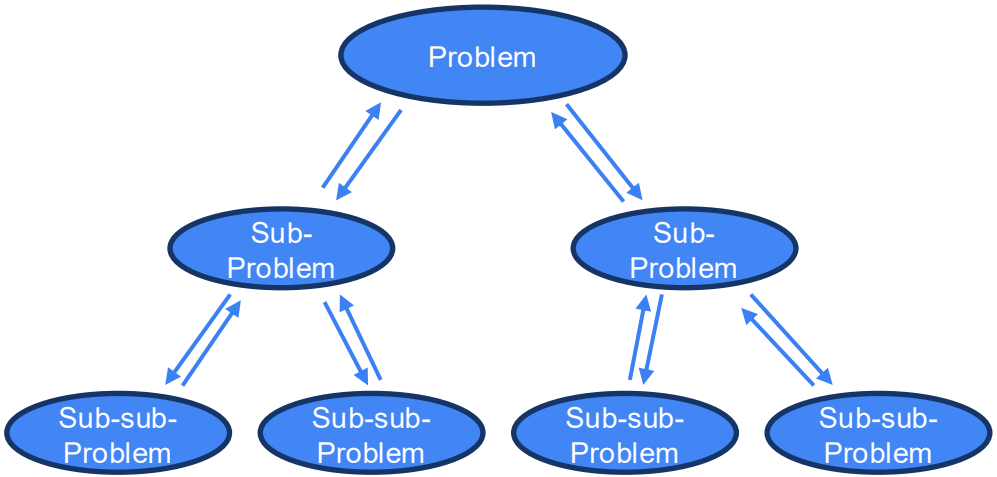
Lecture 16

Divide and Conquer II



Announcements

- Homework 5 due this Sunday night (4/5)
- Group Meetings continue
  - New groups next week
- Individual Project 2 due next Sunday night (4/5)
- Office Hours
  - Today from 3:30pm to 6pm

Last Time

- Worked through MergeSort
- Analyzed MergeSort's complexity
  - Viewed the algorithm as a tree and multiplied the work per layer by the height of the tree (recursion tree method)
  - Solved a recurrence relation (substitution/unrolling method)
  - These method will work for any divide and conquer algorithm

Today

- Work through another example: maximum subsequence sum
- Running time for other D&C patterns

Maximum Subsequence Sum

Input: a list L of n numbers

Goal: find the value of the largest subsequence sum (including empty subsequence)

Example

- $L = [4, -1, 8, -2, 3, -5, 1, -5]$

- MSS: 14 (sum of first five elements)

Exercise 1

Which of the following are 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 subsequence of negative numbers is bordered by a subsequence of positive numbers with a larger sum in absolute value

Exercise 1

Which of the following are 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 subsequence of negative numbers is bordered by a subsequence of positive numbers with a larger sum in absolute value

Baseline MSS Algorithm

Let’s come up with a baseline algorithm to solve the MSS problem

Baseline MSS Algorithm

Let's come up with a baseline algorithm to solve the MSS problem

- Idea: check all subsequences

MSS_baseline(L):
    $n \leftarrow len(L)$
    Initialize A as an $n \times n$ matrix of zeroes
    for $i = 1$ to n **do**
        sum $\leftarrow 0$
        for $j = i$ to n **do**
            sum += L[j]
            $A[i, j] \leftarrow$ sum
    return $\max_{i,j} A[i, j]$

Baseline MSS Algorithm

Let's come up with a baseline algorithm to solve the MSS problem

- Idea: check all subsequences

MSS_baseline(L):
    $n \leftarrow len(L)$
    Initialize A as an $n \times n$ matrix of zeroes
    for $i = 1$ to n **do**
        sum $\leftarrow 0$
        for $j = i$ to n **do**
            sum += L[j]
            $A[i, j] \leftarrow$ sum
    return $\max_{i,j} A[i, j]$

Running time: $O(n^2)$

Baseline MSS Algorithm

Let's come up with a baseline algorithm to solve the MSS problem

- Idea: check all subsequences

MSS_baseline(L):
    $n \leftarrow len(L)$
    Initialize A as an $n \times n$ matrix of zeroes
    for $i = 1$ to n **do**
        sum $\leftarrow 0$
        for $j = i$ to n **do**
            sum += L[j]
            $A[i, j] \leftarrow$ sum
    return $\max_{i,j} A[i, j]$

Running time: $O(n^2)$

Can we do better?

Divide-and-Conquer for MSS

Recursive algorithm for MSS

Divide-and-Conquer for MSS

Recursive algorithm for MSS

Idea:

- Find MSS X in left half of list
- Find MSS Y in right half of list
- Find MSS Z for subsequences that cross the midpoint of list

Divide-and-Conquer for MSS

Recursive algorithm for MSS

Idea:

- Find MSS X in left half of list
- Find MSS Y in right half of list
- Find MSS Z for subsequences that cross the midpoint of list

$$L = [4, -1, 8, -2, 3, -5, 1, -5]$$
$$X = 13;\ Y = 3;\ Z = 14$$

Divide-and-Conquer for MSS

Recursive algorithm for MSS

Idea:

- Find MSS X in left half of list
- Find MSS Y in right half of list
- Find MSS Z for subsequences that cross the midpoint of list

$$L = [4, -1, 8, -2, 3, -5, 1, -5]$$
$$X = 13;\ Y = 3;\ Z = 14$$

- Return maximum of X, Y, Z

Divide-and-Conquer for MSS

Recursive algorithm for MSS

Idea:

- Find MSS X in left half of list
- Find MSS Y in right half of list
- Find MSS Z for subsequences that cross the midpoint of list

$$L = [4, -1, 8, -2, 3, -5, 1, -5]$$
$$X = 13;\ Y = 3;\ Z = 14$$

- Return maximum of X, Y, Z
- How do we find X, Y, Z?

Divide-and-Conquer for MSS

$\text{MSS\_dc}(L, low, high)$:

if $high == low$ **do**

return $L[low]$

$mid \leftarrow \left\lfloor \frac{low+high}{2} \right\rfloor$

$X = \text{MSS\_dc}(L, low, mid)$

$Y = \text{MSS\_dc}(L, mid + 1, high)$

return $\max(X, Y, Z)$

Divide-and-Conquer for MSS

MSS_dc$(L, low, high)$:

if $high == low$ **do**

return $L[low]$

$mid \leftarrow \left\lfloor \frac{low+high}{2} \right\rfloor$

$X = \text{MSS\_dc}(L, low, mid)$

$Y = \text{MSS\_dc}(L, mid + 1, high)$

$sum \leftarrow 0; X' \leftarrow 0$

for $i = mid$ to low **do**

$sum \mathrel{+}= L[i]$

$X' = \max(sum, X')$

$sum \leftarrow 0; Y' \leftarrow 0$

for $j = mid + 1$ to $high$ **do**

$sum \mathrel{+}= L[j]$

$Y' = \max(sum, Y')$

$Z = X' + Y'$

return $\max(X, Y, Z)$

Divide-and-Conquer for MSS

MSS_dc$(L, low, high)$:

if $high == low$ **do**

return $L[low]$

Base case: $O(1)$

$mid \leftarrow \left\lfloor \frac{low+high}{2} \right\rfloor$

$X = \text{MSS\_dc}(L, low, mid)$

$Y = \text{MSS\_dc}(L, mid + 1, high)$

Recursive cases

$sum \leftarrow 0; X' \leftarrow 0$

for $i = mid$ to low **do**

$sum \mathrel{+}= L[i]$

$X' = \max(sum, X')$

Compute left part of Z

$sum \leftarrow 0; Y' \leftarrow 0$

for $j = mid + 1$ to $high$ **do**

$sum \mathrel{+}= L[j]$

$Y' = \max(sum, Y')$

Compute right part of Z

$Z = X' + Y'$

Compute Z

return $\max(X, Y, Z)$

Return max

Running Time

Let $T(n)$ be the running time of $\mathrm{MSS\_dc}(L)$ on a list of size n

- Base case is $O(1)$
- Two recursive calls on arrays of size $n/2$: $2T(n/2)$
- Work outside of recursive calls: $O(n)$

Running Time

Let $T(n)$ be the running time of $\text{MSS\_dc}(L)$ on a list of size n

- Base case is $O(1)$
- Two recursive calls on arrays of size $n/2$: $2T(n/2)$
- Work outside of recursive calls: $O(n)$

Therefore, $T(n) = 2T(n/2) + O(n)$

Running Time

Let $T(n)$ be the running time of $\text{MSS\_dc}(L)$ on a list of size n

- Base case is $O(1)$
- Two recursive calls on arrays of size $n/2$: $2T(n/2)$
- Work outside of recursive calls: $O(n)$

Therefore, $T(n) = 2T(n/2) + O(n)$

Implies the running time is $O(n \log n)$

Other D&C Patterns

What are some other patterns we may consider?

Other D&C Patterns

What are some other patterns we may consider?

We can describe these through a recurrence relation:

$$T(n) = aT\left(n/b\right) + f(n)$$

- $T(n)$ is the running time on an input of size n
- a is the number of subproblems in the recursion
- b is the factor by which the subproblem is reduced
- $f(n)$ is cost of dividing the problem and combining the results

Example

Strassen's algorithm for matrix multiplication

$$T(n) = 8T\left(n/2\right) + \Theta(n^2)$$

Example

Strassen's algorithm for matrix multiplication

$$T(n) = 8T\left(n/2\right) + \Theta(n^2)$$

- 8 subproblems in the recursion
- Subproblem is reduced by a factor of 2
- The cost of dividing the problem and combining the results is exactly quadratic

Example

Strassen's algorithm for matrix multiplication

$$T(n) = 8T\left(n/2\right) + \Theta(n^2)$$

- 8 subproblems in the recursion
- Subproblem is reduced by a factor of 2
- The cost of dividing the problem and combining the results is exactly quadratic
- What does the recursion tree look like?

Example

Strassen's algorithm for matrix multiplication

$$T(n) = 8T\left(n/2\right) + \Theta(n^2)$$

Solving this, we get the following running time

- $T(n) = O\left(n^{\log_2 7}\right) \approx O(n^{2.81})$
- Can solve this using the Master Theorem (or Unified Method) for D&C algorithms
- Set of rules for solving recurrence relations of this form

Next Time

- Dynamic programming