)$

Toward a divide-and-conquer algorithm

Idea (for n a power of two)

- c_ℓ = number of inversions in $A[1..n/2]$
- c_r = number of inversions in $A[n/2+1..n]$
- c_t = number of transverse inversions with $i \leq n/2$ and $j > n/2$
- return $c_\ell + c_r + c_t$

Toward a divide-and-conquer algorithm

Idea (for n a power of two)

- c_ℓ = number of inversions in $A[1..n/2]$
- c_r = number of inversions in $A[n/2+1..n]$
- c_t = number of transverse inversions with $i \le n/2$ and $j > n/2$
- return $c_\ell + c_r + c_t$

Example: with $A = [1, 5, 2, 6, 3, 8, 7, 4]$

- $c_\ell = 1$ $(2, 3)$
- $c_r = 3$ $(6, 7), (6, 8), (7, 8)$
- $c_t = 4$ $(2, 5), (2, 8), (4, 5), (4, 8)$

c_ℓ and c_r done recursively. What about c_t?

Transverse inversions

Goal: how many pairs (i, j) with $i \leq n/2$, $j > n/2$, $A[i] > A[j]$?

Remark: this number does not change if both sides are **sorted**

So assume that we sort left and right after the recursive calls.

Example: starting from $[1, 5, 2, 6, 3, 8, 7, 4]$, we get

[1, 2, 5, 6, 3, 4, 7, 8]

$c_t = \#i$'s greater than 3 + $\#i$'s greater than 4 +
$\#i$'s greater than 7 + $\#i$'s greater than 8

Option 1

Algorithm: take each $i \leq n/2$ and binary-search its position in the right-hand side.

- this is $O(\log(n))$ per i, so total $O(n\log(n))$
- plus another $O(n\log(n))$ for sorting left and right
- recurrence: $T(n) \leq 2T(n/2) + O(n\log(n))$
- gives $T(n) = O(n\log^2(n))$

Option 1

Algorithm: take each $i \leq n/2$ and binary-search its position in the right-hand side.

- this is $O(\log(n))$ per i, so total $O(n\log(n))$
- plus another $O(n\log(n))$ for sorting left and right
- recurrence: $T(n) \leq 2T(n/2) + O(n\log(n))$
- gives $T(n) = O(n\log^2(n))$

Sketchy proof:

$$\begin{aligned}
T(n) &\leq 2T(n/2) + n\log(n) \\
&\leq 4T(n/4) + n\log(n/2) + n\log(n) \\
&\leq 8T(n/8) + n\log(n/4) + n\log(n/2) + n\log(n) \\
&\leq \cdots \leq n(\log(n) + \log(n/2) + \cdots + \log(2)) \\
&\leq n\log^2(n)
\end{aligned}$$

Option 2: enhance mergesort

Idea: find c_t during merge.

Merge($A[1..n]$) (both halves of A assumed sorted)
1. copy A into a new array S
2. $i = 1$; $j = n/2 + 1$;
3. **for** $(k \leftarrow 1; k \leq n; k{+}{+})$ **do**
4. **if** $(i > n/2)$ $A[k] \leftarrow S[j{+}{+}]$
5. **else if** $(j > n)$ $A[k] \leftarrow S[i{+}{+}]$
6. **else if** $(S[i] < S[j])$ $A[k] \leftarrow S[i{+}{+}]$
7. **else** $A[k] \leftarrow S[j{+}{+}]$

Merge$(A[1..n])$ (both halves of A assumed sorted)

copy A into a new array S; $c = 0$

$i = 1$; $j = n/2 + 1$;

for $(k \leftarrow 1;\ k \leq n;\ k{+}{+})$ **do**

    if $(i > n/2)$ $A[k] \leftarrow S[j{+}{+}]$

    else if $(j > n)$ $A[k] \leftarrow S[i{+}{+}]$; $c = c + n/2$

    else if $(S[i] < S[j])$ $A[k] \leftarrow S[i{+}{+}]$; $c = c + j - (n/2 + 1)$

    else $A[k] \leftarrow S[j{+}{+}]$

```
Merge(A[1..n]) (both halves of A assumed sorted)
    copy A into a new array S;  c = 0
    i = 1; j = n/2 + 1;
    for (k ← 1;  k ≤ n;  k++) do
        if (i > n/2) A[k] ← S[j++]
        else if (j > n) A[k] ← S[i++]; c = c + n/2
        else if (S[i] < S[j]) A[k] ← S[i++]; c = c + j − (n/2 + 1)
        else A[k] ← S[j++]
```

Example: with [1, 2, 5, 6, 3, 4, 7, 8]

- when we insert 1 back into A, $j = 5$ so $c = c + 0$
- when we insert 2 back into A, $j = 5$ so $c = c + 0$
- when we insert 5 back into A, $j = 7$ so $c = c + 2$
- when we insert 6 back into A, $j = 7$ so $c = c + 2$

Enhanced merge is still $\Theta(n)$ so total remains $\Theta(n \log(n))$.

Multiplying polynomials

Goal: given $\boldsymbol{F = f_0 + \cdots + f_{n-1}x^{n-1}}$ and $\boldsymbol{G = g_0 + \cdots + g_{n-1}x^{n-1}}$, compute

$$H = FG = f_0 g_0 + (f_0 g_1 + f_1 g_0)x + \cdots + f_{n-1}g_{n-1}x^{2n-2}$$

1. **for** $i = 0, \ldots, n-1$ **do**
2.   **for** $j = 0, \ldots, n-1$ **do**
3. $h_{i+j} = h_{i+j} + f_i g_j$

Divide-and-conquer

Idea: write $\boldsymbol{F = F_0 + F_1 x^{n/2}}$, $\boldsymbol{G = G_0 + G_1 x^{n/2}}$. Then

$$H = F_0 G_0 + (F_0 G_1 + F_1 G_0) x^{n/2} + F_1 G_1 x^n$$

Analysis:

- 4 recursive calls in size $n/2$
- $\Theta(n)$: additions to compute $F_0 G_1 + F_1 G_0$ and etc.

Recurrence: $T(n) = 4T(n/2) + \Theta(n)$

- $a = 4$, $b = 2$, $y = 1$ so $T(n) = \Theta(n^2)$

Not better than the naive algorithm. We do the same operations.

Padlet

Use only one multiplication to write $F_0 G_1 + F_1 G_0$ in terms of $F_0, F_1, G_0, G_1, F_0 G_0, F_1 G_1$(assume $F_0 G_0, F_1 G_1$ are given).
`https://padlet.com/arminjamshidpey/CS341`

Karatsuba's algorithm

Idea: use the identity

$$(F_0 + F_1 x^{n/2})(G_0 + G_1 x^{n/2}) =$$

$$\boldsymbol{F_0 G_0} + ((\boldsymbol{F_0} + \boldsymbol{F_1})(\boldsymbol{G_0} + \boldsymbol{G_1}) - \boldsymbol{F_0 G_0} - \boldsymbol{F_1 G_1})x^{n/2} + \boldsymbol{F_1 G_1} x^n$$

Analysis:

- 3 recursive calls in size $n/2$
- $\Theta(n)$ additions to compute $F_0 + F_1$ and $G_0 + G_1$
- multiplications by $x^{n/2}$ and x^n are free
- $\Theta(n)$ additions and subtractions to combine the results

Recurrence: $T(n) = 3T(n/2) + \Theta(n)$

- $a = 3$, $b = 2$, $c = 1$ so $\boldsymbol{T(n) = \Theta(n^{\log_2(3)})}$
  $\log_2(3) = 1.58\ldots$

Karatsuba's algorithm

Idea: use the identity

$$(F_0 + F_1 x^{n/2})(G_0 + G_1 x^{n/2}) =$$

$$\boldsymbol{F_0 G_0} + ((\boldsymbol{F_0} + \boldsymbol{F_1})(\boldsymbol{G_0} + \boldsymbol{G_1}) - \boldsymbol{F_0 G_0} - \boldsymbol{F_1 G_1})x^{n/2} + \boldsymbol{F_1 G_1} x^n$$

Analysis:

- 3 recursive calls in size $n/2$
- $\Theta(n)$ additions to compute $F_0 + F_1$ and $G_0 + G_1$
- multiplications by $x^{n/2}$ and x^n are free
- $\Theta(n)$ additions and subtractions to combine the results

Recurrence: $T(n) = 3T(n/2) + \Theta(n)$

- $a = 3$, $b = 2$, $c = 1$ so $\boldsymbol{T(n) = \Theta(n^{\log_2(3)})}$
  $\log_2(3) = 1.58\ldots$

Remark: key idea = a formula for degree-1 polymomials that does **3** multiplications

Toom-Cook and FFT

Toom-Cook:

- a family of algorithms based on similar expressions as Karatsuba
- for $k \geq 2$, $2k-1$ recursive calls in size n/k
- so $\boldsymbol{T(n) = \Theta(n^{\log_k(2k-1)})}$
- gets as close to exponent 1 as we want (but very slowly)

FFT:

- if we use complex coefficients, FFT can be used to multiply polynomials
- FFT follows the same recurrence as merge sort, $T(n) = 2T(n/2) + \Theta(n)$
- so we can multiply polynomials in $\boldsymbol{\Theta(n \log(n))}$ ops over $\mathbb{C}$

Multiplying matrices

Goal: given $\boldsymbol{A = [a_{i,j}]_{1 \leq i,j \leq n}}$ and $\boldsymbol{B = [b_{j,k}]_{1 \leq j,k \leq n}}$ compute $C = AB$

Remark: input and output size $\Theta(n^2)$, easy algorithm in $\Theta(n^3)$

```
for i = 1, ..., n do
    for j = 1, ..., n do
        for k = 1, ..., n do
            c_{i,k} = c_{i,k} + a_{i,j} b_{j,k}
```

Divide-and-conquer

Setup: write

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \quad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}$$

with all $A_{i,k}, B_{i,j}$ of size $n/2 \times n/2$. Then

$$C = \begin{pmatrix} A_{1,1}B_{1,1} + A_{1,2}B_{2,1} & A_{1,1}B_{1,2} + A_{1,2}B_{2,2} \\ A_{2,1}B_{1,1} + A_{2,2}B_{2,1} & A_{2,1}B_{1,2} + A_{2,2}B_{2,2} \end{pmatrix}$$

Naively: 8 recursive calls in size $n/2$ + $\Theta(n^2)$ additions $\implies$ $T(n) = \Theta(n^3)$

Padlet

Can we do better than 8 recursive calls?

`https://padlet.com/arminjamshidpey/CS341`

Strassen's algorithm

Compute

$$\left|\begin{array}{lcl} Q_1 & = & (A_{1,1} - A_{1,2})B_{2,2} \\ Q_2 & = & (A_{2,1} - A_{2,2})B_{1,1} \\ Q_3 & = & A_{2,2}(B_{1,1} + B_{2,1}) \\ Q_4 & = & A_{1,1}(B_{1,2} + B_{2,2}) \\ Q_5 & = & (A_{1,1} + A_{2,2})(B_{2,2} - B_{1,1}) \\ Q_6 & = & (A_{1,1} + A_{2,1})(B_{1,1} + B_{1,2}) \\ Q_7 & = & (A_{1,2} + A_{2,2})(B_{2,1} + B_{2,2}) \end{array}\right. \quad \text{and} \quad \left|\begin{array}{lcl} C_{1,1} & = & Q_1 - Q_3 - Q_5 + Q_7 \\ C_{1,2} & = & Q_4 - Q_1 \\ C_{2,1} & = & Q_2 + Q_3 \\ C_{2,2} & = & -Q_2 - Q_4 + Q_5 + Q_6 \end{array}\right.$$

Analysis: 7 recursive calls in size $n/2$ + $\Theta(n^2)$ additions $\implies$ $\boldsymbol{T(n) = \Theta(n^{\log_2(7)})}$

$\log_2(7) = 2.80\ldots$

Padlet

Can we multiply two 2×2 matrices with less than 7 multiplications?

`https://padlet.com/arminjamshidpey/CS341`

What this means

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $T(n) \in \Theta(n^{\log_\ell(k)})$

What this means

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $T(n) \in \Theta(n^{\log_\ell(k)})$

Going beyond

- an algorithm that does k multiplications for matrices of size ℓ, m by m, p gives $T(n) \in \Theta(n^{3 \log_{\ell mp}(k)})$

What this means

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $T(n) \in \Theta(n^{\log_\ell(k)})$

Going beyond

- an algorithm that does k multiplications for matrices of size ℓ, m by m, p gives $T(n) \in \Theta(n^{3 \log_{\ell m p}(k)})$

Best exponent to date (using more than just divide-and-conquer)

- $O(n^{2.37188})$, improves from previous record $O(n^{2.37286})$
- galactic algorithms