Divide and Conquer

Lecture 16

Course Instructor: Sikder Huq
Divide-and-Conquer

Divide-and-conquer.
- Break up problem into several parts.
- Solve each part recursively.
- Combine solutions to sub-problems into overall solution.

Most common usage.
- Break up problem of size \( n \) into \textit{two} equal parts of size \( \frac{1}{2}n \).
- Solve two parts recursively.
- Combine two solutions into overall solution in \textit{linear time}.

Consequence.
- Brute force: \( n^2 \).
- Divide-and-conquer: \( n \log n \).

Divide et impera.
Veni, vidi, vici.
- Julius Caesar
5.1 Mergesort
Sorting

**Sorting.** Given n elements, rearrange in ascending order.

Obvious sorting applications.
- List files in a directory.
- Organize an MP3 library.
- List names in a phone book.
- Display Google PageRank results.

Problems become easier once sorted.
- Find the median.
- Find the closest pair.
- Binary search in a database.
- Identify statistical outliers.
- Find duplicates in a mailing list.

Non-obvious sorting applications.
- Data compression.
- Computer graphics.
- Interval scheduling.
- Computational biology.
- Minimum spanning tree.
- Supply chain management.
- Simulate a system of particles.
- Book recommendations on Amazon.
- Load balancing on a parallel computer.
  
  ...
Mergesort

Mergesort.
- Divide array into two halves.
- Recursively sort each half.
- Merge two halves to make sorted whole.

Jon von Neumann (1945)
Merging

Merging. Combine two pre-sorted lists into a sorted whole.

How to merge efficiently?
- Linear number of comparisons.
- Use temporary array.

Challenge for the bored. In-place merge. [Kronrud, 1969]

using only a constant amount of extra storage
A Useful Recurrence Relation

**Def.** \( T(n) = \) number of comparisons to mergesort an input of size \( n \).

**Mergesort recurrence.**

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
T(\lceil n/2 \rceil) & \text{solve left half} \\
T(\lfloor n/2 \rfloor) & \text{solve right half} \\
\frac{n}{2} & \text{merging}
\end{cases} \text{ otherwise}
\]

**Solution.** \( T(n) = O(n \log_2 n) \).

**Assorted proofs.** We describe several ways to prove this recurrence. Initially we assume \( n \) is a power of 2 and replace \( \leq \) with \( = \).
Proof by Recursion Tree

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + \frac{n}{k} & \text{otherwise}
\end{cases}
\]

sorting both halves
merging
Proof by Telescoping

Claim. If $T(n)$ satisfies this recurrence, then $T(n) = n \log_2 n$.

\[ T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases} \]

assumes $n$ is a power of 2

Pf. For $n > 1$:

\[ \frac{T(n)}{n} = \frac{2T(n/2)}{n} + 1 \]

\[ = \frac{T(n/2)}{n/2} + 1 \]

\[ = \frac{T(n/4)}{n/4} + 1 + 1 \]

\[ \vdots \]

\[ = \frac{T(n/n)}{n/n} + 1 + \cdots + 1 \]

\[ = \log_2 n \]
Proof by Induction

**Claim.** If \( T(n) \) satisfies this recurrence, then \( T(n) = n \log_2 n \).

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
2T(n/2) + n & \text{otherwise}
\end{cases}
\]

assumes \( n \) is a power of 2

\[T(2n) = 2T(n) + 2n = 2n \log_2 n + 2n = 2n(\log_2 (2n) - 1) + 2n = 2n \log_2 (2n)\]

**Pf.** (by induction on \( n \))

- **Base case:** \( n = 1 \).
- **Inductive hypothesis:** \( T(n) = n \log_2 n \).
- **Goal:** show that \( T(2n) = 2n \log_2 (2n) \).

\[
\begin{align*}
T(n) & = 2T(n/2) + n \\
& = 2n \log_2 n + 2n \\
& = 2n(\log_2 (2n) - 1) + 2n \\
& = 2n \log_2 (2n)
\end{align*}
\]
Analysis of Mergesort Recurrence

Claim. If $T(n)$ satisfies the following recurrence, then $T(n) \leq n \lceil \lg n \rceil$.

\[
T(n) \leq \begin{cases} 
0 & \text{if } n = 1 \\
\frac{T(\lceil n/2 \rceil)}{\text{solve left half}} + \frac{T(\lfloor n/2 \rfloor)}{\text{solve right half}} + \frac{n}{\text{merging}} & \text{otherwise}
\end{cases}
\]

Pf. (by induction on $n$)

- **Base case**: $n = 1$.
- **Define** $n_1 = \lceil n / 2 \rceil$, $n_2 = \lfloor n / 2 \rfloor$.
- **Induction step**: assume true for $1, 2, \ldots, n-1$.

\[
T(n) \leq T(n_1) + T(n_2) + n \\
\leq n_1 \lceil \lg n_1 \rceil + n_2 \lfloor \lg n_2 \rfloor + n \\
\leq n_1 \lceil \lg n_2 \rceil + n_2 \lceil \lg n_2 \rceil + n \\
= n \lceil \lg n_2 \rceil + n \\
\leq n(\lceil \lg n \rceil - 1) + n \\
= n \lceil \lg n \rceil
\]

\[
n_2 = \lfloor n/2 \rfloor \\
\leq 2 \lceil \lg n \rceil / 2 \\
= 2 \lceil \lg n \rceil / 2 \\
\Rightarrow \lg n_2 \leq \lceil \lg n \rceil - 1
\]
5.3 Counting Inversions
Counting Inversions

Music site tries to match your song preferences with others.
- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.
- My rank: 1, 2, ..., n.
- Your rank: $a_1, a_2, ..., a_n$.
- Songs i and j inverted if $i < j$, but $a_i > a_j$. 

Counting Inversions

Music site tries to match your song preferences with others.
- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.
- My rank: 1, 2, …, n.
- Your rank: a₁, a₂, …, aₙ.
- Songs i and j inverted if i < j, but aᵢ > aⱼ.

<table>
<thead>
<tr>
<th>Songs</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Me</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>You</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Inversions
3-2, 4-2

Brute force: check all Θ(n²) pairs i and j.
Applications

Applications.
- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.

\[
\begin{array}{cccccccc}
1 & 5 & 4 & 8 & 10 & 2 & 6 & 9 & 12 & 11 & 3 & 7 \\
\end{array}
\]

Divide: \(O(1)\).
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.
- **Divide:** separate list into two pieces.
- **Conquer:** recursively count inversions in each half.

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

5 blue-blue inversions
5-4, 5-2, 4-2, 8-2, 10-2

8 green-green inversions
6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Divide: \(O(1)\).
Conquer: \(2T(n/2)\)
Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.
- **Conquer**: recursively count inversions in each half.
- **Combine**: count inversions where \( a_i \) and \( a_j \) are in different halves, and return sum of three quantities.

<table>
<thead>
<tr>
<th>1</th>
<th>5</th>
<th>4</th>
<th>8</th>
<th>10</th>
<th>2</th>
<th>6</th>
<th>9</th>
<th>12</th>
<th>11</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
</table>

Divide: \( O(1) \).

Conquer: \( 2T(n/2) \)

Combine: ???

5 blue-blue inversions

8 green-green inversions

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

Total = 5 + 8 + 9 = 22.
Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is sorted.
- Count inversions where $a_i$ and $a_j$ are in different halves.
- Merge two sorted halves into sorted whole.

13 blue-green inversions: $6 + 3 + 2 + 2 + 0 + 0$

Count: $O(n)$

Merge: $O(n)$

\[
T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \quad \Rightarrow \quad T(n) = O(n \log n)
\]
Lecture 17
Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.
Post-condition. [Sort-and-Count] L is sorted.

Sort-and-Count(L) {
    if list L has one element
        return 0 and the list L

    Divide the list into two halves A and B
    (r_A, A) ← Sort-and-Count(A)
    (r_B, B) ← Sort-and-Count(B)
    (r, L) ← Merge-and-Count(A, B)

    return r = r_A + r_B + r and the sorted list L
}
Q1: Finding modes

You are given an array $A$ with $n$ entries; each entry is a distinct number. You are told that the sequence $A[1],...,A[n]$ is unimodal. That is, for some index $p$ between 1 and $n$, values in the array increase up to position $p$ in $A$ and then decrease the rest of the way up to position $n$.

Give a $O(\log n)$-time algorithm to find the “peak entry” of the array.
Q2: Significant inversions

Let’s “relax” the inversion-counting problem a bit. Call a pair of numbers \(a_i, a_j\) a *significant inversion* if \(i < j\) and \(a_i > 2 \cdot a_j\). Give an \(O(n \log n)\) algorithm to count the number of significant inversions between two orderings.
The algorithm is similar to the one for counting inversions.

The difference is that in the “conquer” step we merge twice:
- First merge A and B just for sorting
- Then merge A and B’ where for all i, B’[i] = 2 B[i], with the goal of counting significant inversions.
5.4 Closest Pair of Points
**Closest Pair of Points**

**Closest pair.** Given \( n \) points in the plane, find a pair with smallest Euclidean distance between them.

**Fundamental geometric primitive.**

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

*fast closest pair inspired fast algorithms for these problems*

**Brute force.** Check all pairs of points \( p \) and \( q \) with \( \Theta(n^2) \) comparisons.

**1-D version.** \( O(n \log n) \) easy if points are on a line.

**Assumption.** No two points have same \( x \) coordinate.

\[ \uparrow \]

To make presentation cleaner
Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.
Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.
Obstacle. Impossible to ensure n/4 points in each piece.
Closest Pair of Points

Algorithm.
- **Divide**: draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
**Closest Pair of Points**

**Algorithm.**

- **Divide:** draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer:** find closest pair in each side recursively.
**Closest Pair of Points**

**Algorithm.**
- **Divide:** draw vertical line $L$ so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer:** find closest pair in each side recursively.
- **Combine:** find closest pair with one point in each side. $\leftarrow$ seems like $\Theta(n^2)$
- Return best of 3 solutions.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line $L$. 

$\delta = \min(12, 21)$
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$. 
- Observation: only need to consider points within $\delta$ of line $L$.
- Sort points in $2\delta$-strip by their $y$ coordinate.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.

- Observation: only need to consider points within $\delta$ of line $L$.
- Sort points in $2\delta$-strip by their $y$ coordinate.
- Only check distances of those within 11 positions in sorted list!

$\delta = \min(12, 21)$
Lecture 18
Closest Pair of Points

**Def.** Let \( s_i \) be the point in the \( 2\delta \)-strip, with the \( i^{th} \) smallest \( y \)-coordinate.

**Claim.** If \( |i - j| \geq 12 \), then the distance between \( s_i \) and \( s_j \) is at least \( \delta \).

**Pf.**
- No two points lie in same \( \frac{1}{2} \delta \)-by-\( \frac{1}{2} \delta \) box.
- Two points at least 2 rows apart have distance \( \geq 2(\frac{1}{2} \delta) \).

**Fact.** Still true if we replace 12 with 7.
Closest Pair Algorithm

Closest-Pair(p₁, ..., pₙ) {

  Compute separation line L such that half the points are on one side and half on the other side.

  \[ \delta_1 = \text{Closest-Pair(left half)} \]
  \[ \delta_2 = \text{Closest-Pair(right half)} \]
  \[ \delta = \min(\delta_1, \delta_2) \]

  Delete all points further than \( \delta \) from separation line L

  Sort remaining points by y-coordinate.

  Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than \( \delta \), update \( \delta \).

  return \( \delta \).

}
Closest Pair of Points: Analysis

Running time.

\[ T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n) \]

Q. Can we achieve \( O(n \log n) \)?

A. Yes. Don't sort points in strip from scratch each time.
   - Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
   - Sort by **merging** two pre-sorted lists.

\[ T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n) \]
Closest Pair of Points

Algorithm.
- **Divide**: draw vertical line \( L \) so that roughly \( \frac{1}{2}n \) points on each side.
- **Conquer**: find closest pair in each side recursively.
- **Combine**: find closest pair with one point in each side.
- Return best of 3 solutions.
5.5 Integer Multiplication
**Integer Arithmetic**

**Add.** Given two n-digit integers $a$ and $b$, compute $a + b$.
- $O(n)$ bit operations.

**Multiply.** Given two n-digit integers $a$ and $b$, compute $a \times b$.
- Brute force solution: $\Theta(n^2)$ bit operations.
Divide-and-Conquer Multiplication: Warmup

To multiply two n-digit integers:
- Multiply four $\frac{1}{2}n$-digit integers.
- Add two $\frac{1}{2}n$-digit integers, and shift to obtain result.

\[
x = 2^{n/2} \cdot x_1 + x_0 \\
y = 2^{n/2} \cdot y_1 + y_0 \\
xy = \left(2^{n/2} \cdot x_1 + x_0\right) \left(2^{n/2} \cdot y_1 + y_0\right) = 2^n \cdot x_1y_1 + 2^{n/2} \cdot (x_1y_0 + x_0y_1) + x_0y_0
\]

\[
T(n) = 4T(n/2) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^2)
\]

\[\uparrow\]
assumes $n$ is a power of 2
Karatsuba Multiplication

To multiply two n-digit integers:

- Add two \( \frac{1}{2}n \) digit integers.
- Multiply three \( \frac{1}{2}n \)-digit integers.
- Add, subtract, and shift \( \frac{1}{2}n \)-digit integers to obtain result.

\[
x = 2^{n/2} \cdot x_1 + x_0
\]
\[
y = 2^{n/2} \cdot y_1 + y_0
\]
\[
xy = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0
\]
\[
= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 + x_0 y_0
\]

Theorem. [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in \( O(n^{1.585}) \) bit operations.

\[
T(n) \leq T\left(\lfloor n/2 \rfloor \right) + T\left(\lceil n/2 \rceil \right) + T\left(1 + \lceil n/2 \rceil \right) + \Theta(n) \\
\text{recursive calls} \quad \text{add, subtract, shift}
\]

\[
\Rightarrow T(n) = O(n \log_2 3) = O(n^{1.585})
\]
Karatsuba: Recursion Tree

\[
T(n) = \begin{cases} 
0 & \text{if } n = 1 \\
3T(n/2) + n & \text{otherwise}
\end{cases}
\]

Check page 216 form the textbook

\[
T(n) = \sum_{k=0}^{\log_2 n} n \left(\frac{3}{2}\right)^k = \left(\frac{3}{2}\right)^{1+\log_2 n} - 1 = 3^{\log_2 3} - 2
\]

n

3(n/2)

9(n/4)

...
Q3: Two recurrences

Solve the following recurrences:

\[ T(n) = 8 \; T(n/2) + O(n^2) \]
\[ T(n) = 7 \; T(n/2) + O(n^2) \]

Answer: We drew the recursive tree in class. Check Section 2 from the following lecture note to see the solution:

http://www.bowdoin.edu/~ltoma/teaching/cs231/fall07/Lectures/recurrences.pdf
Fast Matrix Multiplication

Lecture 19
Matrix Multiplication

Matrix multiplication. Given two n-by-n matrices $A$ and $B$, compute $C = AB$.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{pmatrix}
  c_{11} & c_{12} & \cdots & c_{1n} \\
  c_{21} & c_{22} & \cdots & c_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  c_{n1} & c_{n2} & \cdots & c_{nn}
\end{pmatrix}
\begin{pmatrix}
  a_{11} & a_{12} & \cdots & a_{1n} \\
  a_{21} & a_{22} & \cdots & a_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\times
\begin{pmatrix}
  b_{11} & b_{12} & \cdots & b_{1n} \\
  b_{21} & b_{22} & \cdots & b_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  b_{n1} & b_{n2} & \cdots & b_{nn}
\end{pmatrix}$$

Brute force. $\Theta(n^3)$ arithmetic operations.

Fundamental question. Can we improve upon brute force?
Matrix Multiplication: Warmup

Divide-and-conquer.

- Divide: partition $A$ and $B$ into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- Conquer: multiply $8$ $\frac{1}{2}n$-by-$\frac{1}{2}n$ recursively.
- Combine: add appropriate products using 4 matrix additions.

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

\[
C_{11} = (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\
C_{12} = (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\
C_{21} = (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\
C_{22} = (A_{21} \times B_{12}) + (A_{22} \times B_{22})
\]

\[
T(n) = 8T(n/2) + \Theta(n^2) \\
\Rightarrow T(n) = \Theta(n^3)
\]
Matrix Multiplication: Key Idea

**Key idea.** multiply 2-by-2 block matrices with only 7 multiplications.

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

\[
egin{align*}
P_1 &= A_{11} \times (B_{12} - B_{22}) \\
P_2 &= (A_{11} + A_{12}) \times B_{22} \\
P_3 &= (A_{21} + A_{22}) \times B_{11} \\
P_4 &= A_{22} \times (B_{21} - B_{11}) \\
P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\
P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\
P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12})
\end{align*}
\]

- 7 multiplications.
- 18 = 10 + 8 additions (or subtractions).
Fast Matrix Multiplication

Fast matrix multiplication. (Strassen, 1969)
- **Divide:** partition A and B into $\frac{1}{2}n$-by-$\frac{1}{2}n$ blocks.
- **Compute:** $14$ $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices via 10 matrix additions.
- **Conquer:** multiply $7$ $\frac{1}{2}n$-by-$\frac{1}{2}n$ matrices recursively.
- **Combine:** 7 products into 4 terms using 8 matrix additions.

**Analysis.**
- Assume $n$ is a power of 2.
- $T(n) = \#$ arithmetic operations.

\[
T(n) = 7T(n/2) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})
\]
Fast Matrix Multiplication in Practice

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n = 128$.

**Common misperception:** "Strassen is only a theoretical curiosity."
- Advanced Computation Group at Apple Computer reports 8x speedup on G4 Velocity Engine when $n \sim 2,500$.
- Range of instances where it's useful is a subject of controversy.

**Remark.** Can "Strassenize" $Ax=b$, determinant, eigenvalues, and other matrix ops.
Fast Matrix Multiplication in Theory

Q. Multiply two 2-by-2 matrices with only 7 scalar multiplications?  
A. Yes!  [Strassen, 1969] \[\Theta(n^{\log_2 7}) = O(n^{2.81})\]

Q. Multiply two 2-by-2 matrices with only 6 scalar multiplications?  
A. Impossible.  [Hopcroft and Kerr, 1971]

Q. Two 3-by-3 matrices with only 21 scalar multiplications?  
A. Also impossible.

Q. Two 70-by-70 matrices with only 143,640 scalar multiplications?  
A. Yes!  [Pan, 1980] \[\Theta(n^{\log_{70} 143640}) = O(n^{2.80})\]

Decimal wars.
- December, 1979:  \(O(n^{2.521813})\).
- January, 1980:  \(O(n^{2.521801})\).
The Master Theorem

For $a \geq 1, b \geq 1, c > 0$

The recurrence $T(n) = aT(n/b) + n^c$ can be solved as follows:

- $T(n) = \Theta(n^{\log_b a})$ if $a > b^c$
- $T(n) = \Theta(n^c \log_b n)$ if $a = b^c$
- $T(n) = \Theta(n^c)$ if $a < b^c$
- If none of these three cases apply, you’re on your own.
Q4: Choosing between algorithms

Suppose you are choosing between 3 algorithms:

- Algorithm A solves problems of size $n$ by dividing them into 5 subproblems of half the size, recursively solving the subproblems, then combining the solutions in linear time.

- Algorithm B solves problems of size $n$ by recursively solving two subproblems of size $(n - 1)$ and then combining the solutions in constant time.

- Algorithm C solves problems of size $n$ by dividing them into 9 subproblems of size $n/3$, recursively solving the problems, then combining in $O(n^2)$ time.

What are the runtimes of these algorithms, and which one would you prefer?
Q5: GCD

Give a divide-and-conquer algorithm for computing the GCD of two n-bit positive integers.
**Answer**

\[ \text{GCD}(a,b) = \]
- \[2 \ \text{GCD}(a/2, b/2)\] if \(a,b\) are even
- \(\text{GCD}(a, b/2)\) if \(a\) is odd, \(b\) is even
- \(\text{GCD}((a - b)/2, b)\) if \(a, b\) are odd

Runtime?? \(O(n^2)\) as we calculated in class.