

COL 351 : ANALYSIS & DESIGN OF ALGORITHMS

LECTURE 6

DIVIDE & CONQUER V :

PROOF AND APPLICATIONS OF MASTER THEOREM

AUG 05, 2026

|

ROHIT VAISH

MASTER METHOD

Black-box for determining the running time of recursive algorithms

RECALL : INTEGER MULTIPLICATION

RECALL : INTEGER MULTIPLICATION

Grade-school multiplication

$$\Theta(n^2)$$

RECALL : INTEGER MULTIPLICATION

Grade-school multiplication

$$\Theta(n^2)$$

Recursive algorithm

$$[x \cdot y = 10^n ac + 10^{n/2}(ad + bc) + bd]$$

RECALL : INTEGER MULTIPLICATION

Grade-school multiplication

$$\Theta(n^2)$$

Recursive algorithm

$$[x \cdot y = 10^n ac + 10^{n/2}(ad + bc) + bd]$$

$T(n)$

|
maximum number of operations

the algorithm needs to multiply

two n -digit numbers

RECALL : INTEGER MULTIPLICATION

Grade-school multiplication

$$\Theta(n^2)$$

Recursive algorithm

$$[x \cdot y = 10^n ac + 10^{n/2}(ad + bc) + bd]$$

$$T(n) \leq 4 \cdot T(n/2) + O(n)$$

RECALL : INTEGER MULTIPLICATION

Grade-school multiplication

$$\Theta(n^2)$$

Recursive algorithm

$$[n \cdot y = 10^n ac + 10^{n/2}(ad + bc) + bd]$$

$$T(n) \leq 4 \cdot T(n/2) + O(n)$$

Karatsuba algorithm

$$[(a+b)(c+d) - ac - bd = ad + bc]$$

$$T(n) \leq 3 \cdot T(n/2) + O(n)$$

MASTER METHOD

Black-box for solving recurrences

Key assumption: All subproblems have the same size.

for unbalanced subproblems and more: Akra-Bazzi method

STANDARD RECURRENCE FORMAT

STANDARD RECURRENCE FORMAT

Base case:

General case:

STANDARD RECURRENCE FORMAT

Base case: $T(n) \leq \text{constant}$ for all sufficiently small n

General case:

STANDARD RECURRENCE FORMAT

Base case: $T(n) \leq \text{constant}$ for all sufficiently small n

General case: for all larger n

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + O(n^d)$$

STANDARD RECURRENCE FORMAT

Base case: $T(n) \leq \text{constant}$ for all sufficiently small n

General case: for all larger n

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + O(n^d)$$

a : number of recursive calls (≥ 1)

b : factor by which input size shrinks (> 1)

d : exponent in running time of "combine" step (≥ 0)

STANDARD RECURRENCE FORMAT

Base case: $T(n) \leq \text{constant}$ for all sufficiently small n

General case: for all larger n

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + O(n^d)$$

independent
of n

- a : number of recursive calls (≥ 1)
- b : factor by which input size shrinks (> 1)
- d : exponent in running time of "combine" step (≥ 0)

MASTER THEOREM

Theorem: If $T(n) \leq a T\left(\frac{n}{b}\right) + O(n^d)$, then

MASTER THEOREM

Theorem: If $T(n) \leq a T\left(\frac{n}{b}\right) + O(n^d)$, then

$$T(n) = \left\{ \begin{array}{l} \end{array} \right.$$

Case 1

Case 2

Case 3

MASTER THEOREM

Theorem: If $T(n) \leq a T\left(\frac{n}{b}\right) + O(n^d)$, then

$$T(n) = \begin{cases} \text{if } a = b^d & \text{Case 1} \\ \text{if } a < b^d & \text{Case 2} \\ \text{if } a > b^d & \text{Case 3} \end{cases}$$

MASTER THEOREM

Theorem: If $T(n) \leq a T\left(\frac{n}{b}\right) + O(n^d)$, then

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d & \text{Case 1} \\ O(n^d) & \text{if } a < b^d & \text{Case 2} \\ O(n^{\log_b a}) & \text{if } a > b^d & \text{Case 3} \end{cases}$$

MASTER THEOREM

Theorem: If $T(n) \leq a T(\frac{n}{b}) + O(n^d)$, then

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d & \text{Case 1} \\ O(n^d) & \text{if } a < b^d & \text{Case 2} \\ O(n^{\log_b a}) & \text{if } a > b^d & \text{Case 3} \end{cases}$$

only upper bounds

EXAMPLES

EXAMPLES

Merge Sort

EXAMPLES

Merge Sort

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

maximum number of operations
the algorithm needs for an
input of size n

EXAMPLES

Merge Sort

$$a=2, b=2, d=1$$

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

EXAMPLES

Merge Sort

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

$$a=2, b=2, d=1$$

(Case 1)

$$\text{Recall } T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d & \text{Case 1} \\ O(n^d) & \text{if } a < b^d & \text{Case 2} \\ O(n^{\log_b a}) & \text{if } a > b^d & \text{Case 3} \end{cases}$$

EXAMPLES

Merge Sort

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

$$a=2, b=2, d=1$$

(Case 1)

$$O(n \log n)$$

BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"

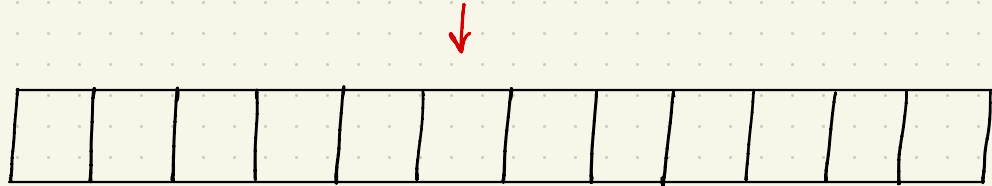


BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"

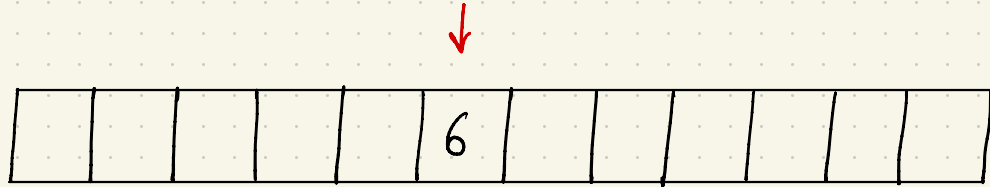


BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"

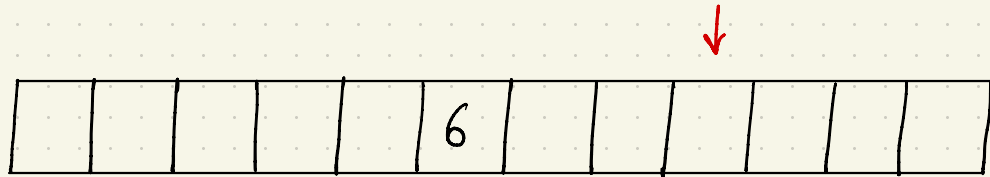


BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"

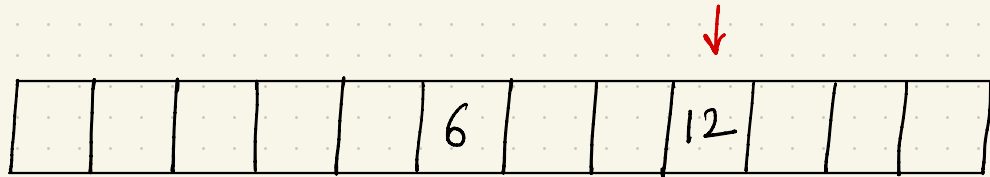


BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"

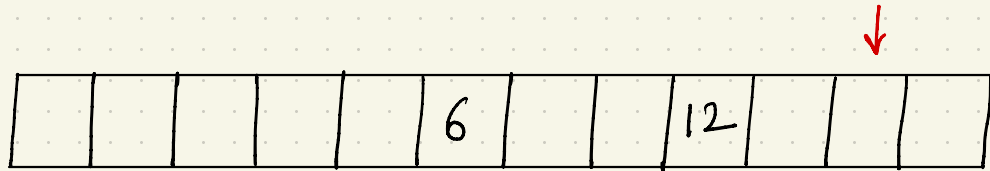


BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"

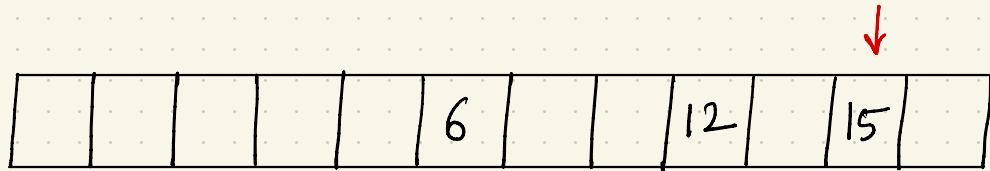


BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"



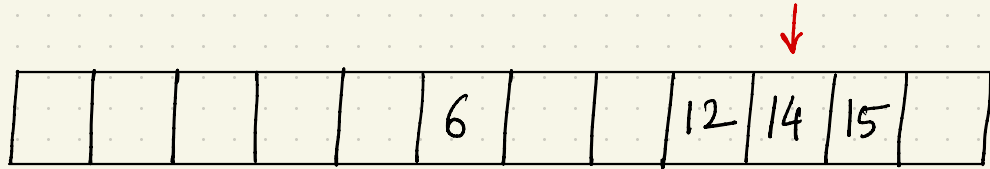
					6			12	15	
--	--	--	--	--	---	--	--	----	----	--

BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"



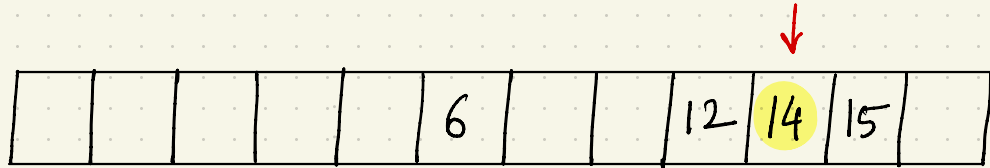
					6			12	14	15	
--	--	--	--	--	---	--	--	----	----	----	--

BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

Look for "14"



					6			12	14	15	
--	--	--	--	--	---	--	--	----	----	----	--

BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

initialize $l := 0$

initialize $h := n-1$

initialize $m := \left\lfloor \frac{l+h}{2} \right\rfloor$

BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

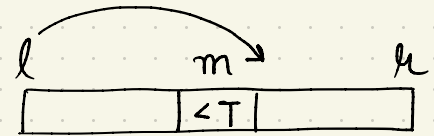
initialize $l := 0$

initialize $h := n-1$

initialize $m := \left\lfloor \frac{l+h}{2} \right\rfloor$

if $A[m] < T$

recuse with $l := m+1$



BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

initialize $l := 0$

initialize $h := n-1$

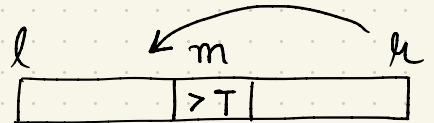
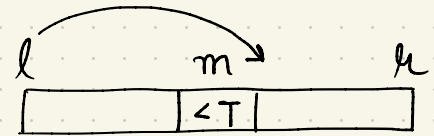
initialize $m := \left\lfloor \frac{l+h}{2} \right\rfloor$

if $A[m] < T$

recuse with $l := m+1$

else if $A[m] > T$

recuse with $h := m-1$



BINARY SEARCH

input: a sorted array A of length n and a target T

output: index of T in A (if exists)

initialize $l := 0$

initialize $h := n-1$

initialize $m := \left\lfloor \frac{l+h}{2} \right\rfloor$

if $A[m] < T$

recuse with $l := m+1$

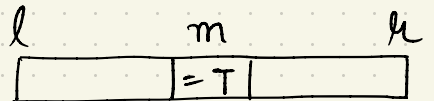
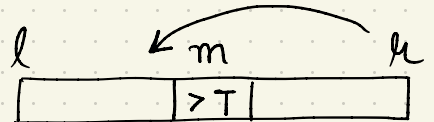
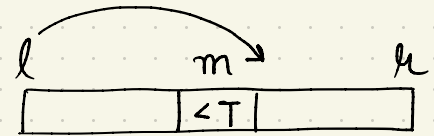
else if $A[m] > T$

recuse with $h := m-1$

else

return m

return No



EXAMPLES

Merge Sort

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

$$a=2, b=2, d=1$$

(Case 1)

$$O(n \log n)$$

Binary Search

$$T(n) \leq T\left(\frac{n}{2}\right) + O(1)$$

$$a=1, b=2, d=0$$

EXAMPLES

Merge Sort

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

$$a=2, b=2, d=1$$

(Case 1)

$$O(n \log n)$$

Binary Search

$$T(n) \leq T\left(\frac{n}{2}\right) + O(1)$$

$$a=1, b=2, d=0$$

(Case 1)

$$O(\log n)$$

EXAMPLES

Merge Sort

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

$$a=2, b=2, d=1$$

(Case 1)

$$O(n \log n)$$

Binary Search

$$T(n) \leq T\left(\frac{n}{2}\right) + O(1)$$

$$a=1, b=2, d=0$$

(Case 1)

$$O(\log n)$$

don't need to read
the entire input to
solve the problem

RECURSIVE ALGORITHM FOR INTEGER MULTIPLICATION

input: two n digit numbers x and y

output: the product $x \cdot y$

if $n = 1$

return $x \cdot y$

else

write $x = 10^{n/2} \cdot a + b$ and $y = 10^{n/2} \cdot c + d$

return $10^n ac + 10^{n/2} (ad + bc) + bd$

EXAMPLES

Recursive integer multⁿ

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

EXAMPLES

Recursive integer multⁿ

$$a = 4, b = 2, d = 1$$

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

EXAMPLES

Recursive integer multⁿ

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

$$a = 4, b = 2, d = 1$$

(Case 3)

$$\text{Recall } T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d & \text{Case 1} \\ O(n^d) & \text{if } a < b^d & \text{Case 2} \\ O(n^{\log_b a}) & \text{if } a > b^d & \text{Case 3} \end{cases}$$

EXAMPLES

Recursive integer multⁿ

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

$$a = 4, b = 2, d = 1$$

(Case 3)

$$O(n^{\log_2 4}) = O(n^2)$$

Same as
grade-school algorithm

KARATSUBA MULTIPLICATION

① compute $a \cdot c = 56 \times 12 = 672$

② compute $b \cdot d = 78 \times 34 = 2652$

③ compute $(a+b) \cdot (c+d) = 134 \times 46 = 6164$

④ compute ③ - ② - ① = 2840

⑤ Compute

672	0000	from ①
2652		from ②
2840	00	from ④
7006652		= $x \cdot y$

$x = \begin{matrix} & a & \\ \text{5} & 6 & \\ & c & \end{matrix} \quad \begin{matrix} & b & \\ \text{7} & 8 & \\ & d & \end{matrix}$

$y = \begin{matrix} \text{1} & 2 & \\ & c & \end{matrix} \quad \begin{matrix} \text{3} & 4 & \\ & d & \end{matrix}$

EXAMPLES

Recursive integer multⁿ

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

$$a = 4, b = 2, d = 1$$

(Case 3)

$$O(n^{\log_2 4}) = O(n^2)$$

Same as
grade-school algorithm

Karatsuba multⁿ

$$T(n) \leq 3T\left(\frac{n}{2}\right) + O(n)$$

$$a = 3, b = 2, d = 0$$

EXAMPLES

Recursive integer multⁿ

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

$$a=4, b=2, d=1$$

(Case 3)

$$O\left(n^{\log_2 4}\right) = O(n^2)$$

Same as
grade-school algorithm

Karatsuba multⁿ

$$T(n) \leq 3T\left(\frac{n}{2}\right) + O(n)$$

$$a=3, b=2, d=0$$

(Case 3)

$$O\left(n^{\log_2 3}\right) = O(n^{1.59})$$

faster!

EXAMPLES

Recursive integer multⁿ

$$T(n) \leq 4T\left(\frac{n}{2}\right) + O(n)$$

$$a=4, b=2, d=1$$

(Case 3)

$$O\left(n^{\log_2 4}\right) = O(n^2)$$

Same as
grade-school algorithm

Karatsuba multⁿ

$$T(n) \leq 3T\left(\frac{n}{2}\right) + O(n)$$

$$a=3, b=2, d=0$$

(Case 3)

$$O\left(n^{\log_2 3}\right) = O(n^{1.59})$$

faster!

Python switches from grade-school to karatsuba at $n=70$.

RECURSIVE MATRIX MULTIPLICATION

$$X = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right]_{n \times n}$$

$$Y = \left[\begin{array}{c|c} E & F \\ \hline G & H \end{array} \right]_{n \times n}$$

Step 1: Recursively compute the eight matrix products

$$X \cdot Y = \left[\begin{array}{c|c} A \cdot E + B \cdot G & A \cdot F + B \cdot H \\ \hline C \cdot E + D \cdot G & C \cdot F + D \cdot H \end{array} \right]$$

Step 2: Do the necessary additions ($\Theta(n^2)$ time)

EXAMPLES

Recursive matrix multⁿ

$$a=8, b=2, d=2$$

$$T(n) \leq 8T\left(\frac{n}{2}\right) + O(n^2)$$

EXAMPLES

Recursive matrix multⁿ

$$T(n) \leq 8T\left(\frac{n}{2}\right) + O(n^2)$$

$$a=8, b=2, d=2$$

(Case 3)

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

Same as the
Strassen and-divide algorithm

STRASSEN'S ALGORITHM

Step 1: Recursively compute only seven matrix products

Step 2: Do the necessary additions and subtractions ($\Theta(n^2)$ time)

$$P_1 = A \cdot (F - H)$$

$$P_2 = (A + B) \cdot H$$

$$P_3 = (C + D) \cdot E$$

$$P_4 = D \cdot (G - E)$$

$$P_5 = (A + D) \cdot (E + H)$$

$$P_6 = (B - D) \cdot (G + H)$$

$$P_7 = (A - C) \cdot (E + F)$$

$$X \cdot Y = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

EXAMPLES

Recursive matrix multⁿ

$$T(n) \leq 8T\left(\frac{n}{2}\right) + O(n^2)$$

$$a=8, b=2, d=2$$

(Case 3)

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

same as the
- and-divide algorithm

Strassen's algorithm

$$T(n) \leq 7T\left(\frac{n}{2}\right) + O(n^2)$$

$$a=7, b=2, d=2$$

(Case 3)

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

faster!

EXAMPLES

Fictitious recurrence

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

EXAMPLES

Fictitious recurrence

$$a = 2, b = 2, d = 2$$

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

$$\text{Recall } T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d & \text{Case 1} \\ O(n^d) & \text{if } a < b^d & \text{Case 2} \\ O(n^{\log_b a}) & \text{if } a > b^d & \text{Case 3} \end{cases}$$

EXAMPLES

Fictitious recurrence

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

$$a = 2, b = 2, d = 2$$

(Case 2)

$$O(n^2)$$

$$\text{Recall } T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d & \text{Case 1} \\ O(n^d) & \text{if } a < b^d & \text{Case 2} \\ O(n^{\log_b a}) & \text{if } a > b^d & \text{Case 3} \end{cases}$$

PROOF OF MASTER THEOREM

MASTER THEOREM

Theorem: If $T(n) \leq a T\left(\frac{n}{b}\right) + O(n^d)$, then

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d & \text{Case 1} \\ O(n^d) & \text{if } a < b^d & \text{Case 2} \\ O(n^{\log_b a}) & \text{if } a > b^d & \text{Case 3} \end{cases}$$

Why three cases?

Three different types of recursion trees.

Formally,

Base case: $T(1) = c$

General case: for $n > 1$

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

Formally,

Base case: $T(1) = c$

can assume same constant

General case: for $n > 1$

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

Formally,

Base case: $T(1) = c$

General case: for $n > 1$ — can be $n > n_0$ in general

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

Formally,

Base case: $T(1) = c$

General case: for $n > 1$

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

will assume that n
is a power of b
($n = b^k$ for some k)

Formally,

Base case: $T(1) = c$

General case: for $n > 1$

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

High level idea: Analyze using recursion trees (like merge sort)

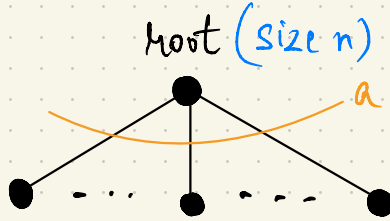
RECURSION TREE

RECURSION TREE

root (size n)
●

level 0 outmost call

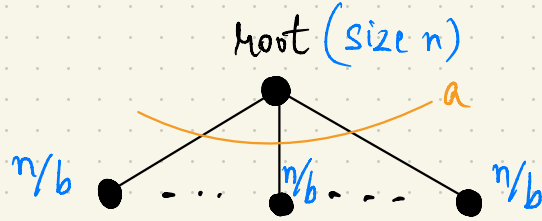
RECURSION TREE



level 0 outmost call

level 1 first batch of recursive
calls

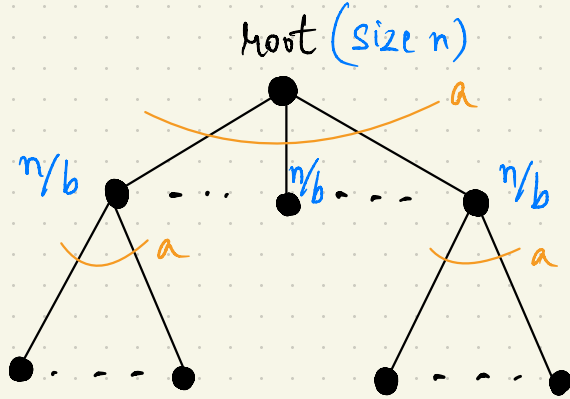
RECURSION TREE



level 0 outmost call

level 1 first batch of recursive calls

RECURSION TREE

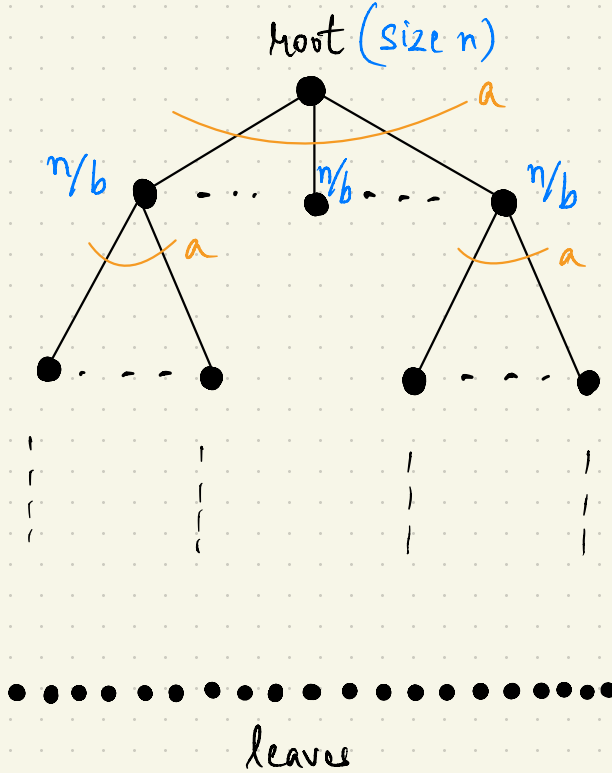


level 0 outmost call

level 1 first batch of recursive calls

level 2 second batch

RECURSION TREE



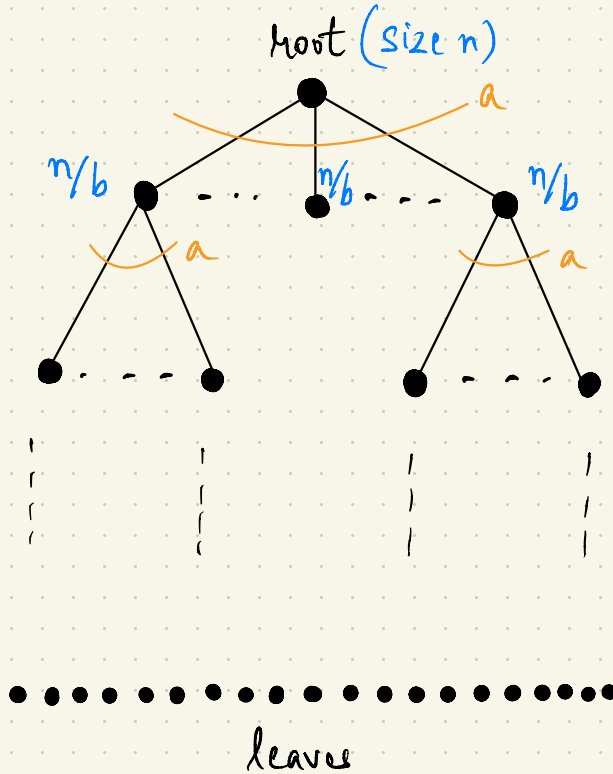
level 0 outmost call

level 1 first batch of recursive calls

level 2 second batch

level ? base cases

RECURSION TREE



level 0 outmost call

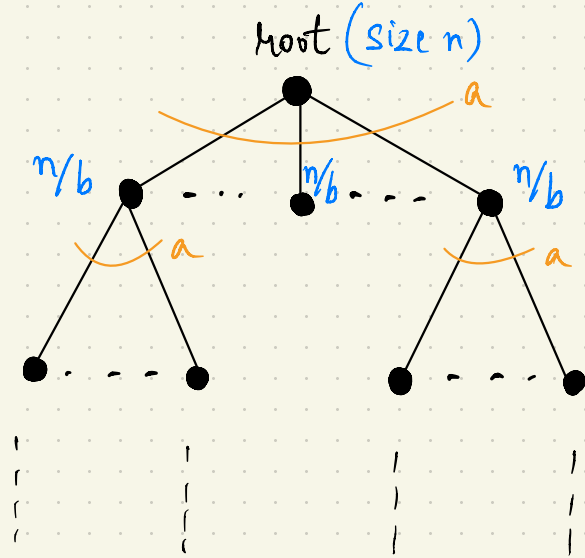
level 1 first batch of recursive calls

level 2 second batch

level $\log_b n$ base cases

└ # times n is divided by b to go below 1

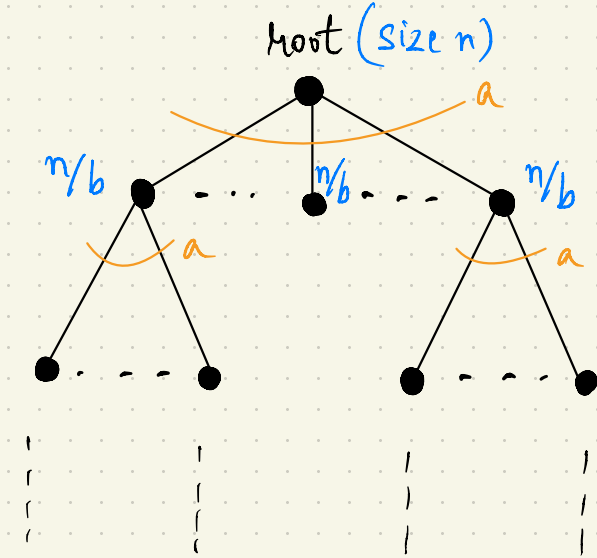
RECURSION TREE



At level j

subproblems =

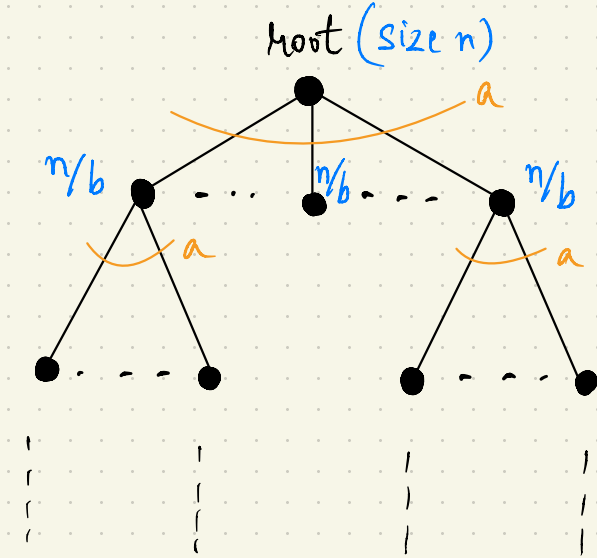
RECURSION TREE



At level j

$$\# \text{ subproblems} = a^j$$

RECURSION TREE

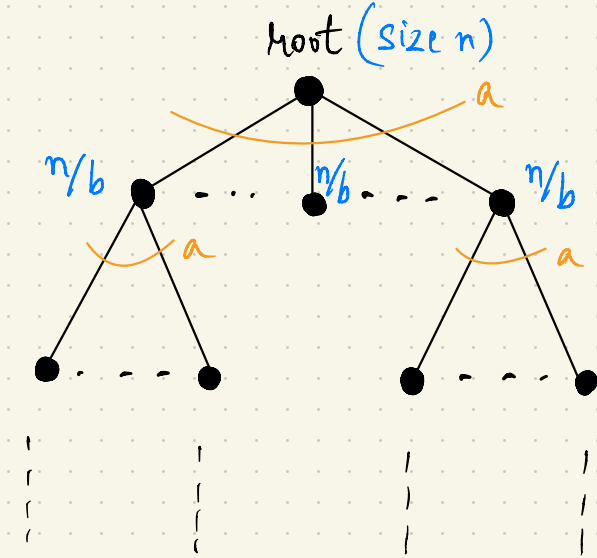


At level j

subproblems = a^j

subproblem size =

RECURSION TREE

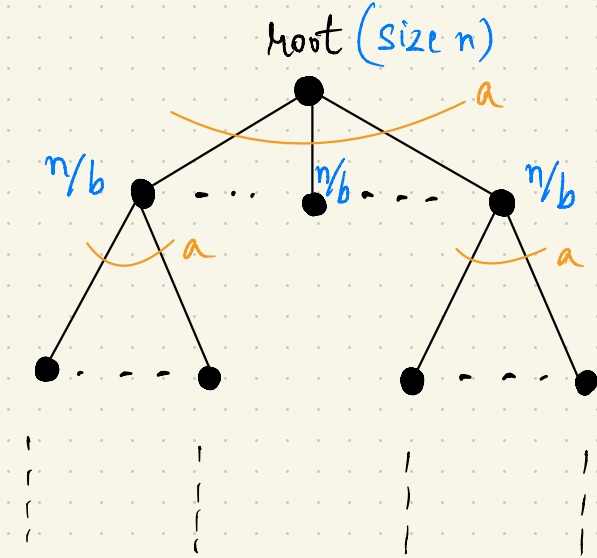


At level j

$$\# \text{ subproblems} = a^j$$

$$\text{subproblem size} = \frac{n}{b^j}$$

RECURSION TREE



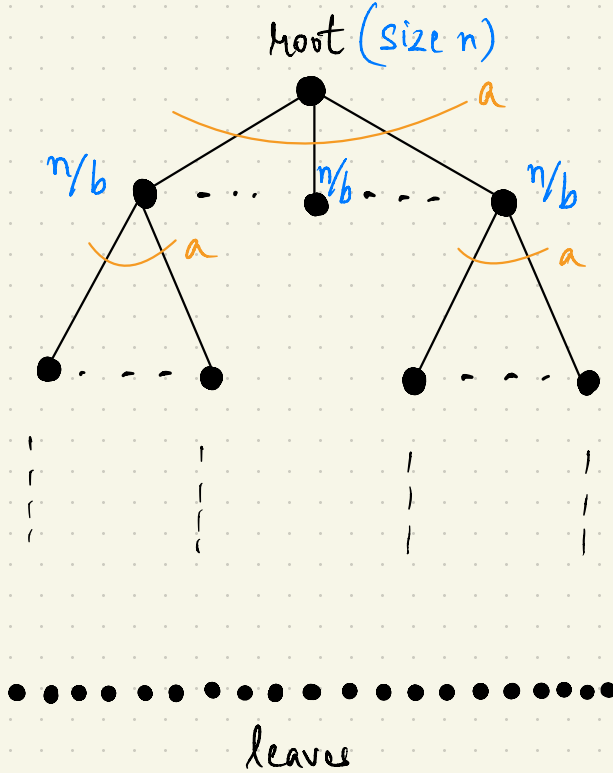
At level j

$$\# \text{ subproblems} = a^j$$

$$\text{subproblem size} = \frac{n}{b^j}$$

Total $\#$ levels =

RECURSION TREE



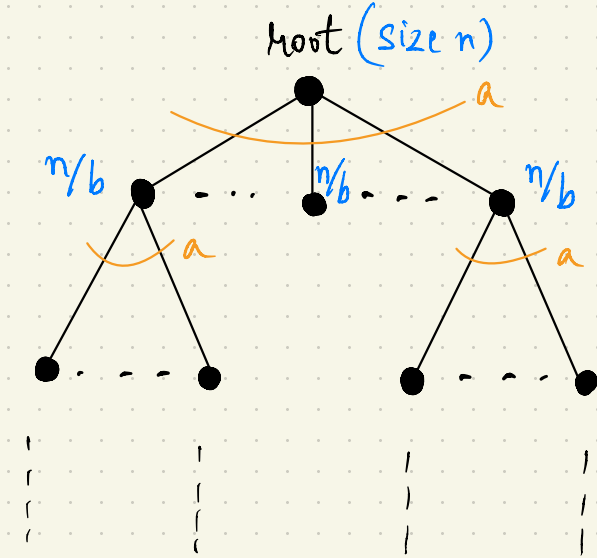
At level j

$$\# \text{ subproblems} = a^j$$

$$\text{subproblem size} = \frac{n}{b^j}$$

$$\text{Total \# levels} = 1 + \log_b n$$

RECURSION TREE

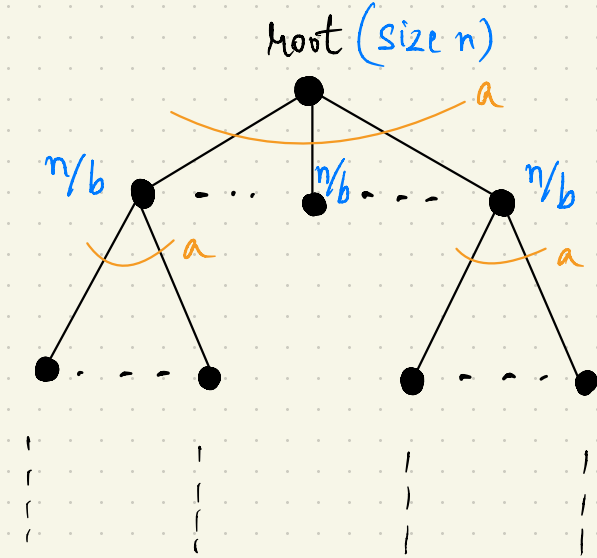


Total work at level j
(ignoring work in recursive calls)

=

.....
leaves

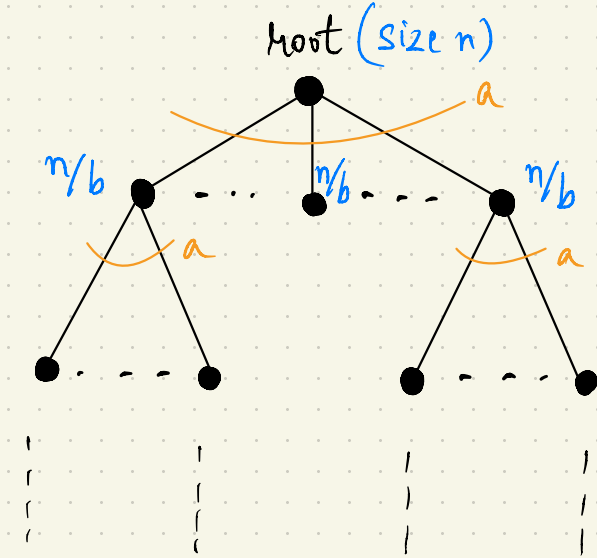
RECURSION TREE



Total work at level j
(ignoring work in recursive calls)

$$= \# \text{ subproblems at level } j \times \text{work done per subproblem}$$

RECURSION TREE

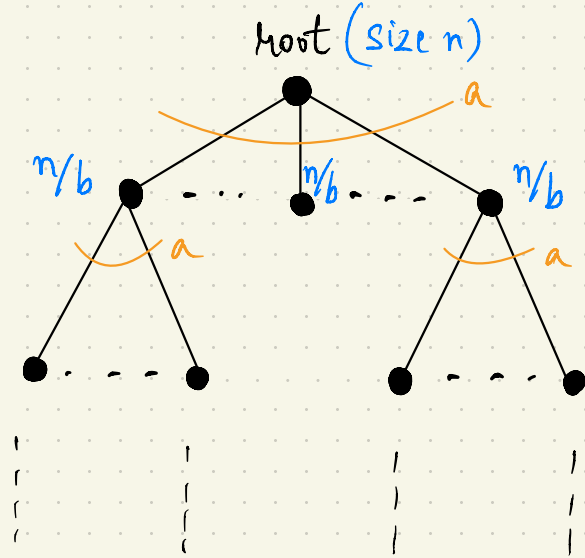


Total work at level j
(ignoring work in recursive calls)

$$= \# \text{ subproblems at level } j \times \text{work done per subproblem}$$

$$= a^j \times$$

RECURSION TREE



Total work at level j
(ignoring work in recursive calls)

$$= \# \text{ subproblems at level } j \times \text{work done per subproblem}$$

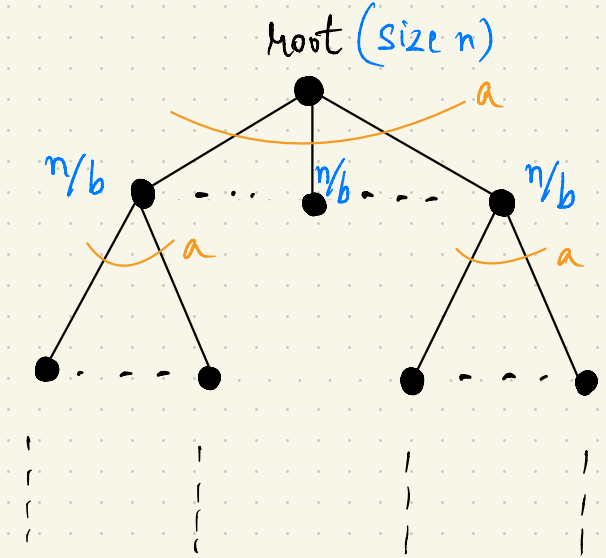
$$= a^j \times$$

.....
leaves

Recall *general case*

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

RECURSION TREE



Total work at level j
(ignoring work in recursive calls)

$$= \# \text{ subproblems at level } j \times \text{work done per subproblem}$$

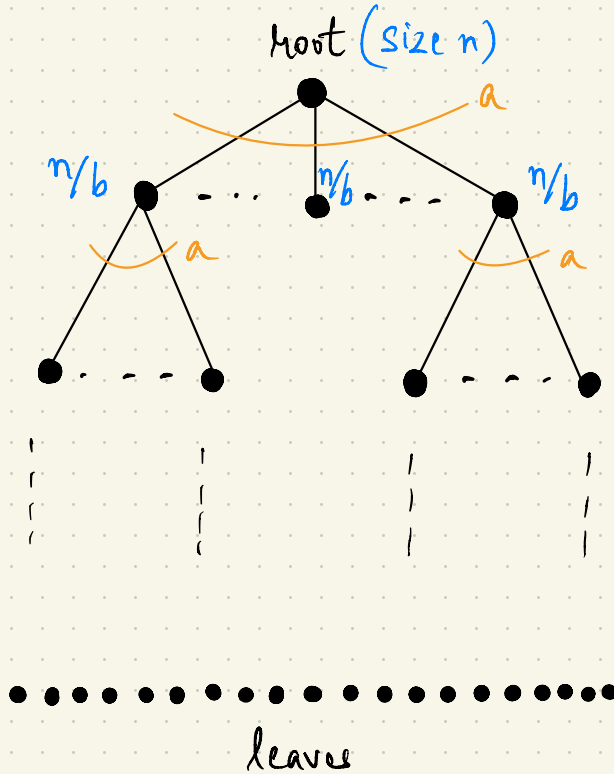
$$= a^j \times c \cdot \left(\frac{n}{b^j} \right)^d$$

.....
leaves

Recall *general case*

$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

RECURSION TREE



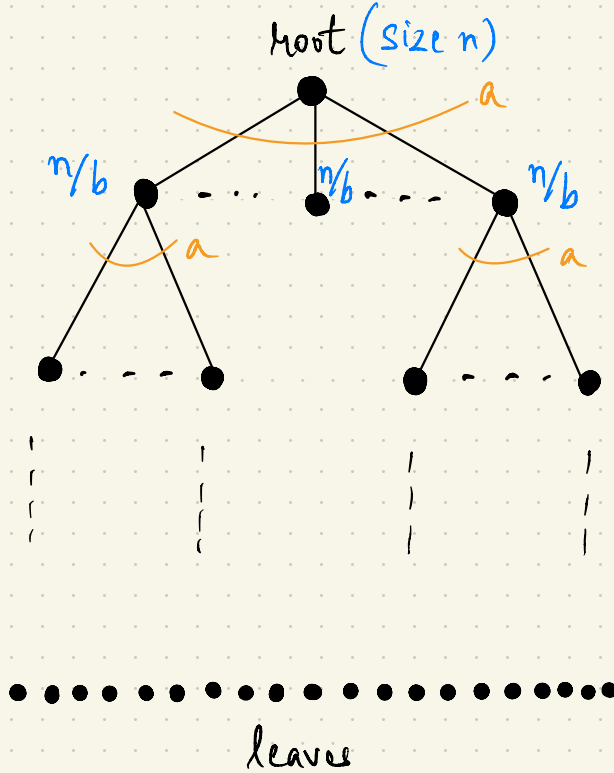
Total work at level j
(ignoring work in recursive calls)

$$= \# \text{ subproblems at level } j \times \text{work done per subproblem}$$

$$= a^j \times c \cdot \left(\frac{n}{b^j}\right)^d$$

$$= \underbrace{cn^d}_{\text{independent of } j} \cdot \underbrace{\left(\frac{a}{b^d}\right)^j}_{\text{depends on } j}$$

RECURSION TREE



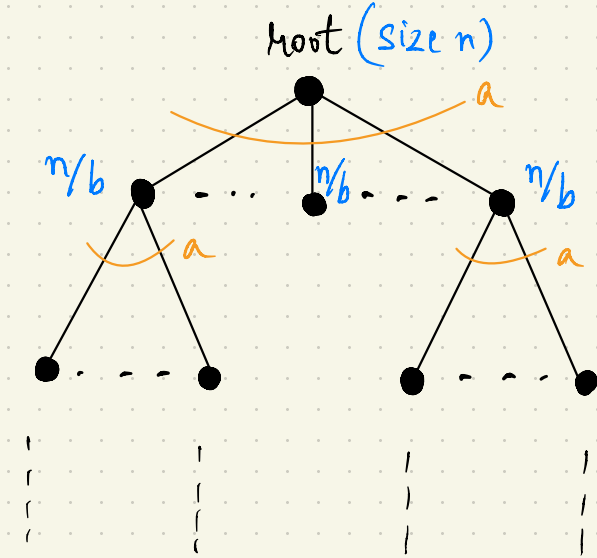
Total work at level j
(ignoring work in recursive calls)

$$= \# \text{ subproblems at level } j \times \text{work done per subproblem}$$

$$= a^j \times c \cdot \left(\frac{n}{b^j}\right)^d$$

$$= \underbrace{cn^d}_{\text{independent of } j} \cdot \underbrace{\left(\frac{a}{b^d}\right)^j}_{\text{depends on } j}$$

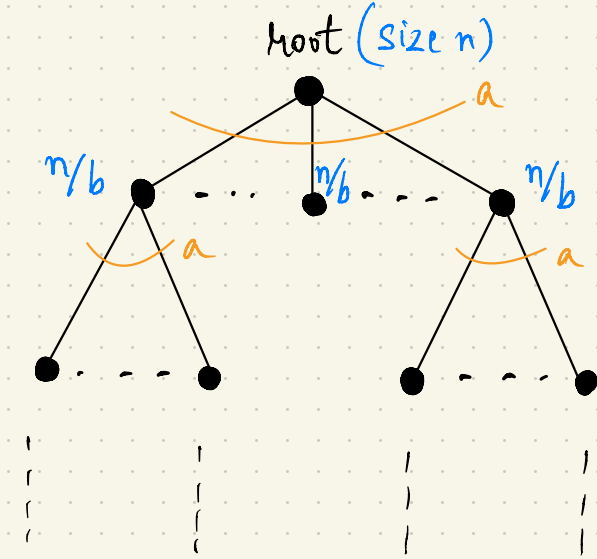
RECURSION TREE



Total work across all levels

$$\leq c n^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d} \right)^j$$

RECURSION TREE



Total work across all levels

$$\leq c n^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d} \right)^j$$

— 😊

INTERPRETING



INTERPRETING 😊

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

INTERPRETING 😊

Recall : Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

a : rate of problem proliferation

b : factor by which input size shrinks

d : exponent in running time of "combine" step

INTERPRETING 😊

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

a : rate of problem proliferation

b : factor by which input size shrinks

d : exponent in running time of "combine" step

b^d : rate of work shrinkage per subproblem

INTERPRETING 😊

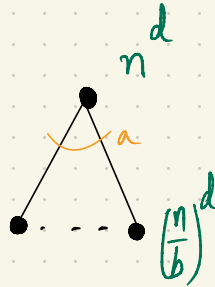
Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

a : rate of problem proliferation

b : factor by which input size shrinks

d : exponent in running time of "combine" step

b^d : rate of work shrinkage per subproblem



INTERPRETING ☺

Recall: Work done at level j = $c n^d \cdot \left(\frac{a}{b^d}\right)^j$

a : rate of problem proliferation

Tag of war between these
two opposing forces

b^d : rate of work shrinkage per subproblem

INTERPRETING 😊

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

$$a > b^d$$

$$a = b^d$$

INTERPRETING ☺

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

$$a > b^d$$

(proliferation > shrinkage)

$$a = b^d$$

(proliferation = shrinkage)

INTERPRETING 😊

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level ↓

$$a > b^d$$

(proliferation > shrinkage)

$$a = b^d$$

(proliferation = shrinkage)

INTERPRETING ☺

Recall: Work done at level j = $c n^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level ↓

$$a > b^d$$

(proliferation > shrinkage)

work done per level ↑

$$a = b^d$$

(proliferation = shrinkage)

INTERPRETING 😊

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level ↓

$$a > b^d$$

(proliferation > shrinkage)

work done per level ↑

$$a = b^d$$

(proliferation = shrinkage)

work done per level independent of j

FORECASTING RUNNING TIME BOUNDS

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level $\downarrow$

$$a > b^d$$

(proliferation > shrinkage)

work done per level $\uparrow$

$$a = b^d$$

(proliferation = shrinkage)

work done per level independent of j

FORECASTING RUNNING TIME BOUNDS

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level $\downarrow$

$$a > b^d$$

(proliferation > shrinkage)

work done per level $\uparrow$

$$a = b^d$$

(proliferation = shrinkage)

work done per level independent of j
e.g., merge sort; expect $O(n^d \log n)$

FORECASTING RUNNING TIME BOUNDS

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level $\downarrow$

most work at root; optimistically $O(n^d)$

$$a > b^d$$

(proliferation > shrinkage)

work done per level $\uparrow$

$$a = b^d$$

(proliferation = shrinkage)

work done per level independent of j

e.g., merge sort; expect $O(n^d \log n)$

FORECASTING RUNNING TIME BOUNDS

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level $\downarrow$

most work at root; optimistically $O(n^d)$

$$a > b^d$$

(proliferation > shrinkage)

work done per level $\uparrow$

most work at the leaves; expect $O(\# \text{leaves})$

$$a = b^d$$

(proliferation = shrinkage)

work done per level independent of j

e.g., merge sort; expect $O(n^d \log n)$

FORECASTING RUNNING TIME BOUNDS

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level $\downarrow$

most work at root; optimistically $O(n^d)$

$$a > b^d$$

(proliferation > shrinkage)

work done per level $\uparrow$

most work at the leaves; expect $O(\# \text{ leaves})$
 $= O(a^{\log_b n})$

$$a = b^d$$

(proliferation = shrinkage)

work done per level independent of j

e.g., merge sort; expect $O(n^d \log n)$

FORECASTING RUNNING TIME BOUNDS

Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

$$a < b^d$$

(proliferation < shrinkage)

work done per level $\downarrow$

most work at root; optimistically $O(n^d)$

$$a > b^d$$

(proliferation > shrinkage)

work done per level $\uparrow$

most work at the leaves; expect $O(\# \text{ leaves})$
 $= O(a^{\log_b n}) = O(n^{\log_b a})$

$$a = b^d$$

(proliferation = shrinkage)

work done per level independent of j

e.g., merge sort; expect $O(n^d \log n)$

THE CALCULATIONS

THE CALCULATIONS

$$\text{Total work done} \leq c n^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



THE CALCULATIONS

$$\text{Total work done} \leq cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 1 : $a = b^d$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 1 : $a = b^d$

$$\text{☺} = cn^d (\log_b n + 1)$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 1 : $a = b^d$

$$\text{☺} = cn^d (\log_b n + 1) = O(n^d \log n)$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

$$\text{Smiley face} =$$

THE CALCULATIONS

$$\text{Total work done} \leq c n^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

$$r = a/b^d < 1$$

$$k = \log_b n$$

=

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

$$r = a/b^d < 1$$

$$k = \log_b n$$

$$\text{Smiley face} = cn^d (1 + r + r^2 + \dots + r^k)$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

$$r = a/b^d < 1$$

$$k = \log_b n$$

$$\begin{aligned} \text{☺} &= cn^d (1 + r + r^2 + \dots + r^k) \\ &\leq cn^d (1 + r + r^2 + \dots + r^k + \dots) \end{aligned}$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

$$r = a/b^d < 1$$

$$k = \log_b n$$

$$\begin{aligned} \text{☺} &= cn^d (1 + r + r^2 + \dots + r^k) \\ &\leq cn^d (1 + r + r^2 + \dots + r^k + \dots) \\ &= cn^d \frac{1}{1-r} \end{aligned}$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

$$r = a/b^d < 1$$

$$k = \log_b n$$

$$\begin{aligned} \text{Smiley face} &= cn^d (1 + r + r^2 + \dots + r^k) \\ &\leq cn^d (1 + r + r^2 + \dots + r^k + \dots) \\ &= cn^d \underbrace{\frac{1}{1-r}}_{\text{constant}} \end{aligned}$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

$$r = a/b^d < 1$$
$$k = \log_b n$$

$$\begin{aligned} \text{Smiley face} &= cn^d (1 + r + r^2 + \dots + r^k) \\ &\leq cn^d (1 + r + r^2 + \dots + r^k + \dots) \\ &= cn^d \underbrace{\frac{1}{1-r}}_{\text{constant}} = O(n^d) \end{aligned}$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 2 : $a < b^d$

$$r = a/b^d < 1$$

$$k = \log_b n$$

dominated by root

$$\begin{aligned} \text{Smiley face} &= cn^d (1 + r + r^2 + \dots + r^k) \\ &\leq cn^d (1 + r + r^2 + \dots + r^k + \dots) \\ &= cn^d \underbrace{\frac{1}{1-r}}_{\text{constant}} = O(n^d) \end{aligned}$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

THE CALCULATIONS

$$\text{Total work done} \leq c n^d \cdot \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

$$\text{Smiley face} = cn^d (1 + r + r^2 + \dots + r^k)$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

$$\begin{aligned} \text{Smiley face} &= cn^d (1 + r + r^2 + \dots + r^k) \\ &= cn^d \left(\frac{r^{k+1} - 1}{r - 1} \right) \end{aligned}$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

$$\text{Smiley face} = cn^d (1 + r + r^2 + \dots + r^k)$$

$$= cn^d \left(\frac{r^{k+1} - 1}{r - 1} \right)$$

$$\leq cn^d \left(\frac{r}{r-1} \right) r^k$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

$$\text{Smiley face} = cn^d (1 + r + r^2 + \dots + r^k)$$

$$= cn^d \left(\frac{r^{k+1} - 1}{r - 1} \right)$$

$$\leq cn^d \left(\frac{r}{r-1} \right) r^k = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right)$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

$$\text{Smiley face} = cn^d (1 + r + r^2 + \dots + r^k)$$

$$= cn^d \left(\frac{r^{k+1} - 1}{r - 1} \right)$$

$$\leq cn^d \left(\frac{r}{r-1} \right) r^k = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O\left(n^{\log_b a}\right).$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

$$\text{Smiley face} = cn^d (1 + r + r^2 + \dots + r^k)$$

$$= cn^d \left(\frac{r^{k+1} - 1}{r - 1} \right)$$

leaves

$$\leq cn^d \left(\frac{r}{r-1} \right) r^k = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O\left(n^{\log_b a}\right).$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

dominated
by leaves



$$\text{☺} = cn^d (1 + r + r^2 + \dots + r^k)$$

$$= cn^d \left(\frac{r^{k+1} - 1}{r - 1} \right)$$

leaves

"

$$\leq cn^d \left(\frac{r}{r-1} \right) r^k = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O\left(n^{\log_b a}\right).$$

THE CALCULATIONS

$$\text{Total work done} \leq cn^d \sum_{j=0}^{\log_b n} \left(\frac{a}{b^d}\right)^j$$



Case 3 : $a > b^d$

$$r = a/b^d > 1$$

$$k = \log_b n$$

dominated
by leaves



$$\text{Smiley face} = cn^d (1 + r + r^2 + \dots + r^k)$$

$$= cn^d \left(\frac{r^{k+1} - 1}{r - 1} \right)$$

leaves

$$\leq cn^d \left(\frac{r}{r-1} \right) r^k = O\left(n^d \left(\frac{a}{b^d}\right)^{\log_b n}\right) = O\left(n^{\log_b a}\right).$$



MASTER THEOREM

Theorem: If $T(n) \leq a T\left(\frac{n}{b}\right) + O(n^d)$, then

$$T(n) = \begin{cases} O(n^d \log n) & \text{if } a = b^d & \text{Case 1} \\ O(n^d) & \text{if } a < b^d & \text{Case 2} \\ O(n^{\log_b a}) & \text{if } a > b^d & \text{Case 3} \end{cases}$$

LIMITATION

Master theorem only applies to **equisized subproblems**

For more general recurrences, use Akra-Bazzi method.