CS170: Lecture 2

CS170: Lecture 2

Last Time: Place value is democratizing!

CS170: Lecture 2

Last Time: Place value is democratizing!
 Like the printing press!

CS170: Lecture 2

Last Time: Place value is democratizing!
Like the printing press!
Reading, writing, arithmetic!

CS170: Lecture 2

Last Time: Place value is democratizing!

Like the printing press!

Reading, writing, arithmetic!

Input size/representation really matters!

CS170: Lecture 2

Last Time: Place value is democratizing!
  Like the printing press!
   Reading, writing, arithmetic!
Input size/representation really matters!

Today: Chapter 2.

CS170: Lecture 2

Last Time: Place value is democratizing!
Like the printing press!
Reading, writing, arithmetic!
Input size/representation really matters!

Today: Chapter 2.
Divide and Conquer

CS170: Lecture 2

Last Time: Place value is democratizing!
Like the printing press!
Reading, writing, arithmetic!
Input size/representation really matters!

Today: Chapter 2.
Divide and Conquer $\equiv$ Recursive.

CS170: Lecture 2

Last Time: Place value is democratizing!
  Like the printing press!
    Reading, writing, arithmetic!
Input size/representation really matters!

Today: Chapter 2.
  Divide and Conquer $\equiv$ Recursive.

Lecture in one minute!

Integer Multiplication: Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Lecture in one minute!

Integer Multiplication: Gauss plus recursion is magic!

$O(n^2) \to O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

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

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Lecture in one minute!

Integer Multiplication: Gauss plus recursion is magic!

$O(n^2) \to O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

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

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Recursive (Divide and Conquer) Matrix Multiplication:

8 subroutine calls of size $n/2 \times n/2$

$\to O(n^3)$.

Strassen:

7 subroutine calls of size $n/2 \times n/2$

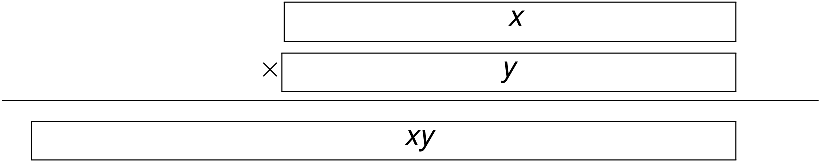
$\to O(n^{\log_2 7}) \approx O(n^{2.8})$.

Chapter 2.

Divide and conquer.

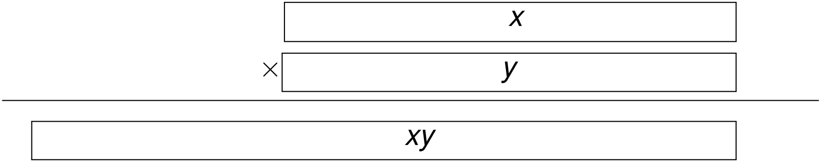
Definition of Multiplication.

n-bit numbers: x, y.



Definition of Multiplication.

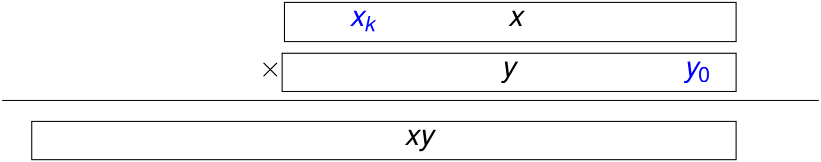
n-bit numbers: x, y.



kth "place" of xy:

Definition of Multiplication.

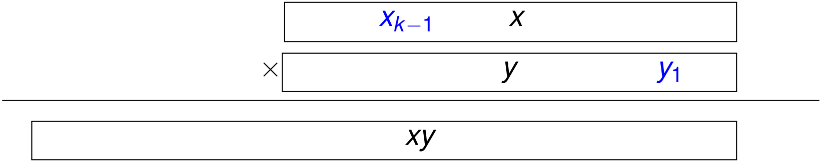
n-bit numbers: x, y.



kth “place” of xy: coefficient of 2^k:

Definition of Multiplication.

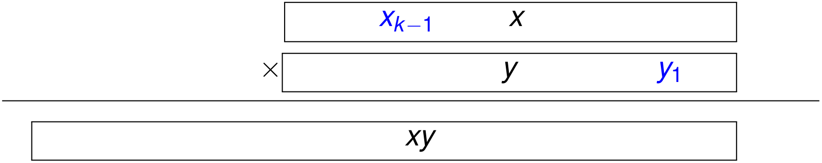
n-bit numbers: x, y.



kth "place" of xy: coefficient of 2^k:

Definition of Multiplication.

n-bit numbers: x, y.

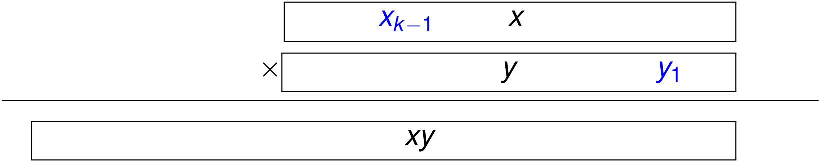


kth "place" of xy: coefficient of 2^k:

$$a_k = \sum_{i \leq k} x_i y_{k-i}.$$

Definition of Multiplication.

n-bit numbers: x, y.



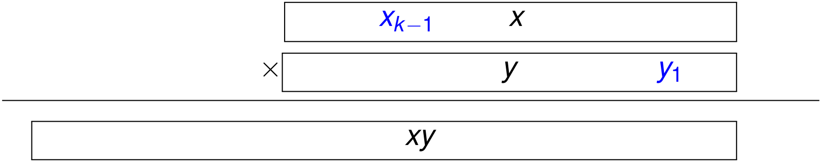
kth "place" of xy: coefficient of 2^k:

$$a_k = \sum_{i \leq k} x_i y_{k-i}.$$

$x * y = \sum_{k=0}^{2n} 2^k a_k$.

Definition of Multiplication.

n-bit numbers: x, y.



kth "place" of xy: coefficient of 2^k:

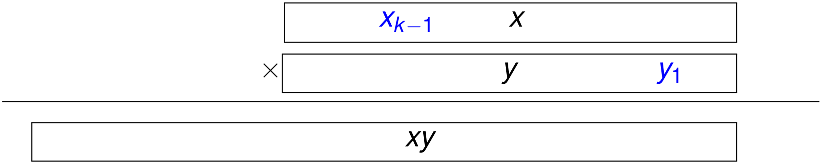
$$a_k = \sum_{i \leq k} x_i y_{k-i}.$$

$x * y = \sum_{k=0}^{2n} 2^k a_k$.

Number of "basic operations":

Definition of Multiplication.

n-bit numbers: x, y.



kth "place" of xy: coefficient of 2^k:

$$a_k = \sum_{i \le k} x_i y_{k-i}.$$

$x * y = \sum_{k=0}^{2n} 2^k a_k$.

Number of "basic operations":

$$\sum_{k \le 2n} \min(k, 2n-k) = \Theta(n^2).$$

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = \boxed{\quad x_L \quad}\boxed{\quad x_R \quad}$$

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = \boxed{\quad x_L \quad}\boxed{\quad x_R \quad} = 2^{n/2}x_L + x_R$$

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

x	$=$	x_L	x_R	$= 2^{n/2} x_L + x_R$
y	$=$	y_L	y_R	

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = \boxed{\quad x_L \quad}\boxed{\quad x_R \quad} = 2^{n/2}x_L + x_R$$

$$y = \boxed{\quad y_L \quad}\boxed{\quad y_R \quad} = 2^{n/2}y_L + y_R$$

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

x	$=$	x_L	x_R	$= 2^{n/2}x_L + x_R$
y	$=$	y_L	y_R	$= 2^{n/2}y_L + y_R$

Multiplying out

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

x	$=$	x_L	x_R	$= 2^{n/2} x_L + x_R$
y	$=$	y_L	y_R	$= 2^{n/2} y_L + y_R$

Multiplying out

$$x \times y$$

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

x	$=$	x_L	x_R	$= 2^{n/2}x_L + x_R$
y	$=$	y_L	y_R	$= 2^{n/2}y_L + y_R$

Multiplying out

$$x \times y = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R)$$

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

x	$=$	x_L	x_R	$= 2^{n/2}x_L + x_R$
y	$=$	y_L	y_R	$= 2^{n/2}y_L + y_R$

Multiplying out

$$
\begin{aligned}
x \times y &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\
&= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R
\end{aligned}
$$

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

$x =$	x_L	x_R	$= 2^{n/2}x_L + x_R$
$y =$	y_L	y_R	$= 2^{n/2}y_L + y_R$

Multiplying out

$$\begin{aligned} x \times y &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\ &= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Four $n/2$-bit multiplications: $x_L y_L$, $x_L y_R$, $x_R y_L$, $x_R y_R$.

Recursive Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = \boxed{\quad x_L \quad}\boxed{\quad x_R \quad} = 2^{n/2}x_L + x_R$$

$$y = \boxed{\quad y_L \quad}\boxed{\quad y_R \quad} = 2^{n/2}y_L + y_R$$

Multiplying out

$$\begin{aligned} x \times y &= (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) \\ &= 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R \end{aligned}$$

Four $n/2$-bit multiplications: $x_L y_L$, $x_L y_R$, $x_R y_L$, $x_R y_R$.
Recurrence:

$$T(n) = 4T(\frac{n}{2}) + O(n)$$

Recurrence for recursive algorithm.

Recurrence:

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

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.
A degree 4 tree of depth $\log_2 n$.

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.
A degree 4 tree of depth $\log_2 n$.
$\Theta(n^2)$ leaves or base cases.

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.
  A degree 4 tree of depth $\log_2 n$.
  $\Theta(n^2)$ leaves or base cases.
One for each pair of digits!

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.

A degree 4 tree of depth $\log_2 n$.

$\Theta(n^2)$ leaves or base cases.

One for each pair of digits!

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.
A degree 4 tree of depth $\log_2 n$.
$\Theta(n^2)$ leaves or base cases.
One for each pair of digits!

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.
  A degree 4 tree of depth $\log_2 n$.
  $\Theta(n^2)$ leaves or base cases.
One for each pair of digits!

Really?

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.
  A degree 4 tree of depth $\log_2 n$.
  $\Theta(n^2)$ leaves or base cases.
One for each pair of digits!

Really? Unfolded recursion in my head?!?!

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.
A degree 4 tree of depth $\log_2 n$.
$\Theta(n^2)$ leaves or base cases.
One for each pair of digits!

Really? Unfolded recursion in my head?!?!
How did I really obtain bound?

Recurrence for recursive algorithm.

Recurrence:

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

$T(n)$ is

(A) $\Theta(n)$.

(B) $\Theta(n^2)$.

(C) $\Theta(n^3)$.

Idea: Think about recursion tree.
 A degree 4 tree of depth $\log_2 n$.
 $\Theta(n^2)$ leaves or base cases.
One for each pair of digits!

Really? Unfolded recursion in my head?!?!
How did I really obtain bound? Soon a formula.

Demo

As number of bits double:

Demo

As number of bits double:

Demo

As number of bits double:

Elementary School Multiply:

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \to 2n$

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \to 2n$
Runtime: $T \to 3T$.

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \rightarrow 2n$

Runtime: $T = cn^2 \rightarrow T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \rightarrow 2n$
Runtime: $T \rightarrow 3T$.

Asymptotics: $T = cn^w \rightarrow c((2n)^w) = T' = 3T = 3(cn^w)$.

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \to 2n$
Runtime: $T \to 3T$.

Asymptotics: $T = cn^w \to c((2n)^w) = T' = 3T = 3(cn^w)$.
.... $\to 2^w = 3$.

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \to 2n$
Runtime: $T \to 3T$.

Asymptotics: $T = cn^w \to c((2n)^w) = T' = 3T = 3(cn^w)$.
.... $\to 2^w = 3$. or $w = \log_2 3 \approx 1.58$.

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \to 2n$
Runtime: $T \to 3T$.

Asymptotics: $T = cn^w \to c((2n)^w) = T' = 3T = 3(cn^w)$.
.... $\to 2^w = 3$. or $w = \log_2 3 \approx 1.58$.

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \to 2n$
Runtime: $T \to 3T$.

Asymptotics: $T = cn^w \to c((2n)^w) = T' = 3T = 3(cn^w)$.
.... $\to 2^w = 3$. or $w = \log_2 3 \approx 1.58$.

Python multiply: $O(n^{\log_2 3})$

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \to 2n$
Runtime: $T \to 3T$.

Asymptotics: $T = cn^w \to c((2n)^w) = T' = 3T = 3(cn^w)$.
.... $\to 2^w = 3$. or $w = \log_2 3 \approx 1.58$.

Python multiply: $O(n^{\log_2 3})$

Demo

As number of bits double:

Elementary School Multiply:

$O(n^2)$
$n \to 2n$

Runtime: $T = cn^2 \to T' = c(2n)^2 = 4(cn^2) = 4T$

Python multiply:

$n \to 2n$
Runtime: $T \to 3T$.

Asymptotics: $T = cn^w \to c((2n)^w) = T' = 3T = 3(cn^w)$.
.... $\to 2^w = 3$. or $w = \log_2 3 \approx 1.58$.

Python multiply: $O(n^{\log_2 3})$

Much better than grade school.

Multiply Complex Numbers

$$(3+2\,\mathbf{i})(4+5\,\mathbf{i}) = 12+(15+8)\,\mathbf{i}+10\,\mathbf{i}^2$$

Multiply Complex Numbers

$(3 + 2\,\mathbf{i})(4 + 5\,\mathbf{i}) = 12 + (15 + 8)\,\mathbf{i} + 10\,\mathbf{i}^2$

Recall, $\mathbf{i}^2 = -1$, so simplifying

Multiply Complex Numbers

$(3+2\,\mathbf{i})(4+5\,\mathbf{i}) = 12+(15+8)\,\mathbf{i}+10\,\mathbf{i}^2$

Recall, $\mathbf{i}^2 = -1$, so simplifying

$$(12-10)+22\,\mathbf{i} = 2+22\,\mathbf{i}.$$

Multiply Complex Numbers

$(3+2\,\mathbf{i})(4+5\,\mathbf{i}) = 12+(15+8)\,\mathbf{i}+10\,\mathbf{i}^2$

Recall, $\mathbf{i}^2 = -1$, so simplifying

$(12-10)+22\,\mathbf{i} = 2+22\,\mathbf{i}.$

What about $(32765+219898\,\mathbf{i})(413764+511110\,\mathbf{i})$?

Gauss's trick.

$$(a + b\,\mathbf{i})(c + d\,\mathbf{i})$$

Gauss's trick.

$$(a+b\ \mathbf{i})(c+d\ \mathbf{i}) = (ac-bd)+(ad+bc)\ \mathbf{i}.$$

Gauss's trick.

$$(a+b\,\mathbf{i})(c+d\,\mathbf{i}) = (ac-bd)+(ad+bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Gauss's trick.

$$(a+b\,\mathbf{i})(c+d\,\mathbf{i}) = (ac-bd)+(ad+bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a+b)(c+d) = ac+ad+bc+bd.$$

Gauss's trick.

$$(a + b\ \mathbf{i})(c + d\ \mathbf{i}) = (ac - bd) + (ad + bc)\ \mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a + b)(c + d) = ac + ad + bc + bd.$$

Four multiplications from one!

Gauss's trick.

$$(a + b\ \mathbf{i})(c + d\ \mathbf{i}) = (ac - bd) + (ad + bc)\ \mathbf{i}.$$

Four multiplications: ac, bd, ad, bd**.**

Drop the i:

$$P_1 = (a + b)(c + d) = ac + ad + bc + bd.$$

Four multiplications from one! **..**but all added up**.**

Gauss's trick.

$$(a + b\,\mathbf{i})(c + d\,\mathbf{i}) = (ac - bd) + (ad + bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a + b)(c + d) = ac + ad + bc + bd.$$

Four multiplications from one! ..but all added up.

Two more multiplications: $P_2 = ac$, $P_3 = bd$.

Gauss's trick.

$$(a + b\,\mathbf{i})(c + d\,\mathbf{i}) = (ac - bd) + (ad + bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a + b)(c + d) = ac + ad + bc + bd.$$

Four multiplications from one! ..but all added up.

Two more multiplications: $P_2 = ac$, $P_3 = bd$.

$$(ac - bd) = P_2 - P_3.$$

Gauss's trick.

$$(a+b\,\mathbf{i})(c+d\,\mathbf{i}) = (ac-bd)+(ad+bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a+b)(c+d) = ac+ad+bc+bd.$$

Four multiplications from one! ..but all added up.

Two more multiplications: $P_2 = ac$, $P_3 = bd$.

$$(ac-bd) = P_2 - P_3.$$

$$(ad+bc) = P_1 - P_2 - P_3.$$

Gauss's trick.

$$(a + b\,\mathbf{i})(c + d\,\mathbf{i}) = (ac - bd) + (ad + bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a+b)(c+d) = ac + ad + bc + bd.$$

Four multiplications from one! ..but all added up.

Two more multiplications: $P_2 = ac$, $P_3 = bd$.

$$(ac - bd) = P_2 - P_3.$$

$$(ad + bc) = P_1 - P_2 - P_3.$$

Only three multiplications.

Gauss's trick.

$$(a + b\,\mathbf{i})(c + d\,\mathbf{i}) = (ac - bd) + (ad + bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a+b)(c+d) = ac + ad + bc + bd.$$

Four multiplications from one! ..but all added up.

Two more multiplications: $P_2 = ac$, $P_3 = bd$.

$$(ac - bd) = P_2 - P_3.$$

$$(ad + bc) = P_1 - P_2 - P_3.$$

Only three multiplications. An extra addition though!

Gauss's trick.

$$(a+b\,\mathbf{i})(c+d\,\mathbf{i}) = (ac-bd)+(ad+bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a+b)(c+d) = ac+ad+bc+bd.$$

Four multiplications from one! ..but all added up.

Two more multiplications: $P_2 = ac$, $P_3 = bd$.

$$(ac-bd) = P_2 - P_3.$$

$$(ad+bc) = P_1 - P_2 - P_3.$$

Only three multiplications. An extra addition though!
Which is harder of multiplication or addition?

Gauss's trick.

$$(a+b\,\mathbf{i})(c+d\,\mathbf{i}) = (ac-bd)+(ad+bc)\,\mathbf{i}.$$

Four multiplications: ac, bd, ad, bd.

Drop the i:

$$P_1 = (a+b)(c+d) = ac+ad+bc+bd.$$

Four multiplications from one! ..but all added up.

Two more multiplications: $P_2 = ac$, $P_3 = bd$.

$$(ac-bd) = P_2 - P_3.$$

$$(ad+bc) = P_1 - P_2 - P_3.$$

Only three multiplications. An extra addition though!
Which is harder of multiplication or addition?
Multiplication!

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R$$

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$
$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = 2^{n/2} x_L + x_R \quad ; \quad y = 2^{n/2} y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Used four $\frac{n}{2}$-bit multiplications: $x_L y_L$, $x_L y_R$, $x_R y_L$, $x_R y_R$.

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Used four $\frac{n}{2}$-bit multiplications: $x_L y_L$, $x_L y_R$, $x_R y_L$, $x_R y_R$.

Can you compute three terms with 3 multiplications?

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Used four $\frac{n}{2}$-bit multiplications: $x_L y_L$, $x_L y_R$, $x_R y_L$, $x_R y_R$.

Can you compute three terms with 3 multiplications?

(A) Yes.

(B) No

Faster Algorithm for Multiplication.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$
$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Used four $\frac{n}{2}$-bit multiplications: $x_L y_L$, $x_L y_R$, $x_R y_L$, $x_R y_R$.

Can you compute three terms with 3 multiplications?

(A) Yes.

(B) No

(A) Yes.

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2} x_L + x_R \quad ; \quad y = 2^{n/2} y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2} x_L + x_R \quad ; \quad y = 2^{n/2} y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R)$$

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R.$$

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$
$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R.$$

Two more: $P_2 = x_L y_L$, $P_3 = x_R y_R$.

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R.$$

Two more: $P_2 = x_L y_L$, $P_3 = x_R y_R$. $(x_L y_R + x_R y_L) = P_1 - P_2 - P_3$

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R.$$

Two more: $P_2 = x_L y_L$, $P_3 = x_R y_R$. $(x_L y_R + x_R y_L) = P_1 - P_2 - P_3$

3 multiplications!

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R.$$

Two more: $P_2 = x_L y_L$, $P_3 = x_R y_R$. $(x_L y_R + x_R y_L) = P_1 - P_2 - P_3$
3 multiplications!

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

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R.$$

Two more: $P_2 = x_L y_L$, $P_3 = x_R y_R$. $(x_L y_R + x_R y_L) = P_1 - P_2 - P_3$

3 multiplications!

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

Technically: $\frac{n}{2} + 1$ bit multiplication.

Three multiplications and faster algorithm.

Two n-bit numbers: x, y.

$$x = 2^{n/2}x_L + x_R \quad ; \quad y = 2^{n/2}y_L + y_R$$

$$x \times y = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

Need 3 terms: $x_L y_L$, $x_L y_R + x_R y_L$, $x_R y_R$.

Compute

$$P_1 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R.$$

Two more: $P_2 = x_L y_L$, $P_3 = x_R y_R$. $(x_L y_R + x_R y_L) = P_1 - P_2 - P_3$
3 multiplications!

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

Technically: $\frac{n}{2} + 1$ bit multiplication. Don't worry.

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$

(B) $\Theta(n^2)$

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$

(B) $\Theta(n^2)$

(C) $\Theta(n^{\log_2 3})$

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$

(B) $\Theta(n^2)$

(C) $\Theta(n^{\log_2 3})$

(C)

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$

(B) $\Theta(n^2)$

(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

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

Analysis of runtime.

Recurrence for "fast algorithm".

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots})$$

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58...})!$$

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58...})!!$$

Analysis of runtime.

Recurrence for "fast algorithm".

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

Runtime is

(A) $\Theta(n)$

(B) $\Theta(n^2)$

(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58...})!!!$$

Analysis of runtime.

Recurrence for "fast algorithm".

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58...})!!!!$$

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58...})!!!!$$

But: all digits have to multiply each other!

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots})!!!!$$

But: all digits have to multiply each other!
They do!

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58...})!!!!$$

But: all digits have to multiply each other!
They do! $(a+b)(c+d) = ac + ac + bc + bd$

Analysis of runtime.

Recurrence for “fast algorithm”.

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

Runtime is

(A) $\Theta(n)$
(B) $\Theta(n^2)$
(C) $\Theta(n^{\log_2 3})$

(C) Idea: number of base cases is $n^{\log_2 3}$.
More soon.

So multiplication algorithm with ..

$$T(n) = 3T(\frac{n}{2}) + \Theta(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58...})!!!!$$

But: all digits have to multiply each other!
They do! $(a+b)(c+d) = ac + ac + bc + bd$
4 products from one multiplication!

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c$

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc}$

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb}$

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log:

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log: $a = b^{\log_b a}$

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log: $a = b^{\log_b a}$

Logarithm Quiz: $a^{\log_b n} = n^{\log_b a}$?

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log: $a = b^{\log_b a}$

Logarithm Quiz: $a^{\log_b n} = n^{\log_b a}$?

Yes!

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log: $a = b^{\log_b a}$

Logarithm Quiz: $a^{\log_b n} = n^{\log_b a}$?

Yes!

$$a^{\log_b n}$$

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log: $a = b^{\log_b a}$

Logarithm Quiz: $a^{\log_b n} = n^{\log_b a}$?

Yes!

$$a^{\log_b n} = (b^{\log_b a})^{\log_b n}$$

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log: $a = b^{\log_b a}$

Logarithm Quiz: $a^{\log_b n} = n^{\log_b a}$?

Yes!

$$a^{\log_b n} = (b^{\log_b a})^{\log_b n} = (b^{\log_b n})^{\log_b a}$$

Logarithms reminder.

Exponents Quiz: $(a^b)^c = (a^c)^b$?

Yes? No?

Yes. $(a^b)^c = a^{bc} = a^{cb} = (a^c)^b$.

Definition of log: $a = b^{\log_b a}$

Logarithm Quiz: $a^{\log_b n} = n^{\log_b a}$?

Yes!

$$a^{\log_b n} = (b^{\log_b a})^{\log_b n} = (b^{\log_b n})^{\log_b a} = n^{\log_b a}$$

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Solving recurrences.

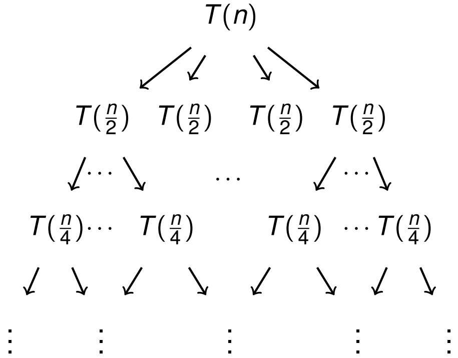
$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \; \swarrow \quad \searrow \; \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots \quad T(\frac{n}{4}) \qquad T(\frac{n}{4}) \quad \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree



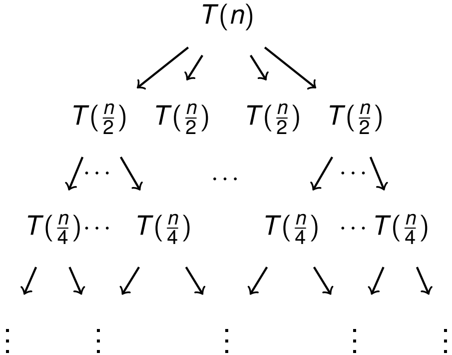
# probs	sz	time/prob	time/level
1	n	cn	cn
4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n$

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree



# probs	sz	time/prob	time/level
1	n	cn	cn
4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \swarrow \quad \searrow \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots \quad T(\frac{n}{4}) \quad T(\frac{n}{4}) \quad \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$4^{\log n}$

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

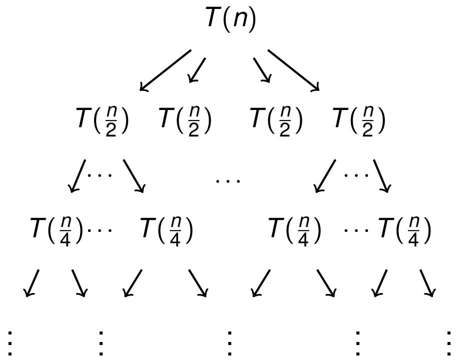
Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \swarrow \searrow \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots \quad T(\frac{n}{4}) \quad T(\frac{n}{4}) \quad \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \searrow \swarrow \searrow \swarrow \searrow \swarrow \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$4^{\log n} = 2^{2 \log n}$

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree



# probs	sz	time/prob	time/level
1	n	cn	cn
4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$4^{\log n} = 2^{2\log n} = n^2$ base case problems.

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$T(\frac{n}{2})$ $T(\frac{n}{2})$ $T(\frac{n}{2})$ $T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$T(\frac{n}{4})\cdots$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $\cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1.

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \; \swarrow \quad \searrow \; \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots \quad T(\frac{n}{4}) \qquad T(\frac{n}{4}) \quad \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$T(\frac{n}{2})\ T(\frac{n}{2})\ T(\frac{n}{2})\ T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$T(\frac{n}{4})\cdots\ T(\frac{n}{4})\ \ T(\frac{n}{4})\ \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c

Work: cn^2.

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \; \swarrow \quad \searrow \; \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots \quad T(\frac{n}{4}) \quad T(\frac{n}{4}) \quad \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c

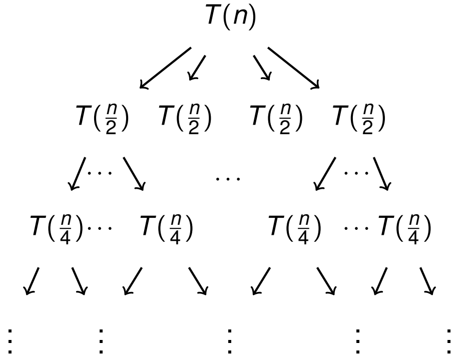
Work: cn^2.

Total Work:

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree



# probs	sz	time/prob	time/level
1	n	cn	cn
4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c

Work: cn^2.

Total Work: cn

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \swarrow \searrow \searrow$				
$T(\frac{n}{2})$ $T(\frac{n}{2})$ $T(\frac{n}{2})$ $T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow$ $\cdots$ $\swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $\cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \searrow \swarrow \searrow \swarrow \searrow \swarrow \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$ $\vdots$ $\vdots$ $\vdots$ $\vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c
Work: cn^2.
Total Work: $cn + 2cn$

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \; \swarrow \quad \searrow \; \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots \quad T(\frac{n}{4}) \qquad T(\frac{n}{4}) \quad \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c

Work: cn^2.

Total Work: $cn + 2cn + 4cn + \cdots$

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \; \swarrow \quad \searrow \; \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots \; T(\frac{n}{4}) \quad T(\frac{n}{4}) \; \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c

Work: cn^2.

Total Work: $cn + 2cn + 4cn + \cdots + cn^2$

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$T(\frac{n}{2})$ $T(\frac{n}{2})$ $T(\frac{n}{2})$ $T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$T(\frac{n}{4})\cdots$ $T(\frac{n}{4})$ $T(\frac{n}{4})$ $\cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots$ $\vdots$ $\vdots$ $\vdots$ $\vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c
Work: cn^2.
Total Work: $cn + 2cn + 4cn + \cdots + cn^2 = O(n^2)$.

Solving recurrences.

$$T(n) = 4T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \; \swarrow \quad \searrow \; \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	4	$\frac{n}{2}$	$c(\frac{n}{2})$	$2cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots \quad T(\frac{n}{4}) \quad T(\frac{n}{4}) \quad \cdots T(\frac{n}{4})$	4^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$4cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	4^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$2^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$4^{\log n} = 2^{2\log n} = n^2$ base case problems. size 1. Work/Prob: c
Work: cn^2.
Total Work: $cn + 2cn + 4cn + \cdots + cn^2 = O(n^2)$. Geometric series.

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n$

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \downarrow \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \searrow \swarrow \searrow \swarrow \searrow \swarrow \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \quad \downarrow \quad \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$3^{\log_2 n}$

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \quad \downarrow \quad \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$3^{\log_2 n} = n^{\log_2 3}$ base case problems.

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1.

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \quad \downarrow \quad \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \quad \searrow \quad \swarrow \quad \searrow \quad \swarrow \quad \searrow \quad \swarrow \quad \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c.

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \quad \downarrow \quad \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c. Work: $cn^{\log_2 3}$.

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c. Work: $cn^{\log_2 3}$.

Total Work:

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c. Work: $cn^{\log_2 3}$.

Total Work: cn

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \quad \downarrow \quad \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \quad \searrow \quad \swarrow \quad \searrow \quad \swarrow \quad \searrow \quad \swarrow \quad \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c. Work: $cn^{\log_2 3}$.

Total Work: $cn + (\frac{3}{2})cn$

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c. Work: $cn^{\log_2 3}$.

Total Work: $cn + (\frac{3}{2})cn + \cdots$

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \; \downarrow \; \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.
$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c. Work: $cn^{\log_2 3}$.

Total Work: $cn + (\frac{3}{2})cn + \cdots + cn^{\log_2 3}$

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c. Work: $cn^{\log_2 3}$.

Total Work: $cn + (\frac{3}{2})cn + \cdots + cn^{\log_2 3} = O(n^{\log_2 3})$

Fast multiplication.

$$T(n) = 3T(\frac{n}{2}) + cn; \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/level
$T(n)$	1	n	cn	cn
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{2}) \quad T(\frac{n}{2}) \quad T(\frac{n}{2})$	3	$\frac{n}{2}$	$c(\frac{n}{2})$	$(\frac{3}{2})cn$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{4}) \cdots T(\frac{n}{4}) \quad T(\frac{n}{4}) \cdots T(\frac{n}{4})$	3^2	$\frac{n}{4}$	$c(\frac{n}{4})$	$(\frac{3}{2})^2 cn$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	3^i	$\frac{n}{2^i}$	$c(\frac{n}{2^i})$	$(\frac{3}{2})^i cn$

$\frac{n}{2^i} = 1$ when $i = \log_2 n \implies$ Depth: $d = \log_2 n$.

$3^{\log_2 n} = n^{\log_2 3}$ base case problems. size 1. Work/Prob: c. Work: $cn^{\log_2 3}$.

Total Work: $cn + (\frac{3}{2})cn + \cdots + cn^{\log_2 3} = O(n^{\log_2 3})$ Geometric series.

Divide and Conquer: In general.

$$T(n) = aT(\frac{n}{b}) + O(n^d); \qquad T(1) = c$$

Divide and Conquer: In general.

$$T(n) = aT(\frac{n}{b}) + O(n^d); \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/lvl
$T(n)$	1	n	cn^d	cn^d
$\swarrow \; \downarrow \; \searrow$				
$T(\frac{n}{b}) \quad T(\frac{n}{b}) \quad T(\frac{n}{b})$	a	$\frac{n}{b}$	$c(\frac{n}{b})^d$	$(\frac{a}{b^d})cn^d$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{b^2}) \cdots T(\frac{n}{b^2}) \quad T(\frac{n}{b^2}) \cdots T(\frac{n}{b^2})$	a^2	$\frac{n}{b^2}$	$c(\frac{n}{b^2})^d$	$(\frac{a}{b^d})^2 cn^d$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	a^i	$\frac{n}{b^i}$	$c(\frac{n}{b^i})^d$	$(\frac{a}{b^d})^i cn^d$

Divide and Conquer: In general.

$$T(n) = aT(\frac{n}{b}) + O(n^d); \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/lvl
$T(n)$	1	n	cn^d	cn^d
$\swarrow \; \downarrow \; \searrow$				
$T(\frac{n}{b}) \quad T(\frac{n}{b}) \quad T(\frac{n}{b})$	a	$\frac{n}{b}$	$c(\frac{n}{b})^d$	$(\frac{a}{b^d})cn^d$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{b^2}) \cdots T(\frac{n}{b^2}) \quad T(\frac{n}{b^2}) \cdots T(\frac{n}{b^2})$	a^2	$\frac{n}{b^2}$	$c(\frac{n}{b^2})^d$	$(\frac{a}{b^d})^2 cn^d$
$\swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow \; \swarrow \; \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	a^i	$\frac{n}{b^i}$	$c(\frac{n}{b^i})^d$	$(\frac{a}{b^d})^i cn^d$

$\frac{n}{b^i} = 1$ when $i = \log_b n$

Divide and Conquer: In general.

$$T(n) = aT(\frac{n}{b}) + O(n^d); \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/lvl
$T(n)$	1	n	cn^d	cn^d
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{b}) \quad T(\frac{n}{b}) \quad T(\frac{n}{b})$	a	$\frac{n}{b}$	$c(\frac{n}{b})^d$	$(\frac{a}{b^d})cn^d$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{b^2}) \cdots T(\frac{n}{b^2}) \quad T(\frac{n}{b^2}) \cdots T(\frac{n}{b^2})$	a^2	$\frac{n}{b^2}$	$c(\frac{n}{b^2})^d$	$(\frac{a}{b^d})^2 cn^d$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	a^i	$\frac{n}{b^i}$	$c(\frac{n}{b^i})^d$	$(\frac{a}{b^d})^i cn^d$

$\frac{n}{b^i} = 1$ when $i = \log_b n \implies$ Depth: $k = \log_b n$.

Divide and Conquer: In general.

$$T(n) = aT(\frac{n}{b}) + O(n^d); \qquad T(1) = c$$

Recursion Tree	# probs	sz	time/prob	time/lvl
$T(n)$	1	n	cn^d	cn^d
$\swarrow \ \downarrow \ \searrow$				
$T(\frac{n}{b}) \quad T(\frac{n}{b}) \quad T(\frac{n}{b})$	a	$\frac{n}{b}$	$c(\frac{n}{b})^d$	$(\frac{a}{b^d})cn^d$
$\swarrow \cdots \searrow \quad \cdots \quad \swarrow \cdots \searrow$				
$T(\frac{n}{b^2}) \cdots T(\frac{n}{b^2}) \quad T(\frac{n}{b^2}) \cdots T(\frac{n}{b^2})$	a^2	$\frac{n}{b^2}$	$c(\frac{n}{b^2})^d$	$(\frac{a}{b^d})^2 cn^d$
$\swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow \ \swarrow \ \searrow$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots$	a^i	$\frac{n}{b^i}$	$c(\frac{n}{b^i})^d$	$(\frac{a}{b^d})^i cn^d$

$\frac{n}{b^i} = 1$ when $i = \log_b n \implies$ Depth: $k = \log_b n$.

Level i work: $(\frac{a}{b^d})^i n^d$.

Master's Theorem

Depth: $\log_b n$.

Master's Theorem

Depth: $\log_b n$.
Level i work:

Master's Theorem

Depth: $\log_b n$.
Level i work:

$$\left(\frac{a}{b^d}\right)^i n^d.$$

Master's Theorem

Depth: $\log_b n$.
Level i work:

$$(\frac{a}{b^d})^i n^d.$$

Total:

$$n^d \sum_{i=0}^{\log_b n} (\frac{a}{b^d})^i$$

Master's Theorem

Depth: $\log_b n$.
Level i work:

$$(\frac{a}{b^d})^i n^d.$$

Total:

$$n^d \sum_{i=0}^{\log_b n} (\frac{a}{b^d})^i$$

Geometric series:

Master's Theorem

Depth: $\log_b n$.
Level i work:

$$(\frac{a}{b^d})^i n^d.$$

Total:

$$n^d \sum_{i=0}^{\log_b n} (\frac{a}{b^d})^i$$

Geometric series: If $\frac{a}{b^d} < 1$ ($d > \log_b a$), first term dominates

$$O(n^d),$$

Master's Theorem

Depth: $\log_b n$.
Level i work:

$$(\frac{a}{b^d})^i n^d.$$

Total:

$$n^d \sum_{i=0}^{\log_b n} (\frac{a}{b^d})^i$$

Geometric series: If $\frac{a}{b^d} < 1$ ($d > \log_b a$), first term dominates

$$O(n^d),$$

if $\frac{a}{b^d} > 1$ ($d < \log_b a$), last term dominates.

$$O(n^{\log_b a}),$$

Master's Theorem

Depth: $\log_b n$.
Level i work:

$$(\frac{a}{b^d})^i n^d.$$

Total:

$$n^d \sum_{i=0}^{\log_b n} (\frac{a}{b^d})^i$$

Geometric series: If $\frac{a}{b^d} < 1$ ($d > \log_b a$), first term dominates

$$O(n^d),$$

if $\frac{a}{b^d} > 1$ ($d < \log_b a$), last term dominates.

$$O(n^{\log_b a}),$$

and if $\frac{a}{b^d} = 1$ ($d = \log_b a$), then all terms are the same

$$O(n^d \log_b n).$$

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n)$.

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a$	$T(n) = O(n^d)$
$d < \log_b a$	$T(n) = O(n^{\log_b a})$
$d = \log_b a$	$T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a$	$T(n) = O(n^d)$
$d < \log_b a$	$T(n) = O(n^{\log_b a})$
$d = \log_b a$	$T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a$	$T(n) = O(n^d)$
$d < \log_b a$	$T(n) = O(n^{\log_b a})$
$d = \log_b a$	$T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a$

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2)$.

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2)$.

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

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have
$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2)$.

$T(n) = T(\frac{n}{2}) + O(n)$ $a = 1$, $b = 2$, and $d = 1$.

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2)$.

$T(n) = T(\frac{n}{2}) + O(n)$ $a = 1$, $b = 2$, and $d = 1$.
$1 > \log_2 1 = 0$

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n).$

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2).$

$T(n) = T(\frac{n}{2}) + O(n)$ $a = 1$, $b = 2$, and $d = 1$.
$1 > \log_2 1 = 0 \implies T(n) = O(n)$

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2)$.

$T(n) = T(\frac{n}{2}) + O(n)$ $a = 1$, $b = 2$, and $d = 1$.
$1 > \log_2 1 = 0 \implies T(n) = O(n)$

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

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2)$.

$T(n) = T(\frac{n}{2}) + O(n)$ $a = 1$, $b = 2$, and $d = 1$.
$1 > \log_2 1 = 0 \implies T(n) = O(n)$

$T(n) = 2T(\frac{n}{2}) + O(n)$ $a = 2$, $b = 2$, and $d = 1$.

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n).$

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2).$

$T(n) = T(\frac{n}{2}) + O(n)$ $a = 1$, $b = 2$, and $d = 1$.
$1 > \log_2 1 = 0 \implies T(n) = O(n)$

$T(n) = 2T(\frac{n}{2}) + O(n)$ $a = 2$, $b = 2$, and $d = 1$.
$1 = \log_2 2$

Master's Theorem: examples.

For a recurrence $T(n) = aT(n/b) + O(n^d)$
We have

$d > \log_b a \quad T(n) = O(n^d)$
$d < \log_b a \quad T(n) = O(n^{\log_b a})$
$d = \log_b a \quad T(n) = O(n^d \log_b n)$.

$T(n) = 4T(\frac{n}{2}) + O(n)$ $a = 4$, $b = 2$, and $d = 1$.
$d = 1 < 2 = \log_2 4 = \log_b a \implies T(n) = O(n^{\log_b a}) = O(n^2)$.

$T(n) = T(\frac{n}{2}) + O(n)$ $a = 1$, $b = 2$, and $d = 1$.
$1 > \log_2 1 = 0 \implies T(n) = O(n)$

$T(n) = 2T(\frac{n}{2}) + O(n)$ $a = 2$, $b = 2$, and $d = 1$.
$1 = \log_2 2 \implies T(n) = O(n \log n)$

Strassen

Matrix multiplication.

Strassen

Matrix multiplication.

Strassen, 1968, visiting Berkeley.

Strassen

Matrix multiplication.

Strassen, 1968, visiting Berkeley.

Berkeley...

Strassen

Matrix multiplication.

Strassen, 1968, visiting Berkeley.

Berkeley...Unite!

Strassen

Matrix multiplication.

Strassen, 1968, visiting Berkeley.

Berkeley...Unite! Resist!

Strassen

Matrix multiplication.

Strassen, 1968, visiting Berkeley.

Berkeley...Unite! Resist!

Strassen:

Strassen

Matrix multiplication.

Strassen, 1968, visiting Berkeley.

Berkeley...Unite! Resist!

Strassen: Divide!

Strassen

Matrix multiplication.

Strassen, 1968, visiting Berkeley.

Berkeley...Unite! Resist!

Strassen: Divide! conquer!

Matrix Multiplication

X and Y are $n \times n$ matrices.

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

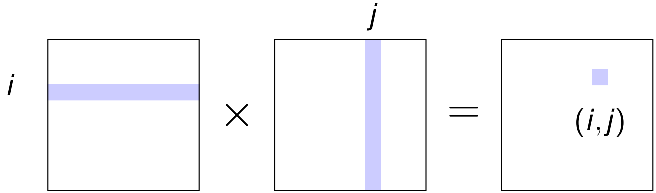
Z_{ij} is dot product of ith row with jth column.

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Z_{ij} is dot product of ith row with jth column.

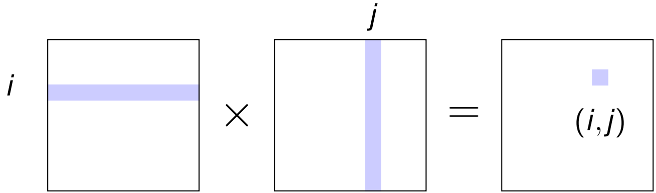


Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Z_{ij} is dot product of ith row with jth column.



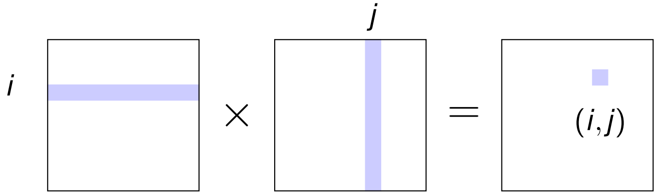
$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Z_{ij} is dot product of ith row with jth column.



$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

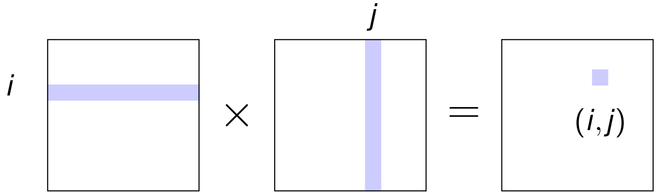
Runtime?

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Z_{ij} is dot product of ith row with jth column.



$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

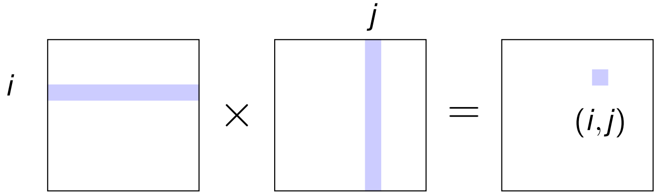
Runtime? $O(n^2)$?

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Z_{ij} is dot product of ith row with jth column.



$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

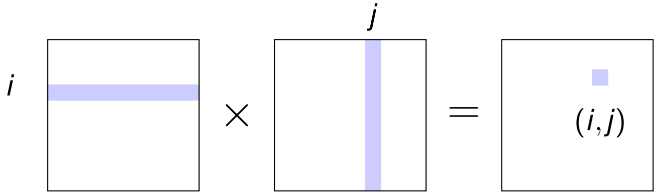
Runtime? $O(n^2)$? $O(n^3)$? ....

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Z_{ij} is dot product of ith row with jth column.



$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

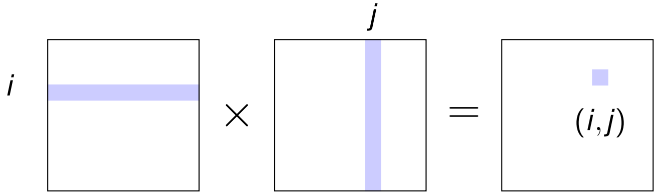
Runtime? $O(n^2)$? $O(n^3)$? ....$O(n^3)$!

Matrix Multiplication

X and Y are $n \times n$ matrices.

$$Z = XY,$$

Z_{ij} is dot product of ith row with jth column.



$$Z_{ij} = \sum_{k=1}^{n} X_{ik} Y_{kj}.$$

Runtime? $O(n^2)$? $O(n^3)$? ....$O(n^3)$!

Divide and Conquer

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Divide and Conquer

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

$A, B, C, \ldots, H$ are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Divide and Conquer

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$A, B, C, \ldots, H$ are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Subproblems?

Divide and Conquer

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

$A, B, C, \ldots, H$ are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Subproblems? $AE, BG, AF, BH, CE, DG, CF, DH$.

Divide and Conquer

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$A, B, C, \ldots, H$ are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Subproblems? $AE, BG, AF, BH, CE, DG, CF, DH$.

Recurrence?

Divide and Conquer

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

$A, B, C, \ldots, H$ are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Subproblems? $AE, BG, AF, BH, CE, DG, CF, DH$.

Recurrence?

$$T(n) = 8T(\frac{n}{2}) +$$

Divide and Conquer

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{bmatrix}$$

$A, B, C, \ldots, H$ are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Subproblems? $AE, BG, AF, BH, CE, DG, CF, DH$.

Recurrence?

$$T(n) = 8T(\frac{n}{2}) + O(n^2).$$

Divide and Conquer

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

$A, B, C, \ldots, H$ are $\frac{n}{2} \times \frac{n}{2}$ matrices.

Subproblems? $AE, BG, AF, BH, CE, DG, CF, DH$.

Recurrence?

$$T(n) = 8T(\frac{n}{2}) + O(n^2).$$

Masters: $O(n^{\log_2 8}) = O(n^3)$.

Strassen

Compute

$$P_1 = A(F - H) \quad P_5 = (A + D)(E + H)$$
$$P_2 = (A + B)H \quad P_6 = (B - D)(G + H)$$
$$P_3 = (C + D)E \quad P_7 = (A - C)(E + F)$$
$$P_4 = D(G - E)$$

Strassen

Compute

$$P_1 = A(F-H) \quad P_5 = (A+D)(E+H)$$
$$P_2 = (A+B)H \quad P_6 = (B-D)(G+H)$$
$$P_3 = (C+D)E \quad P_7 = (A-C)(E+F)$$
$$P_4 = D(G-E)$$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

Strassen

Compute

$$P_1 = A(F-H) \quad P_5 = (A+D)(E+H)$$
$$P_2 = (A+B)H \quad P_6 = (B-D)(G+H)$$
$$P_3 = (C+D)E \quad P_7 = (A-C)(E+F)$$
$$P_4 = D(G-E)$$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

7 multiplies!

Strassen

Compute

$$P_1 = A(F-H) \quad P_5 = (A+D)(E+H)$$
$$P_2 = (A+B)H \quad P_6 = (B-D)(G+H)$$
$$P_3 = (C+D)E \quad P_7 = (A-C)(E+F)$$
$$P_4 = D(G-E)$$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

Strassen

Compute

$$\begin{array}{ll} P_1 = A(F-H) & P_5 = (A+D)(E+H) \\ P_2 = (A+B)H & P_6 = (B-D)(G+H) \\ P_3 = (C+D)E & P_7 = (A-C)(E+F) \\ P_4 = D(G-E) & \end{array}$$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$$T(n) = 7T(\tfrac{n}{2}) + O(n^2)$$

Strassen

Compute

$P_1 = A(F-H) \quad P_5 = (A+D)(E+H)$
$P_2 = (A+B)H \quad P_6 = (B-D)(G+H)$
$P_3 = (C+D)E \quad P_7 = (A-C)(E+F)$
$P_4 = D(G-E)$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$T(n) = 7T(\frac{n}{2}) + O(n^2)$

From Masters:
(A) $O(n^2)$?

Strassen

Compute

$P_1 = A(F-H) \quad P_5 = (A+D)(E+H)$
$P_2 = (A+B)H \quad P_6 = (B-D)(G+H)$
$P_3 = (C+D)E \quad P_7 = (A-C)(E+F)$
$P_4 = D(G-E)$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$T(n) = 7T(\frac{n}{2}) + O(n^2)$

From Masters:
(A) $O(n^2)$? (B) $O(n^{\log_2 7}\log n)$?

Strassen

Compute

$$P_1 = A(F-H) \quad P_5 = (A+D)(E+H)$$
$$P_2 = (A+B)H \quad P_6 = (B-D)(G+H)$$
$$P_3 = (C+D)E \quad P_7 = (A-C)(E+F)$$
$$P_4 = D(G-E)$$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$$T(n) = 7T(\tfrac{n}{2}) + O(n^2)$$

From Masters:
(A) $O(n^2)$? (B) $O(n^{\log_2 7}\log n)$? (C) $T(n) = O(n^{\log_2 7})$?

Strassen

Compute

$P_1 = A(F - H) \quad P_5 = (A + D)(E + H)$
$P_2 = (A + B)H \quad P_6 = (B - D)(G + H)$
$P_3 = (C + D)E \quad P_7 = (A - C)(E + F)$
$P_4 = D(G - E)$

$$\begin{bmatrix} AE + BG = P_5 + P_4 - P_2 + P_6 & AF + BH = P_1 + P_2 \\ CE + DG = P_3 + P_4 & AF + BH = P_1 + P_5 - P_3 + P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$T(n) = 7T(\frac{n}{2}) + O(n^2)$

From Masters:
(A) $O(n^2)$? (B) $O(n^{\log_2 7} \log n)$? (C) $T(n) = O(n^{\log_2 7})$?

Leaf subproblems dominate runtime!

Strassen

Compute

$P_1 = A(F - H) \quad P_5 = (A + D)(E + H)$

$P_2 = (A + B)H \quad P_6 = (B - D)(G + H)$

$P_3 = (C + D)E \quad P_7 = (A - C)(E + F)$

$P_4 = D(G - E)$

$$\begin{bmatrix} AE + BG = P_5 + P_4 - P_2 + P_6 & AF + BH = P_1 + P_2 \\ CE + DG = P_3 + P_4 & AF + BH = P_1 + P_5 - P_3 + P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$T(n) = 7T(\frac{n}{2}) + O(n^2)$

From Masters:

(A) $O(n^2)$? (B) $O(n^{\log_2 7} \log n)$? (C) $T(n) = O(n^{\log_2 7})$?

Leaf subproblems dominate runtime!

(C) $O(n^{\log_2 7})$

Strassen

Compute

$P_1 = A(F-H) \quad P_5 = (A+D)(E+H)$
$P_2 = (A+B)H \quad P_6 = (B-D)(G+H)$
$P_3 = (C+D)E \quad P_7 = (A-C)(E+F)$
$P_4 = D(G-E)$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$T(n) = 7T(\frac{n}{2}) + O(n^2)$

From Masters:
(A) $O(n^2)$? (B) $O(n^{\log_2 7}\log n)$? (C) $T(n) = O(n^{\log_2 7})$?

Leaf subproblems dominate runtime!

(C) $O(n^{\log_2 7}) = O(n^{2.81\ldots})$

Strassen

Compute

$P_1 = A(F-H) \quad P_5 = (A+D)(E+H)$
$P_2 = (A+B)H \quad P_6 = (B-D)(G+H)$
$P_3 = (C+D)E \quad P_7 = (A-C)(E+F)$
$P_4 = D(G-E)$

$$\begin{bmatrix} AE+BG = P_5+P_4-P_2+P_6 & AF+BH = P_1+P_2 \\ CE+DG = P_3+P_4 & AF+BH = P_1+P_5-P_3+P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$T(n) = 7T(\frac{n}{2}) + O(n^2)$

From Masters:
(A) $O(n^2)$? (B) $O(n^{\log_2 7} \log n)$? (C) $T(n) = O(n^{\log_2 7})$?

Leaf subproblems dominate runtime!

(C) $O(n^{\log_2 7}) = O(n^{2.81\cdots})$ Way better than $O(n^3)$.

Strassen

Compute

$P_1 = A(F - H) \quad P_5 = (A + D)(E + H)$
$P_2 = (A + B)H \quad P_6 = (B - D)(G + H)$
$P_3 = (C + D)E \quad P_7 = (A - C)(E + F)$
$P_4 = D(G - E)$

$$\begin{bmatrix} AE + BG = P_5 + P_4 - P_2 + P_6 & AF + BH = P_1 + P_2 \\ CE + DG = P_3 + P_4 & AF + BH = P_1 + P_5 - P_3 + P_7 \end{bmatrix}$$

7 multiplies! Recurrence?

$T(n) = 7T(\frac{n}{2}) + O(n^2)$

From Masters:
(A) $O(n^2)$? (B) $O(n^{\log_2 7} \log n)$? (C) $T(n) = O(n^{\log_2 7})$?

Leaf subproblems dominate runtime!

(C) $O(n^{\log_2 7}) = O(n^{2.81\cdots})$ Way better than $O(n^3)$.

Commonly used in practice!

Current State of the Art: Matrix multiplication.

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^ω multiplications where $\omega = 2.36...$.

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^{ω} multiplications where $\omega = 2.36...$.

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^{ω} multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^ω multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^\omega})$

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^{ω} multiplications where $\omega = 2.36...$.

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^{\omega}}) = O(n^{\omega \log_k k})$

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^{ω} multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^{\omega}}) = O(n^{\omega \log_k k}) = O(n^{\omega})$

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^{ω} multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^{\omega}}) = O(n^{\omega \log_k k}) = O(n^{\omega})$

State of the art: k is very very large...

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^ω multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^\omega}) = O(n^{\omega \log_k k}) = O(n^\omega)$

State of the art: k is very very large... e.g., 10^{100} ...

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^ω multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^\omega}) = O(n^{\omega \log_k k}) = O(n^\omega)$

State of the art: k is very very large... e.g., 10^{100} ...but still a constant.

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^{ω} multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^{\omega}}) = O(n^{\omega \log_k k}) = O(n^{\omega})$

State of the art: k is very very large... e.g., 10^{100} ...but still a constant.

Based on complicated recursive constructions.

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^{ω} multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^{\omega}}) = O(n^{\omega \log_k k}) = O(n^{\omega})$

State of the art: k is very very large... e.g., 10^{100} ...but still a constant.

Based on complicated recursive constructions.

Improvement for constant + recursion gives better algorithm!

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^ω multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^\omega}) = O(n^{\omega \log_k k}) = O(n^\omega)$

State of the art: k is very very large... e.g., 10^{100} ...but still a constant.

Based on complicated recursive constructions.

Improvement for constant + recursion gives better algorithm!

Example:
Gauss + recursion $\implies$ faster multiplication.

Current State of the Art: Matrix multiplication.

$k \times k$ multiplication in k^ω multiplications where $\omega = 2.36....$

E.g., Strassen: 2×2 multiplication in $2^{\log_2 7} = 7$ multiplications.

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

Masters: $O(n^{\log_k k^\omega}) = O(n^{\omega \log_k k}) = O(n^\omega)$

State of the art: k is very very large... e.g., 10^{100} ...but still a constant.

Based on complicated recursive constructions.

Improvement for constant + recursion gives better algorithm!

Example:
Gauss + recursion $\implies$ faster multiplication.
Strassen's 7 multiplies + recursion $\implies$ faster matrix multiplication.

Lecture in five!

Gauss plus recursion is magic!

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem:

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

diminishing by b

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

diminishing by b

working by $O(f(n))$.

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \to O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Recursive (Divide and Conquer) Multiplication:

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Recursive (Divide and Conquer) Multiplication:

8 subroutine calls of size $n/2 \times n/2$

Lecture in five!

Gauss plus recursion is magic!
$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$
Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!
Branching by a
diminishing by b
working by $O(f(n))$.
Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Recursive (Divide and Conquer) Multiplication:
8 subroutine calls of size $n/2 \times n/2$
$\rightarrow O(n^3)$.

Lecture in five!

Gauss plus recursion is magic!
$O(n^2) \to O(n^{\log_2 3}) \approx O(n^{1.58..})$
Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!
Branching by a
diminishing by b
working by $O(f(n))$.
Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Recursive (Divide and Conquer) Multiplication:
8 subroutine calls of size $n/2 \times n/2$
$\to O(n^3)$.
Strassen:

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \rightarrow O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Recursive (Divide and Conquer) Multiplication:

8 subroutine calls of size $n/2 \times n/2$

$\rightarrow O(n^3)$.

Strassen:

7 subroutine calls of size $n/2 \times n/2$

Lecture in five!

Gauss plus recursion is magic!

$O(n^2) \to O(n^{\log_2 3}) \approx O(n^{1.58..})$

Double size, time grows by a factor of 3.

Master's theorem: understand the recursion tree!

Branching by a

diminishing by b

working by $O(f(n))$.

Leaves: $n^{\log_b a}$, Work: $\sum_i a^i f(\frac{n}{b^i})$.

Recursive (Divide and Conquer) Multiplication:

8 subroutine calls of size $n/2 \times n/2$

$\to O(n^3)$.

Strassen:

7 subroutine calls of size $n/2 \times n/2$

$\to O(n^{\log_2 7}) \approx O(n^{2.8})$.