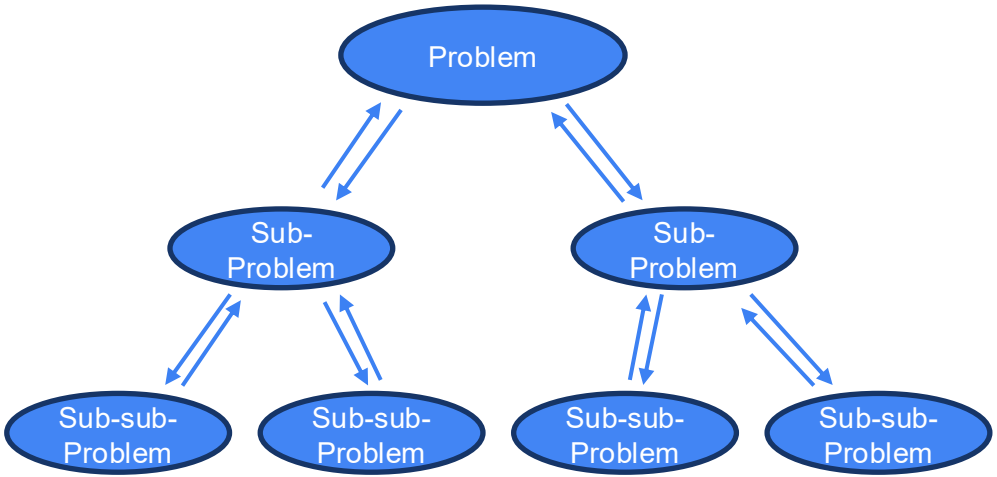
Lecture 14

Divide and Conquer I



Announcements

- Homework 5 due this Sunday night (3/29)
- Group Meetings continue
  - New groups next week
- Individual Project 2 due next Sunday night (4/5)

Greedy Algorithms

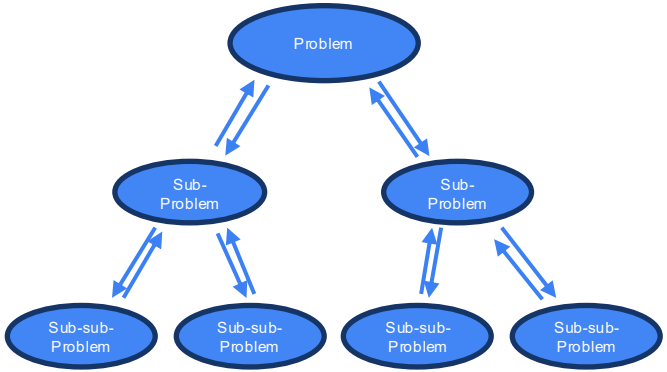
We are moving on to our study of algorithm design

- Greedy
- **Divide-and-conquer**
- Dynamic Programming
- Network Flow

Divide and Conquer Idea

General Recipe (or pattern)

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



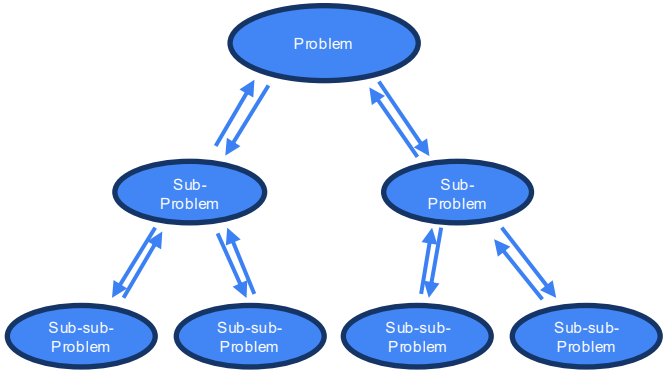
Divide and Conquer Idea

General Recipe (or pattern)

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

Our Focus

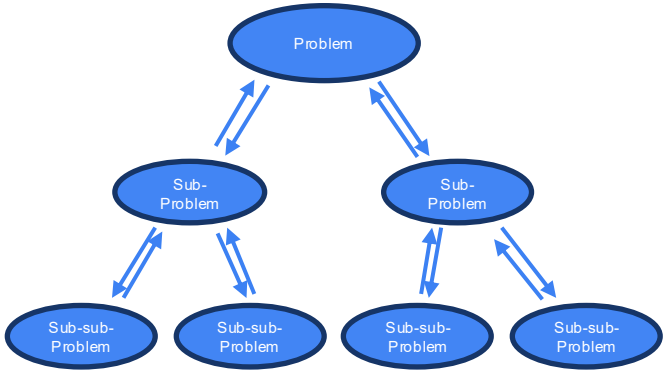
- Algorithm design
- Running time analysis



Divide and Conquer Idea

Typical Setting

- Divide problem of size n into two subproblems of size $n/2$
- Solve two subproblem recursively
- Combine two solutions into overall solution



Running Time Analysis

- Brute force: $\Theta(n^2)$
- Divide-and-conquer: $O(n \log n)$

Merge Sort

Problem: given a list L of n unique items, arranger them in ascending order

$\text{MergeSort}(L)$:
    if L has one item **do**
        return L
    Divide L into two halves A, B
    $A \leftarrow \text{MergeSort}(\text{A})$
    $B \leftarrow \text{MergeSort}(\text{B})$
    $L \leftarrow \text{Merge}(A, B)$
    return L

Merge Sort

Problem: given a list L of n unique items, arranger them in ascending order

Let $T(n)$ be the worst-case running time of $\mathrm{MergeSort(L)}$

$\mathrm{MergeSort}(L)$:
    if L has one item **do**
        return L
    Divide L into two halves A, B
    $A \leftarrow \mathrm{MergeSort(A)}$
    $B \leftarrow \mathrm{MergeSort(B)}$
    $L \leftarrow \mathrm{Merge}(A, B)$
    return L

Base case: $O(1)$

Recursive cases: $T(n/2)$

Combination step: $\Theta(n)$

Merge Sort

Problem: given a list L of n unique items, arranger them in ascending order

Let $T(n)$ be the worst-case running time of $\mathrm{MergeSort(L)}$

$\mathrm{MergeSort}(L)$:

if L has one item **do**

return L

Divide L into two halves A, B

$A \leftarrow \mathrm{MergeSort(A)}$

$B \leftarrow \mathrm{MergeSort(B)}$

$L \leftarrow \mathrm{Merge}(A, B)$

return L

Base case: $O(1)$

Recursive cases: $T(n/2)$

Combination step: $\Theta(n)$

$$T(n) = 2T\left({}^{n}/_{2}\right) + \Theta(n)$$

Recurrence

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

- $T(n)$ is defined in terms of small values of n
- For base case: $T(1)$ is $O(1)$

Goal: "solve" the recurrence by finding a simple expression for $T(n)$ for all n

Recurrence

Recurrence:

- $T(n) = 2T(n/2) + O(n)$
- $T(1) = O(1)$

Let's the use definition on Big-O:

- $T(n) \leq 2T(n/2) + cn$
- $T(1) \leq c$

Exercise 1

Recurrence:

- $T(n) = 2T(n/2) + O(n)$
- $T(1) = O(1)$

Let's the use definition on Big-O:

- $T(n) \leq 2T(n/2) + cn$
- $T(1) \leq c$

Q: Why can we use the same value of c in both instances of the Big-O definition?

a) Take $c = \min(c_1, c_2)$ where c_1 and c_2 are the values from each instance

b) Take $c = \max(c_1, c_2)$ where c_1 and c_2 are the values from each instance

c) It's not okay!

Exercise 1

Recurrence:

- $T(n) = 2T(n/2) + O(n)$
- $T(1) = O(1)$

Let's the use definition on Big-O:

- $T(n) \leq 2T(n/2) + cn$
- $T(1) \leq c$

Q: Why can we use the same value of c in both instances of the Big-O definition?

a) Take $c = \min(c_1, c_2)$ where c_1 and c_2 are the values from each instance

b) Take $c = \max(c_1, c_2)$ where c_1 and c_2 are the values from each instance

c) It's not okay!

Recurrence

Recurrence:

- $T(n) = 2T(n/2) + O(n)$
- $T(1) = O(1)$

Let's the use definition on Big-O:

- $T(n) \leq 2T(n/2) + cn$
- $T(1) \leq c$

Strategies for Solving:

1. Unrolling recursion
2. Recursion tree (more structured unrolling)

Unrolling

Idea: "unroll" the recurrence

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + cn \\
&\leq 2[2T(n/4) + c(n/2)] + cn \\
&= 4T(n/4) + 2cn \\
&\leq 4[2T(n/8) + c(n/4)] + 2cn \\
&= 8T(n/4) + 3cn \\
&\leq \cdots \\
&\ldots \\
&\leq nT(1) + \log_2 n \cdot cn = O(n \log n)
\end{aligned}
$$

Exercise 2

Q: 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. $T(n/4) + T(n/6) + T(n/9)$

Exercise 2

Q: suppose we have the recurrence $T(n) = T(n/2) + T(n/3)$. What do we get after two unrollings?

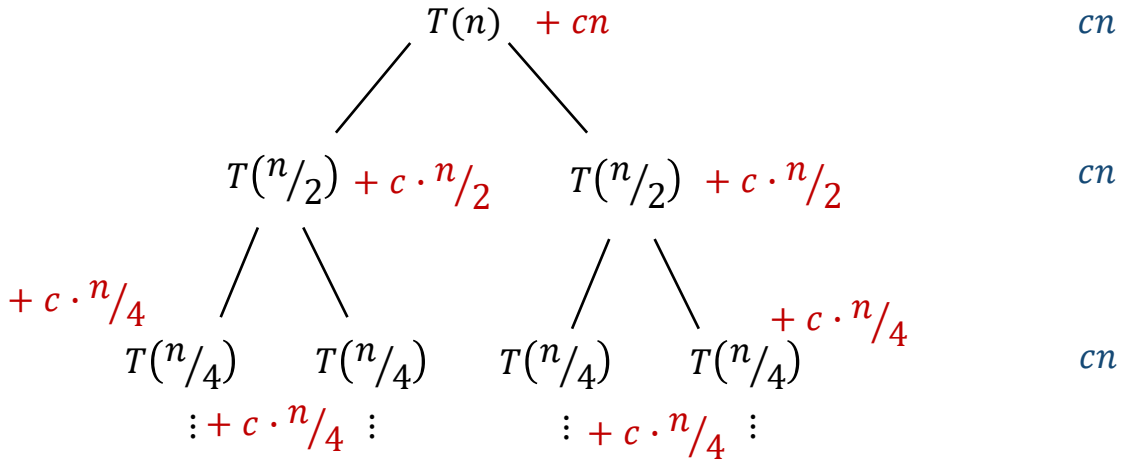
a. $T(n/4) + T(n/9)$

b. $T(n/4) + 2T(n/6) + T(n/9)$

c. $2T(n/6)$

d. $T(n/4) + T(n/6) + T(n/9)$

Recursion Tree



$\log_2 n + 1$ levels each with cn work

- $T(n) \leq cn(\log n + 1)$ implies $T(n)$ is $O(n \log n)$

Guess and Verify

Another approach is to guess the running time and proof it using induction

- We’re going to skip this one but can talk about it during office hours

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 3

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 3

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 $O(n \log n)$

Next Time

- Running time of more complex divide and conquer approaches