

CSE 100: Algorithm Design and Analysis

Chapter 04: Divide-and-Conquer

Sungjin Im

University of California, Merced

Last Update: 01-25-2024

I order you to be supremely happy.
- Audrey Hepburn

Divide and Conquer

- ▶ Divide the problem into a number of smaller subproblems.
- ▶ Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.
- ▶ Combine the solutions to the subproblems into the solution for the original problem.

Sorting via Divide and Conquer: Merge sort

- ▶ Divide: Divide the n -element sequence to be sorted into two subsequences of $n/2$ elements each.
- ▶ Conquer: Sort the two subsequences recursively using merge sort. If there's only one element, do nothing.
- ▶ Combine: Merge the two sorted subsequences to produce the sorted answer.

Chapter Overview

1. See more examples of algorithms based on divide and conquer
 - ▶ The max-subarray problem.
 - ✗ Matrix multiplication.

Chapter Overview

2. Learn how to solve recursions on running time.

$$\text{ex)} \quad T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n \geq 2. \end{cases}$$

- ▶ substitution method: guesses a bound and prove it using induction.
- ✓ recursion-tree method: derive a tree that represents workload at different levels.
- ✗ master theorem: a theorem. powerful but not always applicable.

Part 1: Examples of divide and conquer

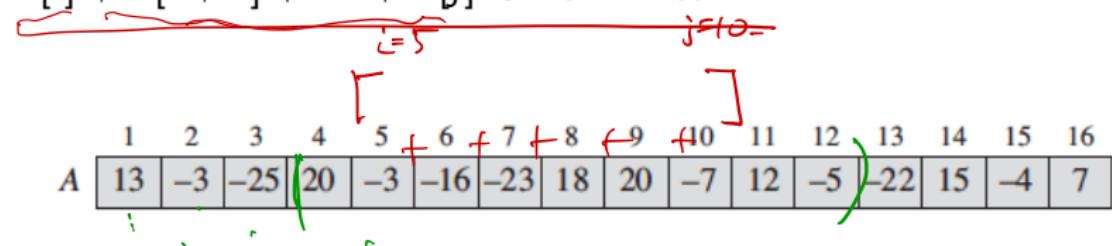
- ▶ The max-subarray problem.
- ▶ Matrix multiplication.

The maximum subarray problem

Input: An array $A[1 \dots n]$ of numbers.

Output: Indices i and j ($1 \leq i \leq j \leq n$) s.t.

$A[i] + A[i + 1] + \dots + A[j]$ is maximized.



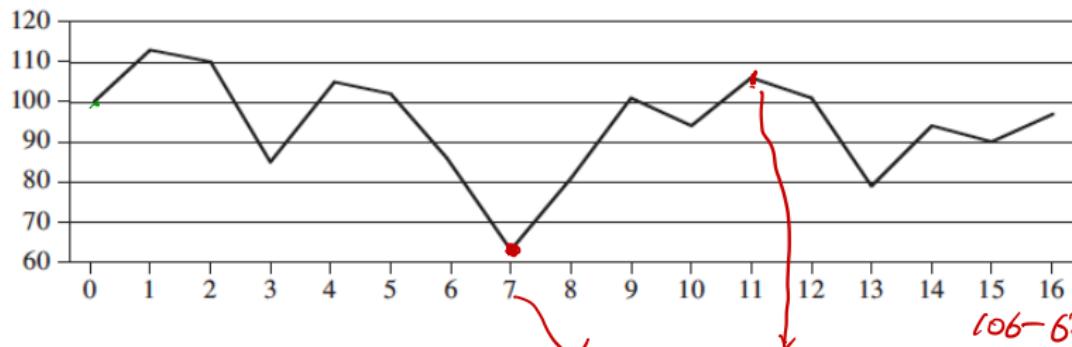
The maximum subarray problem

Stock market view

Buy one stock one day and sell it later at a higher price with the goal of maximizing the profit.

Say stock price = 100 on day 0, i.e. $\text{Price}[0] = 100$

$$\text{Price}[n] = \text{Price}[n-1] + A[n]$$



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7	

Below the table, blue brackets group the changes for days 7, 8, 9, and 10, with a total sum of 43.

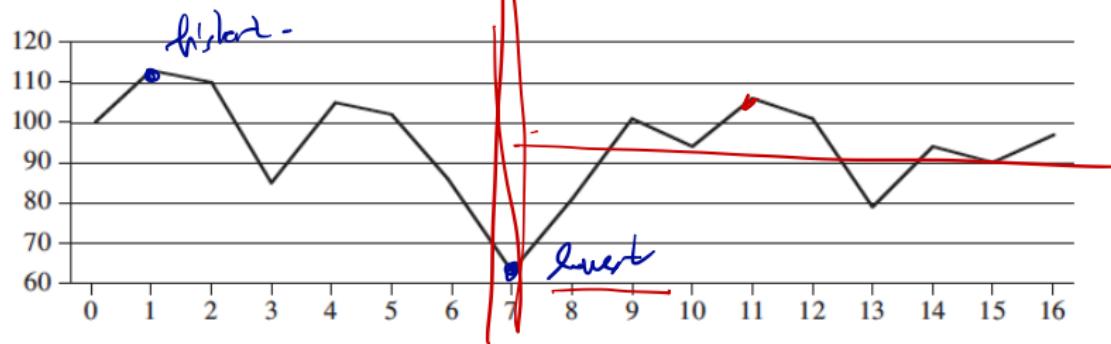
$$= 43$$

The maximum subarray problem

Stock market view

X How about buying a stock at the lowest price and selling it at the highest price?

$$\text{Price}[0] = 100 \text{ and } \text{Price}[n] = \text{Price}[n-1] + A[n]$$

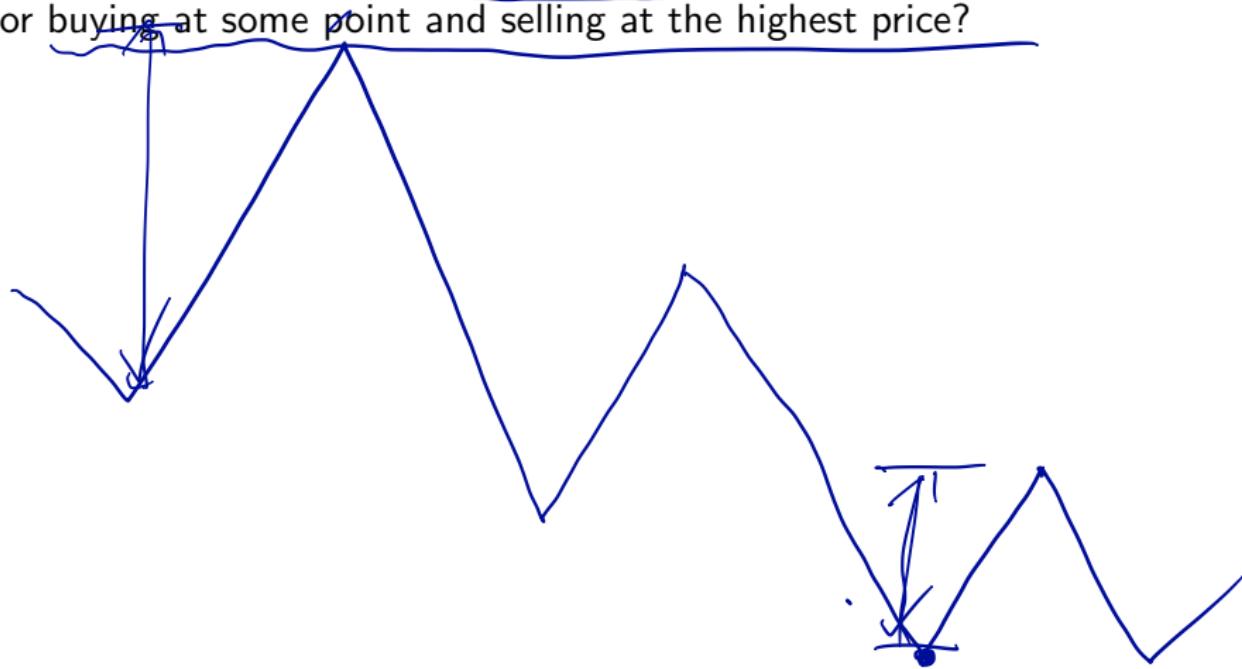


Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

The maximum subarray problem

Stock market view

How about buying a stock at the lowest price and selling it later,
or buying at some point and selling at the highest price?

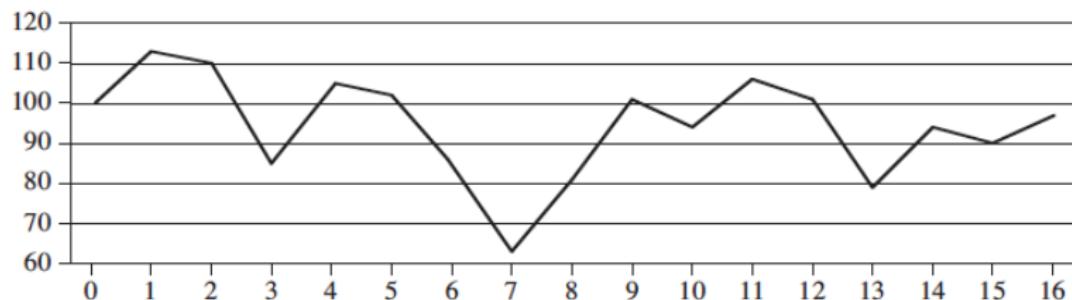


The maximum subarray problem

Stock market view

How about buying a stock at the lowest price and selling it later, or buying at some point and selling at the highest price?
It works in this example, but not always.

$$\text{Price}[0] = 100 \text{ and } \text{Price}[n] = \text{Price}[n-1] + A[n]$$



Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

The maximum subarray problem

Max - Min?

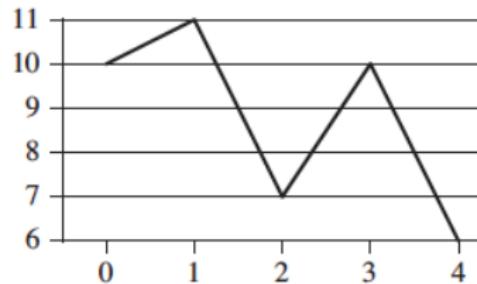
Buying at the lowest price and selling at the highest price?

The maximum subarray problem

Max - Min?

Buying at the lowest price and selling at the highest price?
Doesn't always work...

$$\text{Price}[0] = 100 \text{ and } \text{Price}[n] = \text{Price}[n-1] + A[n]$$



Day	0	1	2	3	4
Price	10	11	7	10	6
Change		1	-4	3	-4

The maximum subarray problem

Naive algorithm?

$$x_1 \left[\begin{array}{c} \\ x_1 \left[\begin{array}{c} \\ O(n) \end{array} \right. \end{array} \right. \Rightarrow$$

for $i=1$ to n .

for $j=i$ to n .

return $\underbrace{S[i, j] = A[i] + \dots + A[j]}$
s.t. $S[i, j]$ is maximized.

$\Rightarrow O(n^3)$

Can try every possible pair (i, j) ($i \leq j$) and compute $A[i] + A[i + 1] + \dots + A[j]$.

$$\begin{aligned} & S[i, n-1] \\ & S[i, n] \end{aligned}$$

$$S[i, n-1] + A[n] = S[i, n].$$

The maximum subarray problem

Naive algorithm?

Can try every possible pair (i, j) ($i \leq j$) and compute $A[i] + A[i + 1] + \dots + A[j]$.

Running time?

The maximum subarray problem

Naive algorithm?

Can try every possible pair (i, j) ($i \leq j$) and compute $A[i] + A[i + 1] + \dots + A[j]$.

Running time? $\Theta(n^3)$.

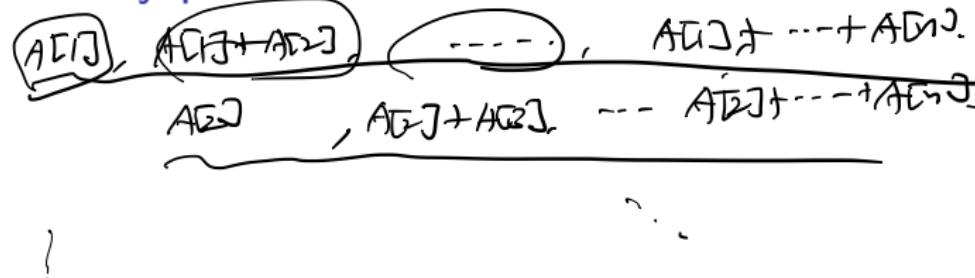
The maximum subarray problem

Naive algorithm?

$O(n^2)$

$O(n)$

$O(n)$



Can try every possible pair (i, j) ($i \leq j$) and compute $A[i] + A[i + 1] + \dots + A[j]$.

Running time? $\Theta(n^3)$. Can be improved to $\Theta(n^2)$. How?

The maximum subarray problem

A simpler version

$$\mathcal{O}(n \log n)$$

Simpler problem:

Definition:

$$\underline{S[i, j] := A[i] + A[i + 1] + \dots + A[j]}.$$

Goal:

Compute

$$\max_{1 \leq i \leq j \leq n} S[i, j].$$

(Let's call this quantity ~~OPT(1, n)~~)

b) optimizer

The maximum subarray problem

A simpler version

Simpler problem:

Definition:

$$S[i, j] := A[i] + A[i + 1] + \cdots + A[j].$$

Goal:

Compute

$$\max_{1 \leq i \leq j \leq n} S[i, j].$$

(Let's call this quantity $OPT(1, n)$)

Original problem:

Find i and j s.t. $1 \leq i \leq j \leq n$ maximizing $S[i, j]$.

The maximum subarray problem

A simpler version

Simpler problem:

Definition:

$$S[i, j] := A[i] + A[i + 1] + \cdots + A[j].$$

Goal:

Compute

$$\max_{1 \leq i \leq j \leq n} S[i, j].$$

(Let's call this quantity $OPT(1, n)$)

Original problem:

Find i and j s.t. $1 \leq i \leq j \leq n$ maximizing $S[i, j]$.

(Once we know such i and j , we can easily compute $OPT(1, n)$.)

The maximum subarray problem

Divide-and-Conquer

Divide the input into two subarrays of an (almost) equal size.

The maximum subarray problem

$\text{OPT}(\text{low}, \text{high})$?

Divide-and-Conquer



Divide the input into two subarrays of an (almost) equal size.
If the input is $A[\text{low} \dots \text{high}]$, then the two inputs to the two subproblems will be $A[\text{low} \dots \text{mid}]$ and $A[\text{mid} + 1 \dots \text{high}]$. m
Any contiguous subarray of $A[\text{low} \dots \text{high}]$ lies in

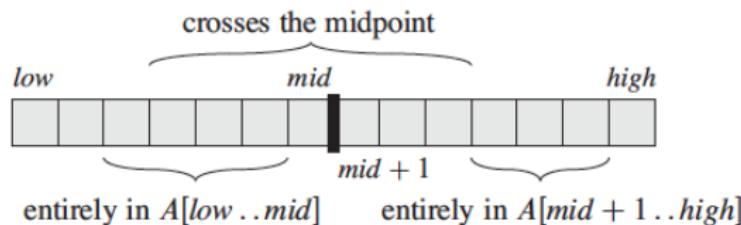
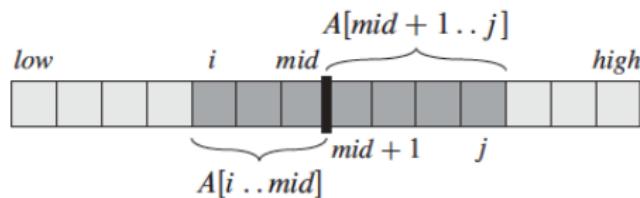
1. entirely in $A[\text{low} \dots \text{mid}]$, so that $\text{low} \leq i \leq j \leq \text{mid}$.
2. entirely in $A[\text{mid} + 1 \dots \text{high}]$, so that $\text{mid} < i \leq j \leq \text{high}$.
3. crossing the midpoint, so that $\text{low} \leq i \leq \text{mid} < j \leq \text{high}$.

The maximum subarray problem

Divide-and-Conquer

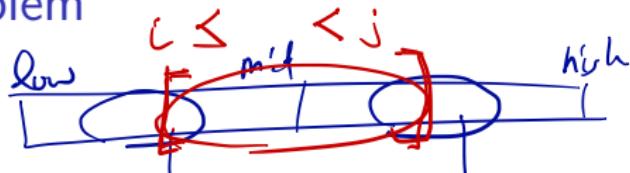
Any contiguous subarray of $A[low \dots high]$

- ▶ entirely lies in $A[low \dots mid]$, so that $low \leq i \leq j \leq mid$.
- ▶ entirely lies in $A[mid + 1 \dots high]$, so that $mid < i \leq j \leq high$.
- ▶ crosses the midpoint, so that $low \leq i \leq mid < j \leq high$.



The maximum subarray problem

Recursion



How to compute $OPT(low, high)$?

$$\max_{low \leq i \leq j \leq high} S[i, j] = \max\{$$



$$\max_{\text{low} \leq i \leq j \leq \text{mid}} S[i, j],$$

$$\max_{mid < i \leq j \leq high} S[i, j],$$

$$\max_{\substack{low \leq i \leq mid < j \leq high}} S[i, j]$$

}

The maximum subarray problem

Recursion

How to compute $OPT(low, high)$?

$$OPT(low, high) = \max\{$$

- $\checkmark OPT(low, mid),$
- $\checkmark OPT(mid + 1, high)$
- $\checkmark \max_{low \leq i \leq mid < j \leq high} S[i, j]$

$$\}$$

The maximum subarray problem

Our Goal: R.T. $O(n \lg n)$.

Recursion

How to compute $OPT(low, high)$?

$$OPT(low, high) = \max\{$$

$$OPT(low, mid),$$

$$OPT(mid + 1, high)$$

$$\max_{low \leq i \leq mid < j \leq high} S[i, j]$$

}

n .

The maximum subarray problem

Recursion

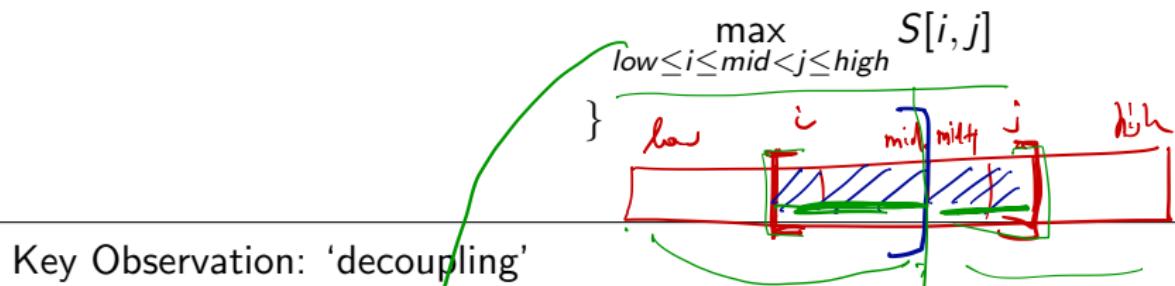
$A[low..high]$

How to compute $OPT(low, high)$?

$$OPT(low, high) = \max\{$$

$$\overbrace{OPT(low, mid)},$$

$$\overbrace{OPT(mid + 1, high)}$$



Key Observation: 'decoupling'

$$\max_{low \leq i \leq mid < j \leq high} S[i, j]$$

$$= \max_{low \leq i \leq mid} S[i, mid] + \max_{mid + 1 \leq j \leq high} S[mid + 1, j]$$

The maximum subarray problem

Recursion

How to compute $OPT(low, high)$?

$$\max_{OPT} OPT(\text{low}, \text{high}) = \max\{$$

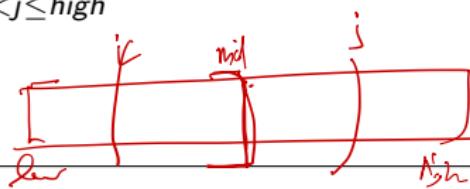
$$f[\mathbf{s}[mid, mid]] = A[mid] \quad b(i) \quad OPT(low, mid),$$

$$S_{[mid+1, mid]} = A_{[mid+1]} + A_{[mid]} \quad \Theta(\cdot) \text{ } OPT(mid+1, high)$$

$$S[\text{mid} \rightarrow, \text{mid} \rightarrow] = A[\text{mid} \rightarrow] + A[\text{mid} \rightarrow] + A[\text{mid} \rightarrow] \max_{\text{low} \leq i \leq \text{mid} \leq j \leq \text{high}} S[i, j]$$

$$S[\text{flow}, \text{M}, d] = A[\text{flow}] + \dots + A[\text{final}] \quad \text{oder}$$

Key Observation: 'decoupling'



$$\max_{\text{low} \leq i \leq \text{mid} \leq j \leq \text{high}} S[i, j]$$

$O(\text{high-mid})$

$$= \max_{low \leq i \leq mid} S[i, mid] + \max_{mid+1 \leq j \leq high} S[mid + 1, j]$$

Can be computed in $O(\text{high} - \text{low})$ time.

The maximum subarray problem

Recursion

How to compute $OPT(low, high)$?

$$OPT(low, high) = \max\{$$
$$\quad OPT(low, mid),$$
$$\quad OPT(mid + 1, high)$$
$$\quad \checkmark \max_{low \leq i \leq mid < j \leq high} S[i, j] \cdots (*)$$
$$\}$$

(*) can be computed in $O(high - low)$ time.

Base case?

The maximum subarray problem

Recursion

How to compute $OPT(low, high)$?

$$OPT(low, high) = \max\{$$
$$\quad OPT(low, mid),$$
$$\quad OPT(mid + 1, high)$$
$$\quad \max_{low \leq i \leq mid < j \leq high} S[i, j] \cdots (*)$$
$$\}$$

(*) can be computed in $O(high - low)$ time.

Base case? When $low = high$. $OPT(low, low = high)$

The maximum subarray problem

Recursion

How to compute $OPT(low, high)$?

$$OPT(low, high) = \max\{$$
$$\quad OPT(low, mid),$$
$$\quad OPT(mid + 1, high)$$
$$\quad \max_{low \leq i \leq mid < j \leq high} S[i, j] \cdots (*)$$
$$\}$$

(*) can be computed in $O(high - low)$ time.

Base case? When $low = high$. $OPT(low, low = high) = A[low]$.

The maximum subarray problem

Recursion

When $low < high$:

$$OPT(low, high) = \max\{$$
$$OPT(low, mid),$$
$$OPT(mid + 1, high)$$
$$\max_{low \leq i \leq mid < j \leq high} S[i, j] \cdots (*)$$

}

When $low = high$. $OPT(low, high) = A[low]$.

(*) can be computed in $O(high - low)$ time.

The maximum subarray problem

Recursion

Any contiguous subarray of $A[low \dots high]$ lies

- ▶ entirely lies in $A[low \dots mid]$, so that $low \leq i \leq j \leq mid$.
- ▶ entirely lies in $A[mid + 1 \dots high]$, so that $mid < i \leq j \leq high$.
- ▶ crosses the midpoint, so that $low \leq i \leq mid < j \leq high$.

The first two cases will be handled by

The maximum subarray problem

Recursion

Any contiguous subarray of $A[low \dots high]$ lies

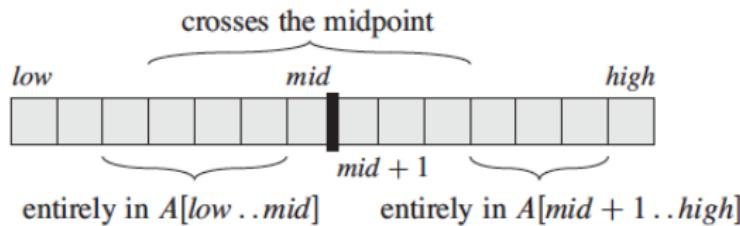
- ▶ entirely lies in $A[low \dots mid]$, so that $low \leq i \leq j \leq mid$.
- ▶ entirely lies in $A[mid + 1 \dots high]$, so that $mid < i \leq j \leq high$.
- ▶ crosses the midpoint, so that $low \leq i \leq mid < j \leq high$.

The first two cases will be handled by recursions. How do we handle the last crossing case?

The maximum subarray problem

Handling the crossing case

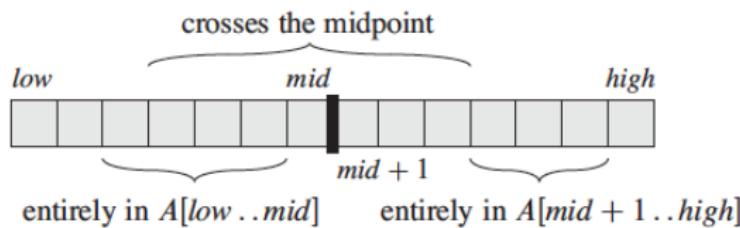
If the best subarray crosses the midpoint, so that $low \leq i \leq mid < j \leq high$, then how do we find it?



The maximum subarray problem

Handling the crossing case

If the best subarray crosses the midpoint, so that $low \leq i \leq mid < j \leq high$, then how do we find it?



Can be done in $\Theta(n)$ time!

The maximum subarray problem

"Semi"-pseudocode for the simpler problem

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

OPT(low , high)
if $\text{low} == \text{high}$ return $A[\text{low}]$
 $\text{mid} = \text{floor of } (\text{low} + \text{high})/2$
 $\text{LV} = \text{OPT}(\text{low}, \text{mid})$
 $\text{RV} = \text{OPT}(\text{mid}+1, \text{high})$
 $\text{CV} = \max \text{ sum of any subarray}$
with starting idx $\geq \text{low}$ and ending idx = mid
+ max sum of any subarray
with starting idx = $\text{mid}+1$ and ending idx = high
return $\max(\text{LV}, \text{RV}, \text{CV})$

$\Rightarrow T(n)$
 $\Rightarrow O(1) +$
 $\Rightarrow O(1) +$
 $\Rightarrow T\left(\frac{n}{2}\right) +$
 $\Rightarrow T\left(\frac{n}{2}\right) +$
 $\Rightarrow O(n, 2 - \text{low}) + = O(n)$
 $T(n) = \Theta(n \lg n)$

Note: The first term in CV can be computed in $\Theta(\text{mid} - \text{low})$ time by computing $S[\text{mid}, \text{mid}]$, $S[\text{mid} - 1, \text{mid}]$, ... $S[1, \text{mid}]$ in this order and taking the max. Similarly (symmetrically), the second term can be computed in $\Theta(\text{high} - \text{mid})$ time.

The maximum subarray problem

Running time

The maximum subarray problem

Running time

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n \geq 2. \end{cases}$$

The maximum subarray problem

Running time

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n \geq 2. \end{cases}$$

Or simply $T(n) = 2T(n/2) + \Theta(n)$

$$\begin{aligned} g &= n^{f_2^2} & f &= \underline{n}. & g &= \Theta(f) \\ &= n. & \Rightarrow T(n) &= \Theta(n \cdot f(n)). \end{aligned}$$

The maximum subarray problem

Running time

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n \geq 2. \end{cases}$$

Or simply $T(n) = 2T(n/2) + \Theta(n)$

$$\Rightarrow T(n) = \Theta(n \log n)$$

We will learn shortly how to solve this recurrence on RT.

The maximum subarray problem

Pseudocode (for the original problem)

FIND-MAXIMUM-SUBARRAY($A, low, high$)

```
1  if  $high == low$ 
2      return ( $low, high, A[low]$ )           // base case: only one element
3  else  $mid = \lfloor (low + high)/2 \rfloor$ 
4      ( $left-low, left-high, left-sum$ ) =
            FIND-MAXIMUM-SUBARRAY( $A, low, mid$ )
5      ( $right-low, right-high, right-sum$ ) =
            FIND-MAXIMUM-SUBARRAY( $A, mid + 1, high$ )
6      ( $cross-low, cross-high, cross-sum$ ) =
            FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )
7  if  $left-sum \geq right-sum$  and  $left-sum \geq cross-sum$ 
8      return ( $left-low, left-high, left-sum$ )
9  elseif  $right-sum \geq left-sum$  and  $right-sum \geq cross-sum$ 
10     return ( $right-low, right-high, right-sum$ )
11  else return ( $cross-low, cross-high, cross-sum$ )
```



The maximum subarray problem

Pseudocode (for the original problem)

If the best subarray crosses the midpoint, so that
 $low \leq i \leq mid < j \leq high$, then how do we find it?

FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)

```
1  left-sum = -∞
2  sum = 0
3  for i = mid downto low
4      sum = sum + A[i]
5      if sum > left-sum
6          left-sum = sum
7          max-left = i
8  right-sum = -∞
9  sum = 0
10 for j = mid + 1 to high
11     sum = sum + A[j]
12     if sum > right-sum
13         right-sum = sum
14         max-right = j
15 return (max-left, max-right, left-sum + right-sum)
```

find max-sum subarray ending at mid.

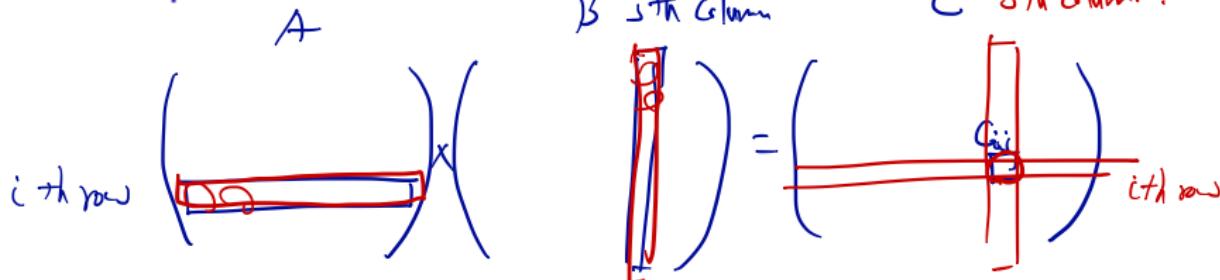
find max-sum subarray starting at mid+1.

Matrix Multiplication

Input: two $n \times n$ matrices A and B .

Output: AB .

Matrix Multiplication



Input: two $n \times n$ matrices A and B .

Output: AB .

Note: Let $A = (a_{ij})$, $B = (b_{ij})$, $C = (c_{ij})$, and $C = A \cdot B$. Then, for any entry c_{ij} , we have $c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$.

$$= \underbrace{a_{i1} b_{1j} + a_{i2} b_{2j} + \dots + a_{in} b_{nj}}$$

Matrix Multiplication

Naive

R.T of computing $A+B$ $\in O(n^2)$.

Lesson: Matrix multiplication is way more expensive than matrix addition.

SQUARE-MATRIX-MULTIPLY(A, B)

1 $n = A.\text{rows}$
2 let C be a new $n \times n$ matrix
3 for $i = 1$ to n
4 for $j = 1$ to n
5 $c_{ij} = 0$
6 for $k = 1$ to n
7 $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
8 return C

$a_{i1} b_{1j}$
 $a_{i2} b_{2j} +$
 \vdots
 $a_{in} b_{nj} +$

if row i th column. of C .

$O(1)$

$*n$

$O(n)$

$$n \times n \times O(n) = O(n^3)$$

$$O(n^3)$$

Matrix Multiplication

Naive

SQUARE-MATRIX-MULTIPLY(A, B)

```
1   $n = A.\text{rows}$ 
2  let  $C$  be a new  $n \times n$  matrix
3  for  $i = 1$  to  $n$ 
4      for  $j = 1$  to  $n$ 
5           $c_{ij} = 0$ 
6          for  $k = 1$  to  $n$ 
7               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
8  return  $C$ 
```

Running time:

Matrix Multiplication

Naive

SQUARE-MATRIX-MULTIPLY(A, B)

```
1   $n = A.\text{rows}$ 
2  let  $C$  be a new  $n \times n$  matrix
3  for  $i = 1$  to  $n$ 
4      for  $j = 1$  to  $n$ 
5           $c_{ij} = 0$ 
6          for  $k = 1$  to  $n$ 
7               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
8  return  $C$ 
```

Running time: $\Theta(\underline{n^3})$.

Matrix Multiplication

Strassen's algorithm

Based on clever algebraic tricks and divide-and-conquer.

Running time: $\Theta(n^{\frac{\log_2 7}{3}})$. ($2.8 < \log_2 7 < 2.81$).

Matrix Multiplication

Basic divide-and-conquer

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

It is well-known that

Suppose we partition each of A , B , and C into four $n/2 \times n/2$ matrices

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}, \quad (4.9)$$

so that we rewrite the equation $C = A \cdot B$ as

$$\begin{aligned} \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} &= \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}. & \text{take it as granted} \\ &= \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix} \end{aligned} \quad (4.10)$$

Matrix Multiplication

Basic divide-and-conquer

assume n is a power of 2

SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

1 $n = A.\text{rows}$
2 let C be a new $n \times n$ matrix
3 if $n == 1$
4 $c_{11} = a_{11} \cdot b_{11}$

$\Rightarrow O(n^2)$

$T(n)$: RT of this algo.
for multiplying two
 $n \times n$ matrices.

$2T(\frac{n}{2}) + O(n^2)$

5 else partition A , B , and C as in equations (4.9)

6 $C_{11} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11})$ $\Rightarrow T(\frac{n}{2})$
7 + $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21})$ //

8 $C_{12} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12})$ //

9 + $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22})$ //

10 $C_{21} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11})$ //

11 + $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21})$ //

12 $C_{22} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12})$ //

13 + $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22})$ //

14 return C

Matrix Multiplication

Basic divide-and-conquer

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

$$\begin{aligned} g &= n^{\log_2 8} & f &= n^2 \\ &= n^3 & &> p \end{aligned}$$

$$\Rightarrow T(n) = \Theta(n^3)$$

Matrix Multiplication

Basic divide-and-conquer

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1 . \end{cases}$$

$$T(n) =$$

Matrix Multiplication

Basic divide-and-conquer

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1 . \end{cases}$$

$T(n) = \Theta(n^3)$. (we will see this later).

Matrix Multiplication

Strassen's algorithm

The key idea is to reduce the number of multiplication of two $n/2 \times n/2$ matrices from 8 to 7. But the details are very non-trivial.

Matrix Multiplication

Strassen's algorithm

we will see Strassen's algorithm will have this recurrence

The key idea is to reduce the number of multiplication of two $n/2 \times n/2$ matrices from 8 to 7. But the details are very non-trivial.
So we will have

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases}$$

$$\approx 2.7 \cdot n^{\log_2 7} > n^2$$

$$n^{\log_2 7} < n^3$$

$$\Rightarrow T(n) = \Theta(n^{\log_2 7})$$

Matrix Multiplication

Strassen's algorithm

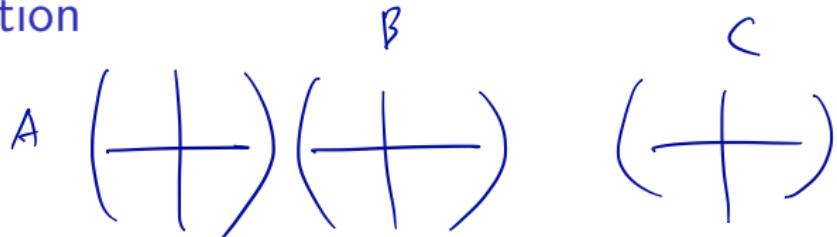
The key idea is to reduce the number of multiplication of two $n/2 \times n/2$ matrices from 8 to 7. But the details are very non-trivial.
So we will have

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1 . \end{cases}$$

Running time: $\Theta(n^{\log_2 7})$. ($2.8 < \log_2 7 < 2.81$).

Matrix Multiplication

Strassen's algorithm



$O(1) \times$ Divide the input matrices A and B and output matrix C into $n/2 \times n/2$ submatrices, as in equation (4.9). This step takes $\Theta(1)$ time by index calculation, just as in SQUARE-MATRIX-MULTIPLY-RECURSIVE.

$O(n^2)$ 2. Create 10 matrices S_1, S_2, \dots, S_{10} , each of which is $n/2 \times n/2$ and is the sum or difference of two matrices created in step 1. We can create all 10 matrices in $\Theta(n^2)$ time. *only addition or differences.*

$7T(\frac{n}{2})$ 3. Using the submatrices created in step 1 and the 10 matrices created in step 2, recursively compute seven matrix products P_1, P_2, \dots, P_7 . Each matrix P_i is $n/2 \times n/2$. *multiplications.*

$O(n^2)$ 4. Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of the P_i matrices. We can compute all four submatrices in $\Theta(n^2)$ time.

Step 3: 7 multiplications of $n/2$ by $n/2$ matrices.

Matrix Multiplication

Strassen's algorithm: Step 2

$$\left. \begin{array}{l} S_1 = B_{12} - B_{22}, \\ S_2 = A_{11} + A_{12}, \\ S_3 = A_{21} + A_{22}, \\ S_4 = B_{21} - B_{11}, \\ S_5 = A_{11} + A_{22}, \\ S_6 = B_{11} + B_{22}, \\ S_7 = A_{12} - A_{22}, \\ S_8 = B_{21} + B_{22}, \\ S_9 = A_{11} - A_{21}, \\ S_{10} = B_{11} + B_{12}. \end{array} \right\}$$

$\Theta(n^2)$

Matrix Multiplication

Strassen's algorithm: Step 3

7 recursive calls.

7 $T(\frac{n}{2})$.

n_2 by n_2 matrices.

$$P_1 = \underbrace{A_{11} \cdot S_1}_{\text{ }} = A_{11} \cdot B_{12} - A_{11} \cdot B_{22},$$

$$P_2 = \underbrace{S_2 \cdot B_{22}}_{\text{ }} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22},$$

$$P_3 = \underbrace{S_3 \cdot B_{11}}_{\text{ }} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11},$$

$$P_4 = \underbrace{A_{22} \cdot S_4}_{\text{ }} = A_{22} \cdot B_{21} - A_{22} \cdot B_{11},$$

$$P_5 = \underbrace{S_5 \cdot S_6}_{\text{ }} = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22},$$

$$P_6 = \underbrace{S_7 \cdot S_8}_{\text{ }} = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22},$$

$$P_7 = \underbrace{S_9 \cdot S_{10}}_{\text{ }} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}.$$

Matrix Multiplication

Strassen's algorithm: Step 4

$$\textcircled{C_{11}} = P_5 + P_4 - P_2 + P_6 . \quad O(n^2)$$

$$\begin{array}{r} A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ \quad - A_{22} \cdot B_{11} \qquad \qquad + A_{22} \cdot B_{21} \\ - A_{11} \cdot B_{22} \qquad \qquad \qquad - A_{12} \cdot B_{22} \\ \quad - A_{22} \cdot B_{22} - A_{22} \cdot B_{21} + A_{12} \cdot B_{22} + A_{12} \cdot B_{21} \end{array}$$

$$A_{11} \cdot B_{11} \qquad \qquad \qquad + A_{12} \cdot B_{21} ,$$

Matrix Multiplication

Strassen's algorithm: Step 4

$$C_{12} = P_1 + P_2 ,$$

and so C_{12} equals

$O(n^{\gamma})$

$$\begin{array}{r} A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\ + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\ \hline A_{11} \cdot B_{12} & + A_{12} \cdot B_{22} , \end{array}$$

Matrix Multiplication

Strassen's algorithm: Step 4

$$C_{21} = P_3 + P_4$$

makes C_{21} equal

$O(n^2)$

$$\begin{array}{r} A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\ - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\ \hline A_{21} \cdot B_{11} & + A_{22} \cdot B_{21}, \end{array}$$

Matrix Multiplication

Strassen's algorithm: Step 4

$$C_{22} = P_5 + P_1 - P_3 - P_7 ,$$

so that C_{22} equals

$O(n^2)$

$$\begin{array}{r} A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ - A_{11} \cdot B_{22} \qquad \qquad \qquad + A_{11} \cdot B_{12} \\ - A_{22} \cdot B_{11} \qquad \qquad \qquad - A_{21} \cdot B_{11} \\ - A_{11} \cdot B_{11} \qquad \qquad \qquad - A_{11} \cdot B_{12} + A_{21} \cdot B_{11} + A_{21} \cdot B_{12} \\ \hline A_{22} \cdot B_{22} \qquad \qquad \qquad + A_{21} \cdot B_{12} , \end{array}$$

Matrix Multiplication

Strassen's algorithm: Step 4

$$C_{22} = P_5 + P_1 - P_3 - P_7,$$

so that C_{22} equals

$$\begin{array}{c} A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ - A_{11} \cdot B_{22} - A_{22} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} \\ - A_{11} \cdot B_{11} - A_{11} \cdot B_{12} + A_{21} \cdot B_{11} + A_{21} \cdot B_{12} \\ \hline A_{22} \cdot B_{22} + A_{21} \cdot B_{12}, \end{array}$$

7 multiplications of $n/2$ by $n/2$ matrices and $O(1)$ additions of such matrices. $\Rightarrow T(n) = 7T(n/2) + \Theta(n^2)$

$$n^{log_2 7} > n^2$$

$$\begin{aligned} T(n) &= \\ \Theta(n^{log_2 7}) \end{aligned}$$

Part 2: Solving recurrences

Three methods:

- ✓ substitution method: guesses a bound and prove it using induction. *most powerful. / least handy.*
- ✓ ▶ recursion-tree method: derive a tree that represents workload at different levels. *intuitive. + & some work.*
- ✓ ▶ master theorem: a theorem. powerful and handy but not always applicable.

We often simplify recursions...

1

We often omit floors, ceilings, and boundary conditions.

ex) $T(n) = \begin{cases} \underline{\Theta(1)} & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n \geq 2. \end{cases}$

We just say $\underline{T(n) = 2T(n/2) + \Theta(n)}$.

Solving recurrences

Substitution method

1. Guess the form of the solution.
2. Use mathematical induction to find the constants and show that the solution works.

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

$\overset{f_2}{\nearrow} \quad \overset{f_1}{\searrow}$

$\Rightarrow T(n) \sim \Theta(n^{\log_2 2})$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess.

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess. $T(n) = O(n \log n)$.

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess. $T(n) = O(n \log n)$.

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $\underline{T(n) \leq cn \log n}$; we will have to choose c later.

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $T(n) \leq cn \log n$; we will have to choose c later.

Induction step: Assume the bound holds for $T(1) \dots T(n-1)$.

$$T(n)$$

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $T(n) \leq cn \log n$; we will have to choose c later.

Induction step: Assume the bound holds for $T(1) \dots T(n-1)$.

$$T(n) \leq 2T(n/2) + bn \tag{4}$$

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $T(n) \leq cn \log n$; we will have to choose c later.

Induction step: Assume the bound holds for $T(1) \dots T(n-1)$.

$$T(n) \leq 2\underline{T(n/2)} + bn \tag{4}$$

$$\leq 2(\underline{c(n/2) \log(n/2)}) + bn \quad \text{I.H.} \tag{5}$$

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $T(n) \leq cn \log n$; we will have to choose c later.

Induction step: Assume the bound holds for $T(1) \dots T(n-1)$.

$$T(n) \leq 2T(n/2) + bn \tag{4}$$

$$\leq 2(c(n/2) \log(n/2)) + bn \tag{5}$$

$$= cn \log n - cn + bn \tag{6}$$

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + bn$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $T(n) \leq cn \log n$; we will have to choose c later.

Induction step: Assume the bound holds for $T(1) \dots T(n-1)$.

$$T(n) \leq 2T(n/2) + bn \tag{4}$$

$$\leq 2(c(n/2) \log(n/2)) + bn \tag{5}$$

$$= cn \log n - cn + bn \tag{6}$$

$$\leq cn \log n \tag{7}$$

It holds as long as $c \geq b$.

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + n \log n$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $T(n) \leq cn \log n$; we will have to choose c later.

Boundary (Base): $\underline{T(1)} \leq \underline{c \log_2 1}$

Solving recurrences

Substitution method: Warm-up example

$$T(n) = 2T(n/2) + \Theta(n)$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + n \log n$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $T(n) \leq cn \log n$; we will have to choose c later.

Boundary (Base): $T(1) \leq c \log_2 1 ???$

Solving recurrences

Substitution method: Warm-up example

$$\underline{T(n) = 2T(n/2) + \Theta(n)}$$

Say we want to show an upper bound. First choose a constant b such that $T(n) \leq 2T(n/2) + n \log n$.

Guess. $T(n) = O(n \log n)$.

So, let's try to prove $T(n) \leq cn \log n$; we will have to choose c later.

Boundary (Base): $T(1) \leq c \log_2 1$? But we only need to show $T(n) \leq cn \log n$ for all $n \geq n_0$ for an appropriate constant n_0 ! Say set $n_0 = 2$. Then,

$$T(2) \leq c \log_2 2$$

as long as $c \geq T(2)$. So unlike usual mathematical inductions, the base case is not important.

Solving recurrences

Substitution method: useful tricks

Solving recurrences

Substitution method: useful tricks

When you guess, usually additive constants are not important.

Solving recurrences

Substitution method: useful tricks

When you guess, usually additive constants are not important. Ex.

$$T(n) = 2T(\lceil n/2 \rceil + 17) + n$$

Solving recurrences

Substitution method: useful tricks

$$\text{Ex. } T(n) = 2T(n/2) + 1$$

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $\underline{T(n) \leq cn}$.

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

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

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

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

$$T(n) = 2T(\underline{n/2}) + 1 \leq 2(c(n/2)) + 1 = \underline{cn} + 1 \leq ?cn$$

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

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

Try $T(n) \leq cn + d$.

$$T(n) = 2T(n/2) + 1 \leq 2(c(n/2) + d) + 1 =$$

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

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

Try $T(n) \leq cn + d$.

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

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

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

Try $T(n) \leq cn + d$.

$$T(n) = 2T(n/2) + 1 \leq 2(c(n/2) + d) + 1 = cn + 2d + 1 \leq? cn + d$$

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

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

Try $T(n) \leq cn + d$.

$$T(n) = 2T(n/2) + 1 \leq 2(c(n/2) + d) + 1 = cn + 2d + 1 \leq? cn + d$$

Use **negative** terms: $T(n) \leq cn - 1$.

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

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

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

Try $T(n) \leq cn + d$.

$$T(n) = 2T(n/2) + 1 \leq 2(c(n/2) + d) + 1 = cn + 2d + 1 \leq? cn + d$$

Use **negative** terms: $T(n) \leq cn - 1$.

$$T(n) = 2T(\underline{n/2}) + 1 \leq \underline{2(cn/2 - 1)} + 1$$

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

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

Try $T(n) \leq cn + d$.

$$T(n) = 2T(n/2) + 1 \leq 2(c(n/2) + d) + 1 = cn + 2d + 1 \leq? cn + d$$

Use **negative** terms: $T(n) \leq cn - 1$.

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

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(n/2) + 1$

Try $T(n) \leq cn$.

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

Try $T(n) \leq cn + d$.

$$T(n) = 2T(n/2) + 1 \leq 2(c(n/2) + d) + 1 = cn + 2d + 1 \leq ?cn + d$$

Use **negative** terms: $T(n) \leq cn - 1$.

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

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(\sqrt{n}) + \log n$

Say $m = \log n$. Then, we have

~~$T(2^m) = 2T(2^{m/2}) + m$~~

Rename ~~$S(m) = T(2^m)$~~ .

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(\sqrt{n}) + \log n$

Say $m = \log n$. Then, we have

$$T(2^m) = 2T(2^{m/2}) + m.$$

Rename $S(m) = T(2^m)$.

$$\rightarrow S(m) = 2S(m/2) + m$$

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(\sqrt{n}) + \log n$

Say $m = \log n$. Then, we have

$$T(2^m) = 2T(2^{m/2}) + m.$$

Rename $S(m) = T(2^m)$.

$$\rightarrow S(m) = 2S(m/2) + m$$

$$\rightarrow S(m) = \Theta(m \log m)$$

Solving recurrences

Substitution method: useful tricks

Ex. $T(n) = 2T(\sqrt{n}) + \log n$

Say $m = \log n$. Then, we have

$$T(2^m) = 2T(2^{m/2}) + m.$$

Rename $S(m) = T(2^m)$.

$$\rightarrow S(m) = 2S(m/2) + m$$

$$\rightarrow S(m) = \Theta(m \log m)$$

$$\rightarrow T(n) = \Theta((\log n) \cdot \log \log n)$$

Changing variables can help!

Solving recurrences

Recursion Tree

$$T(n) = 3 T(\underline{n/3})$$

$$\Rightarrow \underline{T(n)} = \underline{\underline{n}}.$$

divide
+ solve subproblems

Examples:

1. $T(n) = 3 T(n/3) + \Theta(n)$
2. $T(n) = 4 T(n/3) + \Theta(n)$
3. $T(n) = 2 T(n/3) + \Theta(n)$

conquer

$$T(n) = 2 T\left(\frac{n}{4}\right)$$

$$\Rightarrow T(n) = \sqrt{n}$$

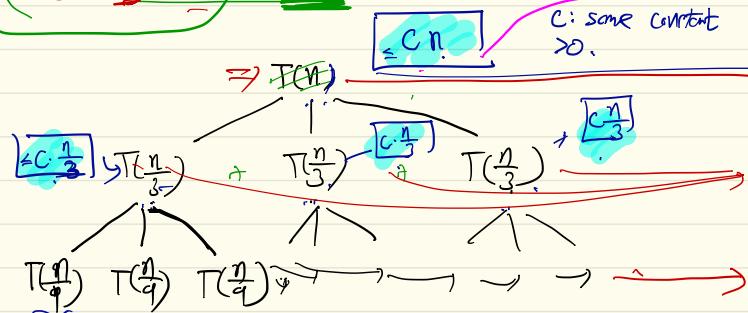
This method can be sloppy if you're not careful. To get full points, you must specify the following key quantities:

1. Tree depth
2. Size of each subproblem at depth d .
3. Number of subproblems/nodes at depth d .
4. Workload per each node at depth d .
5. Total workload at depth d .

$$T(n) = 3T\left(\frac{n}{3}\right) + O(n)$$

$$T\left(\frac{n}{3}\right) = 3T\left(\frac{n}{9}\right) + O\left(\frac{n}{3}\right)$$

$$T(x) \leq 3T(\frac{x}{3}) + cx$$



depth	sub problem size	# of sub problems	workload subproblem	Total workload
0	n	1	$* Cn$	Cn
1	$\frac{n}{3}$	3	$* C \frac{n}{3}$	Cn
2	$\frac{n}{3^2}$	3^2	$* C \frac{n}{3^2}$	Cn
:	:	:		
d	$\frac{n}{3^d}$	3^d	$* C \frac{n}{3^d}$	Cn
				$+$
$D = \log_3 n$	$= \frac{n}{3^D}$	3^D	$* C \frac{n}{3^D}$	Cn
				$+$

base
case

D
= 8.0

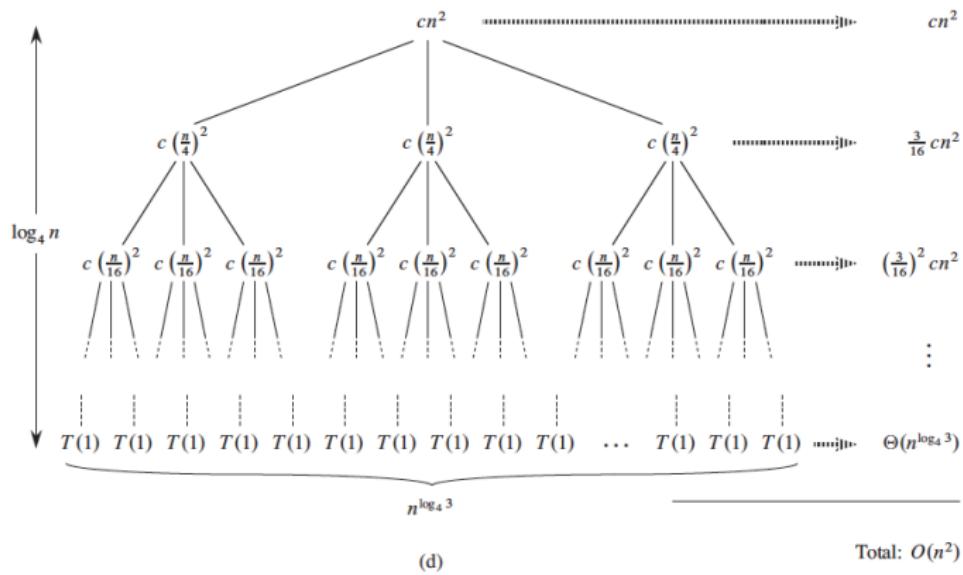
$$D = \frac{r}{3}$$

$$= O(n \log n)$$

Solving recurrences

Recursion Tree in Textbook

$$T(n) = 3T(n/4) + cn^2$$



K terms.

$$a + ar + ar^2 + ar^3 + \dots + ar^{K-1}$$

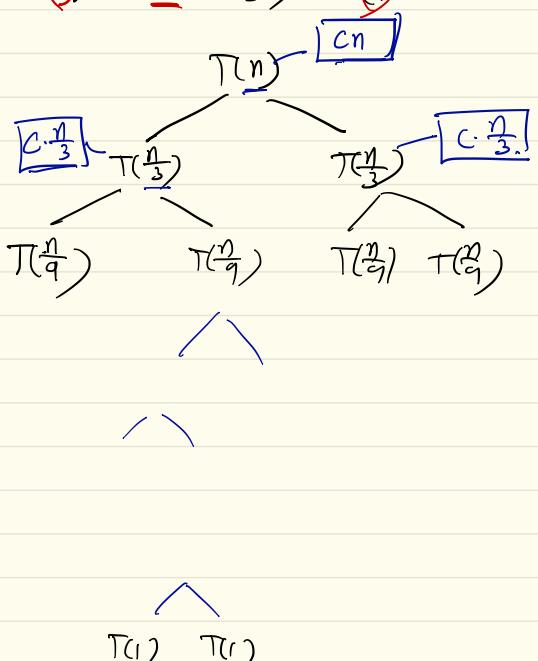
$a > 0$, : a is a function of n .
 $r > 0$, : r is a constant.

$$r=1, a + a + a + \dots + a = ak.$$

$$\begin{aligned} r &< 1, \\ a + ar + ar^2 + \dots + ar^{K-1} &\approx a + ar + ar^2 + \dots + ar^{K-1} \\ &= a \frac{1 - r^K}{1 - r} \\ &\stackrel{K \rightarrow \infty}{\rightarrow} a \frac{1 - r^K}{1 - r} = \frac{a}{1 - r} \end{aligned}$$

$$r \geq 1, a \cdot \frac{r^K}{r^0} = \Theta(a \cdot r^K)$$

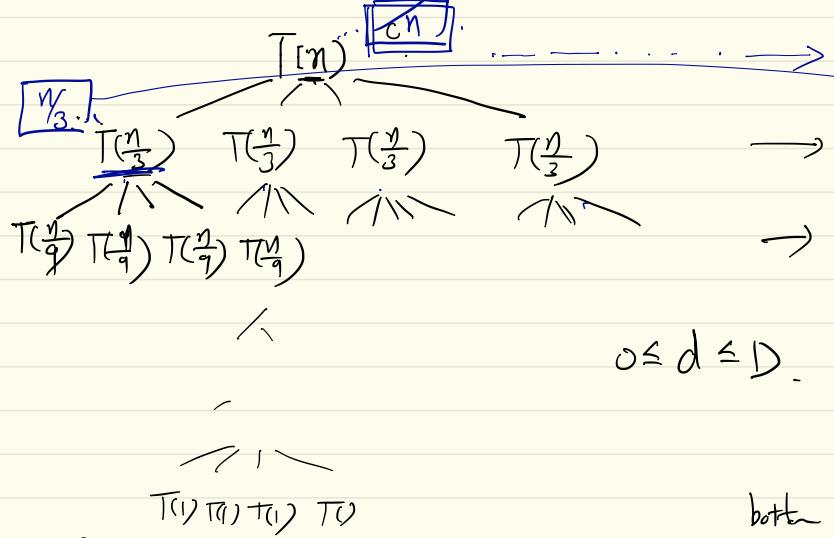
$$T(n) = 2 T\left(\frac{n}{3}\right) + \Theta(n)$$



$$R.T. = cn + cn \cdot \frac{1}{3} + cn \cdot \left(\frac{2}{3}\right)^2 + \dots + cn \left(\frac{2}{3}\right)^D = \Theta(cn) = \Theta(n)$$

depth	sub problem size	#.f sub problems	Workload /sub prob	$T(n)$ workload
0	n	1	$* cn$	cn
1	$\frac{n}{3}$	2	$* c \frac{n}{3}$	$c \frac{2}{3} n$
2	$\frac{n}{3^2}$	2^2	$* c \frac{n}{3^2}$	$c \left(\frac{2}{3}\right)^2 n$
:	:	:	:	+
d	$\frac{n}{3^d}$	2^d	$* c \frac{n}{3^d}$	$c \left(\frac{2}{3}\right)^d n$
;	;	;	;	;
$D = \log_3 n$	$1 = \frac{n}{3^D}$	2^D	$* c \cdot \frac{n}{3^D}$	$c \left(\frac{2}{3}\right)^D n$

$$T(n) = 4 T\left(\frac{n}{3}\right) + \Theta(n)$$



$$\alpha^{\log_3 c} = c^{\log_3 \alpha}$$

depth	subproblem size	# of subproblems	Workload Subproblem	Total Workload
0	n	1	$\cancel{*} n$	$= n$
1	$\frac{n}{3}$	4	$\cancel{*} \frac{n}{3}$	$= \frac{4}{3}n$
2	$\frac{n}{3^2}$	4^2	$\cancel{*} \frac{n}{3^2}$	$= (\frac{4}{3})^2 n$
:	:	:	:	+
d	$\frac{n}{3^d}$	4^d	$\cancel{*} \frac{n}{3^d}$	$= (\frac{4}{3})^d n$
:	:	:	:	+
D	$\frac{n}{3^D}$	4^D	$\frac{n}{3^D}$	$(\frac{4}{3})^D n$

$$R.T = n + (\frac{4}{3})^1 n + (\frac{4}{3})^2 n + \dots + (\frac{4}{3})^D n$$

$$\begin{aligned}
 &= D \left(\left(\frac{4}{3} \right)^D n \right) = \Theta \left(\left(\frac{4}{3} \right)^D n \right) \\
 &= \Theta \left(n^{\frac{4}{3} \cdot D} \right) = \Theta \left(n^{\frac{4}{3} \cdot 3} \right) \\
 &= \Theta \left(n^{4 \cdot \frac{4}{3}} \right)
 \end{aligned}$$

$$\alpha = n, \quad r \geq \frac{4}{3} > 1$$

Solving recurrences

Recursion Tree

Examples:

1. $T(n) = 8T(n/2) + \Theta(n^2)$
2. $T(n) = 7T(n/2) + \Theta(n^2)$
3. $T(n) = 4T(n/2) + \Theta(n^2)$
4. $T(n) = 3T(n/2) + \Theta(n^2)$

\Rightarrow when the subproblem size is n ,
 $W_{\text{subprob}} = n^2$

Solving recurrences

In

Recursion Tree

Examples:

1. $T(n) = 8T(n/2) + \Theta(n^2)$
2. $T(n) = 7T(n/2) + \Theta(n^2)$
3. $T(n) = 4T(n/2) + \Theta(n^2)$
4. $T(n) = 3T(n/2) + \Theta(n^2)$
5. $T(n) = T(n/3) + T(2n/3) + \Theta(n)$

$$T(n) = \delta T\left(\frac{n}{2}\right) + \Theta(n^2)$$

intuition.

① $T(n) = \delta T\left(\frac{n}{2}\right)$

② $T(n) = \Theta(n^2)$

$T(n) = n^3$

$T\left(\frac{n}{2}\right) = \left(\frac{n}{2}\right)^3$

$T(n) = [n^3]$

$T\left(\frac{n}{2}\right) = \left[\left(\frac{n}{2}\right)^3\right]$

$\xrightarrow{k=2}$

$$T(n) = \Theta(n^3)$$

$$T(n) = 9T\left(\frac{n}{3}\right) + \underline{n^2}$$

Tree depth
 $\log_3 n$

Subproblem
size at depth d

$$\frac{n}{3^d}$$

of subproblems
at depth d.

$$q^d$$

w/
subproblem

WL

$$X \left[\begin{array}{c} \frac{n}{3^d} \\ \left(\frac{n}{3^d} \right)^2 \\ \left(\frac{n}{3^d} \right)^3 \end{array} \right] = n^2$$

$$\underline{T(n) = 9T\left(\frac{n}{3}\right) + n^3} : //$$

$$" \quad \left(3^d \right)^2 \quad X \left[\begin{array}{c} \left(3^d \right)^2 \\ \left(3^d \right)^3 \end{array} \right] = \frac{n^3}{3^d}$$

$$\overbrace{T(n) = 9T\left(\frac{n}{3}\right) + n}^{\text{---}} : //$$

$$" \quad \left(3^d \right)^2 \quad X \left[\begin{array}{c} \left(3^d \right)^2 \\ \left(3^d \right)^3 \end{array} \right] = \frac{n^3}{3^d} \cdot n$$

$$T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$T(n) = 4T\left(\frac{n}{2}\right)$$

$$\Rightarrow T(n) = \Theta(n^2)$$

η^2

$$T(n) = \Theta(n^2 \cdot \lg n)$$



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

$$T(n) = 2T\left(\frac{n}{2}\right)$$

$$\Rightarrow T(n) = n$$

$\downarrow \eta^2$

vs. <

$$T(n) = \Theta(n^2)$$

Solving recurrences

Master Theorem

If $f = \Theta(g)$ \Rightarrow case 2.

If $f >_a g$ then

check $f >_p g$ ($\frac{f}{g} \geq n^k$ for some $k > 0$)
case 3.

Theorem 4.1 (Master theorem) If $f <_a g$ and $f <_p g$ case 1

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$T(n) = aT(n/b) + f(n)$, Q. is the given recurrence of this form?
 $(n \rightarrow NA)$

where we interpret n/b to mean either $[n/b]$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$. ■

Solving recurrences

Master Theorem (in other words)

Suppose we have the following recurrence:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n),$$

where $a \geq 1$, $b > 1$ are constants, $f(n)$ is a function. Let $g(n) = n^{\log_b a}$. Then, we have the following:

1. If $f(n) <_{poly} g(n)$, i.e., $f(n) = O\left(\frac{g(n)}{n^\epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) = \Theta(g(n))$.
2. If $f(n) = \Theta(g(n))$, then $T(n) = \Theta(f(n) \log n) = \Theta(g(n) \log n)$.
3. If $f(n) >_{poly} g(n)$, i.e., $f(n) = \Omega(g(n) \cdot n^\epsilon)$ for some constant $\epsilon > 0$ and if $f\left(\frac{n}{b}\right) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$.

Solving recurrences

Master Theorem: examples

$$T(n) = \frac{4T(\frac{n}{2}) + \Theta(n^2)}{f(n)}$$

$$g(n) = n^{\frac{f+1}{2}}$$

$$= n^2$$

$$\frac{f}{2} = \Theta(f)$$

case 2.

$$T(n) = \Theta(f \cdot g(n))$$

$$= \Theta(n^2 \cdot n)$$

~~1. $T(n) = 8T(n/2) + \Theta(n^2)$~~ $\Rightarrow T(n) = \Theta(n^3)$

~~2. $T(n) = 7T(n/2) + \Theta(n^2)$~~ $\Rightarrow T(n) = \Theta(n^{2.5})$

~~3. $T(n) = 4T(n/2) + \Theta(n^2)$~~ $\Rightarrow T(n) = \Theta(n^2 \cdot \ln n)$

~~4. $T(n) = 3T(n/2) + \Theta(n^2)$~~ $\Rightarrow T(n) = \Theta(n^2)$

~~5. $T(n) = T(n/3) + T(2n/3) + \Theta(n)$~~ NA

~~6. $T(n) = 27T(n/3) + n^3$~~ $\Rightarrow T(n) = \Theta(n^3 \ln n)$

~~7. $T(n) = 5T(n/2) + n^3$~~ $\Rightarrow T(n) = \Theta(n^3)$

~~8. $T(n) = 5T(n/2) + n^2$~~ $\Rightarrow T(n) = \Theta(n^{2.5})$

~~9. $T(n) = 4T(n/2) + n^2 / \log n$~~ \Rightarrow NA

$g(n) = n^2$

$f = \frac{n^3}{\ln n}$

$T(n) = 4T(\frac{n}{2}) + n^2 / \ln n$

$g = n^2$ vs $f = \frac{n^3}{\ln n}$

$$T(n) = 4T\left(\frac{n}{2}\right) + \underbrace{\frac{n^3}{f(n)}}_{\text{f(n)}}$$

$$g = n^2 \leq_a f = \frac{n^3}{f(n)}$$

$$g <_p f.$$

Method i). $\frac{f}{g} = \frac{n}{f(n)} \geq_a \sqrt{n}$. a polynomial base factor gap.

method ii)

$$g \leq_a \frac{n^{3-\varepsilon}}{f(n)} \text{ for } \varepsilon=0.01.$$

Answer. $T(n) = \Theta\left(\frac{n^3}{f(n)}\right)$.

Solving recurrences

Master Theorem: examples

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

Solving recurrences

Master Theorem: examples

$$f(n) = n^{\log_2 8 - 3}$$
$$T(n) = 8T(n/2) + \Theta(n^2)$$

Case 1. since $\Theta(n^2) = O(n^{\log_2 8 - \epsilon})$ where $\epsilon = 1$. In other words, $\Theta(n^2)$ is 'polynomially' smaller than n^3 .

$$\frac{f(n)}{f(n)} = \frac{n^3}{n^2} = n > 1$$
$$\Rightarrow T(n) = \Theta(f(n)) = \Theta(n^3)$$

$$\frac{f}{f} = \frac{n^3}{n^2} = n \Rightarrow f \text{ is greater than } f \text{ by a polynomial factor.}$$
$$\Leftrightarrow n^2 = O(n^{3-\epsilon})^{\frac{1}{2}} \text{ for some } \epsilon > 0.$$

Solving recurrences

Master Theorem: examples

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

Case 1. since $\Theta(n^2) = O(n^{\log_2 8 - \epsilon})$ where $\epsilon = 1$. In other words, $\Theta(n^2)$ is ‘polynomially’ smaller than n^3 .

Therefore, $T(n) = \Theta(n^3)$.

This is the running time of the naive divide-and-conquer algorithm for MM.

Solving recurrences

Master Theorem: examples

$$T(n) = 7T(n/2) + \Theta(n^2)$$

Solving recurrences

Master Theorem: examples

$$T(n) = 7T(n/2) + \Theta(n^2)$$

Case 1. since $\Theta(n^2) = O(n^{\log_2 7 - \epsilon})$ for $\epsilon = \log_2 7/6$. In other words, $\Theta(n^2)$ is ‘polynomially’ smaller than $n^{\lg 7}$.

Solving recurrences

Master Theorem: examples

$$T(n) = 7T(n/2) + \Theta(n^2)$$

Case 1. since $\Theta(n^2) = O(n^{\log_2 7 - \epsilon})$ for $\epsilon = \log_2 7/6$. In other words, $\Theta(n^2)$ is ‘polynomially’ smaller than $n^{\lg 7}$. Therefore, $T(n) = \Theta(n^{\lg 7})$.

This is the running time of the Strassen’s algorithm for MM.

Solving recurrences

Master Theorem: examples

$$T(n) = 4T(n/2) + \Theta(n^2)$$

Solving recurrences

Master Theorem: examples

$$T(n) = 4T(n/2) + \Theta(n^2)$$

Case 2. since $\Theta(n^2) = \Theta(n^{\log_2 4})$.

Solving recurrences

Master Theorem: examples

f.

$$T(n) = 4T(n/2) + \Theta(n^2)$$

Case 2. since $\Theta(n^2) = \Theta(n^{\log_2 4})$ Therefore, $T(n) = \Theta(n^2 \log n)$.

g.

Solving recurrences

Master Theorem: examples

$$T(n) = 3T(n/2) + \Theta(n^2)$$

Solving recurrences

Master Theorem: examples

$$f(n) = n^{\frac{\log_2 3}{2}} < n^{\frac{\log_2 4}{2}} = n^2$$

$$T(n) = 3T(n/2) + \Theta(n^2)$$

Case 3. since $\Theta(n^2) = \Omega(n^{\log_2 3 + \epsilon})$ for $\epsilon = \log_2 4/3$. In other words $\Theta(n^2)$ is 'polynomially' greater than $n^{\lg 3}$.

$$\frac{f}{g} = n^{\frac{2 - \log_2 3}{2}} > 0 \Rightarrow f \text{ is polynomially larger than } g.$$

Or

$$n^{\log_2 3 + 0.001} < n^2$$

Solving recurrences

Master Theorem: examples

$$T(n) = 3T(n/2) + \Theta(n^2)$$

Case 3. since $\Theta(n^2) = \Omega(n^{\log_2 3 + \epsilon})$ for $\epsilon = \log_2 4/3$. In other words $\Theta(n^2)$ is ‘polynomially’ greater than $n^{\lg 3}$.

$$3(n/2)^2 \leq (3/4)n^2.$$

Solving recurrences

Master Theorem: examples

$$T(n) = 3T(n/2) + \Theta(n^2)$$

Case 3. since $\Theta(n^2) = \Omega(n^{\log_2 3 + \epsilon})$ for $\epsilon = \log_2 4/3$. In other words $\Theta(n^2)$ is ‘polynomially’ greater than $n^{\lg 3}$.

$$\cancel{3(n/2)^2 \leq (3/4)n^2}.$$

Therefore, $T(n) = \Theta(n^2)$.

Solving recurrences

Master Theorem: examples

$$T(n) = \boxed{a} T\left(\frac{n}{b}\right) + f(n).$$

$$T(n) = T(\underline{n/3}) + T(\underline{2n/3}) + \Theta(n)$$

N/A.

Solving recurrences

Master Theorem: examples

$$T(n) = T(n/3) + T(2n/3) + \Theta(n)$$

The Master Theorem is not applicable.

Solving recurrences

Master Theorem: examples

1. $T(n) = 27T(n/3) + n^3.$
2. $T(n) = 5T(n/2) + n^3.$
3. $T(n) = 5T(n/2) + n^2.$

Solving recurrences

Master Theorem: examples

$$T(n) = 4T(n/2) + n^2/\log n$$

Solving recurrences

Master Theorem: examples

$$g = n^{f_2} = n^2$$
$$T(n) = 4T(n/2) + n^2/\log n$$

The Master Theorem is not applicable. Two functions are not polynomially equal, and one is not polynomially greater than the other.

$g > f$ but g is not polynomially greater than f .

$$\frac{g}{f} = \frac{n^2}{n^2/\log n} = \log n. \Rightarrow N/A$$