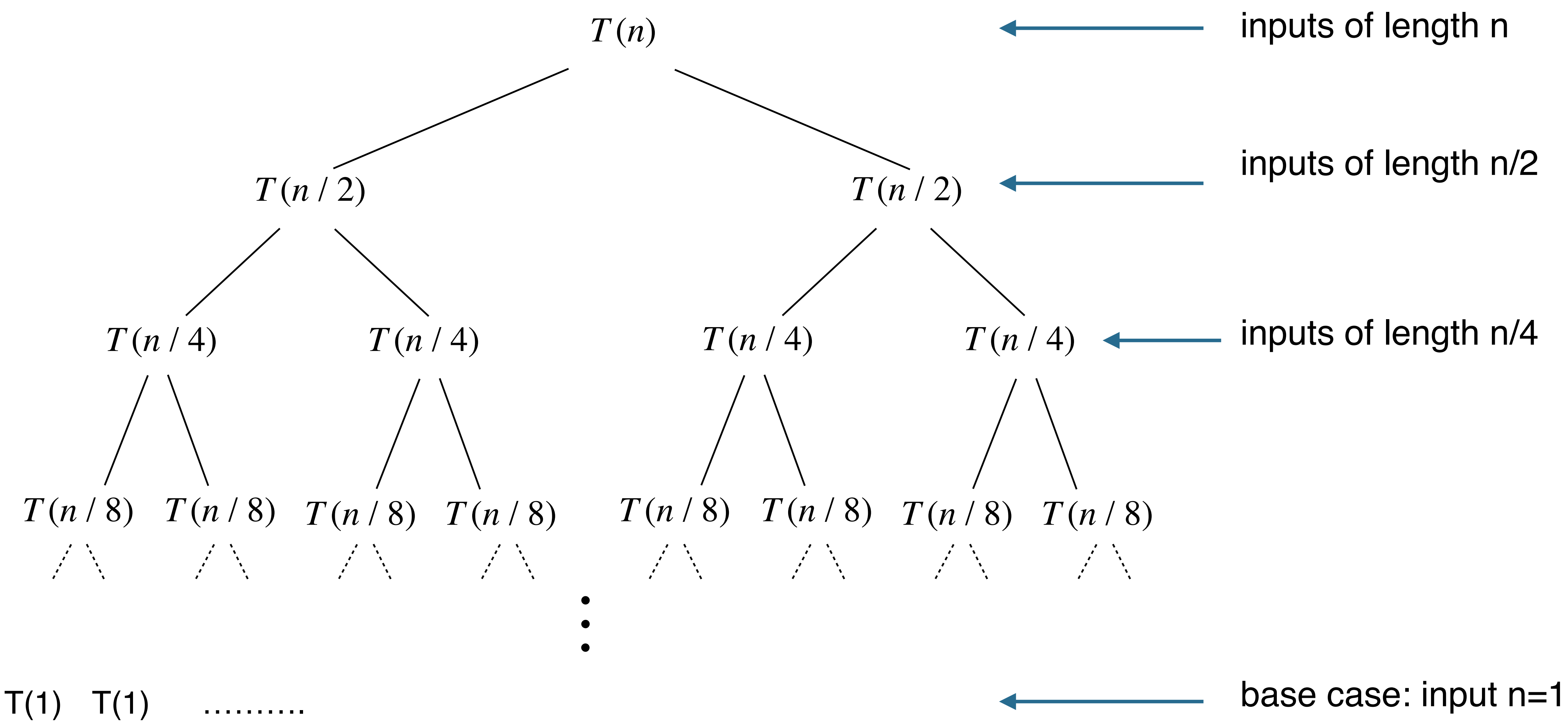
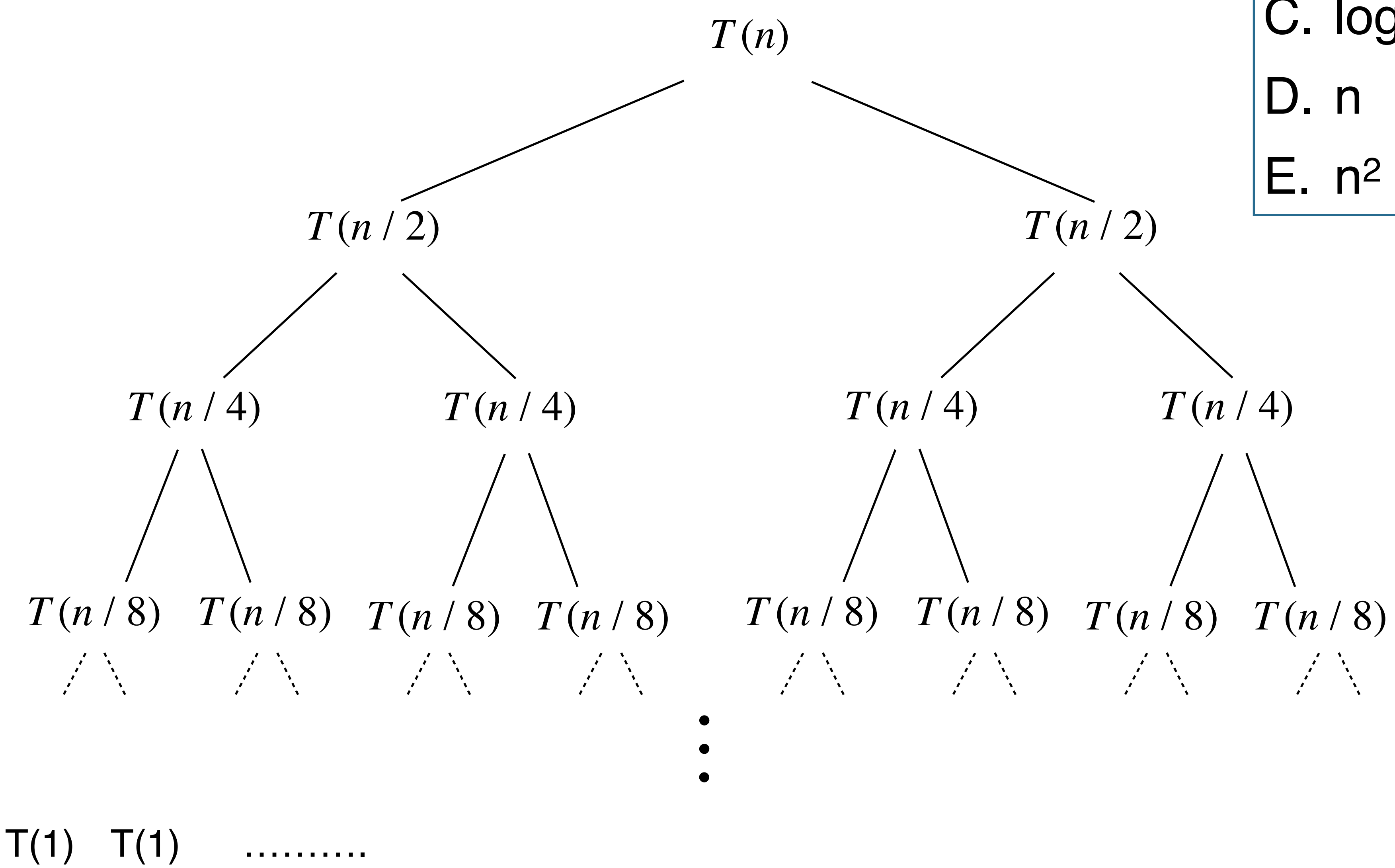
Recursion tree method

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



Recursion tree method – TopHat Question 2

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



Question. How many *layers*?

A. constant

B. $\log_2 n$

C. $\log_4 n$

D. n

E. n^2

Recursion tree method – TopHat Question 3

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

Question. How many *leaves*?

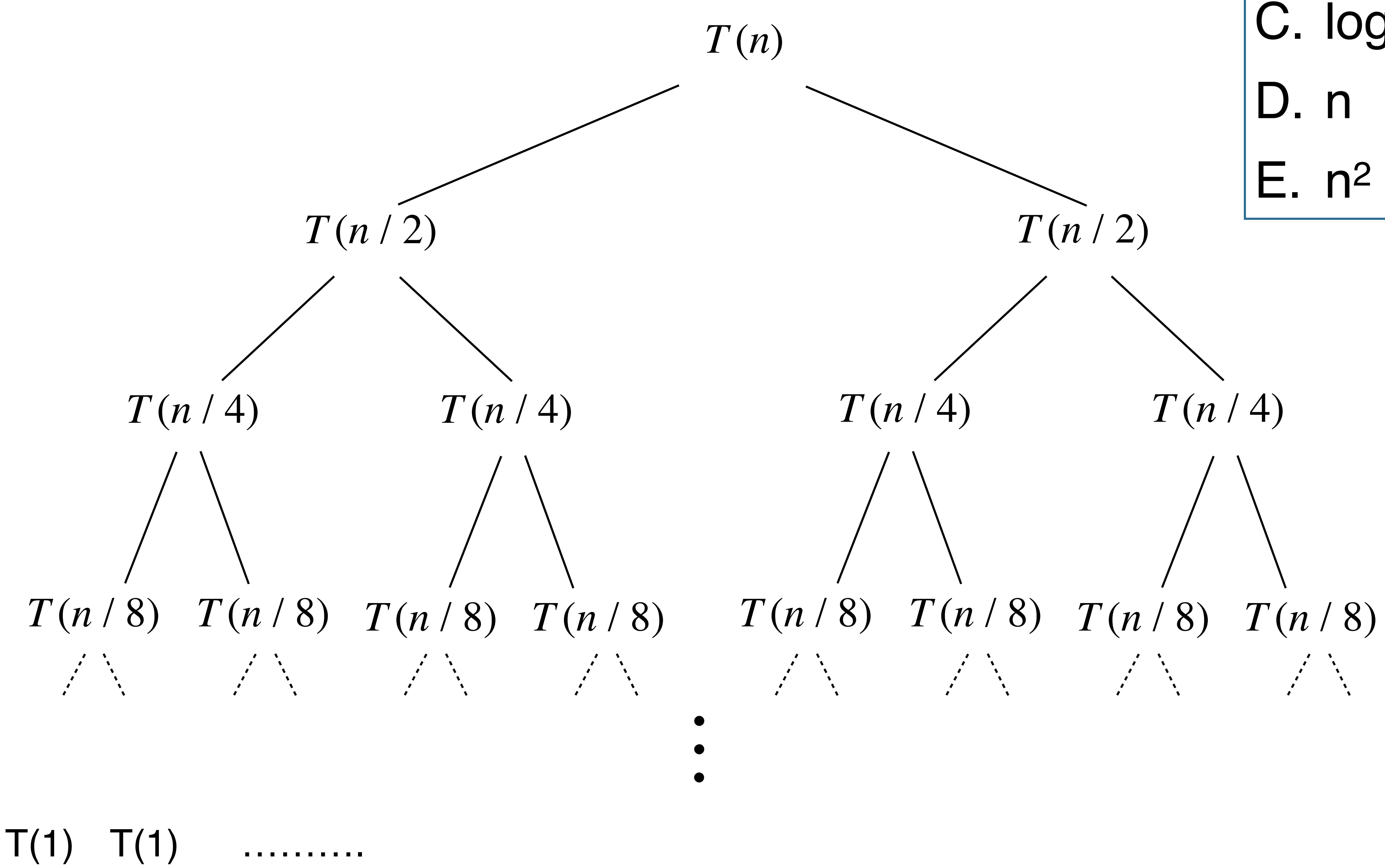
A. constant

B. $\log_2 n$

C. $\log_4 n$

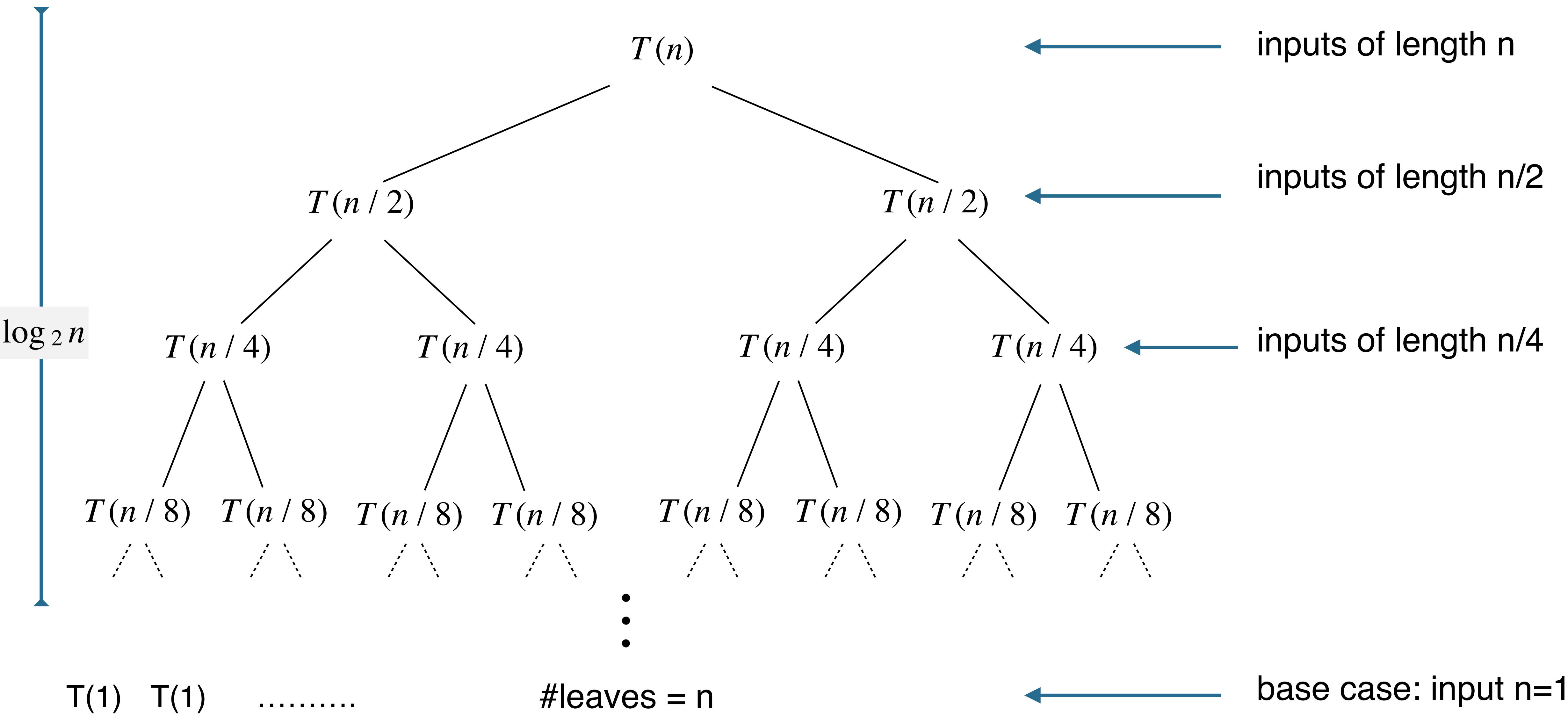
D. n

E. n^2



Recursion tree method

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

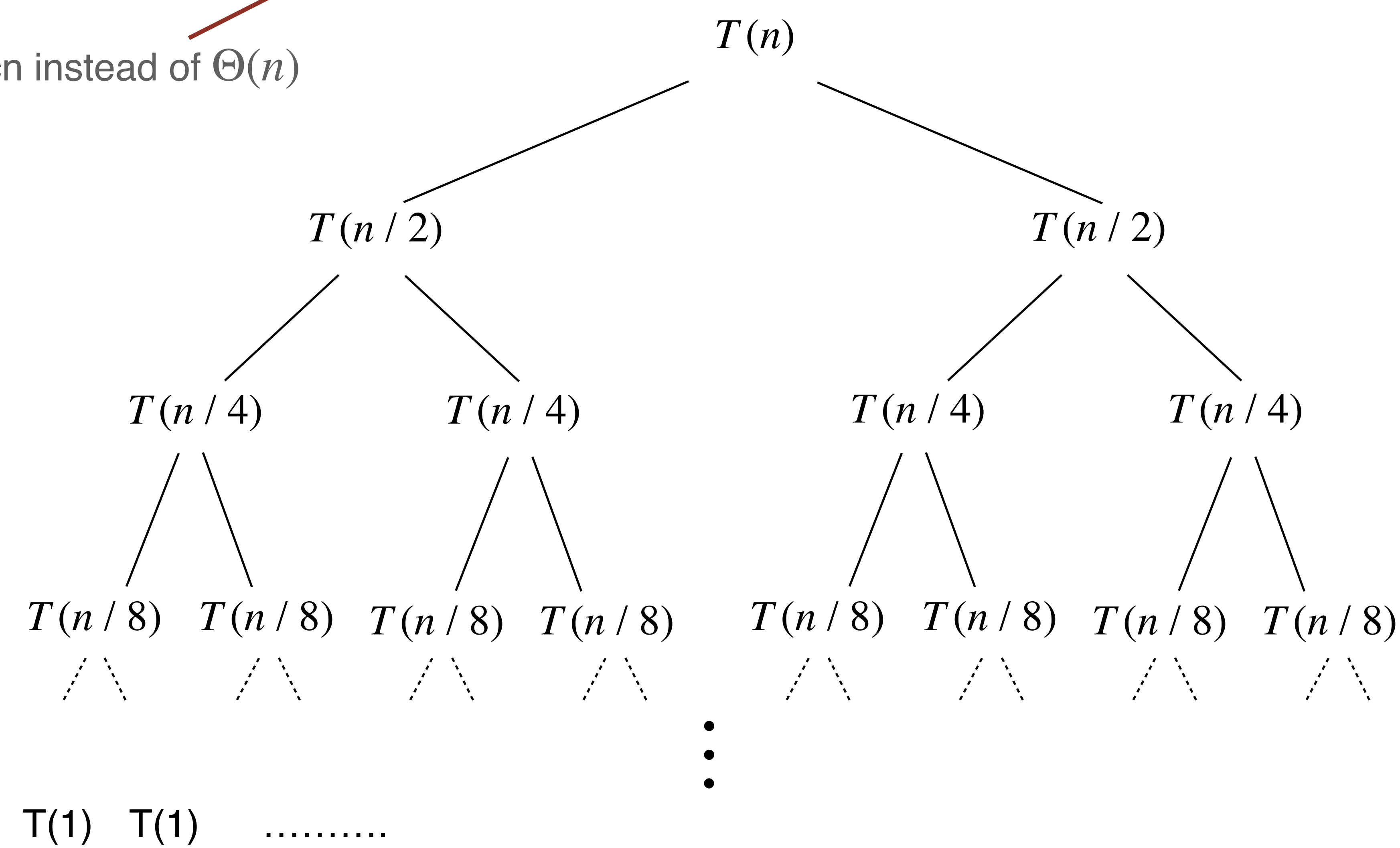


Recursion tree method – analysis idea

- charge each operation to the function call (i.e. tree node) at which it is executed
- for Mergesort on a subarray of length *k* we do *O(k)* work to compute the split point and do the merge
  - work in further recursive calls is counted separately

$$T(n) = 2 \cdot T(n/2) + cn \text{ where } c > 0$$

use cn instead of $\Theta(n)$

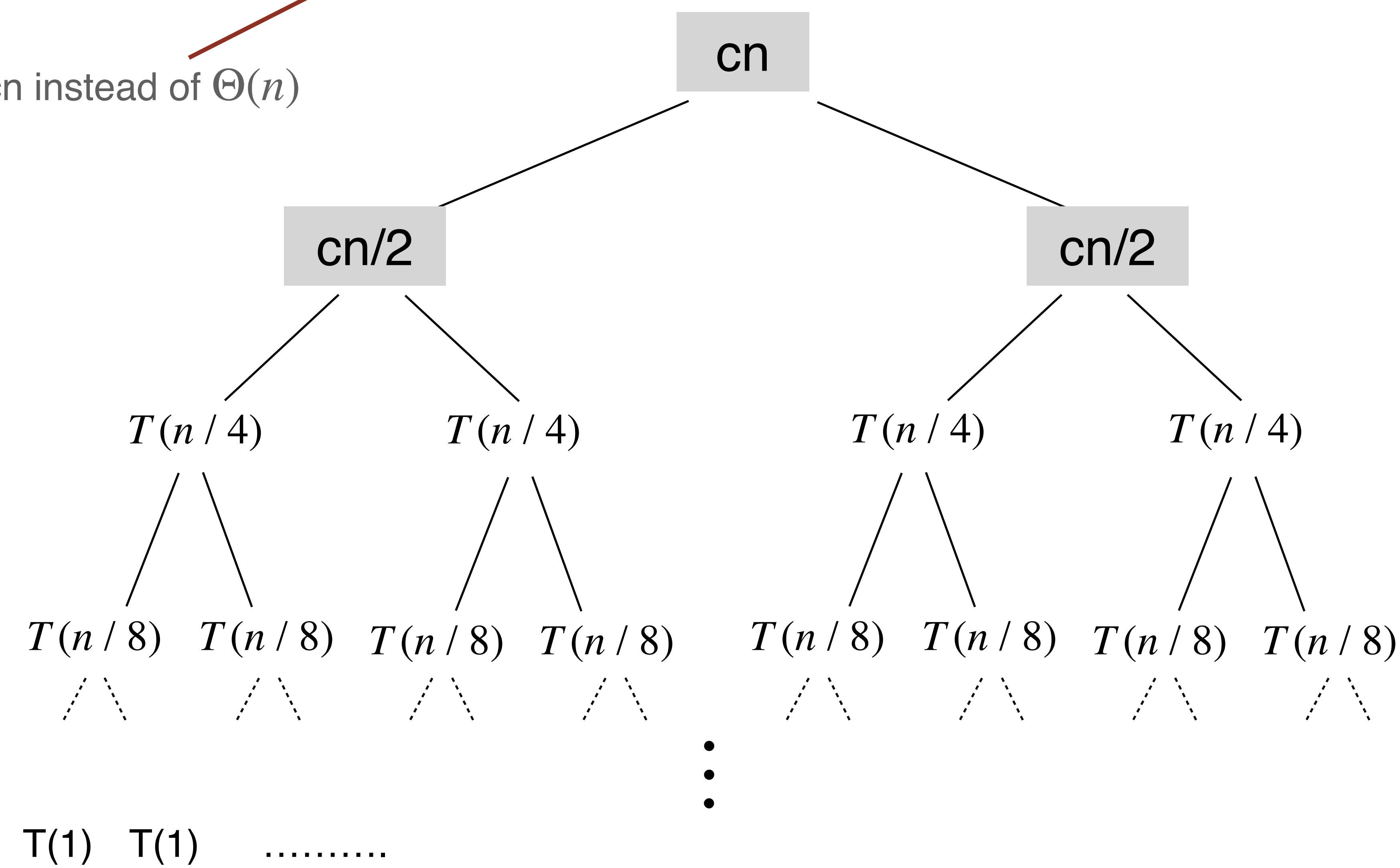


Recursion tree method – analysis idea

- charge each operation to the function call (i.e. tree node) at which it is executed
- for Mergesort on a subarray of length *k* we do *O(k)* work to compute the split point and do the merge
  - work in further recursive calls is counted separately

$T(n) = 2 \cdot T(n/2) + cn$ where $c > 0$

use cn instead of $\Theta(n)$

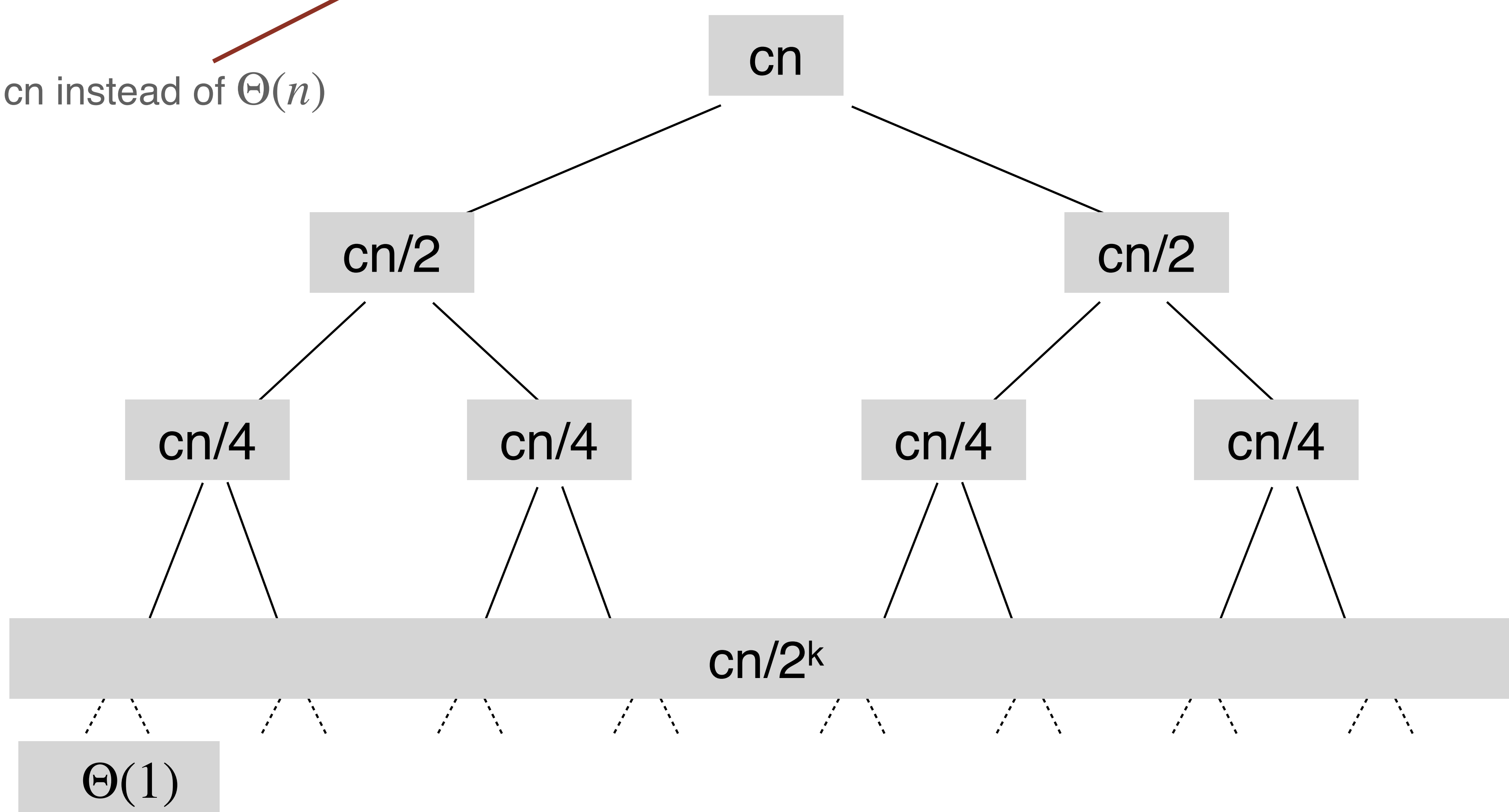


Recursion tree method – analysis idea

- charge each operation to the function call (i.e. tree node) at which it is executed
- for Mergesort on a subarray of length *k* we do *O(k)* work to compute the split point and do the merge
  - work in further recursive calls is counted separately

$T(n) = 2 \cdot T(n/2) + cn$ where $c > 0$

use cn instead of $\Theta(n)$



amount of work per level:

$c \cdot n$

$2 \cdot c \cdot n/2$

$4 \cdot c \cdot n/4$

$2^k \cdot c \cdot n/2^k$

$\Theta(n)$

Total: $\Theta(n \log n)$

binary search

Input: sorted array A, integer x

Output: the index in A where the int x is located or “x is not in A”

1. Divide: compare x to middle element
2. Conquer: recursively search one subarray
3. Combine: trivial

example: find 16

2	3	7	10	11	14	16	18	20	23

binary search - running time

Algorithm 1: BinarySearch(sorted array A, int p, int r, query)

`/* find the index of query in` A `(if present) */`

$q \leftarrow \lfloor \frac{p+r}{2} \rfloor$;
middle $\leftarrow A[q]$;
if *query* $==$ *middle* **then**
    return q;
if *query* $<$ *middle* **then**
    BinarySearch(A, p, q, query);
else
    BinarySearch(A, p, q, query);
return q not in A;

binary search — running time

Algorithm 1: BinarySearch(sorted array A, int p, int r, query)

`/* find the index of query in` A `(if present) */`

$q \leftarrow \lfloor \frac{p+r}{2} \rfloor$;
middle $\leftarrow A[q]$;
if *query* $==$ *middle* **then**
  return q;
if *query* $<$ *middle* **then**
  BinarySearch(A, p, q, query);
else
  BinarySearch(A, p, q, query);
return q not in A;

O(c)

recursive call on array half the size

$$T(n) = 1 \cdot T(n/2) + \Theta(1)$$

#of subproblems

subproblem size

work dividing and combining

find closed form by "telescoping"

Reminder: our goal is to find a closed form formula for the recurrence

method:

- substitute the recursive formula until we get a good guess
- use induction to prove the formula

$$T(n) = T(n/2) + c = T(n/4) + 2c = \ldots$$

$$= T(n/2^3) + 3c = \ldots = T(n/2^{\log n}) + (\log n)c = \Theta(\log n)$$

proof by "telescoping"

claim. $T(n) = \Theta(\log n)$

proof. by induction on n.

- base case: for n = 2 (or any constant) it takes const comparisons and log n = const
- suppose that the formula is true for every k < n
- proof for n: substitute k = n/2

$$T(n) = T(n/2) + \Theta(1) = \Theta(\log(n/2)) + \Theta(1) = \Theta(\log n)$$

Proof by "telescoping" for mergesort

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 1 & n == 1 \\ 2T\left(\frac{n}{2}\right) + cn & otherwise \end{cases}$$

Pf.

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

$$\leq 8 \cdot T\left(\frac{n}{8}\right) + 3cn \leq 2^4 \cdot T\left(\frac{n}{2^4}\right) + 4cn \leq$$

Substitute the formula in to T(n/2)

$$\ldots$$

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

The last line corresponds to the base case. We know that the base case is T(1) = 0

$$T(1) = T(\tfrac{n}{2^k}) \implies k = \log(n)$$

base case

by substituting log(*n)* for *k* we get

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

Master Theorem

For recurrences defined as

$$T(n) = aT(n/b) + f(n) \qquad \text{in which } a > 0, b \geq 1 \text{ are constants}$$

Then the overall computational cost is given by

$$T(n) = \underbrace{\Theta(n^{\log_b a})}_{\text{leaves}} + \underbrace{\sum_{k=1}^{\log_b n} a^{k-1} f(n/b^{k-1})}_{\text{internal nodes}}$$

Such that

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } f(n) = O(n^{\log_b a - \epsilon}) \\ \Theta(n^{\log_b a} \log n) & \text{if } f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{\log_b a + \epsilon}) \text{ and } af(n/b) \leq cf(n) \end{cases}$$

in which $a \geq 1$, $b > 1$, $c < 1$ and $\epsilon > 0$ are constants.

Solve the following recurrences

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

Asymptotic time to execute integer operations

Basic operations:

adding, subtracting, multiplying, dividing

Complexity of mathematical operations:

- Usually in this class we count them as constant time
- Today: we look at algorithms for efficiently implementing these basic operations.
- Today: in our analysis one *computational step* is counted as one bit operation.

Unit of measurement will be bit operations.

Input: two *n-bit* long integers a , b

Output: arithmetic operation on the two integers

Integer addition

Addition. Given two n-bit integers a and b, compute $a + b$.

Subtraction. Given two n-bit integers a and b, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations.

	2	1	3
+	1	2	5

	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1

Integer addition

Addition. Given two n-bit integers a and b, compute $a + b$.

Subtraction. Given two n-bit integers a and b, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations.

	2	1	3
+	1	2	5
	3	3	8

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

Integer addition

Addition. Given two n-bit integers a and b, compute $a + b$.

Subtraction. Given two n-bit integers a and b, compute $a - b$.

Grade-school algorithm. $\Theta(n)$ bit operations.

$$
\begin{array}{r}
a_{n-1}\, a_{n-2} \ldots a_2\, a_1\, a_0 \\
+ \quad b_{n-1}\, b_{n-2} \ldots b_2\, b_1\, b_0 \\
\hline
\boxed{\text{bits of } a + b}
\end{array}
$$

Remark. Grade-school addition and subtraction algorithms are asymptotically optimal.

Integer multiplication

Multiply 213_{10} x 125_{10} = 11010101_2 x 01111101_2

- (subscript refers to decimal and binary representation)

Grade-school algorithm:

			2	1	3
x			1	2	5
+					

Integer multiplication

Multiply 213_{10} x 125_{10} = 11010101_2 x 01111101_2

- (subscript refers to decimal and binary representation)

Grade-school algorithm:

			2	1	3
x			1	2	5
		1	0	6	5
		4	2	6	
+	2	1	3		
	2	6	6	2	5

TopHat

Multiplication. Given two n-bit integers a and b, compute $a \times b$.

Grade-school algorithm.

Question: What is the running time?

A. O(1)

B. O(log n)

C. O(n)

D. D. (n^2)

								1	1	0	1	0	1	0	1
							×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

Integer multiplication

Multiplication. Given two n-bit integers a and b, compute $a \times b$.

Grade-school algorithm. $\Theta(n^2)$ bit operations.

								1	1	0	1	0	1	0	1
							×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

n bits each

total of O(n²) bits

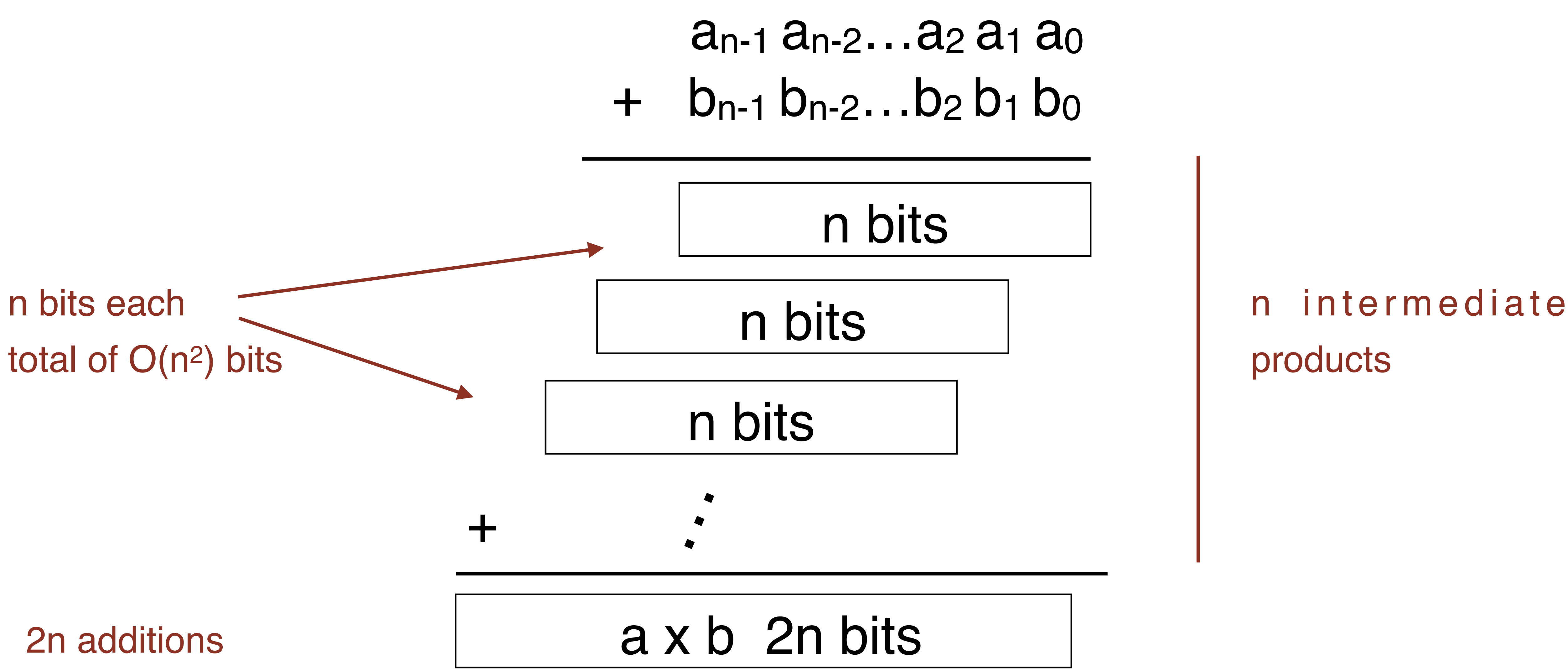
n intermediate products

2n additions

Integer multiplication

Multiplication. Given two n-bit integers a and b, compute $a \times b$.

Grade-school algorithm. $\Theta(n^2)$ bit operations.



Conjecture. [Kolmogorov 1952] Grade-school algorithm is optimal.

Theorem. [Karatsuba 1960] Conjecture is wrong. (his result $O(n^{log_2 3}) = O(n^{1.59\ldots})$)

Multiplying large integers

Input: n bit integers a and b

Compute: ab

computing ab = multiplying two n bit integers ~ $O(n^2)$ time

Divide-and-conquer:

- subproblem: same problem on smaller input
  - input size here is the number of bits in the integers
  - goal: compute ab by multiplying $n/2$ bit integers and then combine them

Divide: n bits to n/2

Input: n-bit integers a , b

Compute: ab

How to divide n-bit integers into n/2-bit ints?

$3557_{10} = 3500 + 57 = 35 \times 10^2 + 57$

$3557_{10} = 110111100101_2 = 110111000000_2 + 100101_2 = 110111_2 \times 2^6 + 100101_2$

Note: in decimal multiplying by 10^k shifts the number k bits to the left. Same in binary, multiplying by 2^k shifts k bits to the left (glues k 0s at the end.)

TopHat

Question. Write the representation of the base-3 number 120120_3 as two parts of n/2 digit base-3 integers.

A. $2 \cdot 120_3$

B. $3 \cdot 120_3 + 120_3$

C. $120_3 \cdot 3^3 + 120_3$

D. $120_3 \cdot 3^6 + 120_3 \cdot 3^3$

Divide: n bits to n/2

Input: integers *a* ,*b* of length *n* bits

How to “divide”? What are the subproblems?

- split *a* in to two substrings A_1, A_0 of length *n/2*

  $$a = a_{n-1}a_{n-2}\dots a_{\frac{n}{2}}a_{\frac{n}{2}-1}\dots a_2a_1a_0$$

  $$A_1 = a_{n-1}a_{n-2}\dots a_{\frac{n}{2}} \qquad A_0 = a_{\frac{n}{2}-1}\dots a_1a_0$$

- then $a = A_1 2^{n/2} + A_0$

- same for b: $b = B_1 2^{n/2} + B_0$

Multiplying large integers using D&C

Input: *n* bit integers *a* and *b*

Compute: *ab*

- A_1, A_0, B_1, B_0 are $n/2$ bit integers

ab consists of multiplying 2 *n* bit integers ~ $O(n^2)$ time

$a = A_1 2^{n/2} + A_0$ $\quad$ $b = B_1 2^{n/2} + B_0$

Compute:

$ab = (A_1 2^{n/2} + A_0)(B_1 2^{n/2} + B_0) =$

Multiplying large integers using D&C

Input: *n* bit integers *a* and *b*

Compute: *ab*

- A_1, A_0, B_1, B_0 are *n/2* bit integers

ab consists of multiplying 2 *n* bit integers ~ $O(n^2)$ time

$a = A_1 2^{n/2} + A_0$ $b = B_1 2^{n/2} + B_0$

Compute $ab = A_1B_1 2^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0$

- this formula consists of 4 multiplications of n/2 bit numbers and 3 additions

Divide and conquer approach: multiply *n/2* bit integers recursively

Multiplying large integers using D&C

(reminder: ab = $A_1B_12^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0$)

Algorithm 1: Multiply(ints a, b, n)

1 **if** $n == 1$ **then**
2   **return** ab;
3 $m \leftarrow \lfloor \frac{n}{2} \rfloor$;
4 $A_0 \leftarrow \lfloor a/2^m \rfloor$ `/* integer division */`
5 $B_0 \leftarrow \lfloor b/2^m \rfloor$ `/* integer division */`
6 $A_1 \leftarrow a \mod 2^m$;
7 $B_1 \leftarrow b \mod 2^m$;
8 $x \leftarrow$ Multiply(A_1, B_1, m);
9 $y \leftarrow$ Multiply(A_1, B_0, m);
10 $z \leftarrow$ Multiply(A_0, B_1, m);
11 $w \leftarrow$ Multiply(A_0, B_0, m);
12 **return** $x2^{2m} + (y + z)2^m + w$;

recursive call on input of size n/2

Note that lines 4-7 can be implemented by simple bit-shifts — no real division is taking place!

Recurrence:

Solve the recurrence

$$T(n) \leq \begin{cases} 1 & n = 1 \\ 4T(\frac{n}{2}) + O(n) & \text{otherwise} \end{cases}$$

Solve the recurrence

$$T(n) \leq \begin{cases} 1 & n = 1 \\ 4T(\frac{n}{2}) + O(n) & \text{otherwise} \end{cases}$$

Use telescoping and solve $T(n) = 4 \cdot T(\frac{n}{2}) + cn$

$$T(n) = 4 \cdot T(\frac{n}{2}) + cn = 4\left(4 \cdot T(\frac{n}{4}) + c\frac{n}{2}\right) + cn = (2^2)^2 \cdot T(\frac{n}{2^2}) + 3cn = \ldots$$

$$= (2^2)^3 \cdot T(\frac{n}{2^3}) + 5cn = \ldots = (2^2)^k \cdot T(\frac{n}{2^k}) + (k+1)cn = \ldots$$

we want to substitute the base case T(1), which happens when k = log n

$$= (2^2)^{\log n} T(1) + (\log n)cn = n^2 + \Theta(1) n \log n = \Theta(n^2)$$

Now we can prove by induction that $T(n) = \Theta(n^2)$

General formula for recurrences of specific form

Recurrence:

$$T(n) = 4T\left(\frac{n}{2}\right) + cn = cn^{log_2 4} = \Theta(n^2)$$

General form:

$$\boxed{T(n) = qT\left(\frac{n}{2}\right) + cn = cn^{\log_2 q} \text{ for } q > 2}$$

Discussed in detail: Kleinberg-Tardos pages 214 - 218

Multiplying large integers using D&C

(reminder: $ab = A_1B_12^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0$)

Algorithm 1: Multiply(ints a, b, n)

---

1 **if** $n == 1$ **then**
2   **return** ab;
3 $m \leftarrow \lfloor \frac{n}{2} \rfloor$;
4 $A_0 \leftarrow \lfloor a/2^m \rfloor$ `/* integer division */`
5 $B_0 \leftarrow \lfloor b/2^m \rfloor$ `/* integer division */`
6 $A_1 \leftarrow a \mod 2^m$;
7 $B_1 \leftarrow b \mod 2^m$;
8 $x \leftarrow \text{Multiply}(A_1, B_1, m)$;
9 $y \leftarrow \text{Multiply}(A_1, B_0, m)$;
10 $z \leftarrow \text{Multiply}(A_0, B_1, m)$;
11 $w \leftarrow \text{Multiply}(A_0, B_0, m)$;
12 **return** $x2^{2m} + (y + z)2^m + w$;

recursive call on input of size n/2

Note that lines 4-7 can be implemented by simple bit-shifts — no real division is taking place!

Recurrence: $T(n) = 4T(\frac{n}{2}) + cn$ $\qquad T(n) = \Theta(n^2)$

Karatsuba's algorithm

Input: n bit integers a and b

Compute: ab

$a = A_1 2^{n/2} + A_0$

$b = B_1 2^{n/2} + B_0$

Compute $ab = A_1 B_1 2^n + (A_1 B_0 + A_0 B_1) 2^{n/2} + A_0 B_0$

Karatsuba's trick: $(A_1+A_0)(B_1+B_0) = A_1 B_1 + A_0 B_0 + (A_1 B_0 + A_0 B_1)$

Karatsuba's algorithm

Input: n bit integers a and b

Compute: ab

$a = A_1 2^{n/2} + A_0$

$b = B_1 2^{n/2} + B_0$

Compute $ab = A_1B_1 2^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0$

Karatsuba's trick: $(A_1+A_0)(B_1+B_0) = A_1B_1 + A_0B_0 + (A_1B_0 + A_0B_1)$

Karatsuba's algorithm

Input: n bit integers a and b

Compute: ab

$a = A_1 2^{n/2} + A_0$

$b = B_1 2^{n/2} + B_0$

Compute $ab = A_1B_1 2^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0$

Karatsuba's trick: $(A_1+A_0)(B_1+B_0) = A_1B_1 + A_0B_0 + (A_1B_0 + A_0B_1)$

Thus we get $x = A_1B_1$, $y = A_0B_0$, $z = \underbrace{(A_1+A_0)}_{\text{n/2 bits!}}(B_1+B_0)$ ← multiply n/2 bit integers

$ab = A_1B_1 2^n + (A_1B_0 + A_0B_1)2^{n/2} + A_0B_0 = x2^n + (z - x - y)2^{n/2} + y$

The new formula contains only 3 multiplications of n/2 bits.

Karatsuba's algorithm

Algorithm 1: Karatsuba(ints a, b, n)

if $n == 1$ **then**
    return ab;
$m \leftarrow \lfloor \frac{n}{2} \rfloor$;
$A_0 \leftarrow \lfloor a/2^m \rfloor$ `/* integer division */`
$B_0 \leftarrow \lfloor b/2^m \rfloor$ `/* integer division */`
$A_1 \leftarrow a \mod 2^m$;
$B_1 \leftarrow b \mod 2^m$;
$x \leftarrow \text{Karatsuba}(A_1, B_1, m)$;
$y \leftarrow \text{Karatsuba}(A_0, B_0, m)$;
$z \leftarrow \text{Karatsuba}(A_1 + A_0, B_1 + B_0, m)$;
return $x2^{2m} + (z - x - y)2^m + y$;

Karatsuba's algorithm - TopHat

Algorithm 1: Karatsuba(ints a, b, n)

if $n == 1$ **then**
    return ab;
$m \leftarrow \lfloor \frac{n}{2} \rfloor$;
$A_0 \leftarrow \lfloor a/2^m \rfloor$ `/* integer division */`
$B_0 \leftarrow \lfloor b/2^m \rfloor$ `/* integer division */`
$A_1 \leftarrow a \mod 2^m$;
$B_1 \leftarrow b \mod 2^m$;
$x \leftarrow \text{Karatsuba}(A_1, B_1, m)$;
$y \leftarrow \text{Karatsuba}(A_0, B_0, m)$;
$z \leftarrow \text{Karatsuba}(A_1 + A_0, B_1 + B_0, m)$;
return $x2^{2m} + (z - x - y)2^m + y$;

Question: What is the recurrence for this algorithm?

A. $T(n) = \Theta(n)$

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

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

D. $T(n) = 3 \cdot T(\frac{n}{3}) + \Theta(1)$

Karatsuba's algorithm

Algorithm 1: Karatsuba(ints a, b, n)

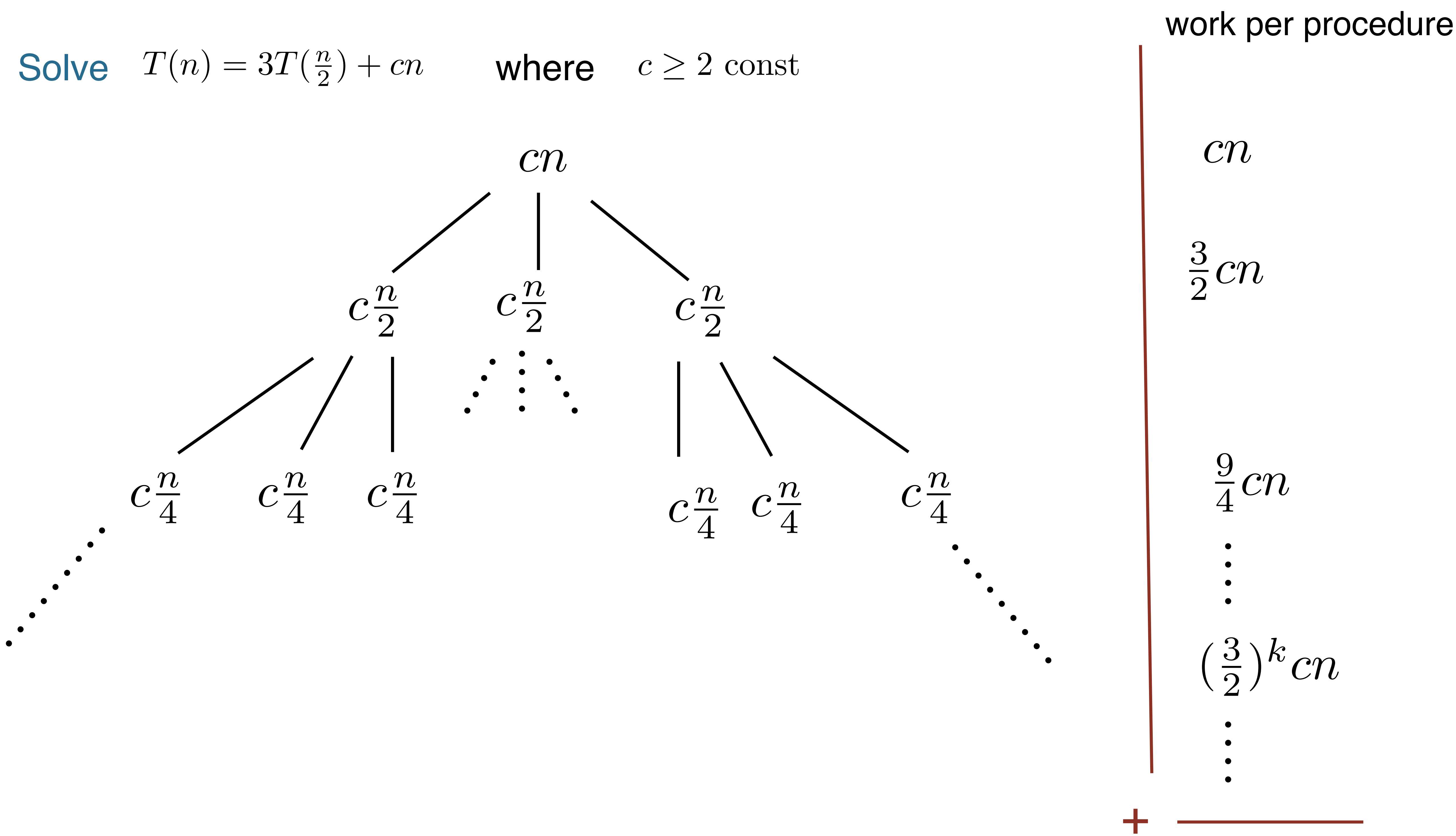
```
if n == 1 then
    return ab;
m ← ⌊n/2⌋;
A0 ← ⌊a/2^m⌋  /* integer division */
B0 ← ⌊b/2^m⌋  /* integer division */
A1 ← a mod 2^m;
B1 ← b mod 2^m;
x ← Karatsuba(A1, B1, m);
y ← Karatsuba(A0, B0, m);
z ← Karatsuba(A1 + A0, B1 + B0, m);
return x·2^(2m) + (z − x − y)·2^m + y;
```

$$T(n) = 3T(\tfrac{n}{2}) + cn$$

$$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59\ldots})$$

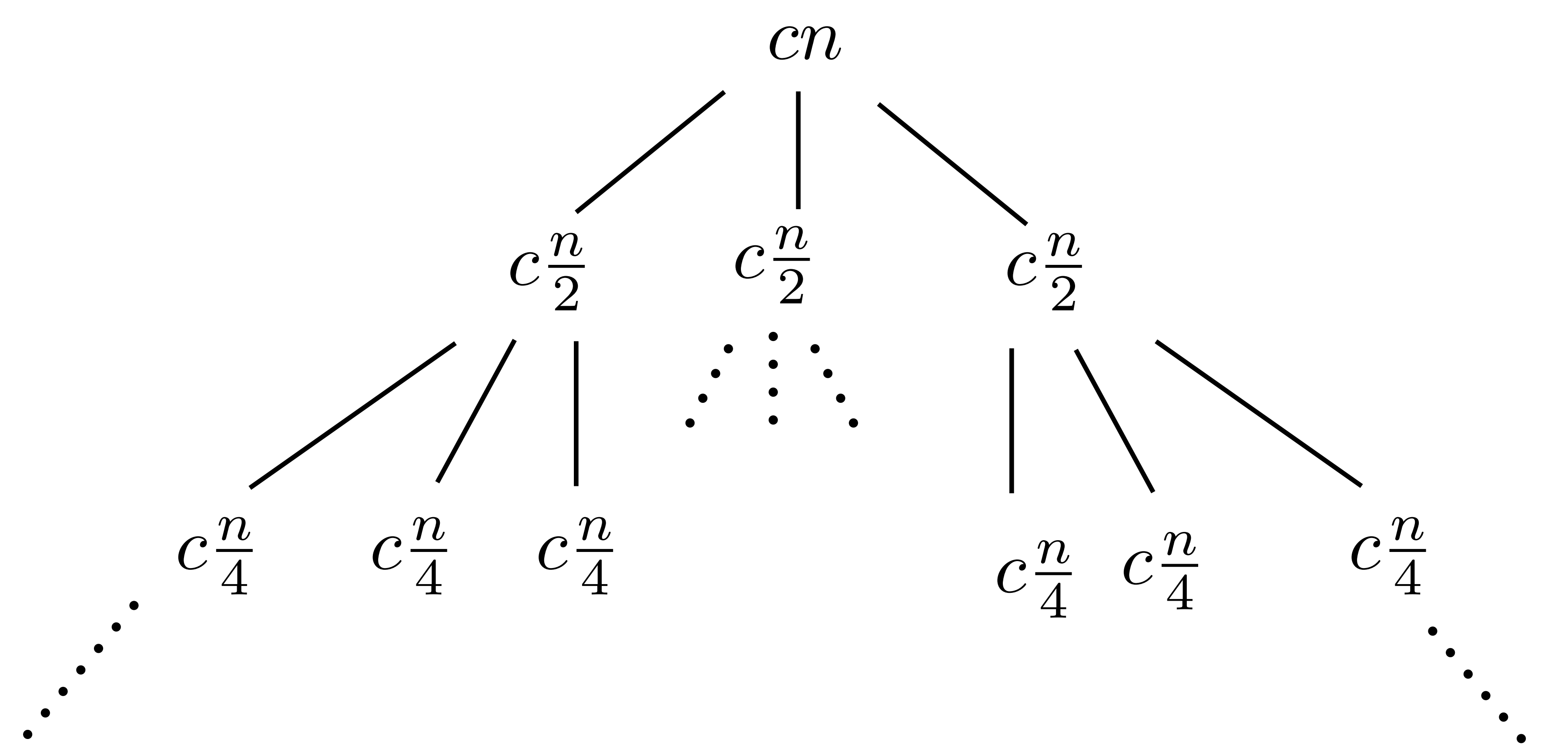
Recursion tree method

Solve $T(n) = 3T(\frac{n}{2}) + cn$ where $c \geq 2$ const



Recursion tree method - TopHat

Solve $T(n) = 3T(\frac{n}{2}) + cn$ where $c \geq 2$ const



work per procedure

cn

$\frac{3}{2}cn$

$\frac{9}{4}cn$

$\vdots$

$\left(\frac{3}{2}\right)^k cn$

$\vdots$

$+$

Question: what is the hight of the tree and number of nodes at layer k? (you may assume the root is at layer 0)

A. $\log_3 n$ & n^k

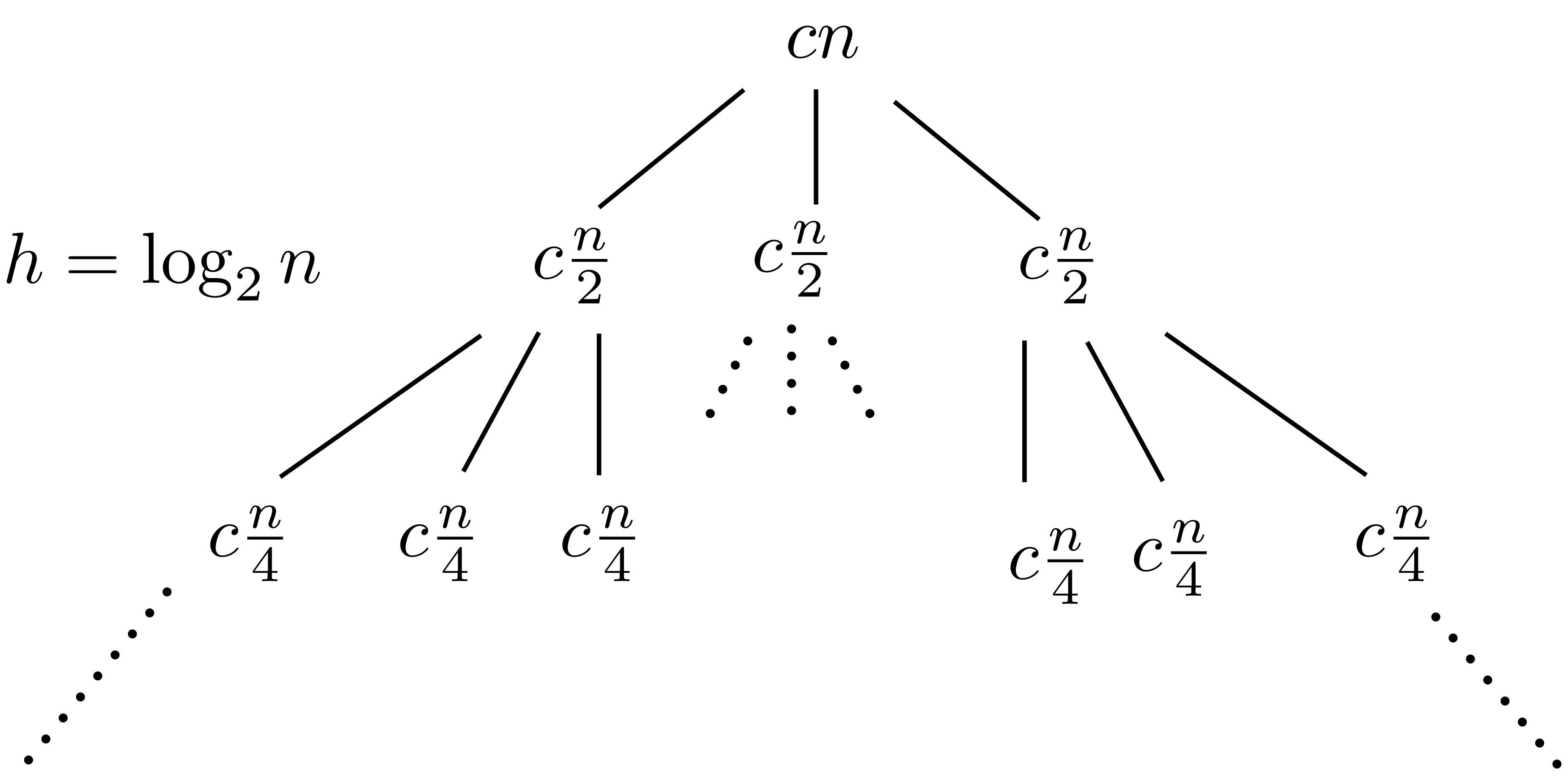
B. $\log_2 n$ & 3^k

C. $\log_2 n$ & n^3

D. $\log_3 n$ & k^3

Recursion tree method

Solve $T(n) = 3T(\frac{n}{2}) + cn$ where $c \geq 2$ const



work per procedure

$$cn$$

$$\frac{3}{2}cn$$

$$\frac{9}{4}cn$$

$$\vdots$$

$$\left(\frac{3}{2}\right)^k cn$$

$$\vdots$$

$+$

$$\left(\frac{3}{2}\right)^{\log_3 n} cn = cn^{\log_2 3} = cn^{1.59\ldots}$$

$$\left(\frac{3}{2}^{log_3 n}\right) = \frac{1}{n} 3^{\log_2 n} =$$

$$= 3^{log_3 n} * 3^{\frac{1}{log_3 2}} = n3^{\log_2 3}$$

General formula for recurrences of specific form

Recurrence for Karatsuba's:

$$T(n) = 3T(\tfrac{n}{2}) + cn = cn^{\log_2 3}$$

General form:

$$\boxed{T(n) = qT(\tfrac{n}{2}) + cn = cn^{\log_2 q} \text{ for } q > 2}$$

Discussed in detail: Kleinberg-Tardos pages 214 - 218

History of asymptotic complexity of integer multiplication

Faster than grade-school algorithm for about 320–640 bits.

year	algorithm	order of growth
?	**grade-school**	$\Theta(n^2)$
1962	**Karatsuba–Ofman**	$\Theta(n^{1.585})$
1963	**Toom-3, Toom-4**	$\Theta(n^{1.465})$, $\Theta(n^{1.404})$
1966	**Toom–Cook**	$\Theta(n^{1+\varepsilon})$
1971	**Schönhage–Strassen**	$\Theta(n \log n \log \log n)$
2007	**Fürer**	$n \log n \, 2^{O(\log^* n)}$
2019	**Harvey, van der Hoeven**	$\Theta(n \log n)$

number of bit operations to multiply two n-bit integers

Remark. GNU Multiple Precision Library uses one of five different algorithms depending on size of operands.