CSE 100: Algorithm Design and Analysis
Chapter 04: Divide-and-Conquer

Sungjin Im

University of California, Merced

Last Update: 01-27-2023

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.

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

- 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 $A[1 \cdots n]$ of numbers.

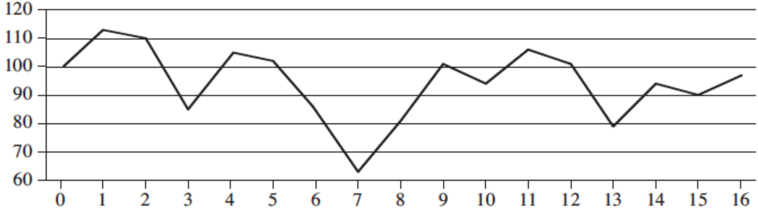
Output: Indices i and j $(1 \leq i \leq j \leq n)$ s.t.
$A[i] + A[i+1] + \cdots + A[j]$ is maximized.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	–3	–25	20	–3	–16	–23	18	20	–7	12	–5	–22	15	–4	7

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. Price[0] = 100
Price[n] = Price[n-1] + A[n]



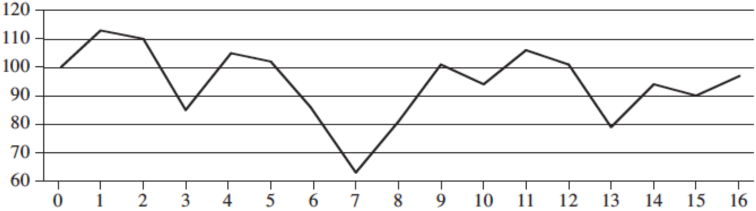
Day	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Price	100	113	110	85	105	102	86	63	81	101	94	106	101	79	94	90	97
Change		13	−3	−25	20	−3	−16	−23	18	20	−7	12	−5	−22	15	−4	7

The maximum subarray problem

Stock market view

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

Price[0] $=$ 100 and Price[n] $=$ 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

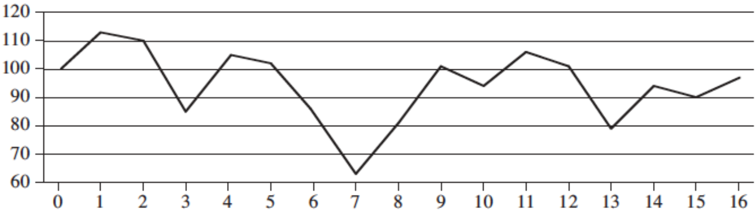
How about buying a stock at the lowest price and selling it later, or buying at some point and selling at the highest price?

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.

Price[0] = 100 and Price[n] = 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?

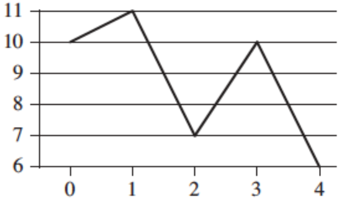
Buying at the lowest price and selling at the highest price?

The maximum subarray problem

Max - Min?

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

Price[0] = 100 and Price[n] = 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?

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

The maximum subarray problem

Naive algorithm?

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

The maximum subarray problem

Naive algorithm?

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

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

The maximum subarray problem

Naive algorithm?

Can try every possible pair (i, j) $(i \leq j)$ and compute $A[i] + A[i+1] + \cdots + A[j]$.
Running time? $\Theta(n^3)$. Can be improved to $\Theta(n^2)$. How?

The maximum subarray problem

A simpler version

Simpler problem:

Definition:

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

Goal:

Compute

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

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

The maximum subarray problem

A simpler version

Simpler problem:

Definition:

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

Goal:

Compute

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

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

Original problem:

Find i and j s.t. $1 \le i \le j \le n$ 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

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

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

Original problem:

Find i and j s.t. $1 \le i \le j \le n$ 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

Divide-and-Conquer

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

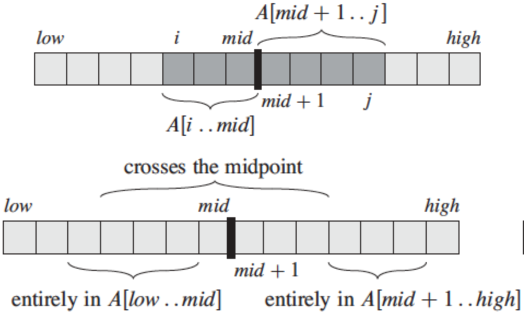
1. entirely in $A[low \cdots mid]$, so that $low \leq i \leq j \leq mid$.
2. entirely in $A[mid + 1 \cdots high]$, so that $mid < i \leq j \leq high$.
3. crossing the midpoint, so that $low \leq i \leq mid < j \leq high$.

The maximum subarray problem

Divide-and-Conquer

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

- entirely lies in $A[low \cdots mid]$, so that $low \leq i \leq j \leq mid$.
- entirely lies in $A[mid + 1 \cdots high]$, 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 $OPT(low, high)$?

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

$$\max_{low \le i \le j \le mid} S[i,j], \tag{1}$$

$$\max_{mid < i \le j \le high} S[i,j], \tag{2}$$

$$\max_{low \le i \le mid < j \le high} S[i,j] \tag{3}$$

$$\}$$

The maximum subarray problem

Recursion

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

$$
\begin{aligned}
OPT(low, high) = \max\{ & \\
& OPT(low, mid), \\
& OPT(mid+1, high) \\
& \max_{low \le i \le mid < j \le high} S[i, j] \\
\} &
\end{aligned}
$$

The maximum subarray problem

Recursion

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

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

$$OPT(low, mid),$$

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

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

$$\}$$

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

$$
\begin{aligned}
OPT(low, high) = \max\{ & \\
& OPT(low, mid), \\
& OPT(mid+1, high) \\
& \max_{low \le i \le mid < j \le high} S[i, j] \\
\} &
\end{aligned}
$$

Key Observation: 'decoupling'

$$
\begin{aligned}
& \max_{low \le i \le mid < j \le high} S[i, j] \\
= & \max_{low \le i \le mid} S[i, mid] + \max_{mid+1 \le j \le high} S[mid+1, j]
\end{aligned}
$$

The maximum subarray problem

Recursion

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

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

Key Observation: 'decoupling'

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

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

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

The maximum subarray problem

Recursion

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

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

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

The maximum subarray problem

Recursion

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

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

$$OPT(low, mid),$$

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

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

$$\}$$

(*) 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 $OPT(low, high)$?

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

(*) 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 $low < high$:

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

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 $A[low \cdots high]$ lies

- entirely lies in $A[low \cdots mid]$, so that $low \leq i \leq j \leq mid$.
- entirely lies in $A[mid + 1 \cdots high]$, 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 $A[low \cdots high]$ lies

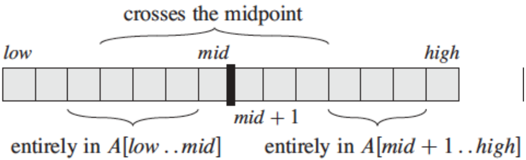
- entirely lies in $A[low \cdots mid]$, so that $low \leq i \leq j \leq mid$.
- entirely lies in $A[mid + 1 \cdots high]$, so that $mid < i \leq j \leq high$.
- crosses the midpoint, so that $low \leq i \leq mid < j \leq high$.

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

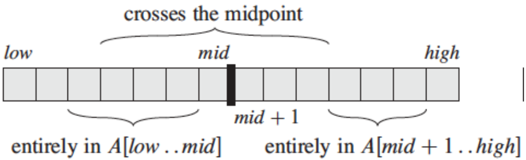
If the best subarray crosses the midpoint, so that $low \leq i \leq mid < j \leq high$, then how do we find it?



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

```
OPT(low, high)
  if low == high return A[low]
  mid = floor of (low + high)/2
  LV = OPT(low, mid)
  RV = OPT(mid+1, high)
  CV = max sum of any subarray
             with starting idx >= low and ending idx = mid
         + max sum of any subarray
             with starting idx = mid+1  and ending idx = high
```

Note: The first term in CV can be computed in $\Theta(mid - low)$ time by computing $S[mid, mid]$, $S[mid-1, mid]$, ... $S[1, mid]$ in this order and taking the max. Similarly (symmetrically), the second term can be computed in $\Theta(high - mid)$ 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 $T(n) = 2T(n/2) + \Theta(n)$

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)
if high == low
    return (low, high, A[low])          // base case: only one element
else mid = ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) =
        FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) =
        FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) =
        FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum
        return (left-low, left-high, left-sum)
    elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
        return (right-low, right-high, right-sum)
    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)
left-sum = -∞
sum = 0
for i = mid downto low
    sum = sum + A[i]
    if sum > left-sum
        left-sum = sum
        max-left = i
right-sum = -∞
sum = 0
for j = mid + 1 to high
    sum = sum + A[j]
    if sum > right-sum
        right-sum = sum
        max-right = j
return (max-left, max-right, left-sum + right-sum)
```

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 $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.

Matrix Multiplication

Naive

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

Matrix Multiplication

Naive

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

Running time:

Matrix Multiplication

Naive

SQUARE-MATRIX-MULTIPLY(A, B)

$n = A.rows$
let C be a new $n \times n$ matrix
for $i = 1$ **to** n
    for $j = 1$ **to** n
        $c_{ij} = 0$
        for $k = 1$ **to** n
            $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
return C

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

Matrix Multiplication

Strassen's algorithm

Based on clever algebraic tricks and divide-and-conquer.
Running time: $\Theta(n^{\log_2 7})$. $(2.8 < \log_2 7 < 2.81)$.

Matrix Multiplication

Basic divide-and-conquer

It is well-known that

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

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

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

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}. \tag{4.10}$$

Matrix Multiplication

Basic divide-and-conquer

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C as in equations (4.9)
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
            + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
            + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
            + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
            + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

Matrix Multiplication

Basic divide-and-conquer

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

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

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

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

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

1. 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.
2. Create 10 matrices $S_1, S_2, \ldots, S_{10}$, 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.
3. Using the submatrices created in step 1 and the 10 matrices created in step 2, recursively compute seven matrix products $P_1, P_2, \ldots, P_7$. Each matrix P_i is $n/2 \times n/2$.
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

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

Matrix Multiplication

Strassen's algorithm: Step 3

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 &&= A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \,, \\
P_2 &= S_2 \cdot B_{22} &&= A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \,, \\
P_3 &= S_3 \cdot B_{11} &&= A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \,, \\
P_4 &= A_{22} \cdot S_4 &&= A_{22} \cdot B_{21} - A_{22} \cdot B_{11} \,, \\
P_5 &= S_5 \cdot S_6 &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \,, \\
P_6 &= S_7 \cdot S_8 &&= A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22} \,, \\
P_7 &= S_9 \cdot S_{10} &&= A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12} \,.
\end{aligned}
$$

Matrix Multiplication

Strassen's algorithm: Step 4

$$C_{11} = P_5 + P_4 - P_2 + P_6 \, .$$

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

Matrix Multiplication

Strassen's algorithm: Step 4

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

and so C_{12} equals

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

Matrix Multiplication

Strassen's algorithm: Step 4

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

makes C_{21} equal

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

Matrix Multiplication

Strassen's algorithm: Step 4

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

so that C_{22} equals

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

Matrix Multiplication

Strassen's algorithm: Step 4

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

so that C_{22} equals

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

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

Part 2: Solving recurrences

Three methods:

- 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 and handy but and not always applicable.

We often simplify recursions...

We often omit floors, ceilings, and boundary conditions.

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

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

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

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

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 $T(n) \leq cn \log n$; 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 $T(1) \cdots T(n-1)$.

$$T(n)$$

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 $T(1) \cdots T(n-1)$.

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

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 $T(1) \cdots T(n-1)$.

$$\begin{aligned} T(n) &\leq 2T(n/2) + bn && (4) \\ &\leq 2(c(n/2)\log(n/2)) + bn && (5) \end{aligned}$$

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 $T(1) \cdots T(n-1)$.

$$\begin{aligned} T(n) &\leq 2T(n/2) + bn & (4) \\ &\leq 2(c(n/2)\log(n/2)) + bn & (5) \\ &= cn \log n - cn + bn & (6) \end{aligned}$$

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 $T(1) \cdots T(n-1)$.

$$\begin{aligned} T(n) &\leq 2T(n/2) + bn && (4) \\ &\leq 2(c(n/2)\log(n/2)) + bn && (5) \\ &= cn\log n - cn + bn && (6) \\ &\leq cn \log n && (7) \end{aligned}$$

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): $T(1) \leq c \log_2 1$

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

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

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

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

Solving recurrences

Substitution method: useful tricks

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

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

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

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

Try $T(n) \le cn + d$.
$T(n) = 2T(n/2) + 1 \le 2(c(n/2) + d) + 1 = cn + 2d + 1$

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

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

Solving recurrences

Substitution method: useful tricks

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

Try $T(n) \le cn$.
$T(n) = 2T(n/2) + 1 \le 2(c(n/2)) + 1 = cn + 1 \le ?cn$

Try $T(n) \le cn + d$.
$T(n) = 2T(n/2) + 1 \le 2(c(n/2) + d) + 1 = cn + 2d + 1 \le ?cn + d$

Use **negative** terms: $T(n) \le cn - 1$.
$T(n) = 2T(n/2) + 1 \le 2(cn/2 - 1) + 1 = cn - 1$

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 $S(m) = T(2^m)$.
$\rightarrow S(m) = 2S(m/2) + m$
$\rightarrow S(m) = \Theta(m \log m)$
$\rightarrow S(n) = \Theta((\log n) \cdot \log \log n)$
Changing variables can help!

Examples:

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

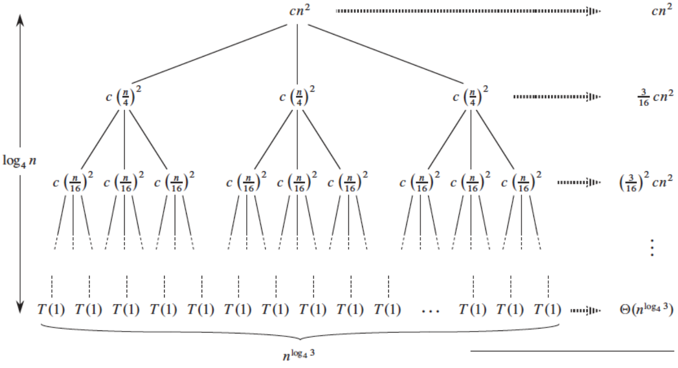
This method can be sloppy if you're not careful. To get full points, you must specify the following key quantities:

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.

Solving recurrences

Recursion Tree in Textbook

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



(d)

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

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

Theorem 4.1 (Master theorem)
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

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ 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:

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

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., $f(n) = O(\frac{g(n)}{n^\epsilon})$ 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 $f(\frac{n}{b}) \leq cf(n)$ for some constant $c < 1$, then $T(n) = \Theta(f(n))$.

Solving recurrences

Master Theorem: 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. $T(n) = T(n/3) + T(2n/3) + \Theta(n)$
6. $T(n) = 27T(n/3) + n^3$
7. $T(n) = 5T(n/2) + n^3$
8. $T(n) = 5T(n/2) + n^2$
9. $T(n) = 4T(n/2) + n^2/\log n$

Solving recurrences

Master Theorem: examples

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

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.

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

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

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

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

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 $n^{\lg 7}$.

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

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

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

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

Solving recurrences

Master Theorem: examples

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

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 $n^{\lg 3}$.

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

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 $n^{\lg 3}$.
$3(n/2)^2 \leq (3/4)n^2$.
Therefore, $T(n) = \Theta(n^2)$.

Solving recurrences

Master Theorem: examples

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

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

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

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