COL 351 : Analysis & Design of Algorithms

LECTURE 6

DIVIDE & CONQUER V :

PROOF AND APPLICATIONS OF MASTER THEOREM

AUG 02, 2024 | ROHIT VAISH

LAST TIME

Counting Inversions

Matrix multiplication

LAST TIME

Counting Inversions via enhanced Merge Sort $O(n \log n)$

Matrix multiplication

LAST TIME

Counting Inversions via enhanced Merge Sort $O(n \log n)$

Matrix multiplication
- Regular $O(n^3)$
- Naive recursive $O(n^3)$ *
- Strassen's $O(n^{2.81})$ *

MASTER THEOREM

MASTER THEOREM

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

$$T(n) = \begin{cases} & \text{Case 1} \\ & \text{Case 2} \\ & \text{Case 3} \end{cases}$$

MASTER THEOREM

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

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

MASTER THEOREM

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

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

MASTER THEOREM

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

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

only upper bounds

Examples

EXAMPLES

Merge Sort

EXAMPLES

Merge Sort

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

$T(n)$: maximum number of operations the algorithm needs for an input of size n

EXAMPLES

Merge Sort

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

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

EXAMPLES

Merge Sort

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

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

(Case 1)

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

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)

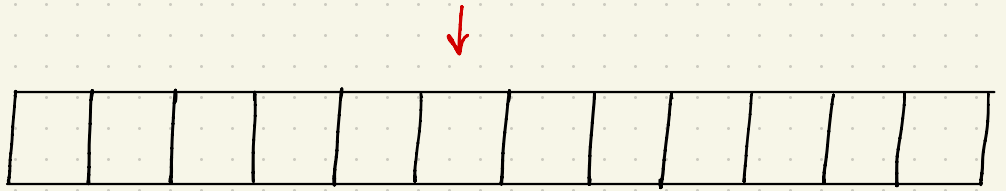
Look for "14"

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"

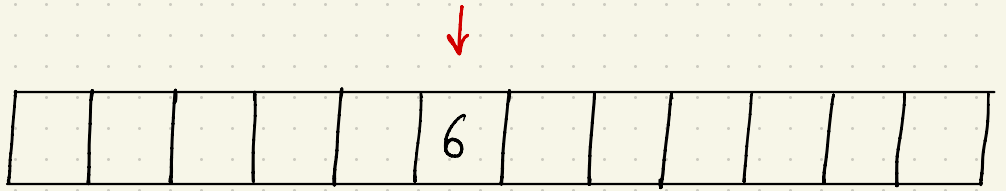


BINARY SEARCH

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

output: index of T in A (if exists)

Look for "14"

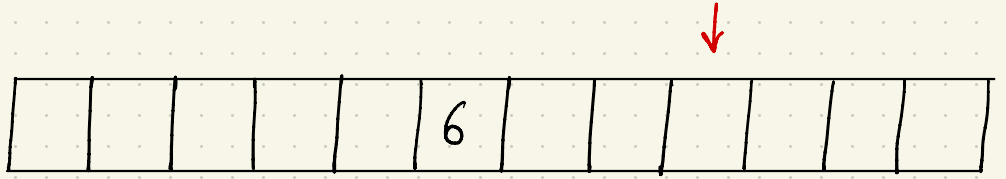


BINARY SEARCH

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

output: index of T in A (if exists)

Look for "14"

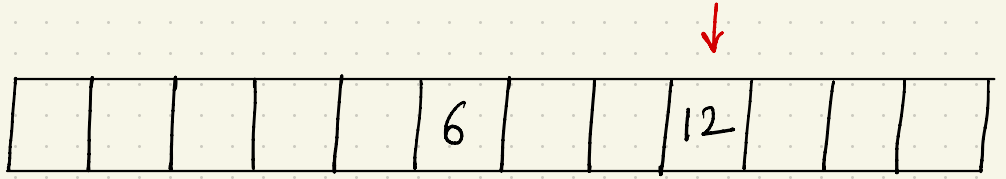


BINARY SEARCH

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

output: index of T in A (if exists)

Look for "14"

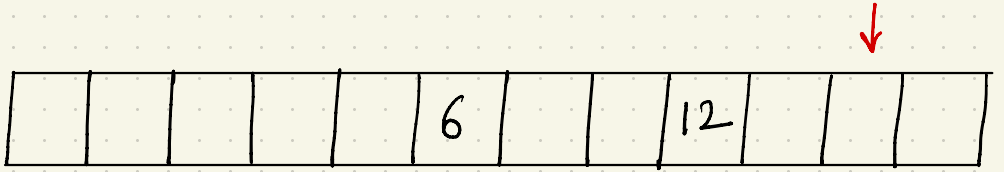


BINARY SEARCH

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

output: index of T in A (if exists)

Look for "14"

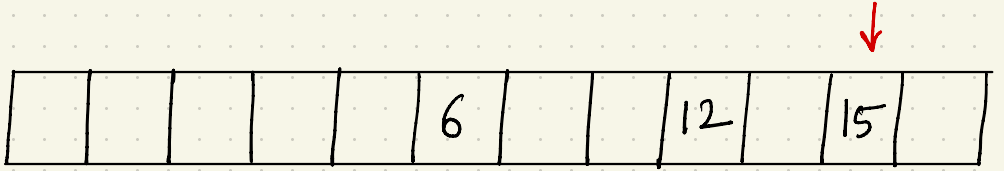


BINARY SEARCH

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

output: index of T in A (if exists)

Look for "14"

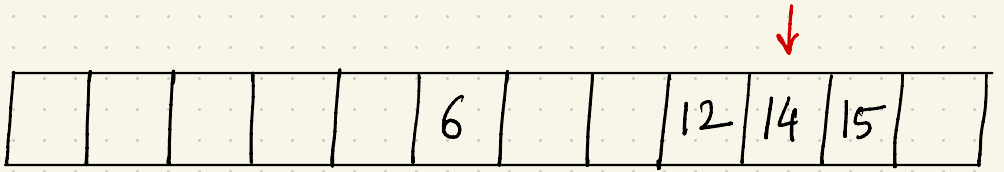


BINARY SEARCH

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

output: index of T in A (if exists)

Look for "14"

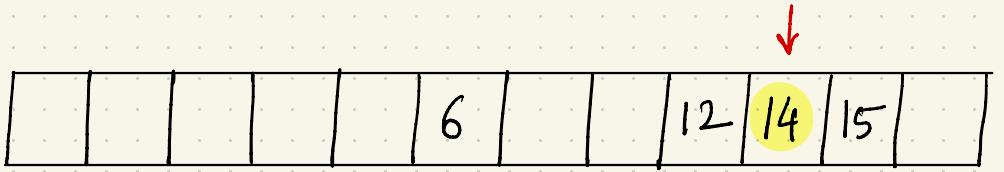


BINARY SEARCH

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)

initialize $l := 0$

initialize $r := n-1$

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

BINARY SEARCH

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

output: index of T in A (if exists)

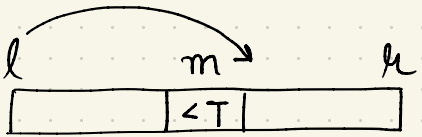
initialize $\ell := 0$

initialize $r := n-1$

initialize $m := \left\lfloor \frac{\ell + r}{2} \right\rfloor$

if $A[m] < T$

recurse with $\ell := m+1$



BINARY SEARCH

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

output: index of T in A (if exists)

initialize $\ell := 0$

initialize $r := n-1$

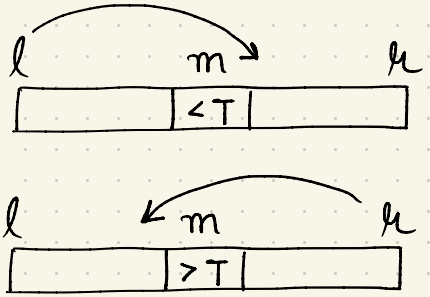
initialize $m := \left\lfloor \frac{\ell + r}{2} \right\rfloor$

if $A[m] < T$

recurse with $\ell := m+1$

else if $A[m] > T$

recurse with $r := m-1$



BINARY SEARCH

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

output: index of T in A (if exists)

initialize $\ell := 0$

initialize $r := n-1$

initialize $m := \left\lfloor \frac{\ell + r}{2} \right\rfloor$

if $A[m] < T$

    recurse with $\ell := m+1$

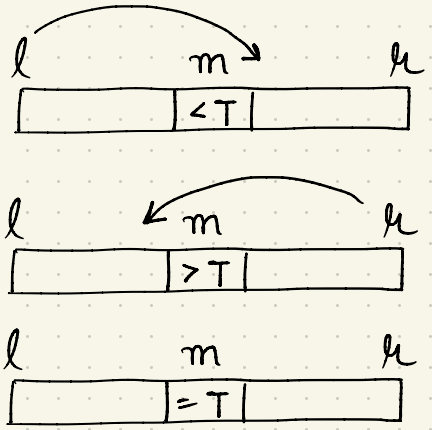
else if $A[m] > T$

    recurse with $r := m-1$

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 mult^n

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

EXAMPLES

Recursive integer mult^n

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

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

EXAMPLES

Recursive integer mult^n

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

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

(Case 3)

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

EXAMPLES

Recursive integer mult^n

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

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

(Case 3)

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

same as grade-school algorithm

KARATSUBA MULTIPLICATION

$x = \overbrace{56}^{a}\ \overbrace{78}^{b}$

$y = \overbrace{12}^{c}\ \overbrace{34}^{d}$

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

$$
\begin{array}{rl}
6720000 & \text{from ①} \\
2652 & \text{from ②} \\
2840000 & \text{from ④} \\
\hline
7006652 & = x \cdot y
\end{array}
$$

EXAMPLES

Recursive integer mult^n

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

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

(Case 3)

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

same as grade-school algorithm

Karatsuba mult^n

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

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

EXAMPLES

Recursive integer mult^n

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

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

(Case 3)

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

same as grade-school algorithm

Karatsuba mult^n

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

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

(Case 3)

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

faster!

EXAMPLES

Recursive integer mult^n

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

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

(Case 3)

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

same as grade-school algorithm

Karatsuba mult^n

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

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

(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

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}_{n \times n} \qquad Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix}_{n \times n}$$

Step 1: Recursively compute the eight matrix products

$$X \cdot Y = \begin{bmatrix} A \cdot E + B \cdot G & A \cdot F + B \cdot H \\ C \cdot E + D \cdot G & C \cdot F + D \cdot H \end{bmatrix}$$

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

EXAMPLES

Recursive matrix mult^n

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

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

EXAMPLES

Recursive matrix mult^n

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

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

(Case 3)

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

same as the Swim-and-dive 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 mult^n

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

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

(Case 3)

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

same as the Swim-and-dive algorithm

Strassen's algorithm

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

$a = 7$, $b = 2$, $d = 1$

(Case 3)

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

faster!

EXAMPLES

Fictitious recurrence

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

EXAMPLES

Fictitious recurrence

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

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

Recall

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

EXAMPLES

Fictitious recurrence

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

$a = 2, b = 2, d = 2$ (Case 2)

$O(n^2)$

Recall

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

Proof of Master Theorem

MASTER THEOREM

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

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

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: $T(1) = c$

General case: for $n > 1$

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

Can assume same constant

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 $n > 1$

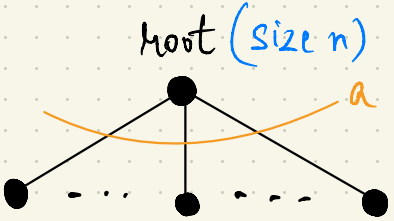
$$T(n) \leq a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

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

Recursion Tree

Recursion Tree

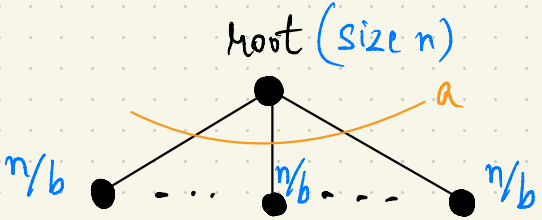
Recursion Tree



level 0 outermost call

level 1 first batch of recursive calls

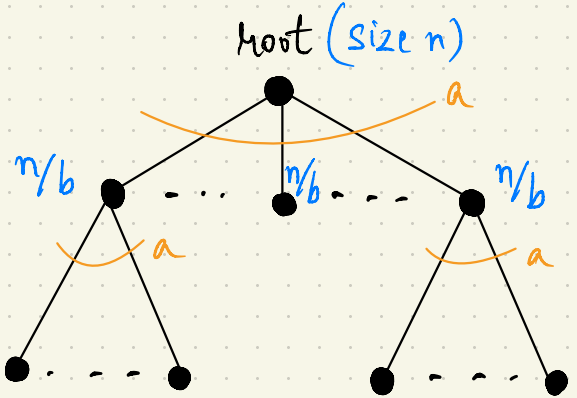
Recursion Tree



level 0 outermost call

level 1 first batch of recursive calls

Recursion Tree

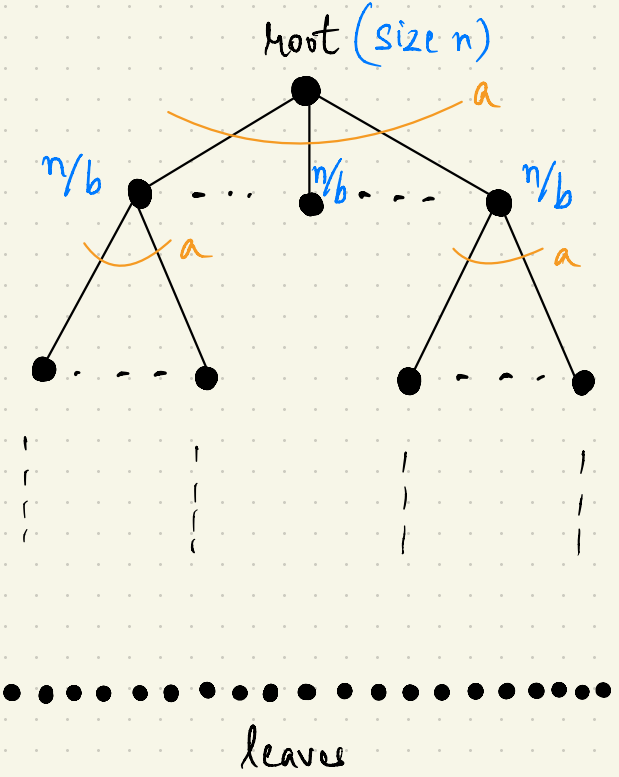


level 0 outermost call

level 1 first batch of recursive calls

level 2 second batch

Recursion Tree



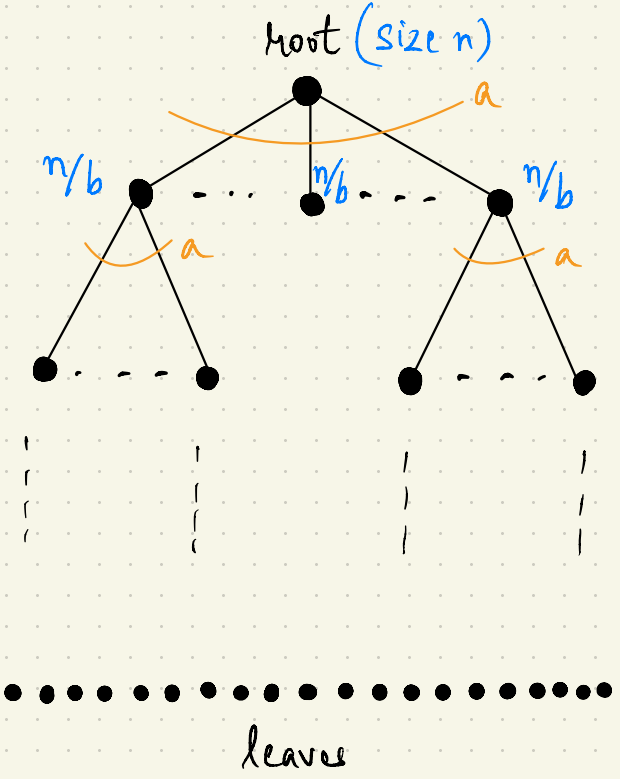
level 0 outermost call

level 1 first batch of recursive calls

level 2 second batch

level ? base cases

Recursion Tree



level 0 outermost call

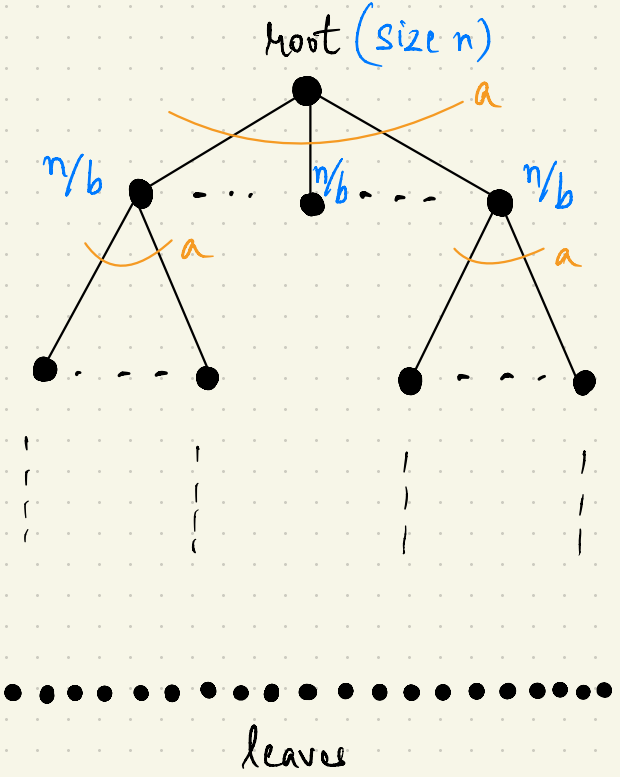
level 1 first batch of recursive calls

level 2 second batch

level $\log_b n$ base cases

$\log_b n$: # times n is divided by b to go below 1

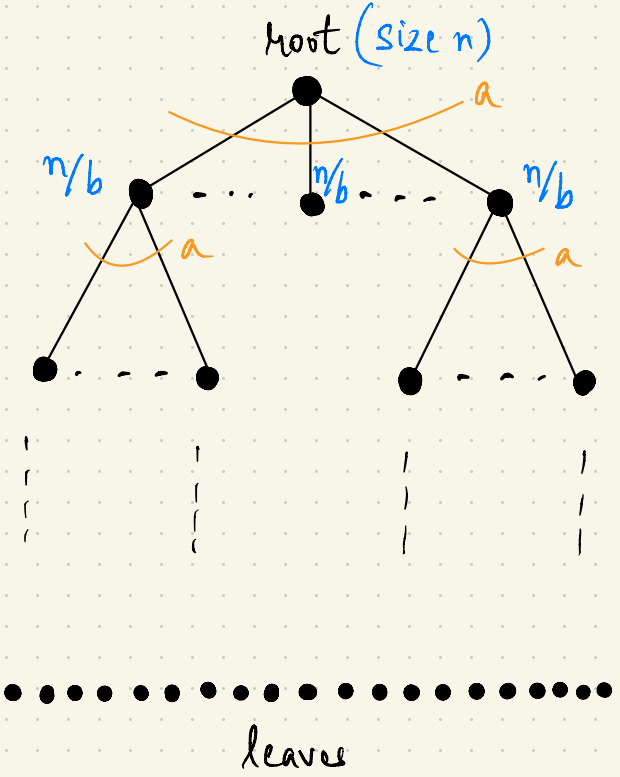
Recursion Tree



At level j

subproblems =

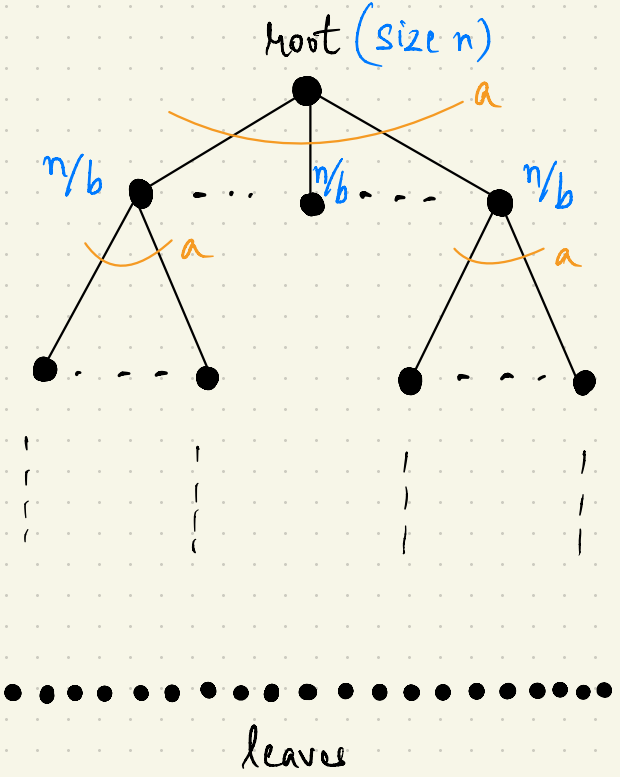
Recursion Tree



At level j

$\#$ subproblems $= a^j$

Recursion Tree

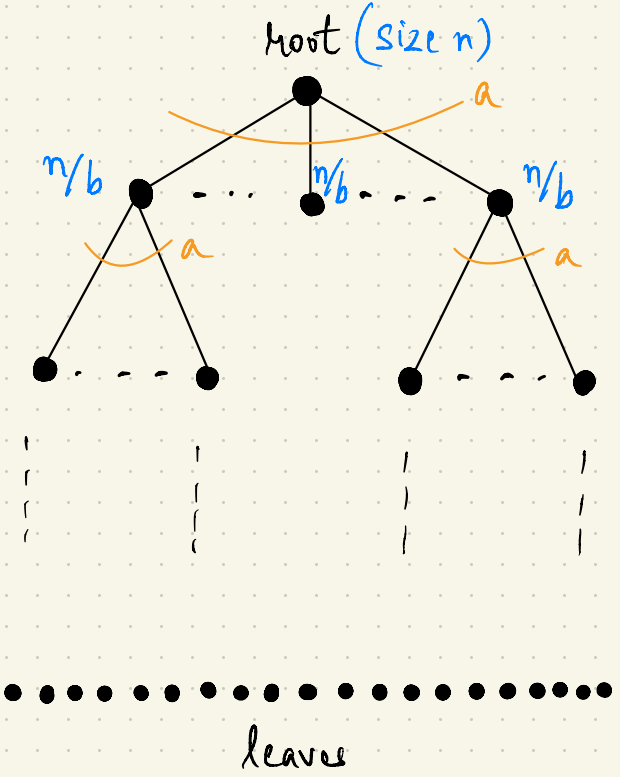


At level j

subproblems = a^j

subproblem size =

Recursion Tree

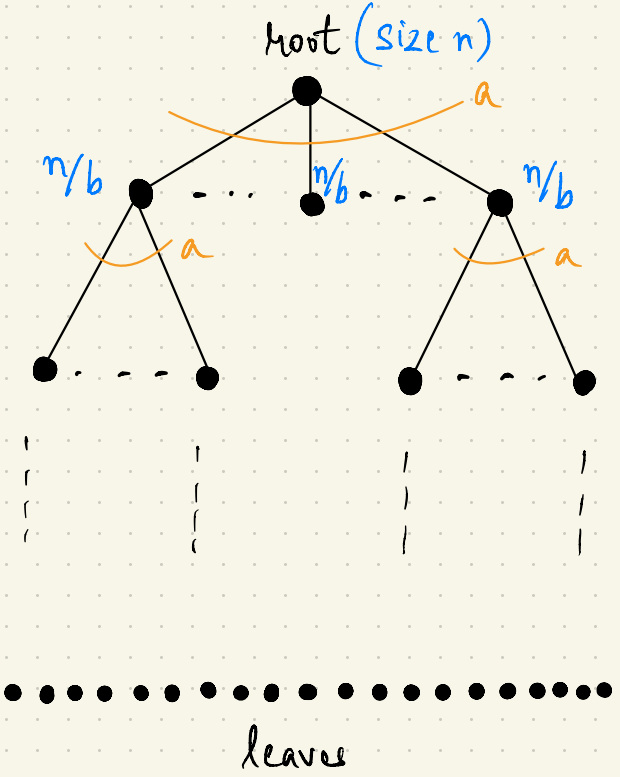


At level j

$\#$ subproblems $= a^j$

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

Recursion Tree



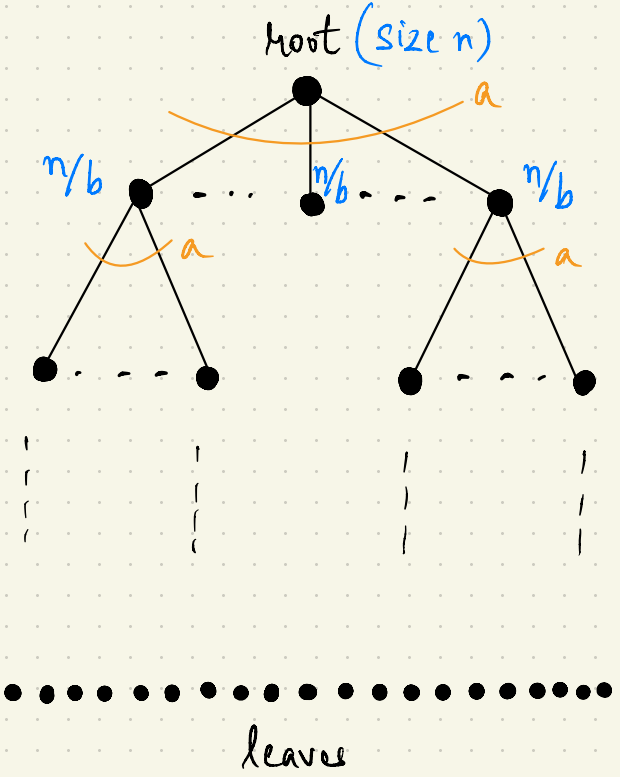
At level j

$\#$ subproblems $= a^j$

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

Total $\#$ levels $=$

Recursion Tree



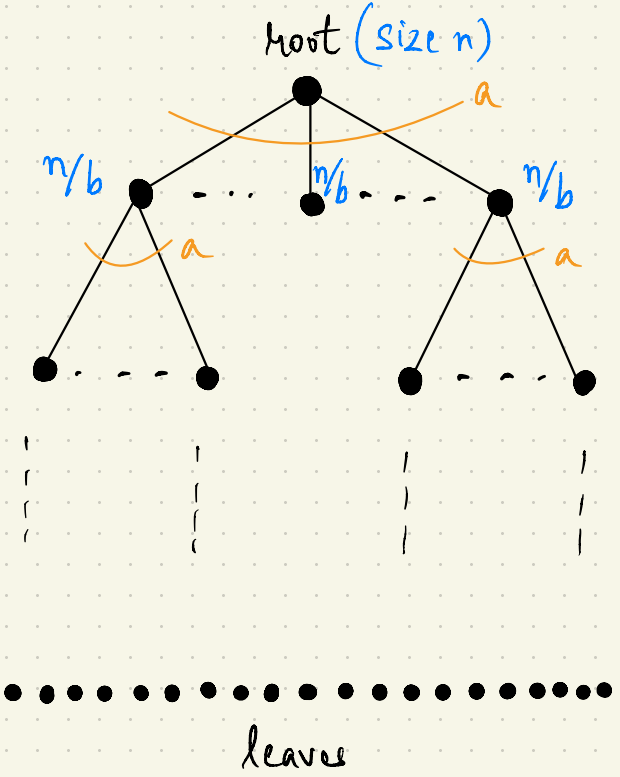
At level j

$\#$ subproblems $= a^j$

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

Total $\#$ levels $= 1 + \log_b n$

Recursion Tree

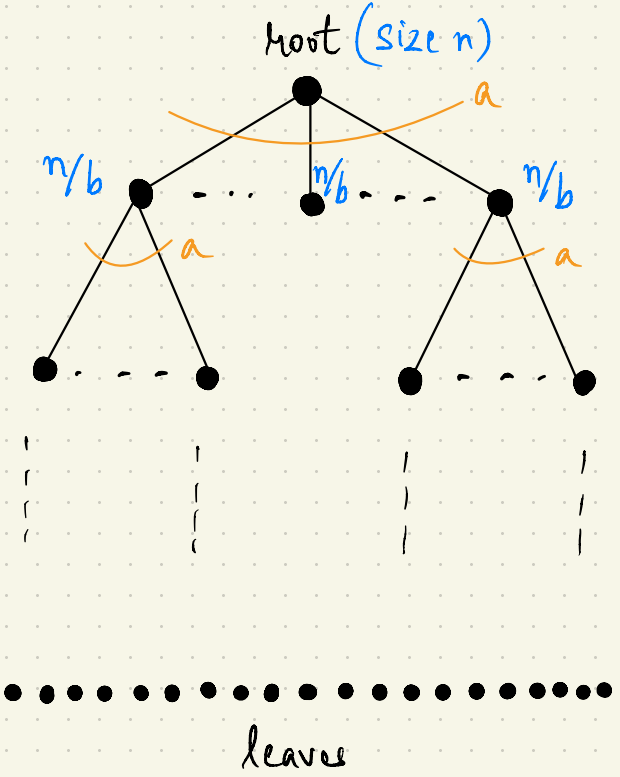


Total work at level j

(ignoring work in recursive calls)

=

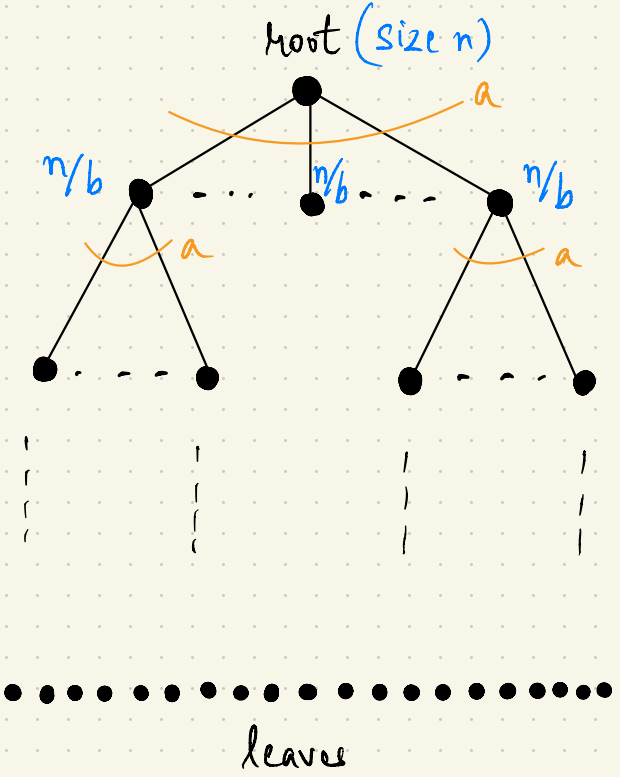
Recursion Tree



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

= # subproblems at level j × work done per subproblem

Recursion Tree

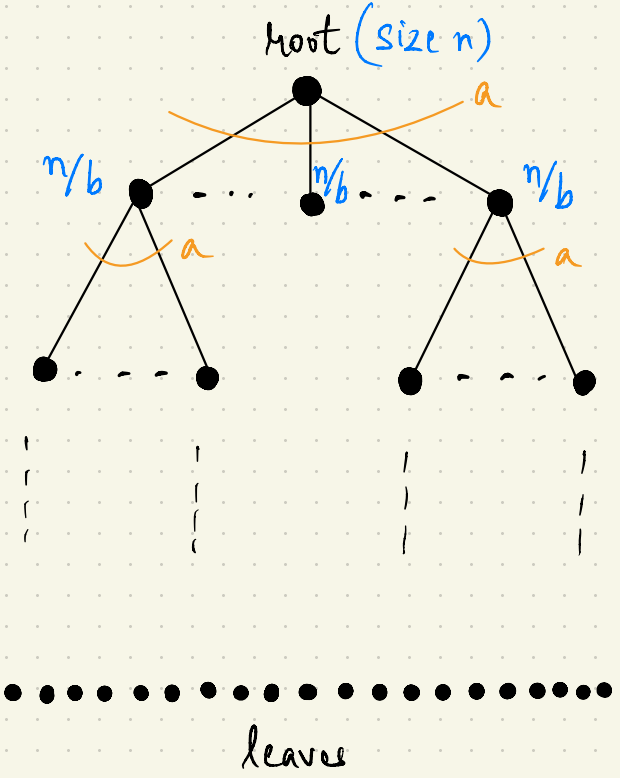


Total work at level j
(ignoring work in recursive calls)

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

$= a^j \times$

Recursion Tree



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

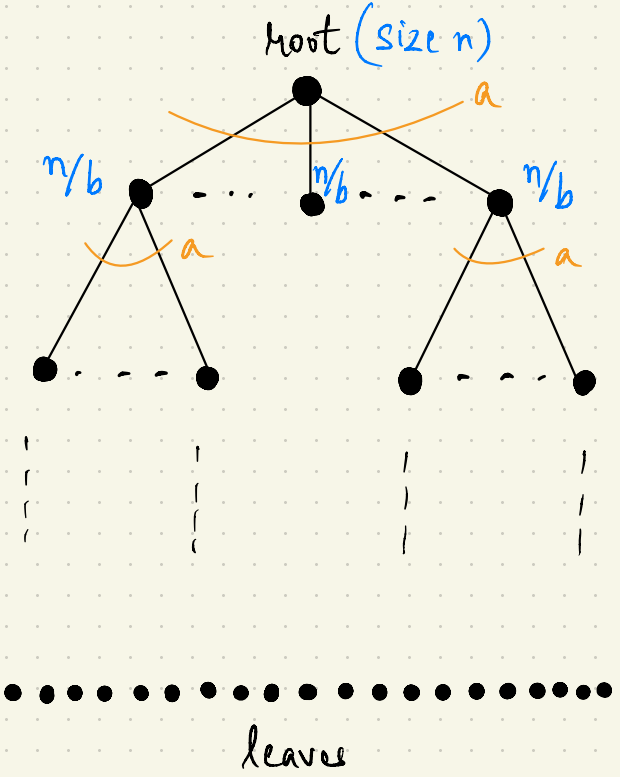
$=$ # subproblems at level j $\times$ work done per subproblem

$= a^j \times$

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)

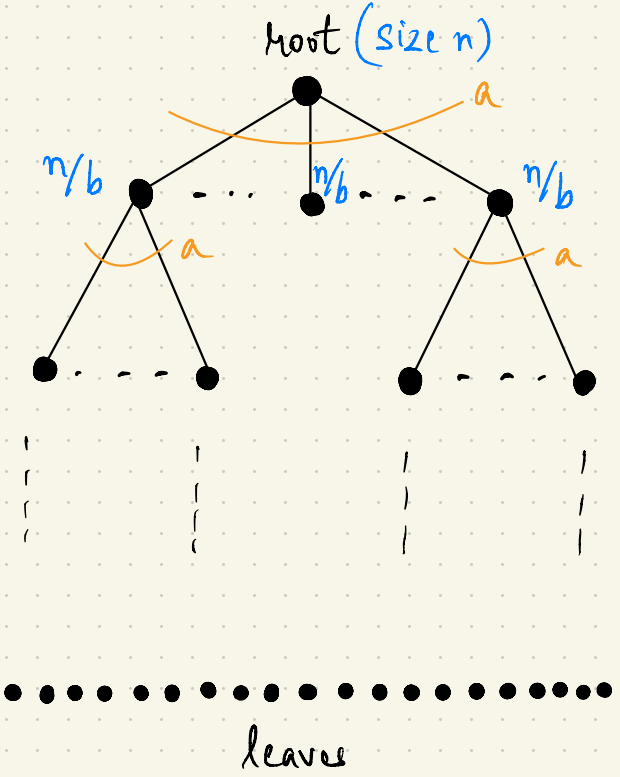
$$= \#\text{ subproblems at level } j \times \text{work done per subproblem}$$

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

Recall general case

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

Recursion Tree



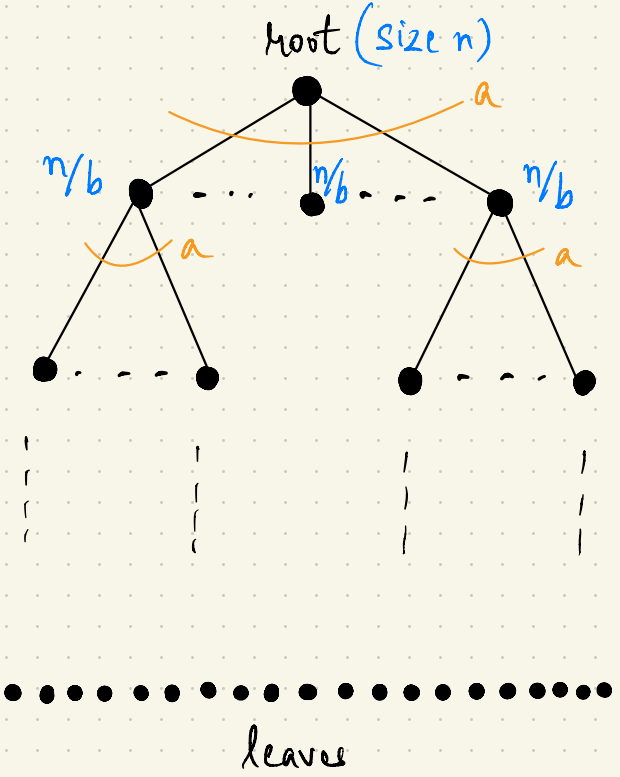
Total work at level j
(ignoring work in recursive calls)

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

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

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

Recursion Tree



Total work at level j

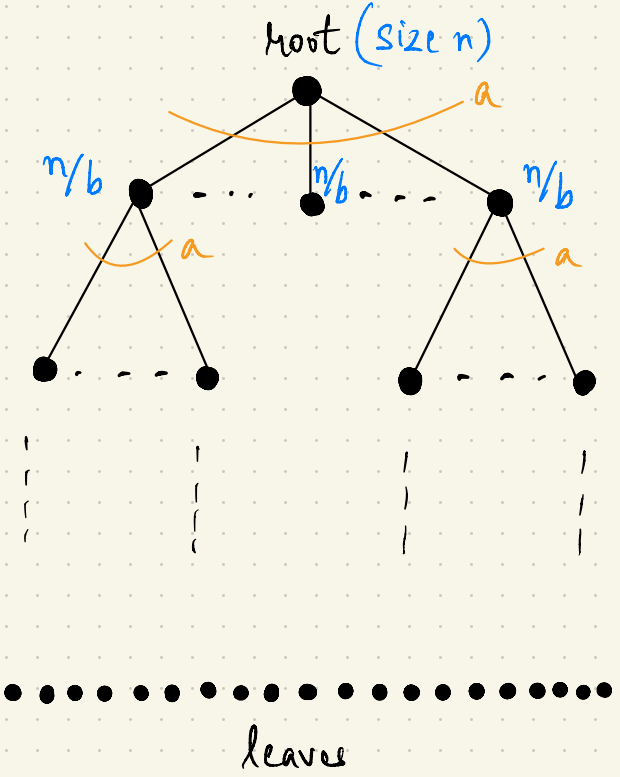
(ignoring work in recursive calls)

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

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

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

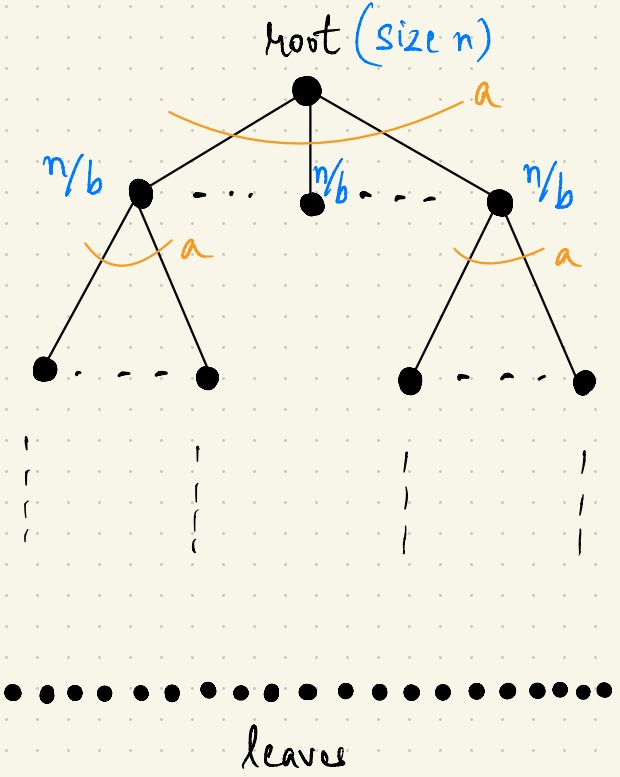
Recursion Tree



Total work across all levels

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

Recursion Tree



Total work across all levels

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

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 ☺

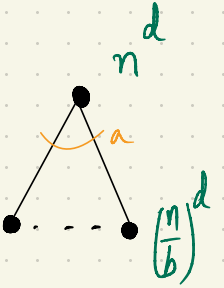
Recall: Work done at level $j = cn^d \cdot \left(\frac{a}{b^d}\right)^j$

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

a : rate of problem proliferation

b^d : rate of work shrinkage per subproblem

Tug of war between these two opposing forces

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

$a < b^d$ (proliferation < shrinkage) — work done per level ↓

$a > b^d$ (proliferation > shrinkage) — work done per level ↑

$a = b^d$ (proliferation = shrinkage)

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 ↓

$a > b^d$ (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 ↓

$a > b^d$ (proliferation > shrinkage) — work done per level ↑

$a = b^d$ (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 ↓

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

$a > b^d$ (proliferation > shrinkage)

work done per level ↑

$a = b^d$ (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 ↓

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

$a > b^d$ (proliferation > shrinkage)

work done per level ↑

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 ↓

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

$a > b^d$

(proliferation > shrinkage)

work done per level ↑

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

$= O\left(a^{\log_b n}\right)$

$a = b^d$

(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 ↓

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

$a > b^d$ (proliferation > shrinkage)

work done per level ↑

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

$= O\left(a^{\log_b n}\right) = O\left(n^{\log_b a}\right)$

$a = b^d$ (proliferation = shrinkage)

work done per level independent of j

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

The Calculations

The Calculations

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

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

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

Case 1 : $a = b^d$

 $= cn^d (\log_b n + 1)$

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 : $a = b^d$

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

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 : $a < b^d$

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 : $a < b^d$

 =

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 : $a < b^d$

$r = a/b^d < 1$

$k = \log_b n$

 =

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 : $a < b^d$

$r = a/b^d < 1$

$k = \log_b n$

 $= cn^d \left(1 + r + r^2 + \cdots + r^k\right)$

THE CALCULATIONS

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

Case 2 : $a < b^d$

$r = a/b^d < 1$

$k = \log_b n$

$$☺ = cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

$$\leq cn^d \left(1 + r + r^2 + \cdots + r^k + \cdots\right)$$

THE CALCULATIONS

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

Case 2 : $a < b^d$

$r = a/b^d < 1$

$k = \log_b n$

$$☺ = cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

$$\leq cn^d \left(1 + r + r^2 + \cdots + r^k + \cdots\right)$$

$$= cn^d \frac{1}{1-r}$$

THE CALCULATIONS

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

Case 2 : $a < b^d$

$r = a/b^d < 1$

$k = \log_b n$

$$= cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

$$\leq cn^d \left(1 + r + r^2 + \cdots + r^k + \cdots\right)$$

$$= cn^d \underbrace{\frac{1}{1-r}}_{\text{constant}}$$

THE CALCULATIONS

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

Case 2 : $a < b^d$

$r = a/b^d < 1$

$k = \log_b n$

$$☺ = cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

$$\le cn^d \left(1 + r + r^2 + \cdots + r^k + \cdots\right)$$

$$= cn^d \underbrace{\frac{1}{1-r}}_{\text{constant}} = O(n^d)$$

THE CALCULATIONS

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

Case 2 : $a < b^d$ $\qquad r = a/b^d < 1$

$k = \log_b n$

dominated by root

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

$$\le cn^d \left(1 + r + r^2 + \cdots + r^k + \cdots\right)$$

$$= cn^d \underbrace{\frac{1}{1-r}} = O(n^d)$$

constant

THE CALCULATIONS

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

Case 3 : $a > b^d$

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

$$☺ = cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

$$☺ = cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

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

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

$$= cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

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

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

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

$$☺ = cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

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

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

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

$$☺ = cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

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

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

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

$$☺ = cn^d \left(1 + r + r^2 + \cdots + r^k\right)$$

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

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

leaves = $n^{\log_b a}$

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

dominated by leaves ↘

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

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

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

leaves = $n^{\log_b a}$

THE CALCULATIONS

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

Case 3 : $a > b^d$

$r = a/b^d > 1$

$k = \log_b n$

dominated by leaves ↘

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

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

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

leaves = $n^{\log_b a}$

∎

MASTER THEOREM

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

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

LIMITATION

Master theorem only applies to equisized subproblems

For more general recurrences, use Akra-Bazzi method.