Module 3 Divide and Conquer

Some slides borrowed from CS 240.
Thanks to Anna Lubiw and other previous CS 341 instructors.

- Divide and Conquer Algorithm Basics
- Examples from previous courses
- Counting Inversions
- Multiplying Large Numbers
- Matrix Multiplication
- Centrality of Matrix Multiplication
- Finding the Closest Pair

Divide and Conquer Algorithm Basics

In previous courses, we covered a few **Divide and Conquer** algorithms but maybe didn't specifically define the term.

Divide and Conquer algorithms are broken into 3 basic steps:

1. **Divide** - break the problem into smaller instances of the problem
2. **Recurse** - use recursion to solve the smaller problems
3. **Conquer** - combine the results of the smaller problems to solve the initial larger problem

Examples from previous courses

Binary Search: Search for an element k in a sorted array A.
Note: we may have implemented this iteratively but it has a natural recursive implementation as well.

- Compute the middle index m of A and compare with k.
- If $k = A[m]$ then return FOUND
  Else If k is smaller than $A[m]$ then recurse on left half of A
  Else (k is larger than $A[m]$) so recurse on right half of A

Binary Search only recurses on one of the subproblems and simply returns what the subproblem returns.

Analysis (worst-case): $T(n) = 1 + T(n/2)$ but this assumes n always divides evenly.
Rigorously, $T(n) = 1 + max\{T(\lfloor n/2 \rfloor), T(\lceil n/2 \rceil)\}$.
These resolve to: $T(n) \in O(\log n)$.

Examples from previous courses - Sorting

QuickSort(A):

- Partition array based on a given pivot $\Rightarrow O(n)$ work to divide.
- 2 Subproblems: Pivot divides A into Left and right subarrays, recurse on both.
- Conquer step is easy - does nothing if algorithm is "in-place".

Analysis:

Worst-case: $T(n) = O(n) + T(n-1) \in O(n^2)$
Best-case: $T(n) = O(n) + 2T(n/2) \in O(n \log n)$
Average Case: $O(n \log n)$
Randomized (each pivot choice is equally likely): Expected $O(n \log n)$

Examples from previous courses - Sorting

MergeSort(A):

- Divide array into left and right halves.
- 2 Subproblems: Left and right half subarrays, recurse on both.
- Merge the two sorted arrays $\Rightarrow O(n)$ work to conquer.

Analysis:

Best-case/Worst-case: $T(n) = 2T(n/2) + O(n) \in O(n \log n)$
Rigorously, $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + cn$

New Problem - Example

Some websites attempt to make suggestions (or target advertising) to you based on matching you with previous users and observing what they were interested in.

Similarity of users is based on similarity of preferences. Given a set of items A, B, C, D, a user ranks them by most favourite to least favourite.

Given 2 rankings, how do you compare similarity?

Ranking 1: B D C A

Ranking 2: A D B C

New Problem - Example

Some websites attempt to make suggestions (or target advertising) to you based on matching you with previous users and observing what they were interested in.

Similarity of users is based on similarity of preferences. Given a set of items A, B, C, D, a user ranks them by most favourite to least favourite.

Given 2 rankings, how do you compare similarity?

Ranking 1:	B D C A
Ranking 2:	A D B C

How many exact matches are there?

New Problem - Example

Some websites attempt to make suggestions (or target advertising) to you based on matching you with previous users and observing what they were interested in.

Similarity of users is based on similarity of preferences. Given a set of items A, B, C, D, a user ranks them by most favourite to least favourite.

Given 2 rankings, how do you compare similarity?

Ranking 1:	B D C A
Ranking 2:	A D B C

How many exact matches are there? 1 (only D at the same index)
Maybe try ordering of pairs? How many pairs of distinct items?

New Problem - Example

Some websites attempt to make suggestions (or target advertising) to you based on matching you with previous users and observing what they were interested in.

Similarity of users is based on similarity of preferences. Given a set of items A, B, C, D, a user ranks them by most favourite to least favourite.

Given 2 rankings, how do you compare similarity?

Ranking 1:	B D C A
Ranking 2:	A D B C

How many exact matches are there? 1 (only D at the same index)
Maybe try ordering of pairs? How many pairs of distinct items? $\binom{4}{2} = 6$

Are these 2 rankings similar? How many pairs (X, Y) have the same order?
2: Both prefer B over C and D over C $\rightarrow$ pairs BC and CD.
Not very similar - *4* pairs are inverted: AB AC AD BD, *not very similar*.

Example: Counting Inversions

Problem

Counting Inversions

Instance: *Given 2 rankings of items* $\{a_1, \ldots, a_n\}$

Find: *The number of inverted pairs of items between the rankings; i.e. pairs where one ranking prefers* a_i *over* a_j *but the other prefers* a_j *over* a_i.

Observe: If we draw edges between the same items in both rankings, the number of edge crossings is the number of inversions.

Example: Counting Inversions

Problem

Counting Inversions

Instance: *Given 2 rankings of items* $\{a_1, \ldots, a_n\}$

Find: *The number of inverted pairs of items between the rankings; i.e. pairs where one ranking prefers* a_i *over* a_j *but the other prefers* a_j *over* a_i*.*

Observe: If we draw edges between the same items in both rankings, the number of edge crossings is the number of inversions.

An equivalent formulation is to assign numbers to each item, then compare the order of numbers. For simplicity, assign the numbers in order to the first ranking:

B D C A $\Rightarrow$ 1 2 3 4

Using the same mapping, the second ranking becomes:

A D B C

Example: Counting Inversions

Problem

Counting Inversions

Instance: *Given 2 rankings of items* $\{a_1, \ldots, a_n\}$

Find: *The number of inverted pairs of items between the rankings; i.e. pairs where one ranking prefers* a_i *over* a_j *but the other prefers* a_j *over* a_i*.*

Observe: If we draw edges between the same items in both rankings, the number of edge crossings is the number of inversions.

An equivalent formulation is to assign numbers to each item, then compare the order of numbers. For simplicity, assign the numbers in order to the first ranking:

B D C A $\Rightarrow$ 1 2 3 4

Using the same mapping, the second ranking becomes:

A D B C $\Rightarrow$ 4 2 1 3

The problem of counting the number of inversions now becomes ...

Example: Counting Inversions

The problem of counting the number of inversions now becomes:
count the number of pairs that are out of order in the 2nd list.

Brute Force:

Example: Counting Inversions

The problem of counting the number of inversions now becomes:
count the number of pairs that are out of order in the 2nd list.

Brute Force: Check all $\binom{n}{2}$ pairs, requires $O(n^2)$ time.
Does sorting help? Not really since we check all pairs.

Divide and Conquer Method: Given a list $L = a_1, \ldots, a_n$ of numbers, count the number of inversions.

Example: Counting Inversions

The problem of counting the number of inversions now becomes:
count the number of pairs that are out of order in the 2nd list.

Brute Force: Check all $\binom{n}{2}$ pairs, requires $O(n^2)$ time.
Does sorting help? Not really since we check all pairs.

Divide and Conquer Method: Given a list $L = a_1, \ldots, a_n$ of numbers, count the number of inversions.

- Divide L into 2 lists at $m = \lceil \frac{n}{2} \rceil$: $A = a_1, \ldots, a_m$ and $B = a_{m+1}, \ldots, a_n$
- Recursively count number of inversions in A and $B \Rightarrow$ return counts r_A and r_B
- Combine the results: $r_A + r_B + r$. What is r?

Example: Counting Inversions

The problem of counting the number of inversions now becomes:
count the number of pairs that are out of order in the 2nd list.

Brute Force: Check all $\binom{n}{2}$ pairs, requires $O(n^2)$ time.
Does sorting help? Not really since we check all pairs.

Divide and Conquer Method: Given a list $L = a_1, \ldots, a_n$ of numbers, count the number of inversions.

- Divide L into 2 lists at $m = \lceil \frac{n}{2} \rceil$: $A = a_1, \ldots, a_m$ and $B = a_{m+1}, \ldots, a_n$
- Recursively count number of inversions in A and $B \Rightarrow$ return counts r_A and r_B
- Combine the results: $r_A + r_B + r$. What is r?
  $r :=$ number of inversions with one element in A and one in B; i.e. number of pairs (a_i, a_j) with $a_i \in A$ and $a_j \in B$ and $a_i > a_j$

Example: Counting Inversions

How do we find r?

Count: For each $a_j \in B$, count the number of items, r_j, in A that are larger than a_j; i.e. $r = \sum_{a_j \in B} r_j$

Example: Counting Inversions

How do we find r?

Count: For each $a_j \in B$, count the number of items, r_j, in A that are larger than a_j; i.e. $r = \sum_{a_j \in B} r_j$

Now, it would help if A and B are sorted and also, for the combine step to return a sorted list. Think about how we can modify *mergesort* to compute r; modify the *merge* process.

When a_j is copied into the merged list (combined sorted list), determine r_j and add to r.

Example: Counting Inversions

Algorithm: Sort-and-Count(L) returns a sorted L and number of inversions.

- Divide L at midpoint into A and B
- Sort-and-Count(A) returns $(\textit{sorted } A, r_A)$
  Sort-and-Count(B) returns $(\textit{sorted } B, r_B)$
- $r \leftarrow 0$
  Merge(A, B) and when an element of B is chosen to merge,
  $r \leftarrow r +$ number of elements remaining in A
  return $(\textit{sorted } A \cup B, r_A + r_B + r)$

Analysis:

Example: Counting Inversions

Algorithm: Sort-and-Count(L) returns a sorted L and number of inversions.

- Divide L at midpoint into A and B
- Sort-and-Count(A) returns ($\textit{sorted } A, r_A$)
  Sort-and-Count(B) returns ($\textit{sorted } B, r_B$)
- $r \leftarrow 0$
  Merge(A, B) and when an element of B is chosen to merge,
  $r \leftarrow r +$ number of elements remaining in A
  return ($\textit{sorted } A \cup B, r_A + r_B + r$)

Analysis: Similar to mergesort: $T(n) = 2T(n/2) + O(n) \in O(n \log n)$
Better Algorithms?

Multiplying Large Numbers

Recall: grade 2 method $\Rightarrow$ multiplying two $n-$digit numbers $\in O(n^2)$
How about an $n-$digit number with an $m-$digit number? *Exercise*

Divide and Conquer Method:
Idea: Split numbers in half (by digits), multiply smaller components.
Easier if both have same number of digits $\Rightarrow$ pad with 0 if necessary.

Multiplying Large Numbers

Recall: grade 2 method $\Rightarrow$ multiplying two $n-$digit numbers $\in O(n^2)$
How about an $n-$digit number with an $m-$digit number? *Exercise*

Divide and Conquer Method:
Idea: Split numbers in half (by digits), multiply smaller components.
Easier if both have same number of digits $\Rightarrow$ pad with 0 if necessary.

Example: Multiply 667 (0667) with 1234
$06|67 \times 12|34$ becomes the sum of:

$06 \times 12 \Rightarrow 72\mathit{0000}$ (72 shifted 4 digits)
$06 \times 34 \Rightarrow 204\mathit{00}$ (204 shifted 2 digits)
$67 \times 12 \Rightarrow 804\mathit{00}$ (804 shifted 2 digits)
$67 \times 34 \Rightarrow 2278$
Total sum: 823078

Use recursion until numbers are small enough: $0|6 \times 1|2$, etc

Recall Master Theorem

Theorem: Given

$$T(n) = aT(\frac{n}{b}) + cn^k$$

where $a \geq 1, b > 1, c > 0, k \geq 0$, then

$$T(n) \in \begin{cases} \Theta(n^k) & \text{if } a < b^k \text{ i.e. } \log_b a < k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

Proof: For a rigorous proof, use induction.
Less rigorous, think about the recursion tree.

Multiplying Large Numbers

Analysis: $T(n) = 4T(n/2) + O(n)$
Apply the Master Theorem: $a = 4, b = 2, k = 1$ and compare a with b^k

$\Rightarrow 4 > 2$, so Case 3: $T(n) \in \Theta(n^{\log_b a})$

Multiplying Large Numbers

Analysis: $T(n) = 4T(n/2) + O(n)$
Apply the Master Theorem: $a = 4, b = 2, k = 1$ and compare a with b^k

$\Rightarrow 4 > 2$, so Case 3: $T(n) \in \Theta(n^{\log_b a}) \in \Theta(n^2)$

Karatsuba's Algorithm (1960)

Idea: Avoid one of the four multiplications!

Consider 0667×1234 where $w = 06, x = 67, y = 12, z = 34$, then

$wx \times yz \Rightarrow w|x \times y|z$

$$= (10^2 w + x) \times (10^2 y + z)$$
$$= 10^4 wy + 10^2(wz + xy) + xz$$

Don't need wz, xy individually, only the sum $(wz + xy)$.

Karatsuba's Algorithm (1960)

Idea: Avoid one of the four multiplications!

Consider 0667×1234 where $w = 06, x = 67, y = 12, z = 34$, then

$wx \times yz \Rightarrow w|x \times y|z$

$= (10^2 w + x) \times (10^2 y + z)$
$= 10^4 wy + 10^2(wz + xy) + xz$

Don't need wz, xy individually, only the sum $(wz + xy)$.

$(w + x) \times (y + z) = wy + (wz + xy) + xz$
$\Rightarrow (wz + xy) = (w + x) \times (y + z) - wy - xz$

We already compute wy and xz. Only 3 multiplications.

Karatsuba's Algorithm (1960)

Idea: Avoid one of the four multiplications!

Consider 0667×1234 where $w = 06, x = 67, y = 12, z = 34$, then

$wx \times yz \Rightarrow w|x \times y|z$

$$= (10^2 w + x) \times (10^2 y + z)$$
$$= 10^4 wy + 10^2(wz + xy) + xz$$

Don't need wz, xy individually, only the sum $(wz + xy)$.

$$(w + x) \times (y + z) = wy + (wz + xy) + xz$$
$$\Rightarrow (wz + xy) = (w + x) \times (y + z) - wy - xz$$

We already compute wy and xz. Only 3 multiplications.

Algorithm

```
$p \Rightarrow wy$
$q \Rightarrow xz$
$r \Rightarrow (w + x) \times (y + z)$
**return** $10^4 p + 10^2(r - p - q) + q$
```

Karatsuba's Algorithm (1960)

Note: Additions are only linear in number of digits, O(n)

Analysis: $T(n) = 3T(n/2) + O(n)$

Master Theorem: $a = 3, b = 2, k = 1$ and compare a with b^k

$a = 3 > b^k = 2$, so Case 3: $T(n) \in \Theta(n^{\log_b a})$

Karatsuba's Algorithm (1960)

Note: Additions are only linear in number of digits, O(n)

Analysis: $T(n) = 3T(n/2) + O(n)$

Master Theorem: $a = 3, b = 2, k = 1$ and compare a with b^k

$a = 3 > b^k = 2$, so Case 3: $T(n) \in \Theta(n^{\log_b a})$

$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 3}) \approx \Theta(n^{1.585})$

Better Algorithms? Asymptotically faster methods for larger n.
Schönhage and Strassen (1971): $O(n(\log n)(\log\log n))$ (often used)
More recent: $O(n \log n)$

Karatsuba's Algorithm (1960)

Implementation Concerns

1. Numbers of different lengths - sometimes large differences.
   Example: A has n digits, B has m digits and $n \gg m$.
   Break A into $O(n/m)$ blocks of m digits, e.g.

$$A = 342|3794|3749|4379|4297|7294|9742 \times 3422 = B$$

Karatsuba's Algorithm (1960)

Implementation Concerns

1. Numbers of different lengths - sometimes large differences.
   Example: A has n digits, B has m digits and $n \gg m$.
   Break A into $O(n/m)$ blocks of m digits, e.g.
   $$A = 342|3794|3749|4379|4297|7294|9742 \times 3422 = B$$
   Multiply each block by B
   Sum all products (remember to do the shifts)
   Analysis: $O((n/m)m^{\log_2 3})$ or $O(nm^{0.585})$

Karatsuba's Algorithm (1960)

Implementation Concerns

1. Numbers of different lengths - sometimes large differences.
   Example: A has n digits, B has m digits and $n \gg m$.
   Break A into $O(n/m)$ blocks of m digits, e.g.
   $$A = 342|3794|3749|4379|4297|7294|9742 \times 3422 = B$$
   Multiply each block by B
   Sum all products (remember to do the shifts)
   Analysis: $O((n/m)m^{\log_2 3})$ or $O(nm^{0.585})$
2. Which base to use? Base 10 nice for humans. In practice, for computers, Base 2^{64}.
   Store large numbers as an array of 64-bit integers (unsigned long):
   $$A = a_0 + a_1(2^{26}) + a_2(2^{26})^2 + \ldots + a_{n-1}(2^{26})^{n-1}$$
   $$\Rightarrow A = a_0|a_1|a_2|\ldots|a_{n-1}$$

Multiplying Matrices

Problem

Matrix Multiplication

Instance: *Two n by n matrices, A and B.*

Question: *Compute the n by n matrix product $C = AB$.*

The naive algorithm (row by column for each of n^2 locations) has complexity $\Theta(n^3)$.

Divide and Conquer: Divide into Submatrices of size $n/2 \times n/2$.

Matrix Multiplication - Simple Divide and Conquer

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}, \quad C = AB = \begin{bmatrix} r & s \\ t & u \end{bmatrix}$$

If A, B are n by n matrices, then $a, b, \ldots, h, r, s, t, u$ are $\frac{n}{2}$ by $\frac{n}{2}$ matrices, where

$$r = a\,e + b\,g \qquad s = a\,f + b\,h$$
$$t = c\,e + d\,g \qquad u = c\,f + d\,h$$

requiring 8 multiplications of $\frac{n}{2}$ by $\frac{n}{2}$ matrices to compute $C = AB$.

Analysis: $T(n) = 8T(n/2) + O(n^2)$
Master Theorem: $a = 8, b = 2, k = 2$ compare $a = 8 > b^k = 4$
$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^3)$

Strassen's Algorithm (1969)

Idea: Similar to multiplication, algebraic genius (or trickery)!
$\Rightarrow$ 7 subproblems instead of 8 to compute $C = AB$!

Define

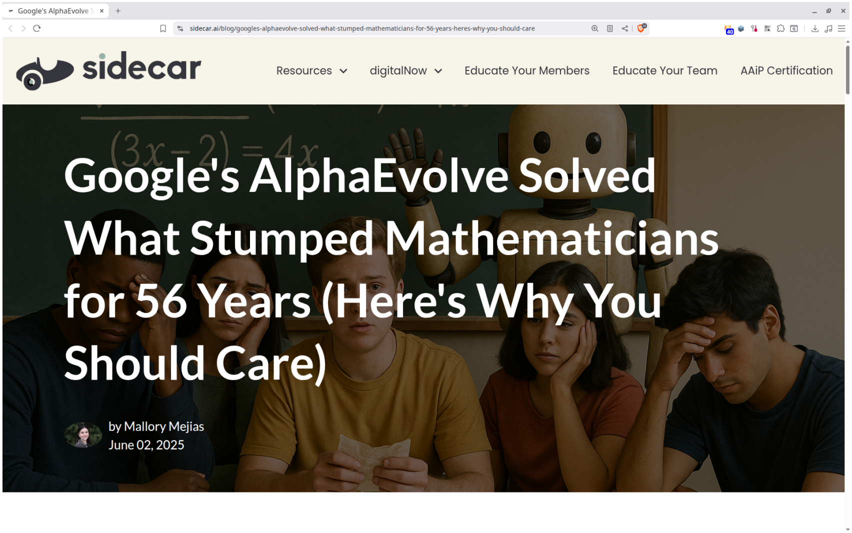
$$\begin{aligned} P_1 &= a(f-h) & P_2 &= (a+b)h \\ P_3 &= (c+d)e & P_4 &= d(g-e) \\ P_5 &= (a+d)(e+h) & P_6 &= (b-d)(g+h) \\ P_7 &= (a-c)(e+f). \end{aligned}$$

Then, compute

$$\begin{aligned} r &= P_5 + P_4 - P_2 + P_6 & s &= P_1 + P_2 \\ t &= P_3 + P_4 & u &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

Analysis: $T(n) = 7T(n/2) + O(n^2)$
Master Theorem: $a = 7, b = 2, k = 2$ compare $a = 7 > b^k = 4$
$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_2 7}) \approx \Theta(n^{2.808})$

Faster algorithms: AI to the rescue


Google's AlphaEvolve S
sidecar.ai/blog/googles-alphaevolve-solved-what-stumped-mathematicians-for-56-years-heres-why-you-should-care
sidecar
Resources
digitalNow
Educate Your Members
Educate Your Team
AAiP Certification

Google's AlphaEvolve Solved What Stumped Mathematicians for 56 Years (Here's Why You Should Care)

by Mallory Mejias
June 02, 2025

Beyond Strassen

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $T(n) \in \Theta(n^{\log_\ell(k)})$ (can prove: $k > \ell^2$, so no log)

Beyond Strassen

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $T(n) \in \Theta(n^{\log_\ell(k)})$ (can prove: $k > \ell^2$, so no log)

A challenge: find best k for small values of ℓ

- SAT solving, gradient descent, ...
- AlphaEvolve found 48 multiplications for 4×4 (instead of $7 \times 7 = 49$)

Beyond Strassen

Direct generalization

- an algorithm that does k multiplications for matrices of size ℓ gives $T(n) \in \Theta(n^{\log_\ell(k)})$ (can prove: $k > \ell^2$, so no log)

A challenge: find best k for small values of ℓ

- SAT solving, gradient descent, ...
- AlphaEvolve found 48 multiplications for 4×4 (instead of $7 \times 7 = 49$)

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

- $O(n^{2.371552})$, improves from previous record $O(n^{2.37188})$
- galactic algorithms

Centrality of Matrix Multiplication

Suppose two $n \times n$ matrices can be multiplied in $O(n^{\omega})$ where $2 \leq \omega \leq 3$.

Many other problems can also then be solved in $O(n^{\omega})$:

- Solving $Ax = b$
- Determinant of A
- Inverse of A, A^{-1}

Many problems are at least as difficult as matrix multiplication.

Example: Reduction of triangular matrix inversion to matrix multiplication.
Compute the inverse of an $n \times n$ upper triangular matrix T.
Divide and Conquer: Decompose T into blocks of size $n/2$

Analysis: $T(n) = 2T(n/2) + O(n^{\omega})$
Master Theorem: $a = 2, b = 2, k = \omega$ and $a = 2 < b^k = 2^{\omega} \geq 4$
So, $T(n) \in \Theta(n^{\omega})$

Also, Reduction of matrix multiplication to triangular matrix inversion.

Finding the Closest Pair

Problem

Closest Pair

Instance: *A set of n distinct points in the plane.*

Find: *Two distinct points p, q such that the distance between p and q,*

$$d(p, q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

is minimized.

Brute Force: try all pairs, $O(n^2)$

Special case: 1D (points on a line): sort and compare consecutive pairs: $O(n \log n)$

Closest Pair - Divide and Conquer

Idea:

- Divide points in half: left half Q, right half R, dividing line L
- Recursively find closest pair in Q, R
- Combine - must consider points with an endpoint on each side of L

Closest Pair - Divide and Conquer

Idea:

- Divide points in half: left half Q, right half R, dividing line L
- Recursively find closest pair in Q, R
- Combine - must consider points with an endpoint on each side of L

Note: To divide points, it helps to sort by x-coord - once only!
Then extract the points you need in linear time (they will also be sorted).