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)
- Can we extract any observations?

Clicker Question

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

A Problem from HW1 (replace max with sum)

\[
\text{MSS}(A) \\
\text{Initialize all entries of } n \times n \text{ array } B \text{ to zero} \\
\text{for } i = 1 \text{ to } n \text{ do} \\
\quad \text{sum} = 0 \\
\quad \text{for } j = i \text{ to } n \text{ do} \\
\quad \quad \text{compute sum of } A[i], A[i+1], \ldots, A[j] \\
\quad \quad B[i,j] = \text{sum} \\
\quad \text{end for} \\
\text{end for} \\
\text{Return maximum value among all } B[i,j] \\
\]  

**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

  \[
  A = \begin{cases} 
  4, -3, 5, -2, -1, 2, 6, -2 \\
  \text{L=6} \\
  \text{R=8} 
  \end{cases}
  \]

- **Return** \( \max(L, R, M) \)
- **Exercise:** change one entry to make MSS = \( R = -2 \to -10 \)
- **How to find** \( L, R, M \) ?

### Recurrence

- **Recurrence**
  - \( T(n) = 2T(n/2) + O(n) \)
  - **Recurrence relation**
    - sequence \( T(0), T(1), T(2), \ldots \)
    - \( T(n) \) defined in terms of smaller values
    - Technically \( T(\cdot) \) only valid for integers, but...
  - **Need base case(s)**
    - \( T(0) = O(1) \)
    - \( T(1) = O(1) \)
    - \( T(2) = O(1) \)
  - **Choose convenient one**

### 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)

1. **Base case**
   - if \( \text{right} - \text{left} \leq 2 \) then
     - Solve directly and return MSS
   - end if
2. **Recursion on left and right halves**
   - mid = \([\text{left} + \text{right}] / 2\)
   - \( L = \text{MSS}(A, \text{left}, \text{mid}) \)
   - \( R = \text{MSS}(A, \text{mid} + 1, \text{right}) \)
   - Set \( \text{sum} = 0 \) and \( L' = \max(L, \text{sum}) \)
   - for \( i = \text{mid} \) down to \( 1 \) do
     - \( \text{sum} += A[i] \)
     - \( L' = \max(L', \text{sum}) \)
   - end for
   - \( R' = \max(R, \text{sum}) \)
   - for \( i = \text{mid} + 1 \) to \( \text{right} \) do
     - \( \text{sum} += A[i] \)
     - \( R' = \max(R', \text{sum}) \)
   - end for
   - \( M = L' + R' \)
   - return \( \max(L, R, M) \)

### Clicker Question

- **Recurrence**
  - \( T(n) = 2T(n/2) + O(n) \)
  - **Recurrence relation**
    - sequence \( T(0), T(1), T(2), \ldots \)
    - \( T(n) \) defined in terms of smaller values
    - Technically \( T(\cdot) \) only valid for integers, but...
  - **Need base case(s)**
    - \( T(0) = O(1) \)
    - \( T(1) = O(1) \)
    - \( T(2) = O(1) \)
  - **Choose convenient one**

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 definition.
- C. Take \( c = \max\{c_1, c_2\} \) where \( c_1 \) and \( c_2 \) are the values from each definition.
Recurrence

- Same recurrence with change of variable
  \[ T(m) \leq 2T(m/2) + cn, \quad m \geq 2 \]
  \[ T(1) \leq c \]
- 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
  \[ T(n) \leq 2T(n/2) + cn \]
  \[ m = n \]
  \[ \leq 2T(n/4) + cn \]
  \[ m = n/2 \]
  \[ = 4T(n/4) + 2cn \]
  \[ m = n/4 \]
  \[ \leq 4T(n/8) + 3cn \]
  \[ m = n/8 \]
  \[ \leq \ldots \]
- Do you see a pattern? \( T(n) \leq 2^kT(n/2^k) + k \cdot cn \)
- When does this stop?
  - Base case: \( n/2^k = 1 \) \( \implies k = \log_2 n \)
  - \( T(n) \leq 2^{\log_2 n} \cdot T(1) + \log n \cdot cn = cn + cn \log n = O(n \log n) \)

Recurrence Solving (2): Recursion Tree

- Each level adds work to expand nodes
- Conclusion: \( T(n) \leq cn \log n \)

Recurrence Solving (3): Guess and Verify

- Guess solution. \( T(n) \leq cn \log n \)
- Prove by (strong) induction
  - Base case
    \[ T(2) \leq c < c \cdot 2 = c \cdot 2 \log 2 \]
  - Why \( T(2) \) instead of \( T(1) \)?
Induction step

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

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

Selecting Constants

Base case: \( T(2) \leq a \) \( \Rightarrow \) \( a \cdot 2 \log 2 = 2k \). Need \( k \geq a/2 \).

Induction step:

\[
T(n) \leq 2T(n/2) + cn \\
\leq 2 \cdot kn \log n + (c-k)n \\
\leq 2 \cdot kn \log n
\]
Need \( k \geq c \). Induction proof works for \( k = \max(c, a/2) \) (constant!)

A More General Recurrence

\[
T(n) \leq q \cdot T(n/2) + cn
\]

- What does the algorithm look like?
  - \( q \) recursive calls to itself on problems of half the size
  - \( O(n) \) work outside of the recursive calls
- Exercises: \( q = 1, q > 2 \) (recursion trees)
- Useful fact (geometric sum): if \( r \neq 1 \) then

\[
1 + r + r^2 + \ldots + r^d = \frac{1 - r^{d+1}}{1-r}
\]

General Case

Work at level \( j \) of recursion tree:

\[
\begin{align*}
\text{num. subproblems} & \quad \times \quad \text{work per subproblem} \\
q^j & \quad \times \quad cn/2^j \\
= (q/2)^j \cdot cn
\end{align*}
\]

Total work:

\[
T(n) = cn \sum_{j=0}^{d} \left( \frac{q}{2} \right)^j \quad (d = \log_2 n)
\]

\( q = 1? \) Easy to see \( T(n) \leq 2cn = O(n) \) for \( q = 1 \).

General Case \( (q > 2) \)

\[
T(n) = cn \sum_{j=0}^{d} \left( \frac{q}{2} \right)^j \quad (d = \log_2 n)
\]

Let \( r = q/2 > 1 \). Then

\[
\sum_{j=0}^{d} r^j = \frac{r^{d+1} - 1}{r-1}
\]

Therefore,

\[
T(n) = cn \cdot O(n^{\log_2 q - 1}) = O(n^{\log_2 q})
\]

E.g., \( q = 3, T(n) \) is \( O(n^{1.59}) \)

Wow, that worked out well...

Constants were chosen carefully \( \Rightarrow \) everything simplifies nicely.
If you don’t know correct constants, leave them as variables and set them at the end (important for more complicated cases).

Recurrence:

\[
T(n) \leq 2T(n/2) + cn \\
T(2) \leq a
\]

Guess:

\[
T(n) \leq kn \log n
\]
Summary

Useful general recurrence and its solutions:

\[ T(n) \leq q \cdot T(n/2) + cn \]

1. \( q = 1 \): \( T(n) = O(n) \)
2. \( q = 2 \): \( T(n) = O(n \log n) \)
3. \( q > 2 \): \( T(n) = O(n \log_2 q) \)

Algorithms with these running times?

1. ???
2. MSS, Mergesort
3. Integer multiplication (next time)