Dynamic Programming
CSCI 532
Divide and Conquer Battle Plan

1. Divide problem into subproblems that are smaller instances of the same problem.

2. Conquer the subproblems by solving them recursively.

3. Combine the solutions to the subproblems into the solution for the original problem.
Divide and Conquer – Merge Sort

1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.
Divide and Conquer – Merge Sort

1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.
Divide and Conquer – Merge Sort

1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.
Divide and Conquer – Merge Sort

1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.
Divide and Conquer – Merge Sort

1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.
Divide and Conquer – Merge Sort

1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.
Divide and Conquer – Merge Sort

1. Divide array in half.
2. Sort sub arrays.
3. Merge into sorted array.
Divide and Conquer Battle Plan

1. Divide problem into subproblems that are smaller instances of the same problem.

2. Conquer the subproblems by solving them recursively.

3. Combine the solutions to the subproblems into the solution for the original problem.
Square Matrix Multiplication

\[
\begin{array}{ccc}
5 & 8 & 1 \\
3 & 6 & 7 \\
4 & 1 & 5 \\
\end{array}
\times
\begin{array}{ccc}
2 & 4 & 7 \\
5 & 1 & 2 \\
3 & 6 & 5 \\
\end{array}
= \begin{array}{ccc} \text{?} \end{array}
\]
Square Matrix Multiplication

\[
\begin{bmatrix}
5 & 8 & 1 \\
3 & 6 & 7 \\
4 & 1 & 5
\end{bmatrix} \quad \times \quad \begin{bmatrix}
2 & 4 & 7 \\
5 & 1 & 2 \\
3 & 6 & 5
\end{bmatrix} = \begin{bmatrix}
53 & 34 & 56 \\
57 & 60 & 68 \\
36 & 47 & 55
\end{bmatrix}
\]

\[
Z_{ij} = \sum_{k=1}^{n} X_{ik}Y_{kj} \Rightarrow Z_{01} = 5 \times 4 + 8 \times 1 + 1 \times 6 = 34
\]
Square Matrix Multiplication

\[ Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj} \]

\( X, Y, Z \) are \( n \times n \) matrices. Running time?
Square Matrix Multiplication

\[ Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj} \]

\[ X, Y, Z \text{ are } n \times n \text{ matrices. Running time?} \]

\[ \text{How big is the solution - ?} \]

\[ \text{How long to calculate a single entry - ?} \]
Square Matrix Multiplication

\[
\begin{bmatrix}
5 & 8 & 1 \\
3 & 6 & 7 \\
4 & 1 & 5
\end{bmatrix}
\begin{bmatrix}
2 & 4 & 7 \\
5 & 1 & 2 \\
3 & 6 & 5
\end{bmatrix}
= \begin{bmatrix}
53 & 34 & 56 \\
57 & 60 & 68 \\
36 & 47 & 55
\end{bmatrix}
\]

\[Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}\]

\(X, Y, Z\) are \(n \times n\) matrices. Running time?

How big is the solution - \(n^2\)

How long to calculate a single entry - ?
Square Matrix Multiplication

\[
\begin{array}{ccc}
5 & 8 & 1 \\
3 & 6 & 7 \\
4 & 1 & 5 \\
\end{array}
\quad \times \quad
\begin{array}{ccc}
2 & 4 & 7 \\
5 & 1 & 2 \\
3 & 6 & 5 \\
\end{array}
= 
\begin{array}{ccc}
53 & 34 & 56 \\
57 & 60 & 68 \\
36 & 47 & 55 \\
\end{array}
\]

\[
Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}
\]

\(X, Y, Z\) are \(n \times n\) matrices. Running time?
How big is the solution - \(n^2\)
How long to calculate a single entry - \(O(n)\)
Total time - \(O(n^3)\)
Square Matrix Addition

\[
\begin{array}{ccc}
5 & 8 & 1 \\
3 & 6 & 7 \\
4 & 1 & 5 \\
\end{array}
\quad + \quad
\begin{array}{ccc}
2 & 4 & 7 \\
5 & 1 & 2 \\
3 & 6 & 5 \\
\end{array}
\quad = \quad
\begin{array}{ccc}
7 & 12 & 8 \\
8 & 7 & 9 \\
7 & 7 & 10 \\
\end{array}
\]

\[Z_{ij} = X_{ij} + Y_{ij}\]

\(X, Y, Z\) are \(n \times n\) matrices. Running time? How big is the solution? How long to calculate a single entry? Total time?
Square Matrix Addition

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

\[Z_{ij} = X_{ij} + Y_{ij}\]

\[X, Y, Z \text{ are } n \times n \text{ matrices. Running time? How big is the solution - } n^2 \]

How long to calculate a single entry - \(O(1)\)

Total time - \(O(n^2)\)
Square Matrix Multiplication

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\times
\begin{bmatrix}
E & F \\
G & H
\end{bmatrix}
=
\begin{bmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{bmatrix}
\]

Alternate approach:
1. Recursively break \(X\) and \(Y\) into \(\frac{n}{2} \times \frac{n}{2}\) blocks.
2. Multiply blocks to get \(Z\).
Square Matrix Multiplication

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\times
\begin{bmatrix}
E & F \\
G & H
\end{bmatrix}
=
\begin{bmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{bmatrix}
\]

Running time?
Master Theorem

Master theorem: If $T(n) = aT(n/b) + O(n^d)$ for constants $a \geq 1, b > 1, d \geq 0$, then

$$T(n) = \begin{cases} 
O(n^d), & d > \log_b a \\
O(n^d \log n), & d = \log_b a \\
O(n^{\log_b a}), & d < \log_b a
\end{cases}$$
Square Matrix Multiplication

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\times
\begin{bmatrix}
E & F \\
G & H
\end{bmatrix}
= 
\begin{bmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{bmatrix}
\]

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

- \(a\) – Number of subproblems
- \(n/b\) – Size of subproblem
- \(D(n)\) – time to divide problems
- \(C(n)\) – time to combine problems
Square Matrix Multiplication

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\times
\begin{bmatrix}
E & F \\
G & H
\end{bmatrix}
=
\begin{bmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{bmatrix}
\]

X
Y
Z

Running time?
How many subproblems?
How long to combine solutions?

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

- \(a\) – Number of subproblems
- \(n/b\) – Size of subproblem
- \(D(n)\) – time to divide problems
- \(C(n)\) – time to combine problems
Square Matrix Multiplication

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
E & F \\
G & H
\end{bmatrix}
= 
\begin{bmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{bmatrix}
\]

Running time?
How many subproblems - 8
How long to combine solutions - \(O(n^2)\)
Four additions = \(O(n^2)\)

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

\[
\begin{align*}
    a & \quad \text{Number of subproblems} \\
    n/b & \quad \text{Size of subproblem} \\
    D(n) & \quad \text{time to divide problems} \\
    C(n) & \quad \text{time to combine problems}
\end{align*}
\]
Square Matrix Multiplication

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\times
\begin{bmatrix}
E & F \\
G & H
\end{bmatrix}
=
\begin{bmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{bmatrix}
\]

Running time?
How many subproblems - 8
How long to combine solutions - $O(n^2)$

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

- $a$ – Number of subproblems
- $n/b$ – Size of subproblem
- $D(n)$ – time to divide problems
- $C(n)$ – time to combine problems
Square Matrix Multiplication

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \times \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Running time?
How many subproblems - 8
How long to combine solutions - $O(n^2)$

$T(n) = 8T(n/2) + O(n^2)$

Master theorem:
If $T(n) = aT(n/b) + O(n^d)$, then

- $T(n) = O(n^d)$, $d > \log_b a$
- $T(n) = O(n^d \log n)$, $d = \log_b a$
- $T(n) = O(n^{\log_b a})$, $d < \log_b a$
Square Matrix Multiplication

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix} \times \begin{bmatrix}
E & F \\
G & H
\end{bmatrix} = \begin{bmatrix}
AE + BG & AF + BH \\
CE + DG & CF + DH
\end{bmatrix}
\]

\[ X \quad Y \quad Z \]

Running time?
How many subproblems - 8
How long to combine solutions - \(O(n^2)\)

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

\[ \log_b a = \log_2 8 = 3 > 2 = d \]

\[ T(n) = O(n^3) \]

Master theorem:
If \(T(n) = aT(n/b) + O(n^d)\), then

\[
T(n) = \begin{cases} 
O(n^d), & d > \log_b a \\
O(n^d \log n), & d = \log_b a \\
O(n^{\log_b a}), & d < \log_b a
\end{cases}
\]
Square Matrix Multiplication

\[
Z = \begin{bmatrix}
    P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\
    P_3 + P_4 & P_1 + P_5 - P_3 - P_7
\end{bmatrix}
\]

for,

\[
\begin{align*}
    P_1 &= A(F - H) & P_5 &= (A + D)(E + H) \\
    P_2 &= (A + B)H & P_6 &= (B - D)(G + H) \\
    P_3 &= (C + D)E & P_7 &= (A - C)(E + F) \\
    P_4 &= D(G - E)
\end{align*}
\]

\[
\begin{bmatrix}
    A & B \\
    C & D
\end{bmatrix}
\begin{bmatrix}
    E & F \\
    G & H
\end{bmatrix}
= \begin{bmatrix}
    AE + BG & AF + BH \\
    CE + DG & CF + DH
\end{bmatrix}
\]
Square Matrix Multiplication

\[ Z = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 \\ P_3 + P_4 \\ P_1 + P_2 \\ P_1 + P_5 - P_3 - P_7 \end{bmatrix} \]

for,

\[ P_1 = A(F - H) \]
\[ P_2 = (A + B)H \]
\[ P_3 = (C + D)E \]
\[ P_4 = D(G - E) \]
\[ P_5 = (A + D)(E + H) \]
\[ P_6 = (B - D)(G + H) \]
\[ P_7 = (A - C)(E + F) \]

Running time?
How many smaller problems - ?
How long to combine solutions - ?

Master theorem:
If \( T(n) = aT(n/b) + O(n^d) \), then

\[ T(n) = \begin{cases} 
O(n^d), & d > \log_b a \\
O(n^d \log n), & d = \log_b a \\
O(n^{\log_b a}), & d < \log_b a 
\end{cases} \]
Square Matrix Multiplication

\[ Z = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 \\ P_3 + P_4 \\ P_1 + P_2 \\ P_1 + P_5 - P_3 - P_7 \end{bmatrix} \]

for,

\[
\begin{align*}
P_1 &= A(F - H) \\
P_2 &= (A + B)H \\
P_3 &= (C + D)E \\
P_4 &= D(G - E) \\
P_5 &= (A + D)(E + H) \\
P_6 &= (B - D)(G + H) \\
P_7 &= (A - C)(E + F)
\end{align*}
\]

Running time?
How many smaller problems - 7
How long to combine solutions - ?

Master theorem:
If \( T(n) = aT(n/b) + O(n^d) \), then

\[
T(n) = \begin{cases} 
O(n^d), & d > \log_b a \\
O(n^d \log n), & d = \log_b a \\
O(n^{\log_b a}), & d < \log_b a 
\end{cases}
\]
Square Matrix Multiplication

\[
Z = \begin{bmatrix}
    P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\
    P_3 + P_4 & P_1 + P_5 - P_3 - P_7
\end{bmatrix}
\]

for,

\[
\begin{align*}
P_1 &= A(F - H) \\
P_2 &= (A + B)H \\
P_3 &= (C + D)E \\
P_4 &= D(G - E) \\
P_5 &= (A + D)(E + H) \\
P_6 &= (B - D)(G + H) \\
P_7 &= (A - C)(E + F)
\end{align*}
\]

Running time?
How many smaller problems - 7
How long to combine solutions - ?
18 additions/subtractions

Master theorem:
If \( T(n) = aT(n/b) + O(n^d) \), then

\[
T(n) = \begin{cases} 
O(n^d), & d > \log_b a \\
O(n^d \log n), & d = \log_b a \\
O(n^{\log_b a}), & d < \log_b a
\end{cases}
\]
Square Matrix Multiplication

\[ Z = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix} \]

for,

\[ P_1 = A(F - H) \quad P_5 = (A + D)(E + H) \]
\[ P_2 = (A + B)H \quad P_6 = (B - D)(G + H) \]
\[ P_3 = (C + D)E \quad P_7 = (A - C)(E + F) \]
\[ P_4 = D(G - E) \]

Running time?
How many smaller problems - 7
How long to combine solutions - \( O(n^2) \)
18 additions/subtractions

Master theorem:
If \( T(n) = aT(n/b) + O(n^d) \), then

\[ T(n) = \begin{cases} 
O(n^d), & d > \log_b a \\
O(n^d \log n), & d = \log_b a \\
O(n^{\log_b a}), & d < \log_b a 
\end{cases} \]
Square Matrix Multiplication

\[ Z = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 \\ P_3 + P_4 \\ P_1 + P_5 - P_3 - P_7 \end{bmatrix} \]

for,

\[ P_1 = A(F - H) \]
\[ P_2 = (A + B)H \]
\[ P_3 = (C + D)E \]
\[ P_4 = D(G - E) \]
\[ P_5 = (A + D)(E + H) \]
\[ P_6 = (B - D)(G + H) \]
\[ P_7 = (A - C)(E + F) \]

Running time?

How many smaller problems - 7

How long to combine solutions - \( O(n^2) \)

\[ T(n) = 7T(n/2) + O(n^2) \]
\[ \log_b a = \log_2 7 \approx 2.81 > 2 = d \]
\[ \Rightarrow T(n) = O(n^{2.81}) \]

Master theorem:

If \( T(n) = aT(n/b) + O(n^d) \), then

\[ T(n) = \begin{cases} 
O(n^d), & d > \log_b a \\
O(n^d \log n), & d = \log_b a \\
O(n^{\log_b a}), & d < \log_b a 
\end{cases} \]
Square Matrix Multiplication

\[ Z = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix} \]

for,

\[
\begin{align*}
P_1 &= A(F - H) \\
P_2 &= (A + B)H \\
P_3 &= (C + D)E \\
P_4 &= D(G - E) \\
P_5 &= (A + D)(E + H) \\
P_6 &= (B - D)(G + H) \\
P_7 &= (A - C)(E + F) 
\end{align*}
\]

\[ O(n^{2.81}) \text{ vs } O(n^3)? \]

- 1 second vs 2.6 seconds
- 1 hour vs 4.5 hours
- 1 day vs 5.5 days
- 3 days vs 18 days
Square Matrix Multiplication

\[ \mathbf{Z} = \mathbf{P} \mathbf{B} + \mathbf{P} \mathbf{B} - \mathbf{P} \mathbf{C} + \mathbf{P} \mathbf{D} \]

for,

\[ \mathbf{P} = \mathbf{A} \mathbf{F} - \mathbf{H} \]
\[ \mathbf{P} = \mathbf{A} + \mathbf{B} \mathbf{H} \]
\[ \mathbf{P} = \mathbf{C} + \mathbf{D} \mathbf{E} \]
\[ \mathbf{P} = \mathbf{D} \mathbf{G} - \mathbf{E} \]
\[ \mathbf{P} = \mathbf{A} + \mathbf{D} \mathbf{E} + \mathbf{H} \]
\[ \mathbf{P} = \mathbf{B} - \mathbf{D} \mathbf{G} + \mathbf{H} \]
\[ \mathbf{P} = \left( \mathbf{A} - \mathbf{C} \right) \left( \mathbf{E} + \mathbf{F} \right) \]

1 second vs 2.6 seconds
1 hour vs 4.5 hours
1 day vs 5.5 days
3 days vs 18 days

History of Matrix Multiplication Algorithms

\[ O(n^3) - 1812 \ (Definition \ of \ matrix \ multiplication) \]
\[ O(n^{2.81}) - 1969 \]
\[ O(n^{2.3755}) - 1990 \]
\[ O(n^{2.3737}) - 2010 \]
\[ O(n^{2.3729}) - 2013 \]
\[ O(n^{2.3728639}) - 2014 \]
\[ O(n^{2.3728596}) - 2020 \]
\[ O(n^{2.371866}) - 2022^* \]

Lower bound: \( O(n^2) \). Need to look at all elements of both matrices.
Fundamental Algorithmic Techniques

“Sledgehammers”:
• Dynamic Programming
• Linear Programming

Precision Tools:
• Greedy
• Randomization
• Reductions
Dynamic Programming vs Divide and Conquer

Dynamic Programming
• Optimal substructure.
• Overlapping subproblems.

Divide and Conquer
• Independent, recursive subproblems (mergesort).

Dynamic Programming Process:
1. Characterize structure of optimal solution.
2. Recursively define value of optimal solution.
3. Compute value of optimal solution.
4. Construct optimal solution from computed information.
Making Change

How can I represent 37 cents with the smallest number of coins?
Making Change

How can I represent 37 cents with the smallest number of coins?
Quarter, dime, two pennies (25 + 10 + 2 = 37) – four coins.
Algorithm: ?
Making Change

How can I represent 37 cents with the smallest number of coins?  
Quarter, dime, two pennies \((25 + 10 + 2 = 37)\) – four coins.

Algorithm: Max quarters + max dimes + ...

What if there were also an 18 cent coin?
Making Change

How can I represent 37 cents with the smallest number of coins? Quarter, dime, two pennies \((25 + 10 + 2 = 37)\) – four coins.

Algorithm: Max quarters + max dimes + ...

What if there were also an 18-cent coin? Two 18-cent coin, penny – three coins.
Making Change

In general, suppose a country has coins with denominations:

\[ 1 = d_1 < d_2 < \cdots < d_k \]  

(US coins: \( d_1 = 1, d_2 = 5, d_3 = 10, d_4 = 25 \))

Algorithm: To make change for \( p \) cents, we are going to figure out change for every value \( x < p \). We will build solution for \( p \) out of smaller solutions.
Making Change – Dynamic Programming

Central tenant of Dynamic Programming:
Leverage optimal sub-structure.
Central tenant of Dynamic Programming:
Leverage optimal sub-structure.

1 broom + 2 “cool rocks” + 1 chair are the smallest number of items possible to get to $17. One broom costs $10. What can you conclude?
Central tenant of Dynamic Programming:
Leverage optimal sub-structure.

1 broom + 2 “cool rocks” + 1 chair are the smallest number of items possible to get to $17. One broom costs $10. What can you conclude?

2 “cool rocks” + 1 chair are the smallest number of items possible to get to $7.
Making Change

\( C(p) \) – minimum number of coins to make \( p \) cents.
\( x \) – value (e.g. $0.25) of a coin used in the optimal solution.
Making Change

\[ C(p) \] – minimum number of coins to make \( p \) cents.
\[ x \] – value (e.g. $0.25) of a coin used in the optimal solution.

Can we characterize \( C(p) \) in terms of \( C(p - x) \)?
Making Change

\[ C(p) \] – minimum number of coins to make \( p \) cents.
\[ x \] – value (e.g. $0.25) of a coin used in the optimal solution.

\[ C(p) = 1 + C(p - x). \]
Making Change

\[ C(p) \] – minimum number of coins to make \( p \) cents.
\[ x \] – value (e.g. $0.25) of a coin used in the optimal solution.

\[ C(p) = 1 + C(p - x). \]

\[
C(p) = \begin{cases} 
\min_{i:d_i \leq p} C(p - d_i) + 1, & p > 0 \\
0, & p = 0
\end{cases}
\]

Least change for 20 cents = minimum of:
- least change for 20-10 = 10 cents
- least change for 20-5 = 15 cents
- least change for 20-1 = 19 cents
Making Change - Recursive

\[
\text{change}(p) \\
\quad \text{if } p == 0 \\
\quad \quad \text{return } 0 \\
\quad \text{else } \\
\quad \quad \text{min} = \infty \\
\quad \quad \text{for } d_i \leq p \\
\quad \quad \quad a = \text{change}(p-d_i) \\
\quad \quad \quad \text{if } a < \text{min} \\
\quad \quad \quad \quad \text{min} = a \\
\quad \quad \text{return } 1 + \text{min}
\]
Making Change - Recursive

change(p)
    if p == 0
        return 0
    else
        min = ∞
        for d_i ≤ p
            a = change(p - d_i)
            if a < min
                min = a
        return 1 + min

Running time?