Divide and Conquer: Recipe

- Divide problem into several parts
- Solve each part recursively
- Combine solutions to sub-problems into overall solution

Learning Goals

<table>
<thead>
<tr>
<th></th>
<th>Greedy</th>
<th>Divide and Conquer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formulate problem</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Design algorithm</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Prove correctness</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Analyze running time</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Specific algorithms</td>
<td>Dijkstra, MST</td>
<td></td>
</tr>
</tbody>
</table>

Motivating Problem: Maximum Subsequence Sum (MSS)

- **Input**: array $A$ of $n$ numbers, e.g.
  
  $A = 4, -3, 5, -2, -1, 2, 6, -2$

- **Find**: value of the largest *subsequence sum*
  

  - (empty subsequence allowed and has sum zero)
  - MSS in example? 11 (first 7 elements)

Clicker

Which of the following is true for a maximum-sum subsequence?

A. It has more positive than negative numbers
B. It does not start or end with a negative number
C. Any maximal sequence of negative numbers is bordered by a sequence of positive numbers with sum larger in absolute value

A Simple MSS Algorithm

Brute force in $\Theta(n^2)$ (c.f K&T Chapter 2, Exercise 6)

MSS($A$)

- Initialize all entries of $n \times n$ array $B$ to zero
- $\text{for } i = 1 \text{ to } n \text{ do}$
  - $\text{sum } = 0$
  - $\text{for } j = i \text{ to } n \text{ do}$
    - Compute sum of $A[i] \ldots A[j]$
    - $B[i,j] = \text{sum}$
  - Return maximum value among all $B[i,j]$

Running time? $O(n^2)$. Can we do better?
Divide-and-conquer for MSS

- Recursive solution for MSS

- Idea:
  - Find MSS \( L \) in left half of array
  - Find MSS \( R \) in right half of array
  - Find MSS \( M \) for sequence that crosses the midpoint

\[
A = \begin{cases} 4, -3, 5, -2, -1, 2, 6 & \text{for } i = \text{mid} + 1 \text{ to right} \\ -2 & \text{for } i = 1 \text{ to mid} \end{cases}
\]

- Return \( \max(L, R, M) \)
- Change one entry to make MSS-=R. \(-2 \rightarrow -10\)
- How to find \( L, R, M \)?

Recurrence

- Recurrence
  \[ T(n) = 2T(n/2) + O(n) \]
  \[ T(1) = O(1) \]
- First, let’s use definition of Big-O:
  \[ T(n) \leq 2T(n/2) + cn \]
  \[ T(1) \leq c \]
- Running time?
  - Let \( T(n) \) be running time of MSS on array of size \( n \)
  - Two recursive calls on arrays of size \( n/2 \): \( 2T(n/2) \)
  - Work outside of recursive calls: \( O(n) \)
  - Running time
    \[ T(n) = 2T(n/2) + O(n) \]

MSS(A, left, right)

\[
\begin{align*}
\text{mid} &= \lfloor \text{left} + \text{right} \rfloor / 2 \\
L &= \text{MSS}(A, \text{left}, \text{mid}) \\
R &= \text{MSS}(A, \text{mid+1}, \text{right}) \\
\text{Set} \ q &= 0 \text{ and } L' = 0 \\
\text{for } i &= \text{mid down to } 1 \text{ do} \\
& \quad \text{sum} += A[i] \\
& \quad L' = \max(L', \text{sum}) \\
\text{Set} \ q &= 0 \text{ and } R' = 0 \\
\text{for } i &= \text{mid+1 to right} \text{ do} \\
& \quad \text{sum} += A[i] \\
& \quad R' = \max(R', \text{sum}) \\
M &= L' + R' \\
\text{return max}(L, R, M)
\end{align*}
\]

Recurrence

- Recurrence
  \[ T(n) = 2T(n/2) + O(n) \]
  \[ T(1) = O(1) \]
- sequence \( T(0), T(1), T(2), \ldots \)
- \( T(n) \) defined in terms of smaller values
- For running time, choose any convenient base case:
  \( T(1) = O(1), T(2) = O(1) \)

Goal: “solve” the recurrence = find simple expression for \( T(n) \) for all \( n \)

Clicker

- Recurrence
  \[ T(n) = 2T(n/2) + O(n) \]
  \[ T(1) = O(1) \]
- Why is it OK to use the same value of \( c \) in both instances of the big-O definition?
  A. It’s not OK. You just took a shortcut. (By the way, you forgot about \( n_0 \).)
  B. Take \( c = \min(c_1, c_2) \) where \( c_1 \) and \( c_2 \) are the values from each instance.
  C. Take \( c = \max(c_1, c_2) \) where \( c_1 \) and \( c_2 \) are the values from each instance.
Recurrence

- Same recurrence with change of variable
  \[ T(m) \leq 2T(m/2) + cm, \quad m \geq 2 \]
  \[ T(1) \leq c \]
  - no difference, but sometimes helpful conceptually
  - \( n \) = original input size, \( m \) = generic input size
- Three approaches to solve it
  1. Unrolling
  2. Recursion tree (another version of unrolling)
  3. Guess and verify (proof by induction)

Recurrence Solving (1): Unrolling

- **Idea 1:** “unroll” the recurrence
  \[ T(n) \leq 2T(n/2) + cn \]
  \[ m = n \]
  \[ \leq 2 \left[ 2T(n/4) + c(n/2) \right] + cn \]
  \[ m = n/2 \]
  \[ = 4T(n/4) + 2cn \]
  \[ m = n/4 \]
  \[ \leq 4 \left[ 2T(n/8) + c(n/4) \right] + 2cn \]
  \[ \leq \ldots \]
  \[ \leq nT(1) + \log_2(n) \cdot cn = O(n \log n) \]

Recurrence Solving (2): Recursion Tree

- Idea 1: “unroll” the recurrence
- **Idea 2:** Recursion tree
- **Idea 3:** Guess and verify (proof by induction)

Recurrence Solving (3): Guess and Verify

- **Idea 1:** “unroll” the recurrence
- **Idea 2:** Recursion tree
- **Idea 3:** Guess and verify (proof by induction)

Clicker

Suppose we have the recurrence \( T(n) = T(n/2) + T(n/3) \). What do we get after two unrollings?

A. \( T(n/4) + T(n/9) \)
B. \( T(n/4) + 2T(n/6) + T(n/9) \)
C. \( 2T(n/6) \)
D. \( T(n/4) + T(n/6) + T(n/9) \)

Recurrence Solving (2): Recursion Tree

- **Idea 1:** “unroll” the recurrence
- **Idea 2:** Recursion tree
- **Idea 3:** Guess and verify (proof by induction)
- **Conclusion:** \( T(n) \leq cn \log n \)

Induction step

Strong induction:
- Assume \( T(m) \leq c \cdot m \log m \) for all \( m < n \)
- Prove \( T(n) \leq c \cdot n \log n \)

\[ T(n) \leq 2T(n/2) + cn \]
\[ \leq 2 (c(n/2) \log(n/2) + cn) \]
\[ \leq 2c(n/2) \log(n/2) + c2n \]
\[ = cn(\log n - 1) + cn \]
\[ = cn \log n \]
Summary

Three approaches to solve first recurrence:

1. Unrolling ✓
2. Recursion tree ✓
3. Guess and verify (proof by induction) ✓

Next: other recurrences!