Divide and Conquer

Textbook Reading

Chapters 4, 7 & 33.4

Overview

Design principle

- Divide and conquer

Proof technique

- Induction, induction, induction

Analysis technique

- Recurrence relations

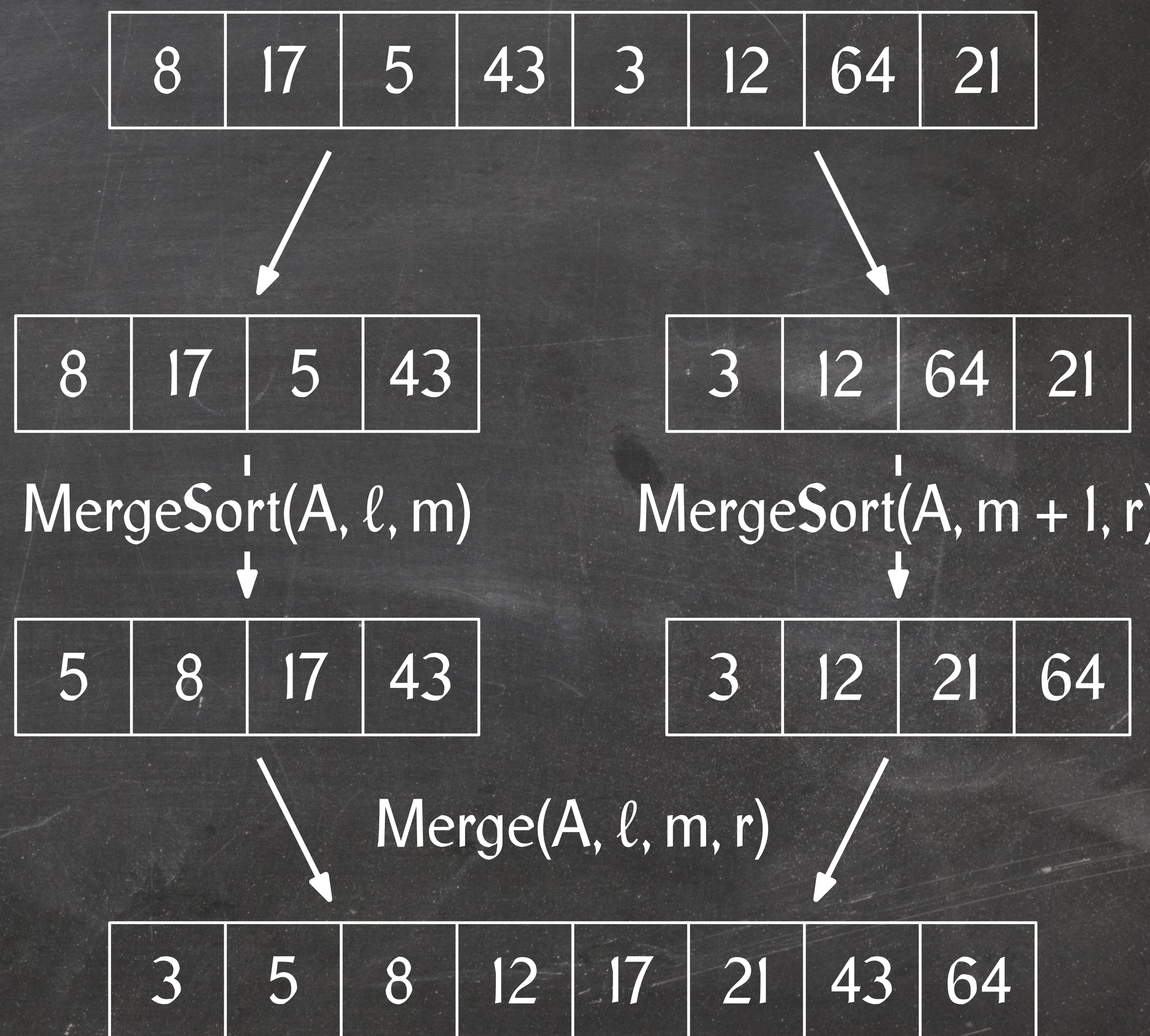
Problems

- Sorting (Merge Sort and Quick Sort)
- Selection
- Matrix multiplication
- Finding the two closest points

Merge Sort

MergeSort(A, ℓ, r)

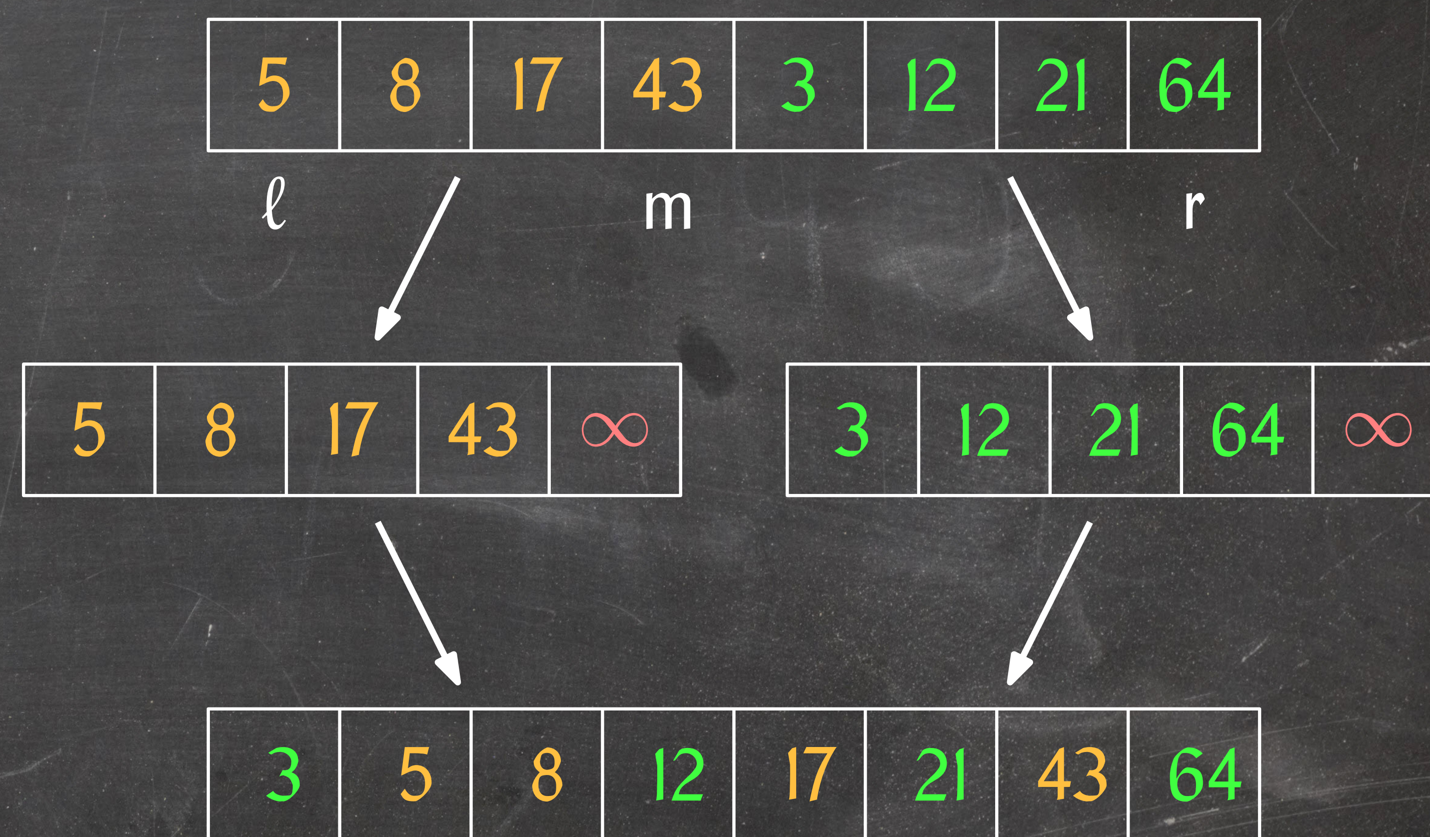
```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```



Merging Two Sorted Lists

Merge(A, ℓ, m, r)

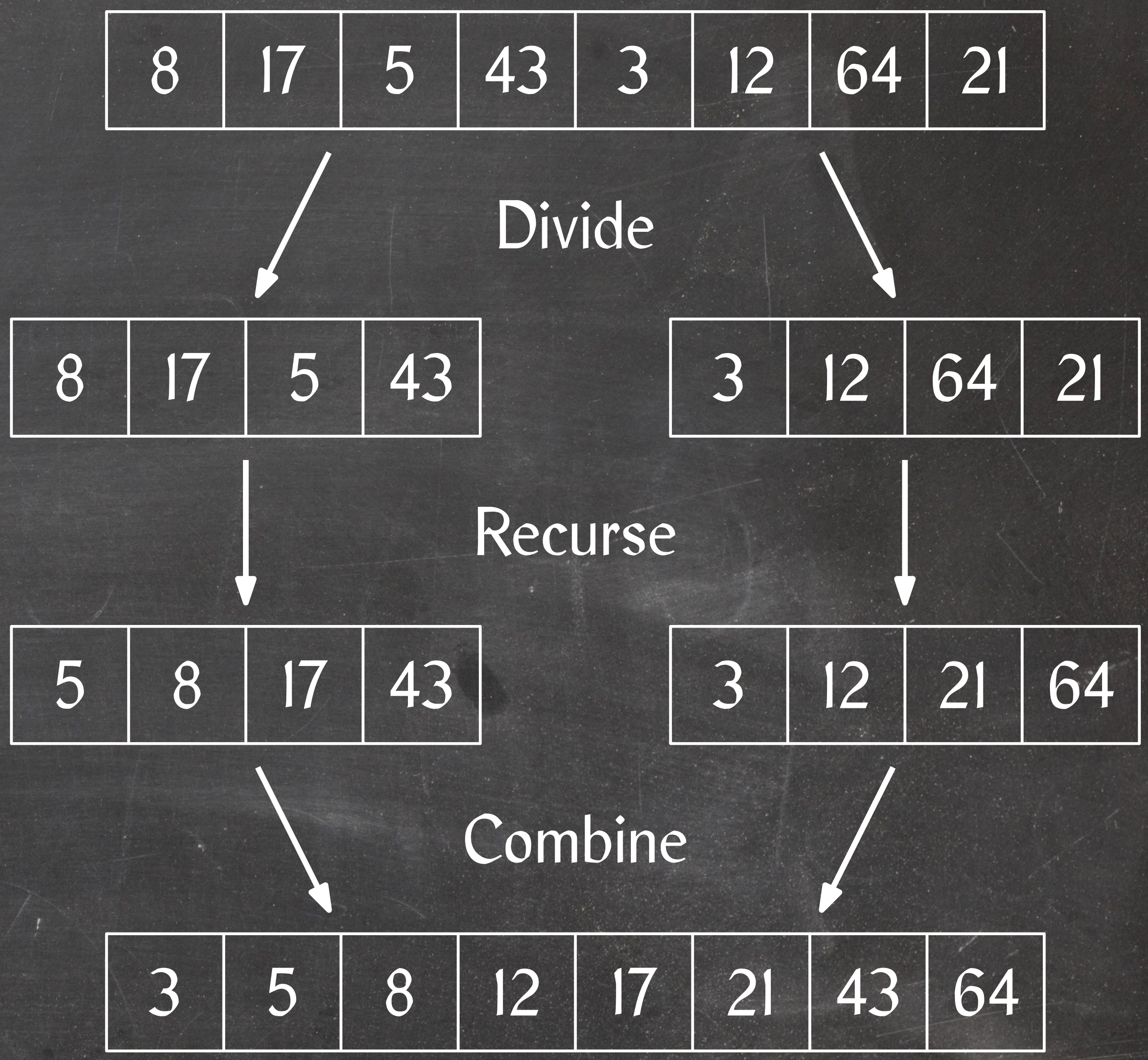
```
n1 = m − ℓ + 1
n2 = r − m
for i = 1 to n1
    do L[i] = A[l + i − 1]
for i = 1 to n2
    do R[i] = A[m + i]
L[n1 + 1] = R[n2 + 1] = +∞
i = j = 1
for k = ℓ to r
    do if L[i] ≤ R[j]
          then A[k] = L[i]
               i = i + 1
          else A[k] = R[j]
               j = j + 1
```



Divide and Conquer

Three steps:

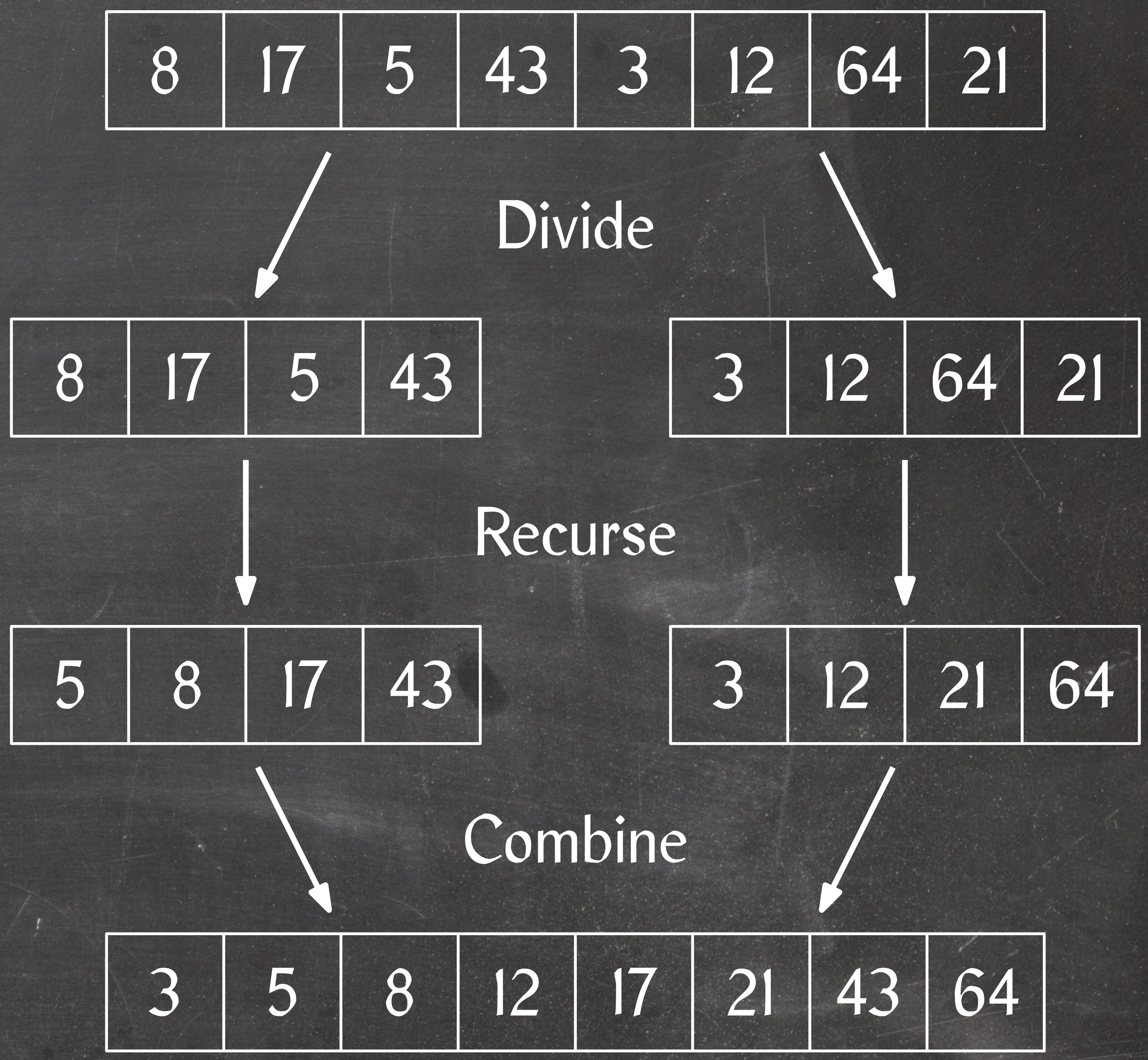
- Divide the input into smaller parts.
- Recursively solve the same problem on these smaller parts.
- Combine the solutions computed by the recursive calls to obtain the final solution.



Divide and Conquer

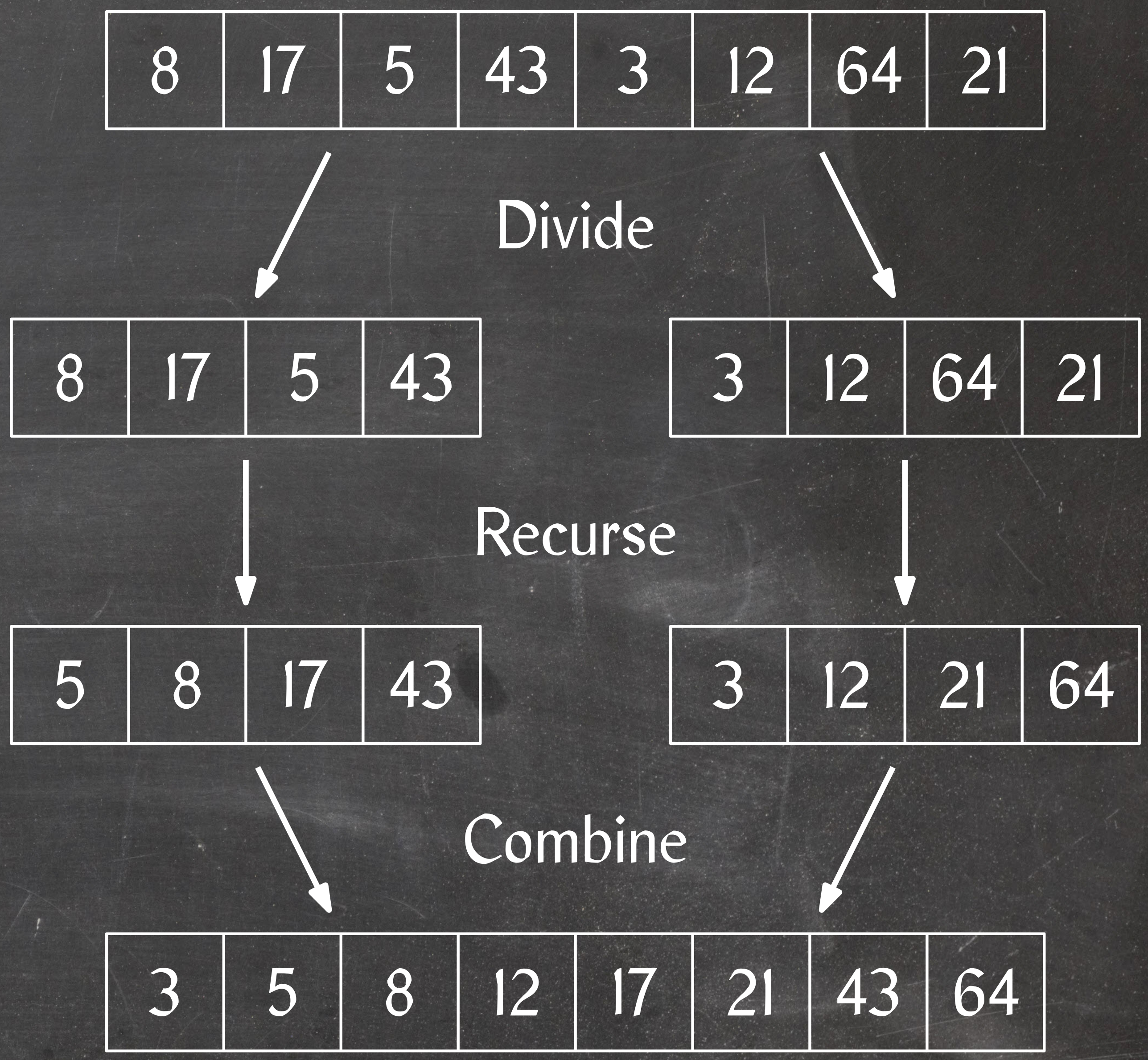
Three steps:

- Divide the input into smaller parts.
- Recursively solve the same problem on these smaller parts.
- Combine the solutions computed by the recursive calls to obtain the final solution.



This can be viewed as a reduction technique, reducing the original problem to simpler problems to be solved in the divide and/or combine steps.

Divide and Conquer



Three steps:

- Divide the input into smaller parts.
- Recursively solve the same problem on these smaller parts.
- Combine the solutions computed by the recursive calls to obtain the final solution.

This can be viewed as a reduction technique, reducing the original problem to simpler problems to be solved in the divide and/or combine steps.

Example: Once we unfold the recursion of Merge Sort, we're left with nothing but Merge steps. Thus, we reduce sorting to the simpler problem of merging sorted lists.

Loop Invariants

… are a technique for proving the correctness of an iterative algorithm.

The invariant states conditions that should hold before and after each iteration.

Correctness proof using a loop invariant:

Initialization: Prove the invariant holds before the first iteration.

Maintenance: Prove each iteration maintains the invariant.

Termination: Prove that the correctness of the invariant after the last iteration implies correctness of the algorithm.

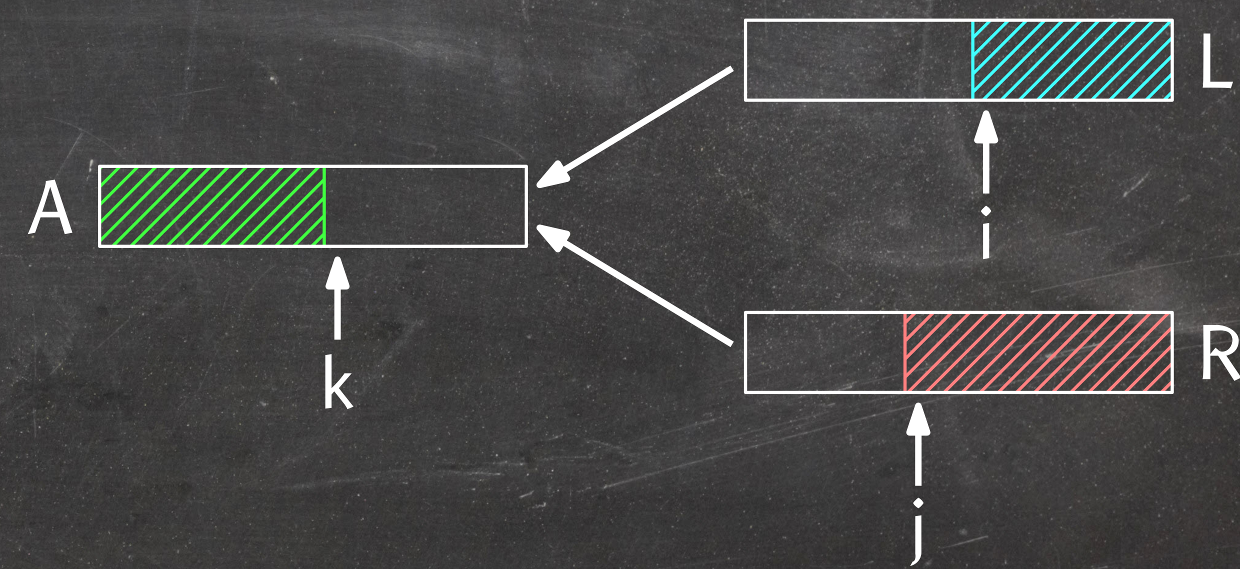
Correctness of Merge

Merge(A, ℓ, m, r)

```
n₁ = m − ℓ + 1
n₂ = r − m
for i = 1 to n₁
    do L[i] = A[l + i − 1]
for i = 1 to n₂
    do R[i] = A[m + i]
L[n₁ + 1] = R[n₂ + 1] = +∞
i = j = 1
for k = ℓ to r
    do if L[i] ≤ R[j]
          then A[k] = L[i]
               i = i + 1
          else A[k] = R[j]
               j = j + 1
```

Loop invariant:

- $A[\ell\,..\,k-1] \cup L[i\,..\,n_1] \cup R[j\,..\,n_2]$ is the set of elements originally in $A[\ell\,..\,r]$.
- $A[\ell\,..\,k-1]$, $L[i\,..\,n_1]$, and $R[j\,..\,n_2]$ are sorted.
- $x \leq y$ for all $x \in A[\ell\,..\,k-1]$ and $y \in L[i\,..\,n_1] \cup R[j\,..\,n_2]$.



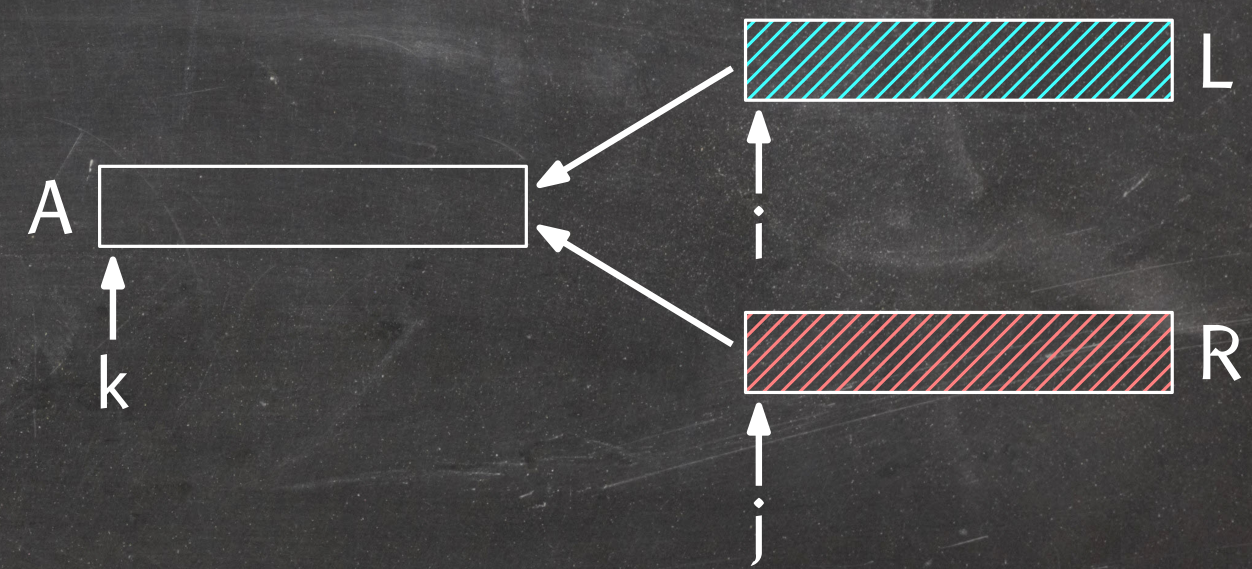
Correctness of Merge

Merge(A, ℓ, m, r)

```
n1 = m − ℓ + 1
n2 = r − m
for i = 1 to n1
    do L[i] = A[l + i − 1]
for i = 1 to n2
    do R[i] = A[m + i]
L[n1 + 1] = R[n2 + 1] = +∞
i = j = 1
for k = ℓ to r
    do if L[i] ≤ R[j]
          then A[k] = L[i]
               i = i + 1
          else A[k] = R[j]
               j = j + 1
```

Initialization:

- $A[\ell \,..\, m]$ is copied to $L[1 \,..\, n_1]$.
- $A[m+1 \,..\, r]$ is copied to $R[1 \,..\, n_2]$.
- $i = 1$, $j = 1$, $k = 1$.



Correctness of Merge

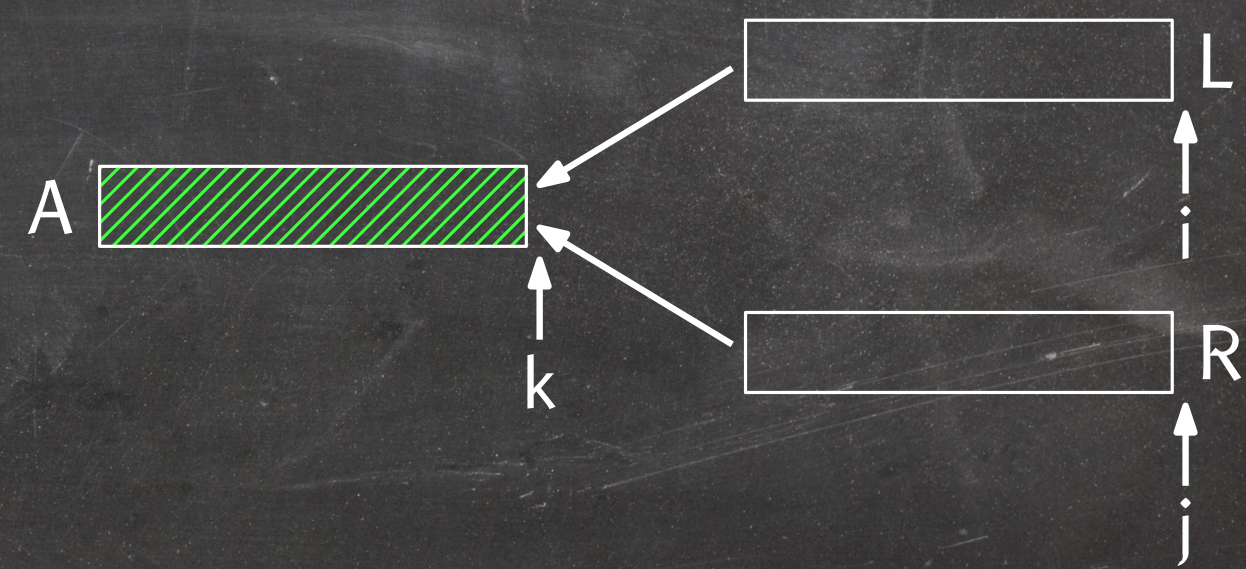
Merge(A, ℓ, m, r)

```
n1 = m − ℓ + 1
n2 = r − m
for i = 1 to n1
    do L[i] = A[l + i − 1]
for i = 1 to n2
    do R[i] = A[m + i]
L[n1 + 1] = R[n2 + 1] = +∞
i = j = 1
for k = ℓ to r
    do if L[i] ≤ R[j]
          then A[k] = L[i]
               i = i + 1
          else A[k] = R[j]
               j = j + 1
```

Termination:

- $k = r + 1$

$\Rightarrow$ $A[\ell \mathinner{..} r]$ contains all items it contained initially, in sorted order.



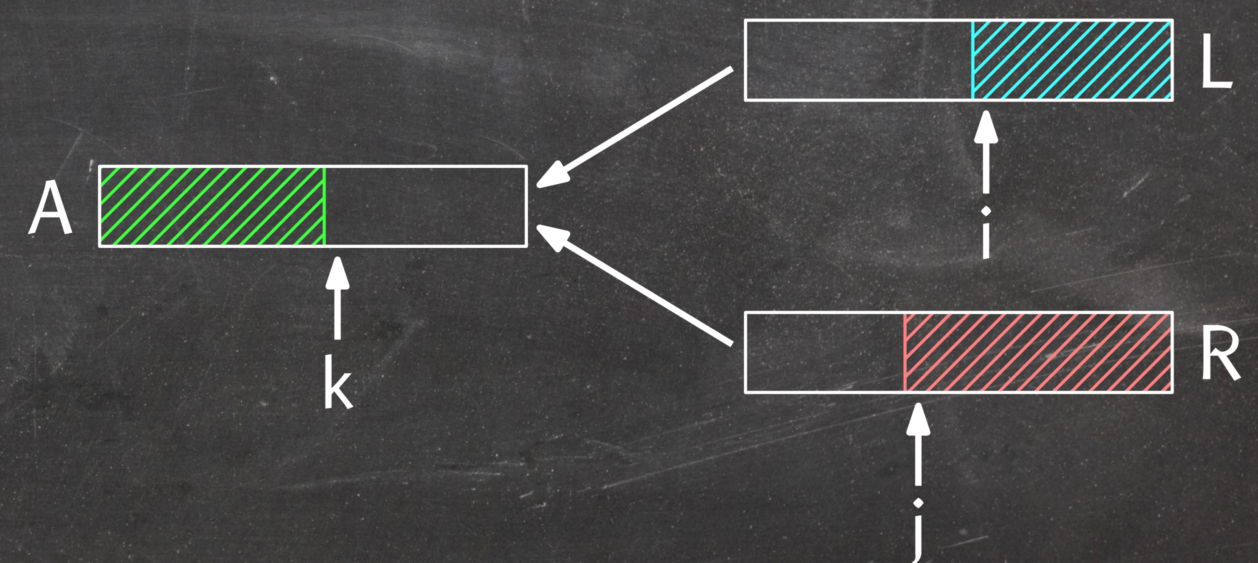
Correctness of Merge

Merge(A, ℓ, m, r)

```
n₁ = m − ℓ + 1
n₂ = r − m
for i = 1 to n₁
    do L[i] = A[l + i − 1]
for i = 1 to n₂
    do R[i] = A[m + i]
L[n₁ + 1] = R[n₂ + 1] = +∞
i = j = 1
for k = ℓ to r
    do if L[i] ≤ R[j]
          then A[k] = L[i]
               i = i + 1
          else A[k] = R[j]
               j = j + 1
```

Maintenance:

- $A[k'] \le L[i]$ for all $k' < k$
- $L[i] \le L[i']$ for all $i' > i$
- $L[i] \le R[j] \le R[j']$ for all $j' > j$



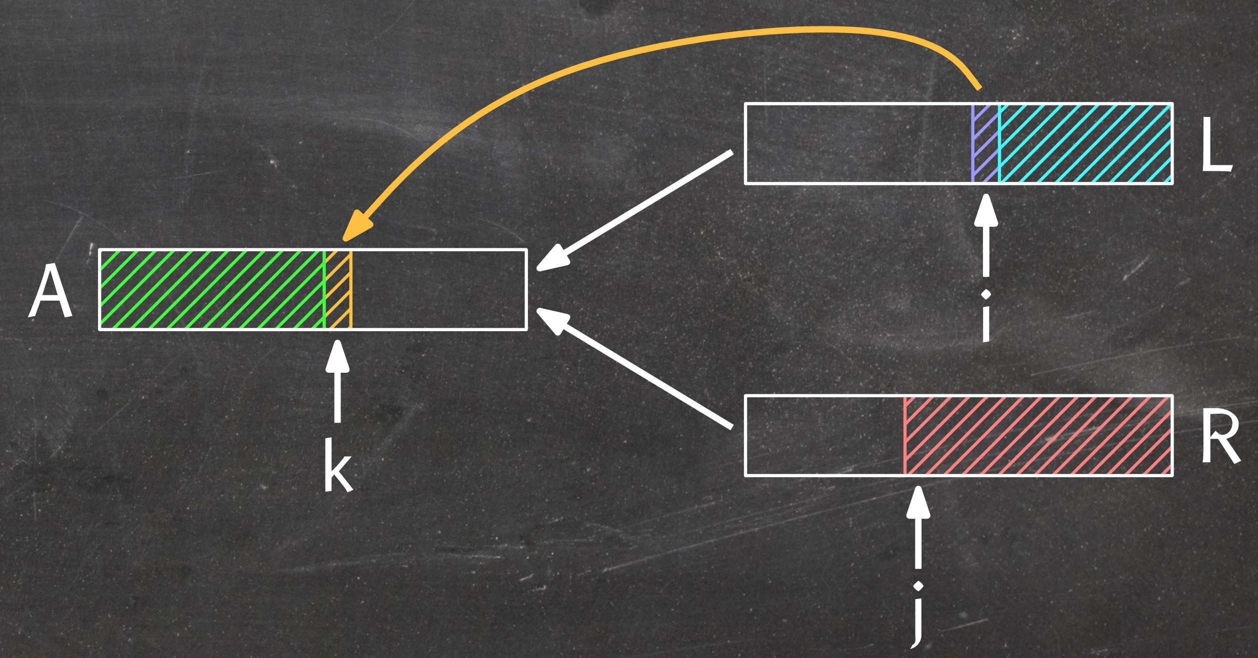
Correctness of Merge

Merge(A, ℓ, m, r)

```
n1 = m − ℓ + 1
n2 = r − m
for i = 1 to n1
    do L[i] = A[l + i − 1]
for i = 1 to n2
    do R[i] = A[m + i]
L[n1 + 1] = R[n2 + 1] = +∞
i = j = 1
for k = ℓ to r
    do if L[i] ≤ R[j]
          then A[k] = L[i]
               i = i + 1
          else A[k] = R[j]
               j = j + 1
```

Maintenance:

- $A[k'] \le L[i]$ for all $k' < k$
- $L[i] \le L[i']$ for all $i' > i$
- $L[i] \le R[j] \le R[j']$ for all $j' > j$



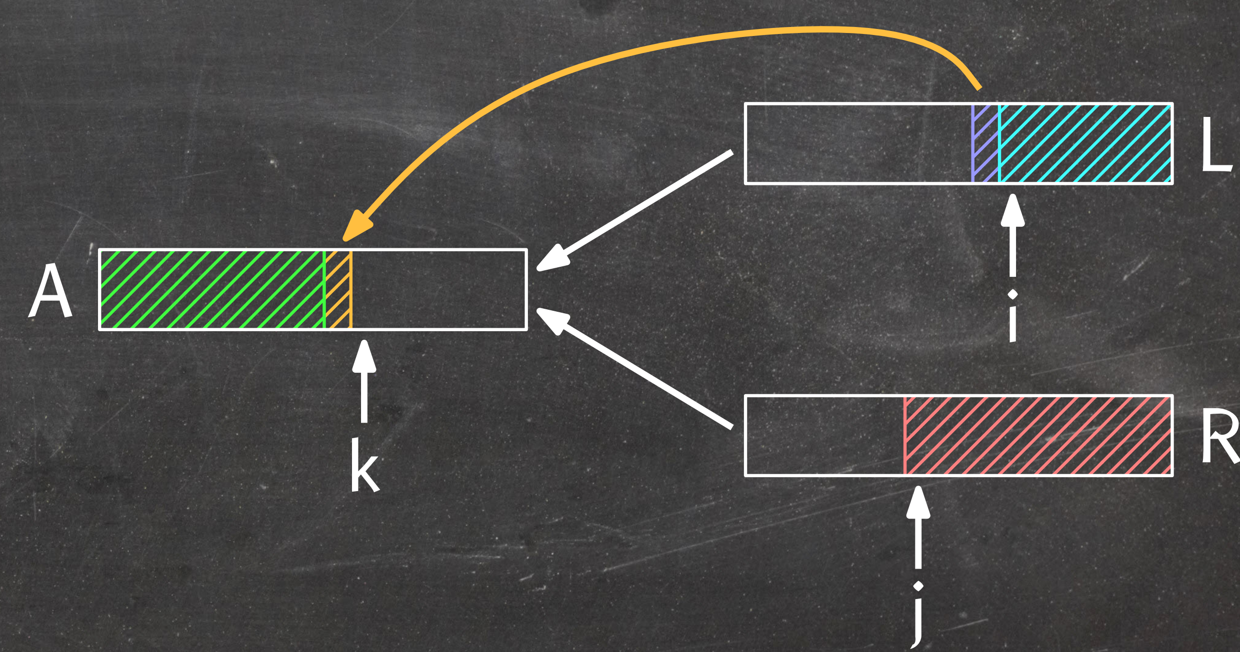
Correctness of Merge

Merge(A, ℓ, m, r)

```
n1 = m − ℓ + 1
n2 = r − m
for i = 1 to n1
    do L[i] = A[l + i − 1]
for i = 1 to n2
    do R[i] = A[m + i]
L[n1 + 1] = R[n2 + 1] = +∞
i = j = 1
for k = ℓ to r
    do if L[i] ≤ R[j]
          then A[k] = L[i]
               i = i + 1
          else A[k] = R[j]
               j = j + 1
```

Maintenance:

- $A[k'] \leq L[i]$ for all $k' < k$
- $L[i] \leq L[i']$ for all $i' > i$
- $L[i] \leq R[j] \leq R[j']$ for all $j' > j$



Correctness of Merge Sort

Lemma: Merge Sort correctly sorts any input array.

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Correctness of Merge Sort

Lemma: Merge Sort correctly sorts any input array.

Proof by induction on n.

MergeSort(A, ℓ, r)

if $r \leq \ell$
    then return
$m = \lfloor(\ell + r)/2\rfloor$
MergeSort(A, ℓ, m)
MergeSort(A, $m + 1$, r)
Merge(A, ℓ, m, r)

Correctness of Merge Sort

Lemma: Merge Sort correctly sorts any input array.

Proof by induction on n.

Base case: ($n = 1$)

- An empty or one-element array is already sorted.
- Merge sort does nothing.

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Correctness of Merge Sort

Lemma: Merge Sort correctly sorts any input array.

Proof by induction on n.

Base case: $(n = 1)$

- An empty or one-element array is already sorted.
- Merge sort does nothing.

Inductive step: $(n > 1)$

- The left and right halves have size less than n each.
- By the inductive hypothesis, the recursive calls sort them correctly.
- Merge correctly merges the two sorted sequences.

MergeSort(A, ℓ, r)

if $r \leq \ell$
    then return
$m = \lfloor (\ell + r)/2 \rfloor$
MergeSort(A, ℓ, m)
MergeSort($A, m + 1, r$)
Merge(A, ℓ, m, r)

Correctness of Divide and Conquer Algorithms

Divide and conquer algorithms are the algorithmic incarnation of induction:

Base case: Solve trivial instances directly, without recursing.

Inductive step: Reduce the solution of a given instance to the solution of smaller instances, by recursing.

Correctness of Divide and Conquer Algorithms

Divide and conquer algorithms are the algorithmic incarnation of induction:

Base case: Solve trivial instances directly, without recursing.

Inductive step: Reduce the solution of a given instance to the solution of smaller instances, by recursing.

$\Rightarrow$ Induction is the natural proof method for divide and conquer algorithms.

Recurrence Relations

A recurrence relation defines the value $f(n)$ of a function $f(\cdot)$ for argument n in terms of the values of $f(\cdot)$ for arguments smaller than n.

Recurrence Relations

A recurrence relation defines the value $f(n)$ of a function $f(\cdot)$ for argument n in terms of the values of $f(\cdot)$ for arguments smaller than n.

Examples:

Fibonacci numbers:

$$F_n = \begin{cases} 1 & n = 0 \text{ or } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Recurrence Relations

A recurrence relation defines the value f(n) of a function f(·) for argument n in terms of the values of f(·) for arguments smaller than n.

Examples:

Fibonacci numbers:

$$F_n = \begin{cases} 1 & n = 0 \text{ or } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

Binomial coefficients

$$B(n, k) = \begin{cases} 1 & k = 1 \text{ or } k = n \\ B(n-1, k-1) + B(n-1, k) & \text{otherwise} \end{cases}$$

A Recurrence for Merge Sort

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Recurrence:

$$T(n) =$$

A Recurrence for Merge Sort

Analysis:

If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n \le 1 \\ & \end{cases}$$

A Recurrence for Merge Sort

Analysis:

If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.

If $n > 1$, we

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \\ & n > 1 \end{cases}$$

A Recurrence for Merge Sort

Analysis:

If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.

If $n > 1$, we

- Spend constant time to determine the middle index m,

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
   then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n \le 1 \\ \Theta(1) & n > 1 \end{cases}$$

A Recurrence for Merge Sort

Analysis:

If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.

If $n > 1$, we

- Spend constant time to determine the middle index m,
- Make one recursive call on the left half, which has size $\lceil n/2 \rceil$,

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n \leq 1 \\ T(\lceil n/2 \rceil) + \Theta(1) & n > 1 \end{cases}$$

A Recurrence for Merge Sort

Analysis:

If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.

If $n > 1$, we

- Spend constant time to determine the middle index m,
- Make one recursive call on the left half, which has size $\lceil n/2 \rceil$,
- Make one recursive call on the right half, which has size $\lfloor n/2 \rfloor$, and

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n \le 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(1) & n > 1 \end{cases}$$

A Recurrence for Merge Sort

Analysis:

If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.

If $n > 1$, we

- Spend constant time to determine the middle index m,
- Make one recursive call on the left half, which has size $\lceil n/2 \rceil$,
- Make one recursive call on the right half, which has size $\lfloor n/2 \rfloor$, and
- Spend linear time to merge the two resulting sorted sequences.

MergeSort(A, ℓ, r)

if $r \le \ell$
    then return
$m = \lfloor (\ell + r)/2 \rfloor$
MergeSort(A, ℓ, m)
MergeSort(A, $m + 1$, r)
Merge(A, ℓ, m, r)

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n \le 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(1) & n > 1 \end{cases}$$

A Recurrence for Merge Sort

Analysis:

If $n = 0$ or $n = 1$, we spend constant time to figure out that there is nothing to do and then exit.

If $n > 1$, we

- Spend constant time to determine the middle index m,
- Make one recursive call on the left half, which has size $\lceil n/2 \rceil$,
- Make one recursive call on the right half, which has size $\lfloor n/2 \rfloor$, and
- Spend linear time to merge the two resulting sorted sequences.

MergeSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = ⌊(ℓ + r)/2⌋
MergeSort(A, ℓ, m)
MergeSort(A, m + 1, r)
Merge(A, ℓ, m, r)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n \le 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & n > 1 \end{cases}$$

A Recurrence for Binary Search

BinarySearch(A, ℓ, r, x)

```
if r < ℓ
    then return "no"
m = ⌊(ℓ + r)/2⌋
if x = A[m]
    then return "yes"
if x < A[m]
    then return BinarySearch(A, ℓ, m − 1, x)
    else  return BinarySearch(A, m + 1, r, x)
```

Recurrence:

$$T(n) =$$

A Recurrence for Binary Search

Analysis:

If $n = 0$, we spend constant time to answer "no".

BinarySearch(A, ℓ, r, x)

```
if r < ℓ
    then return "no"
m = ⌊(ℓ + r)/2⌋
if x = A[m]
    then return "yes"
if x < A[m]
    then return BinarySearch(A, ℓ, m − 1, x)
    else return BinarySearch(A, m + 1, r, x)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n = 0 \\ & \end{cases}$$

A Recurrence for Binary Search

Analysis:

If $n = 0$, we spend constant time to answer "no".

If $n > 0$, we

BinarySearch(A, ℓ, r, x)

```
if r < ℓ
    then return "no"
m = ⌊(ℓ + r)/2⌋
if x = A[m]
    then return "yes"
if x < A[m]
    then return BinarySearch(A, ℓ, m − 1, x)
    else return BinarySearch(A, m + 1, r, x)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n = 0 \\ & n > 0 \end{cases}$$

A Recurrence for Binary Search

Analysis:

If $n = 0$, we spend constant time to answer "no".

If $n > 0$, we

- Spend constant time to find the middle element and compare it to x.

BinarySearch(A, ℓ, r, x)

```
if r < ℓ
    then return "no"
m = ⌊(ℓ + r)/2⌋
if x = A[m]
    then return "yes"
if x < A[m]
    then return BinarySearch(A, ℓ, m − 1, x)
    else return BinarySearch(A, m + 1, r, x)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n = 0 \\ \Theta(1) & n > 0 \end{cases}$$

A Recurrence for Binary Search

Analysis:

If $n = 0$, we spend constant time to answer "no".

If $n > 0$, we

- Spend constant time to find the middle element and compare it to x.
- Make one recursive call on one of the two halves, which has size at most $\lfloor n/2 \rfloor$.

BinarySearch(A, ℓ, r, x)

```
if r < ℓ
    then return "no"
m = ⌊(ℓ + r)/2⌋
if x = A[m]
    then return "yes"
if x < A[m]
    then return BinarySearch(A, ℓ, m − 1, x)
    else return BinarySearch(A, m + 1, r, x)
```

Recurrence:

$$T(n) = \begin{cases} \Theta(1) & n = 0 \\ T(\lfloor n/2 \rfloor) + \Theta(1) & n > 0 \end{cases}$$

Simplified Recurrence Notation

The recurrences we use to analyze algorithms all have a base case of the form

$$T(n) \leq c \ \forall n \leq n_0.$$

The exact choices of c and n_0 affect the value of $T(n)$ for any n by only a constant factor.

Simplified Recurrence Notation

The recurrences we use to analyze algorithms all have a base case of the form

$$T(n) \leq c \;\forall n \leq n_0.$$

The exact choices of c and n_0 affect the value of $T(n)$ for any n by only a constant factor.

Floors and ceilings usually affect the value of $T(n)$ by only a constant factor.

Simplified Recurrence Notation

The recurrences we use to analyze algorithms all have a base case of the form

$$T(n) \leq c \ \forall n \leq n_0.$$

The exact choices of c and n_0 affect the value of $T(n)$ for any n by only a constant factor.

Floors and ceilings usually affect the value of $T(n)$ by only a constant factor.

So we are lazy and write

- Merge Sort: $T(n) = 2T(n/2) + \Theta(n)$
- Binary search: $T(n) = T(n/2) + \Theta(1)$

"Solving" Recurrences

Given two algorithms A and B with running times

$$T_A(n) = 2T(n/2) + \Theta(n)$$
$$T_B(n) = 3T(n/2) + \Theta(\lg n),$$

which one is faster?

"Solving" Recurrences

Given two algorithms A and B with running times

$$T_A(n) = 2T(n/2) + \Theta(n)$$
$$T_B(n) = 3T(n/2) + \Theta(\lg n),$$

which one is faster?

A recurrence for $T(n)$ precisely defines $T(n)$, but it is hard for us to look at the function and say which one grows faster.

"Solving" Recurrences

Given two algorithms A and B with running times

$$T_A(n) = 2T(n/2) + \Theta(n)$$
$$T_B(n) = 3T(n/2) + \Theta(\lg n),$$

which one is faster?

A recurrence for $T(n)$ precisely defines $T(n)$, but it is hard for us to look at the function and say which one grows faster.

$\Rightarrow$ We want a closed-form expression for $T(n)$, that is, one of the form $T(n) \in \Theta(n)$, $T(n) \in \Theta(n^2)$, ..., one that does not depend on $T(n')$ for any $n' < n$.

Methods for Solving Recurrences

Substitution:

- Guess the solution.
  (Intuition, experience, black magic, recursion trees, trial and error)
- Use induction to prove that the guess is correct.

Methods for Solving Recurrences

Substitution:

- Guess the solution.
  (Intuition, experience, black magic, recursion trees, trial and error)
- Use induction to prove that the guess is correct.

Recursion trees:

- Draw a tree that visualizes how the recurrence unfolds.
- Sum up the costs of the nodes in the tree to
  - Obtain an exact answer if the analysis is done rigorously enough or
  - Obtain a guess that can then be verified rigorously using substitution.

Methods for Solving Recurrences

Substitution:

- Guess the solution.
  (Intuition, experience, black magic, recursion trees, trial and error)
- Use induction to prove that the guess is correct.

Recursion trees:

- Draw a tree that visualizes how the recurrence unfolds.
- Sum up the costs of the nodes in the tree to
  - Obtain an exact answer if the analysis is done rigorously enough or
  - Obtain a guess that can then be verified rigorously using substitution.

Master Theorem:

- Cook book recipe for solving common recurrences.
- Immediately tells us the solution after we verify some simple conditions to determine which case of the theorem applies.

Substitution: Merge Sort

Lemma: The running time of Merge Sort is in $O(n \lg n)$.

Substitution: Merge Sort

Lemma: The running time of Merge Sort is in $O(n \lg n)$.

Recurrence:

$$T(n) = 2T(n/2) + O(n)$$

Substitution: Merge Sort

Lemma: The running time of Merge Sort is in $O(n \lg n)$.

Recurrence:

$$T(n) = 2T(n/2) + O(n), \text{ that is,}$$

$$T(n) \le 2T(n/2) + an, \text{ for some } a > 0 \text{ and all } n \ge n_0.$$

Substitution: Merge Sort

Lemma: The running time of Merge Sort is in $O(n \lg n)$.

Recurrence:

$$T(n) = 2T(n/2) + O(n), \text{ that is,}$$
$$T(n) \leq 2T(n/2) + an, \text{ for some } a > 0 \text{ and all } n \geq n_0.$$

Guess:

$$T(n) \leq cn \lg n, \text{ for some } c > 0 \text{ and all } n \geq n_1.$$

Substitution: Merge Sort

Lemma: The running time of Merge Sort is in $O(n \lg n)$.

Recurrence:

$$T(n) = 2T(n/2) + O(n), \text{ that is,}$$
$$T(n) \leq 2T(n/2) + an, \text{ for some } a > 0 \text{ and all } n \geq n_0.$$

Guess:

$$T(n) \leq cn \lg n, \text{ for some } c > 0 \text{ and all } n \geq n_1.$$

Base case:

For $2 \leq n < 4$, $T(n) \leq c' \leq c'n \leq c'n \lg n$, for some $c' > 0$.

Substitution: Merge Sort

Lemma: The running time of Merge Sort is in $O(n \lg n)$.

Recurrence:

$$T(n) = 2T(n/2) + O(n), \text{ that is,}$$
$$T(n) \le 2T(n/2) + an, \text{ for some } a > 0 \text{ and all } n \ge n_0.$$

Guess:

$$T(n) \le cn \lg n, \text{ for some } c > 0 \text{ and all } n \ge n_1.$$

Base case:

For $2 \le n < 4$, $T(n) \le c' \le c'n \le c'n \lg n$, for some $c' > 0$.

$\Rightarrow$ $T(n) \le cn \lg n$ as long as $c \ge c'$.

Substitution: Merge Sort

Inductive step: $(n \geq 4)$

Substitution: Merge Sort

Inductive step: $(n \geq 4)$

$$T(n) \leq 2T(n/2) + an$$

Substitution: Merge Sort

Inductive step: $(n \geq 4)$

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + an \\
&\leq 2 \cdot \left(\frac{cn}{2} \lg \frac{n}{2}\right) + an
\end{aligned}
$$

Substitution: Merge Sort

Inductive step: $(n \geq 4)$

$$\begin{aligned} T(n) &\leq 2T(n/2) + an \\ &\leq 2 \cdot \left(\frac{cn}{2} \lg \frac{n}{2}\right) + an \\ &= cn(\lg n - 1) + an \end{aligned}$$

Substitution: Merge Sort

Inductive step: $(n \geq 4)$

$$\begin{aligned} T(n) &\leq 2T(n/2) + an \\ &\leq 2 \cdot \left(\frac{cn}{2} \lg \frac{n}{2}\right) + an \\ &= cn(\lg n - 1) + an \\ &= cn \lg n + (a - c)n \end{aligned}$$

Substitution: Merge Sort

Inductive step: $(n \geq 4)$

$$\begin{aligned} T(n) &\leq 2T(n/2) + an \\ &\leq 2 \cdot \left(\frac{cn}{2} \lg \frac{n}{2}\right) + an \\ &= cn(\lg n - 1) + an \\ &= cn \lg n + (a - c)n \\ &\leq cn \lg n, \text{ for all } c \geq a. \end{aligned}$$

Substitution: Merge Sort

Inductive step: $(n \geq 4)$

$$\begin{aligned} T(n) &\leq 2T(n/2) + an \\ &\leq 2 \cdot \left(\frac{cn}{2} \lg \frac{n}{2}\right) + an \\ &= cn(\lg n - 1) + an \\ &= cn \lg n + (a - c)n \\ &\leq cn \lg n, \text{ for all } c \geq a. \end{aligned}$$

Notes:

- We only proved the upper bound. The lower bound, $T(n) \in \Omega(n \lg n)$ can be proven analogously.

Substitution: Merge Sort

Inductive step: $(n \geq 4)$

$$\begin{aligned}
T(n) &\leq 2T(n/2) + an \\
&\leq 2 \cdot \left(\frac{cn}{2} \lg \frac{n}{2}\right) + an \\
&= cn(\lg n - 1) + an \\
&= cn \lg n + (a - c)n \\
&\leq cn \lg n, \text{ for all } c \geq a.
\end{aligned}$$

Notes:

- We only proved the upper bound. The lower bound, $T(n) \in \Omega(n \lg n)$ can be proven analogously.
- Since the base case is valid only for $n \geq 2$ and we use the inductive hypothesis for $n/2$ in the inductive step, the inductive step is valid only for $n \geq 4$. Hence, a base case for $2 \leq n < 4$.

Substitution: Binary Search

Lemma: The running time of binary search is in $O(\lg n)$.

Substitution: Binary Search

Lemma: The running time of binary search is in $O(\lg n)$.

Recurrence:

$$T(n) = T(n/2) + O(1), \text{ that is,}$$

$$T(n) \leq T(n/2) + a, \text{ for some } a > 0 \text{ and all } n \geq n_0.$$

Substitution: Binary Search

Lemma: The running time of binary search is in $O(\lg n)$.

Recurrence:

$$T(n) = T(n/2) + O(1), \text{ that is,}$$
$$T(n) \le T(n/2) + a, \text{ for some } a > 0 \text{ and all } n \ge n_0.$$

Guess:

$$T(n) \le c \lg n, \text{ for some } c > 0 \text{ and all } n \ge n_1.$$

Substitution: Binary Search

Lemma: The running time of binary search is in $O(\lg n)$.

Recurrence:

$$T(n) = T(n/2) + O(1), \text{ that is,}$$
$$T(n) \le T(n/2) + a, \text{ for some } a > 0 \text{ and all } n \ge n_0.$$

Guess:

$$T(n) \le c \lg n, \text{ for some } c > 0 \text{ and all } n \ge n_1.$$

Base case:

For $2 \le n < 4$, $T(n) \le c' \le c' \lg n$, for some $c' > 0$.

Substitution: Binary Search

Lemma: The running time of binary search is in $O(\lg n)$.

Recurrence:

$$T(n) = T(n/2) + O(1), \text{ that is,}$$
$$T(n) \leq T(n/2) + a, \text{ for some } a > 0 \text{ and all } n \geq n_0.$$

Guess:

$$T(n) \leq c \lg n, \text{ for some } c > 0 \text{ and all } n \geq n_1.$$

Base case:

For $2 \leq n < 4$, $T(n) \leq c' \leq c' \lg n$, for some $c' > 0$.

$\Rightarrow$ $T(n) \leq c \lg n$ as long as $c \geq c'$.

Substitution: Binary Search

Inductive step: $(n \geq 4)$

Substitution: Binary Search

Inductive step: $(n \geq 4)$

$$T(n) \leq T(n/2) + a$$

Substitution: Binary Search

Inductive step: $(n \geq 4)$

$$\begin{aligned} T(n) &\leq T(n/2) + a \\ &\leq c \lg \frac{n}{2} + a \end{aligned}$$

Substitution: Binary Search

Inductive step: $(n \geq 4)$

$$\begin{aligned} T(n) &\leq T(n/2) + a \\ &\leq c \lg \frac{n}{2} + a \\ &= c(\lg n - 1) + a \end{aligned}$$

Substitution: Binary Search

Inductive step: $(n \geq 4)$

$$\begin{aligned} T(n) &\leq T(n/2) + a \\ &\leq c \lg \frac{n}{2} + a \\ &= c(\lg n - 1) + a \\ &= c \lg n + (a - c) \end{aligned}$$

Substitution: Binary Search

Inductive step: $(n \geq 4)$

$$\begin{aligned}
T(n) &\leq T(n/2) + a \\
&\leq c \lg \frac{n}{2} + a \\
&= c(\lg n - 1) + a \\
&= c \lg n + (a - c) \\
&\leq c \lg n, \text{ for all } c \geq a.
\end{aligned}$$

Substitution and Asymptotic Notation

Why did we expand the Merge Sort recurrence

$$T(n) = 2T(n/2) + O(n) \quad \text{to} \quad T(n) \leq 2T(n/2) + an$$

and the claim

$$T(n) \in O(n \lg n) \quad \text{to} \quad T(n) \leq cn \lg n?$$

Substitution and Asymptotic Notation

Why did we expand the Merge Sort recurrence

$$T(n) = 2T(n/2) + O(n) \quad \text{to} \quad T(n) \le 2T(n/2) + an$$

and the claim

$$T(n) \in O(n \lg n) \quad \text{to} \quad T(n) \le cn \lg n?$$

If we're not careful, we may "prove" nonsensical results:

Recurrence: $T(n) = T(n-1) + n$

Claim: $T(n) \in O(n)$

Note that $T(n) \in \Theta(n^2)$!

Substitution and Asymptotic Notation

Why did we expand the Merge Sort recurrence

$$T(n) = 2T(n/2) + O(n) \quad \text{to} \quad T(n) \le 2T(n/2) + an$$

and the claim

$$T(n) \in O(n \lg n) \quad \text{to} \quad T(n) \le cn \lg n?$$

If we're not careful, we may "prove" nonsensical results:

Recurrence: $T(n) = T(n-1) + n$

Claim: $T(n) \in O(n)$

Base case: $(n = 1)$

$T(n) = T(1) = 1 \in O(n)$

Substitution and Asymptotic Notation

Why did we expand the Merge Sort recurrence

$$T(n) = 2T(n/2) + O(n) \quad \text{to} \quad T(n) \leq 2T(n/2) + an$$

and the claim

$$T(n) \in O(n \lg n) \quad \text{to} \quad T(n) \leq cn \lg n?$$

If we're not careful, we may "prove" nonsensical results:

Recurrence: $T(n) = T(n-1) + n$

Claim: $T(n) \in O(n)$

Base case: $(n = 1)$

$T(n) = T(1) = 1 \in O(n)$

Inductive step: $(n > 1)$

$$\begin{aligned} T(n) &= T(n-1) + n = O(n-1) + n = \\ &\quad O(n) + n = O(n) \end{aligned}$$

Substitution and Asymptotic Notation

Why did we expand the Merge Sort recurrence

$$T(n) = 2T(n/2) + O(n) \quad \text{to} \quad T(n) \le 2T(n/2) + an$$

and the claim

$$T(n) \in O(n \lg n) \quad \text{to} \quad T(n) \le cn \lg n?$$

If we're not careful, we may "prove" nonsensical results:

Recurrence: $T(n) = T(n-1) + n$

Claim: $T(n) \in O(n)$ $\qquad T(n) \le cn$

Base case: $(n = 1)$

$T(n) = T(1) = 1 \in O(n)$

Inductive step: $(n > 1)$

$$\begin{aligned} T(n) = {} & T(n-1) + n = O(n-1) + n = \\ & O(n) + n = O(n) \end{aligned}$$

Substitution and Asymptotic Notation

Why did we expand the Merge Sort recurrence

$$T(n) = 2T(n/2) + O(n) \quad \text{to} \quad T(n) \le 2T(n/2) + an$$

and the claim

$$T(n) \in O(n \lg n) \quad \text{to} \quad T(n) \le cn \lg n?$$

If we're not careful, we may "prove" nonsensical results:

Recurrence: $T(n) = T(n-1) + n$

Claim: $T(n) \in O(n)$ $\qquad$ $T(n) \le cn$

Base case: $(n = 1)$

$T(n) = T(1) = 1 \in O(n)$ $\qquad$ $T(n) = T(1) = c \le cn$

Inductive step: $(n > 1)$

$$\begin{aligned} T(n) &= T(n-1) + n = O(n-1) + n = \\ &\quad O(n) + n = O(n) \end{aligned}$$

Substitution and Asymptotic Notation

Why did we expand the Merge Sort recurrence

$$T(n) = 2T(n/2) + O(n) \quad \text{to} \quad T(n) \le 2T(n/2) + an$$

and the claim

$$T(n) \in O(n \lg n) \quad \text{to} \quad T(n) \le cn \lg n?$$

If we're not careful, we may "prove" nonsensical results:

Recurrence: $T(n) = T(n-1) + n$

Claim: $T(n) \in O(n)$ $\qquad$ $T(n) \le cn$

Base case: $(n = 1)$

$T(n) = T(1) = 1 \in O(n)$ $\qquad$ $T(n) = T(1) = c \le cn$

Inductive step: $(n > 1)$

$T(n) = T(n-1) + n = O(n-1) + n = O(n) + n = O(n)$

$T(n) = T(n-1) + n \le c(n-1) + n = cn + (n - c) > cn!$

A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

Strategy: Expand the recurrence all the way down to the base case

A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

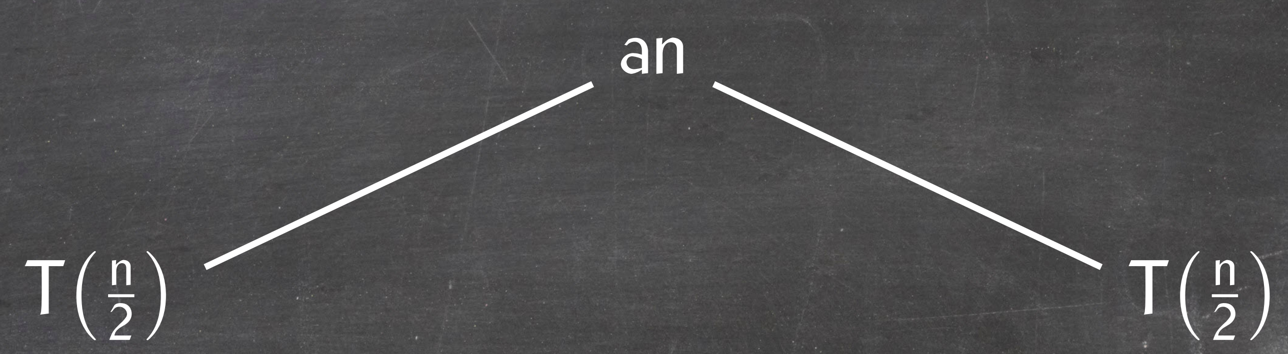
Strategy: Expand the recurrence all the way down to the base case

$$T(n)$$

A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

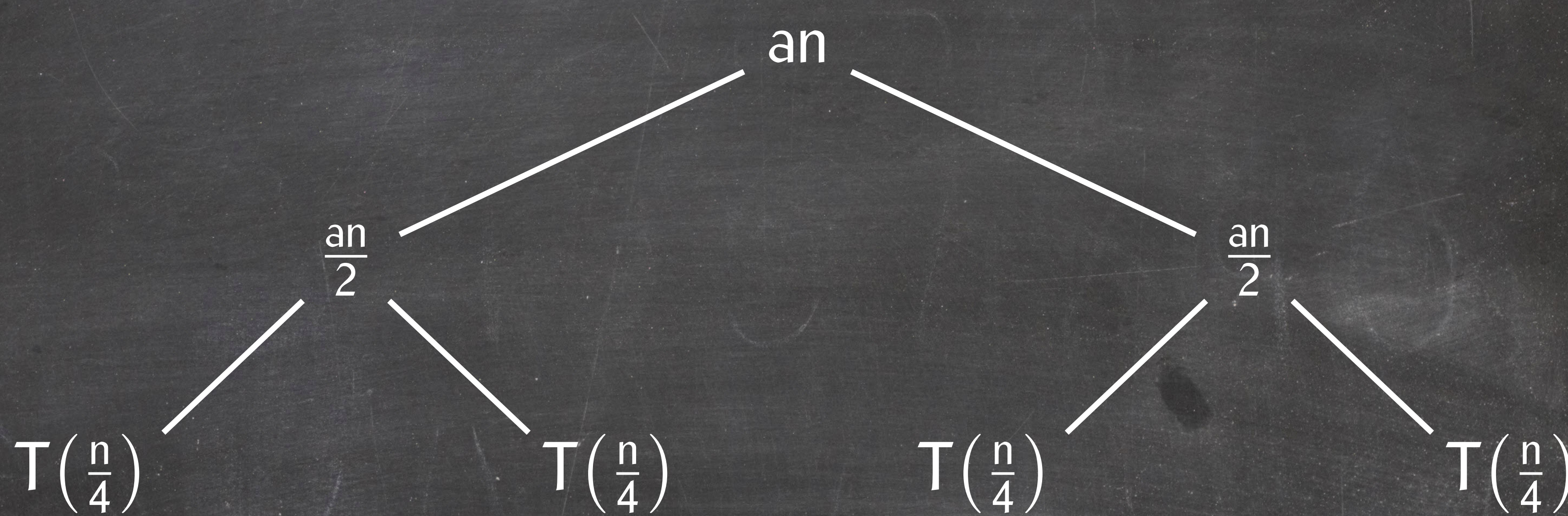
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

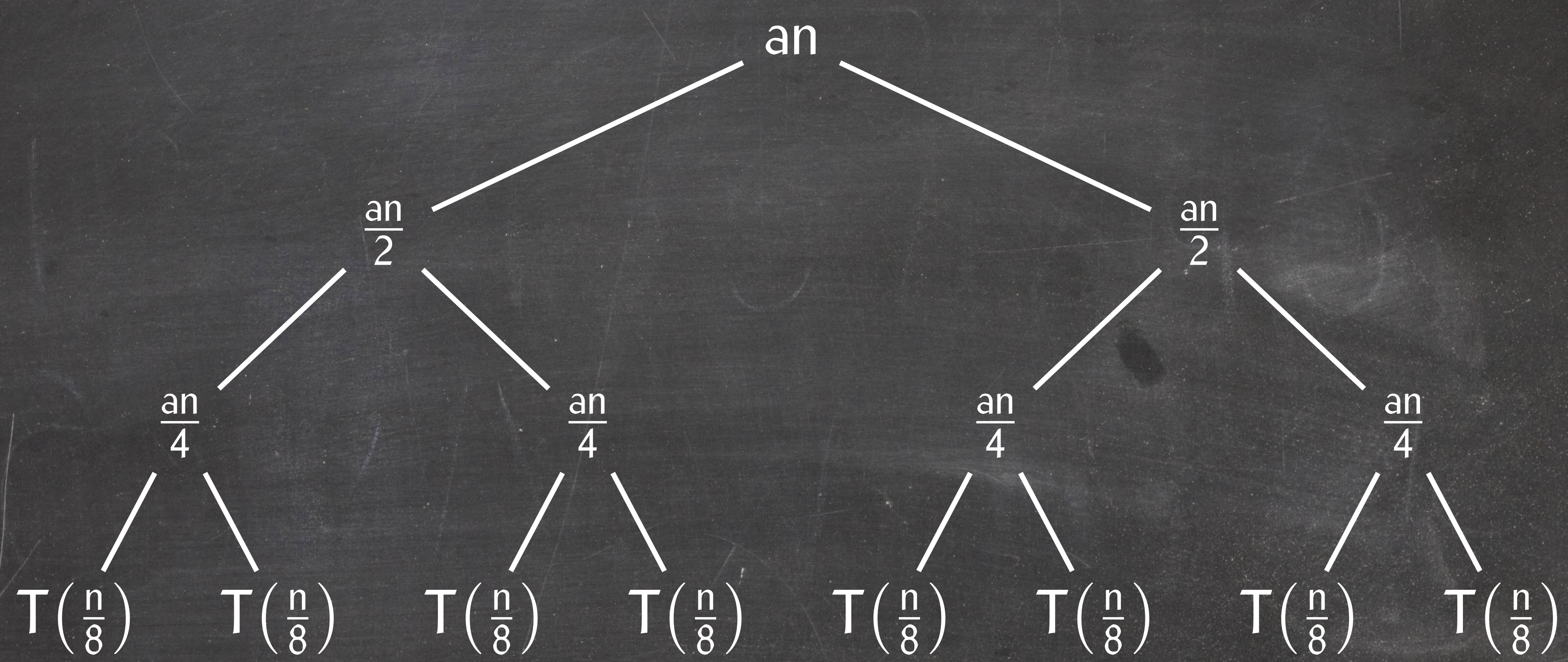
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

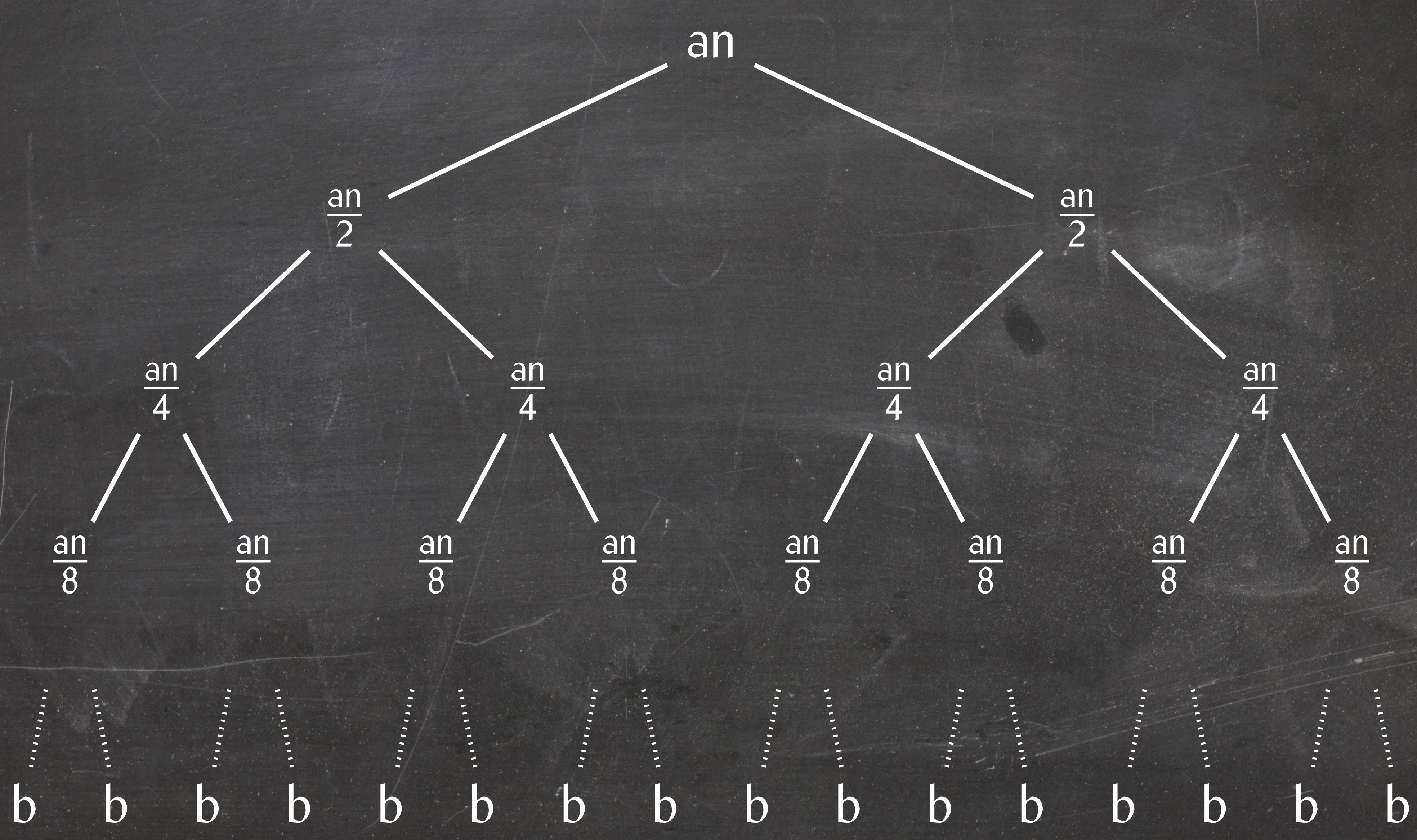
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

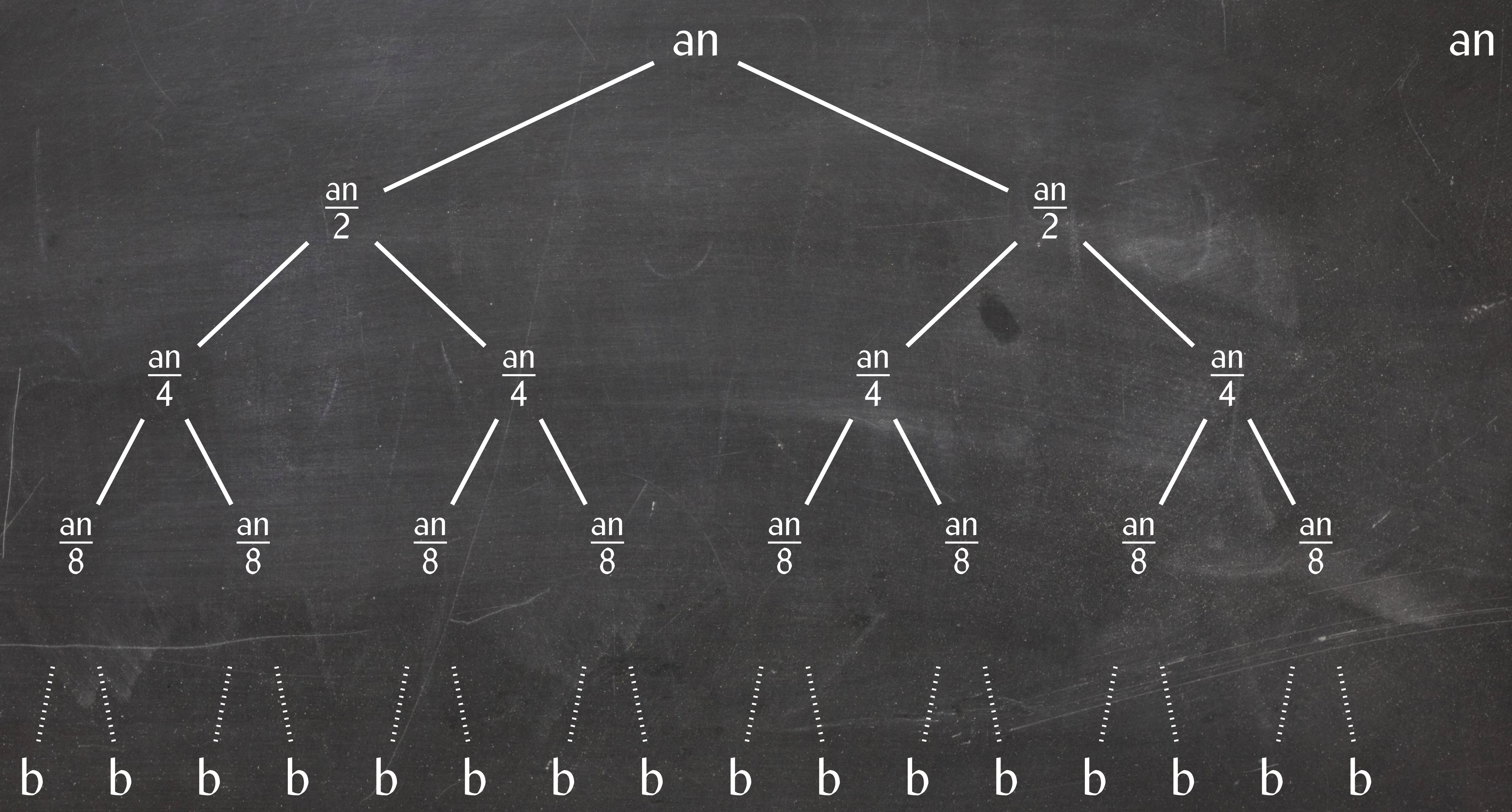
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

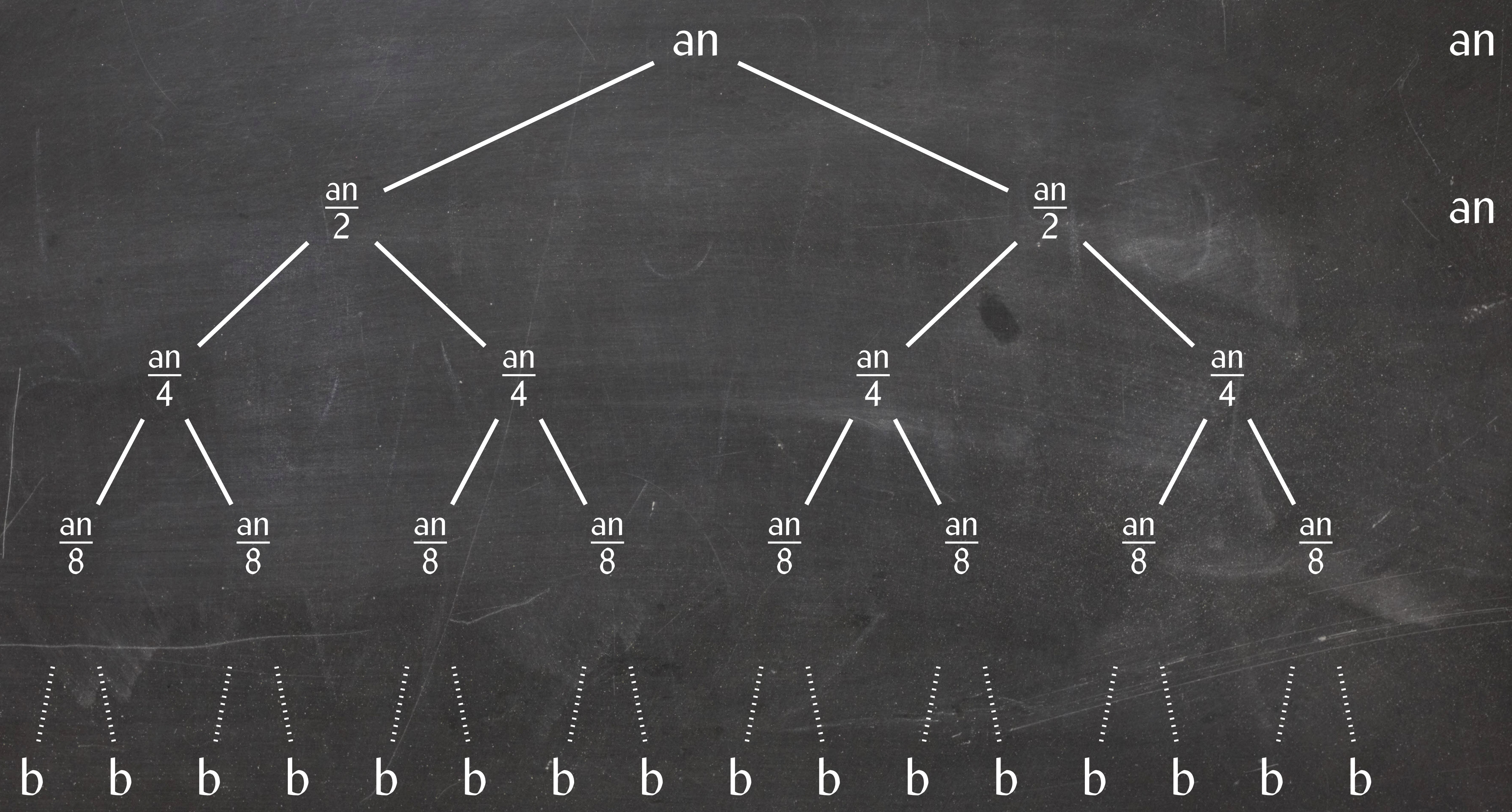
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

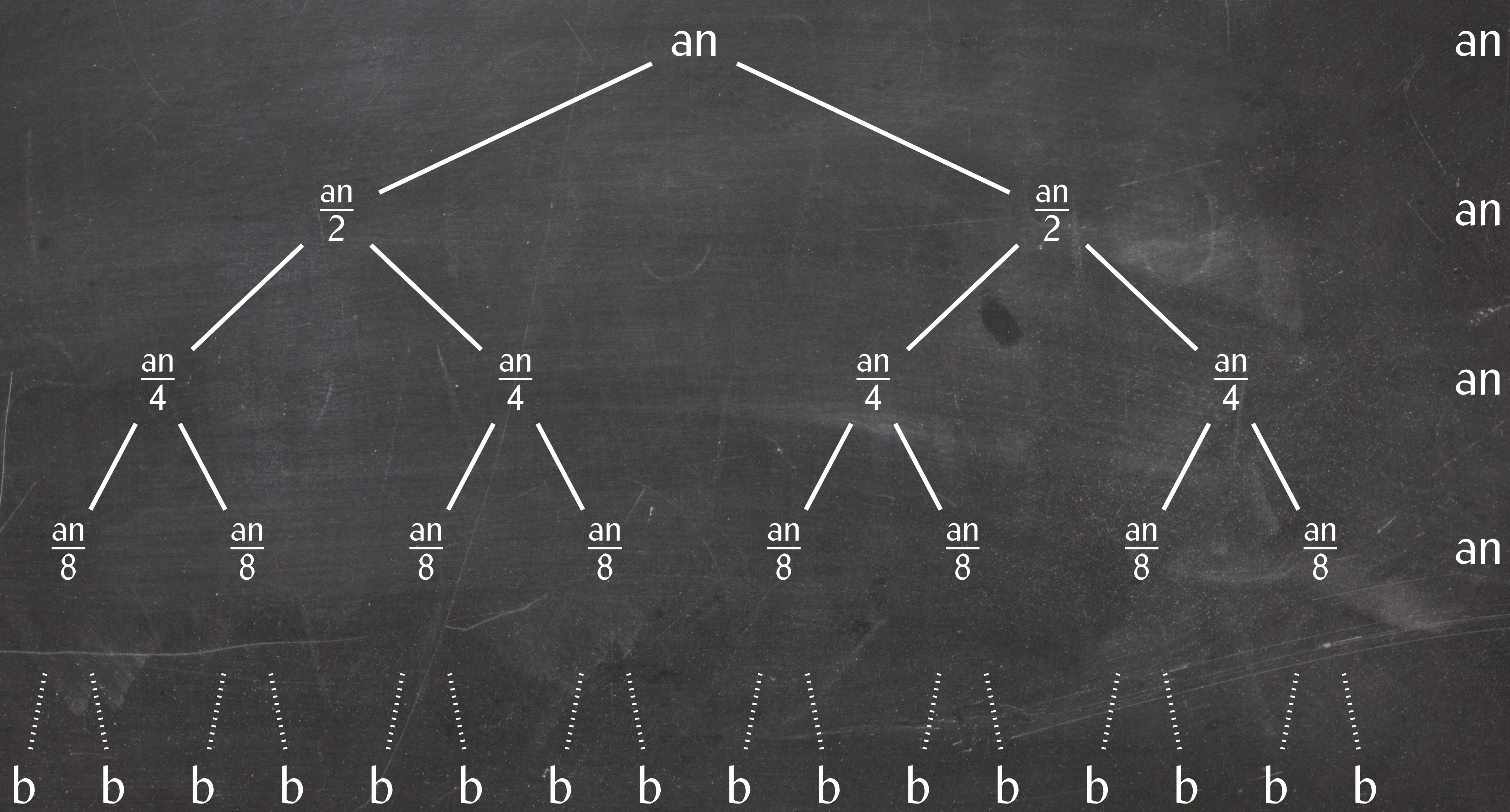
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

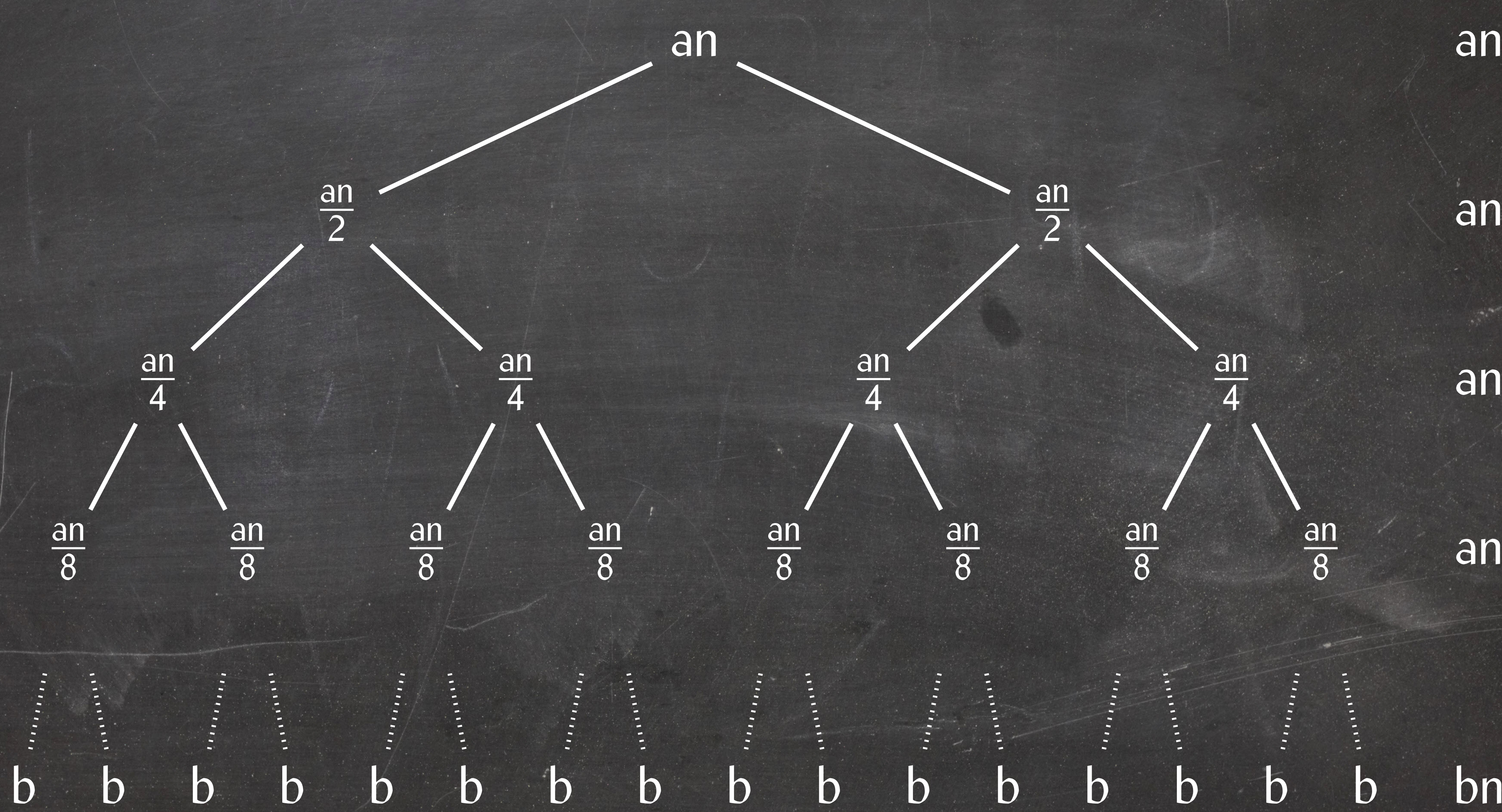
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

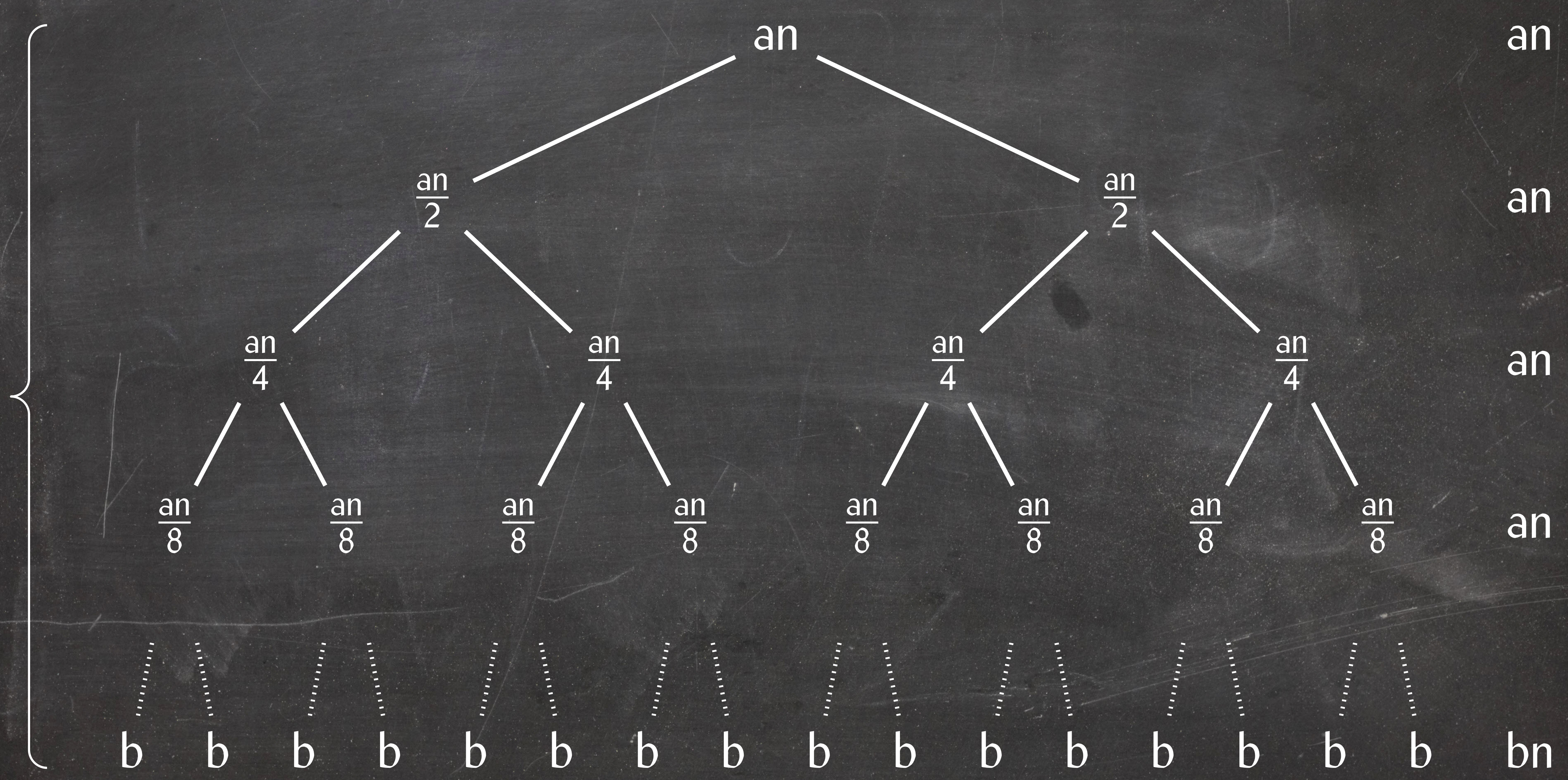
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

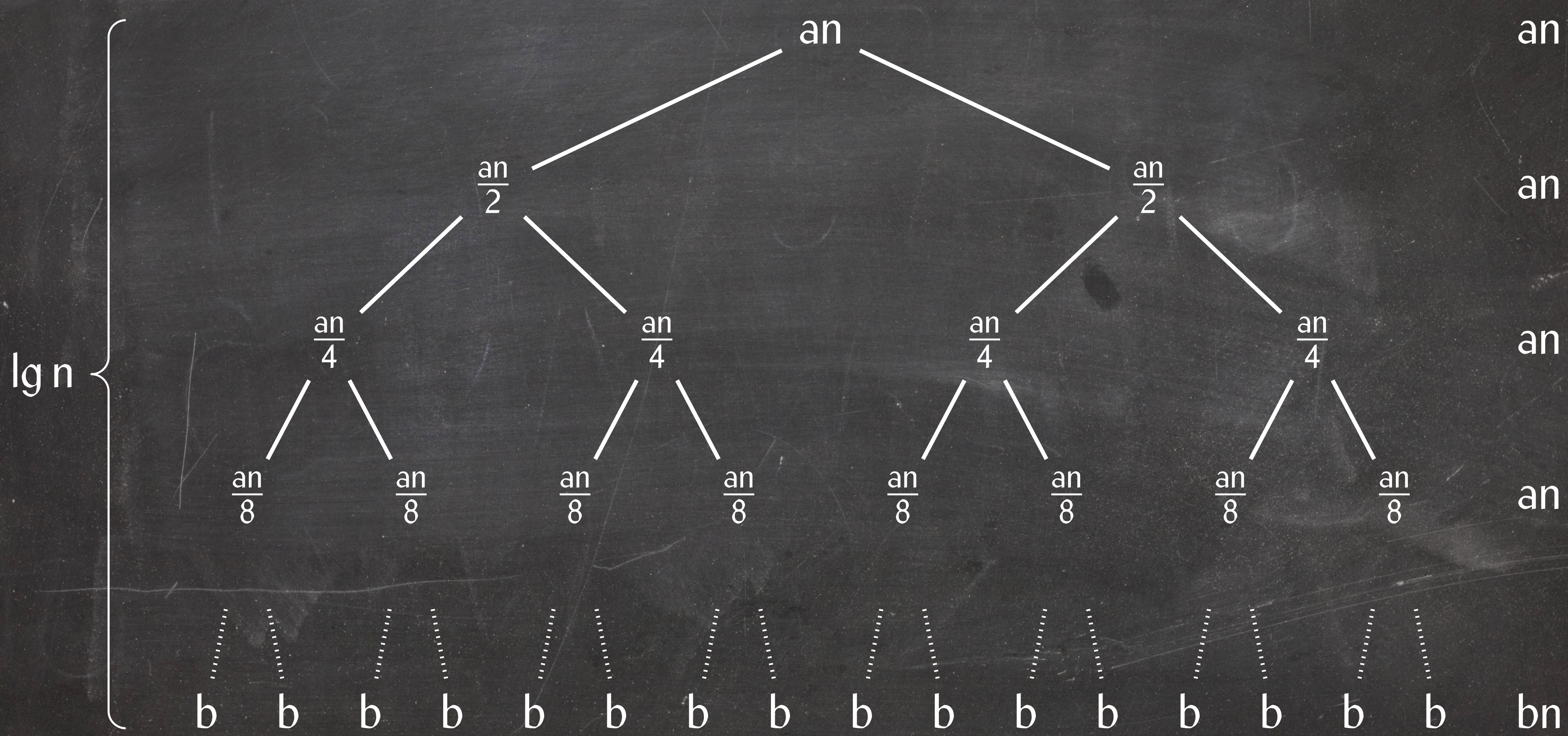
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

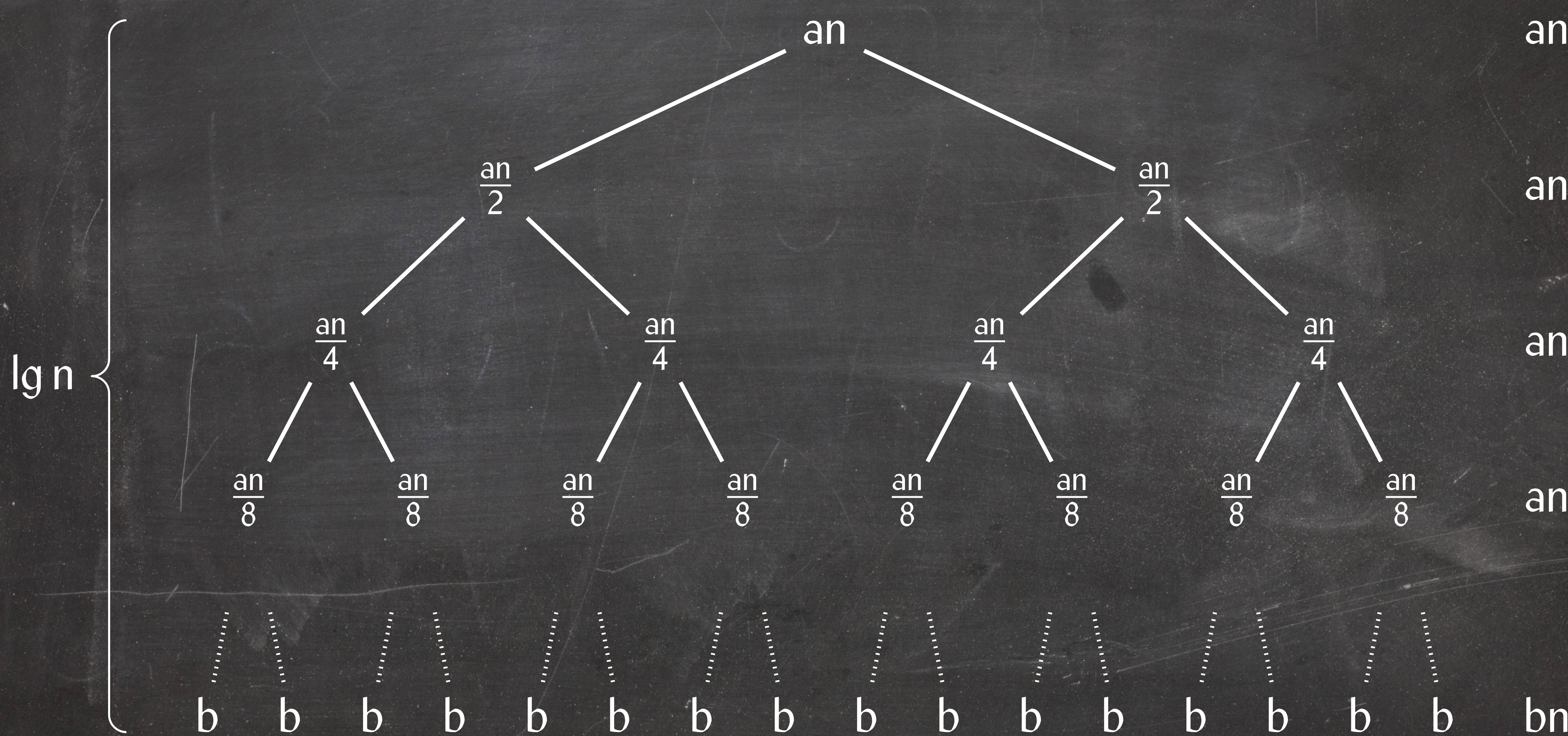
Strategy: Expand the recurrence all the way down to the base case



A Recursion Tree for Merge Sort

Recurrence: $T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = 2T(n/2) + an$

Strategy: Expand the recurrence all the way down to the base case



Solution: $T(n) \in \Theta(n \lg n)$

A Recursion Tree for Binary Search

Recurrence: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = T(n/2) + a$

A Recursion Tree for Binary Search

Recurrence: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = T(n/2) + a$

$$T(n)$$

A Recursion Tree for Binary Search

Recurrence: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = T(n/2) + a$

$$\begin{array}{c} a \\ | \\ T\left(\frac{n}{2}\right) \end{array}$$

A Recursion Tree for Binary Search

Recurrence: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = T(n/2) + a$

$$\begin{array}{c} a \\ | \\ a \\ | \\ T\left(\frac{n}{4}\right) \end{array}$$

A Recursion Tree for Binary Search

Recurrence: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = T(n/2) + a$

$$\begin{array}{c} a \\ | \\ a \\ | \\ a \\ | \\ T\left(\frac{n}{8}\right) \end{array}$$

A Recursion Tree for Binary Search

Recurrence: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = T(n/2) + a$

$$
\begin{array}{c}
a \\
| \\
a \\
| \\
a \\
| \\
a \\
\vdots \\
b
\end{array}
$$

A Recursion Tree for Binary Search

Recurrence: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = T(n/2) + a$

$$\lg n \left\{ \begin{array}{c} a \\ | \\ a \\ | \\ a \\ | \\ a \\ \vdots \\ b \end{array} \right.$$

A Recursion Tree for Binary Search

Recurrence: $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = T(n/2) + a$

$$\lg n \left\{ \begin{array}{c} a \\ | \\ a \\ | \\ a \\ | \\ a \\ \vdots \\ b \end{array} \right.$$

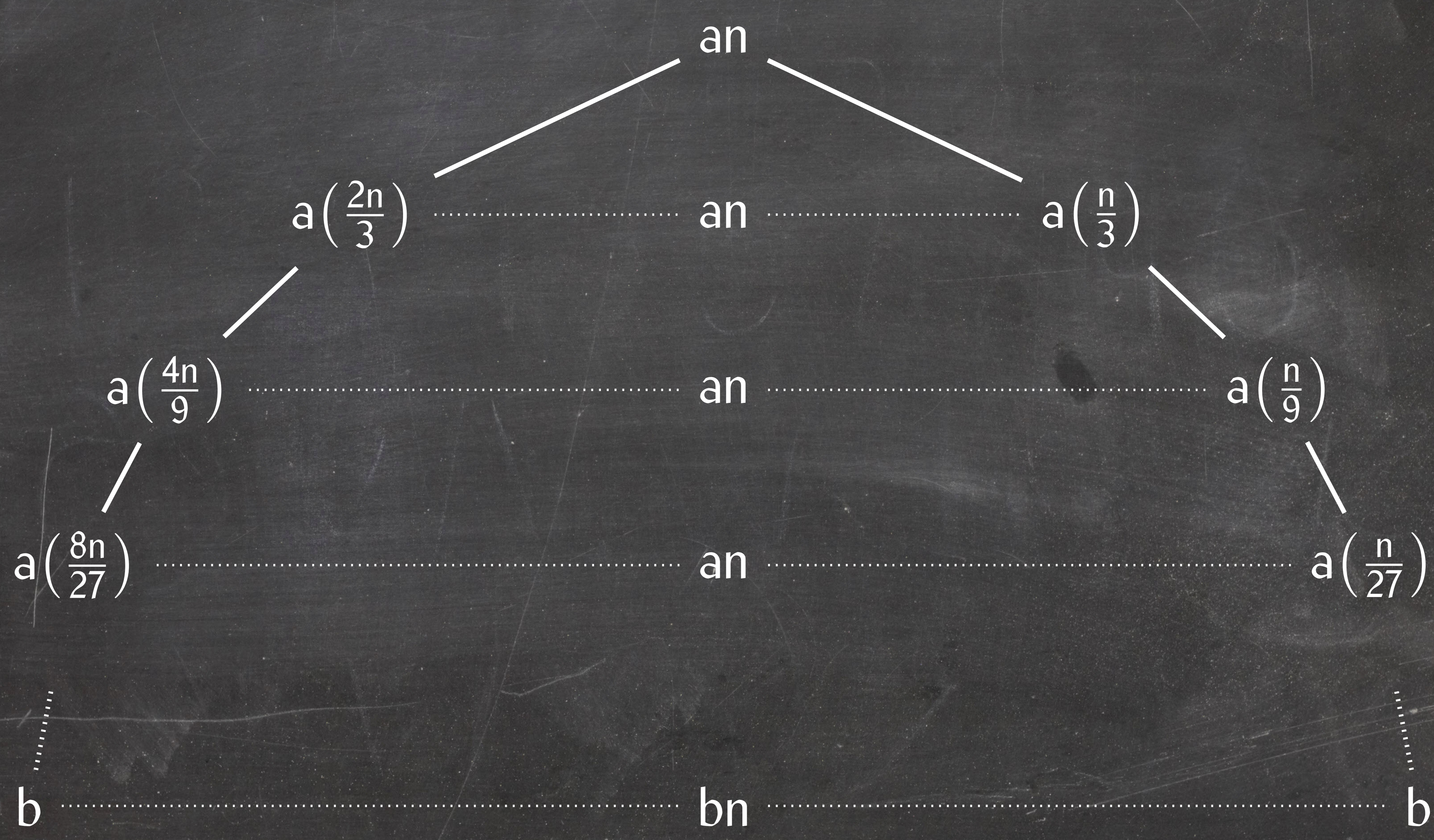
Solution: $T(n) \in \Theta(\lg n)$

A Less Obvious Recursion Tree

Recurrence: $T(n) = T(2n/3) + T(n/3) + \Theta(n) \Rightarrow T(n) = T(2n/3) + T(n/3) + an$

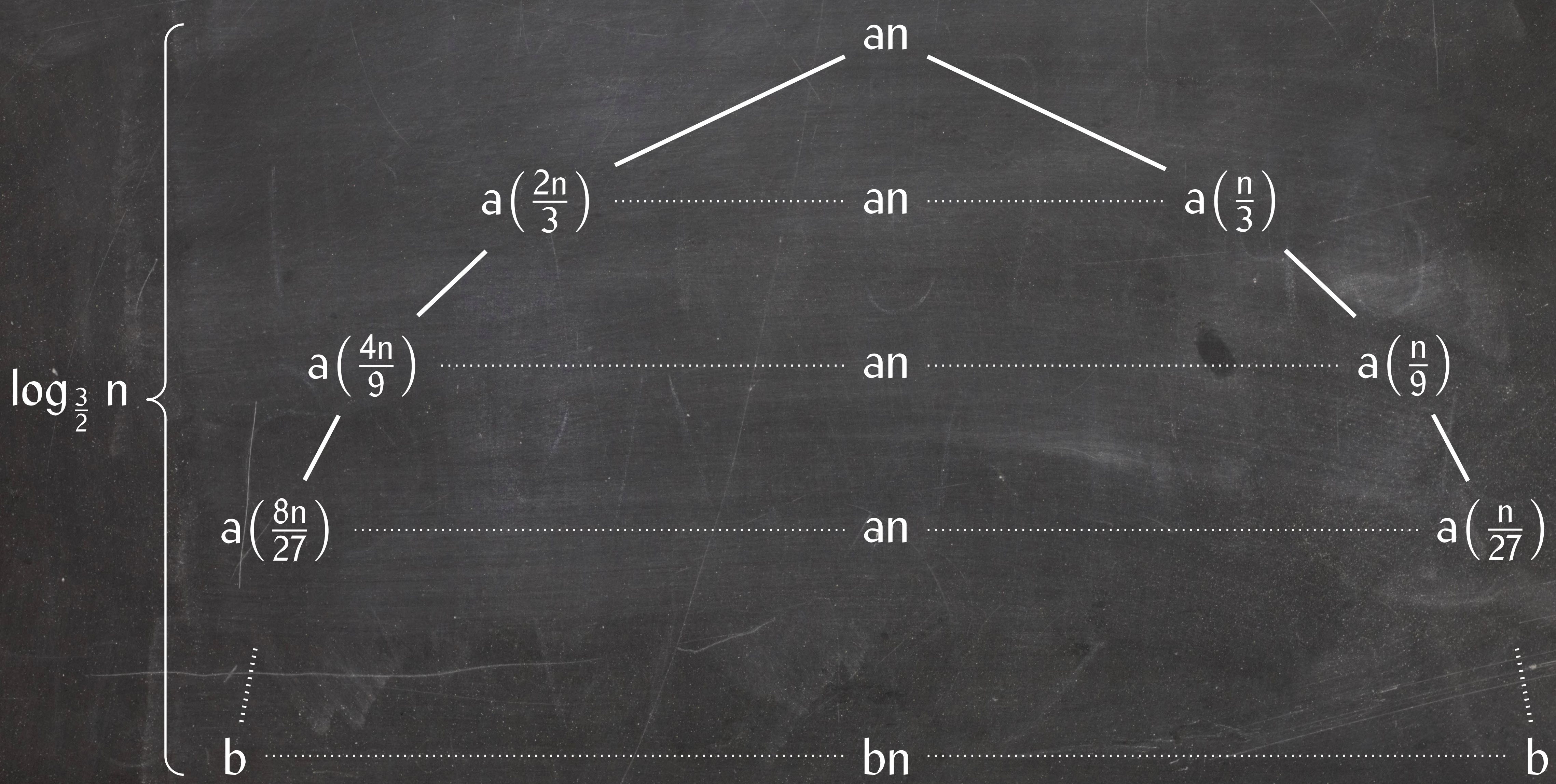
A Less Obvious Recursion Tree

Recurrence: $T(n) = T(2n/3) + T(n/3) + \Theta(n) \Rightarrow T(n) = T(2n/3) + T(n/3) + an$



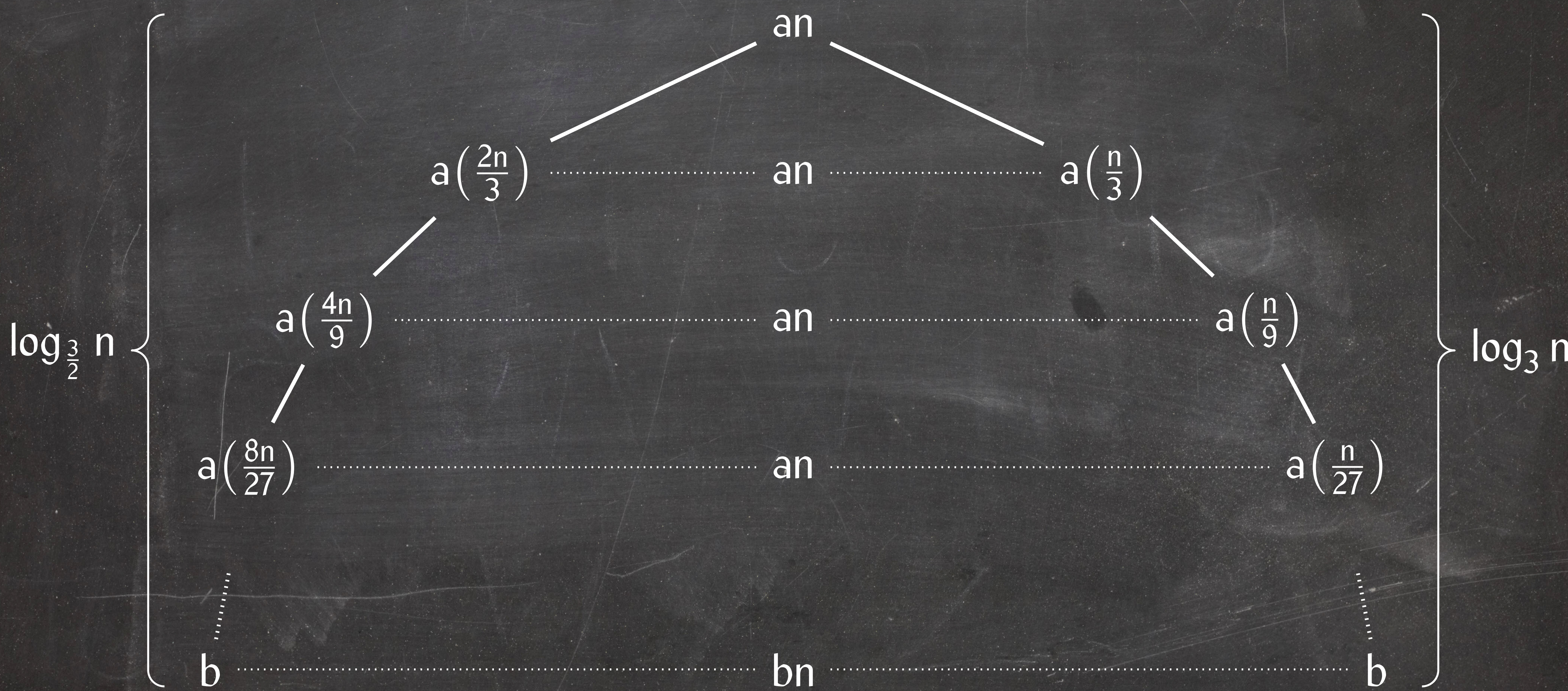
A Less Obvious Recursion Tree

Recurrence: $T(n) = T(2n/3) + T(n/3) + \Theta(n) \Rightarrow T(n) = T(2n/3) + T(n/3) + an$



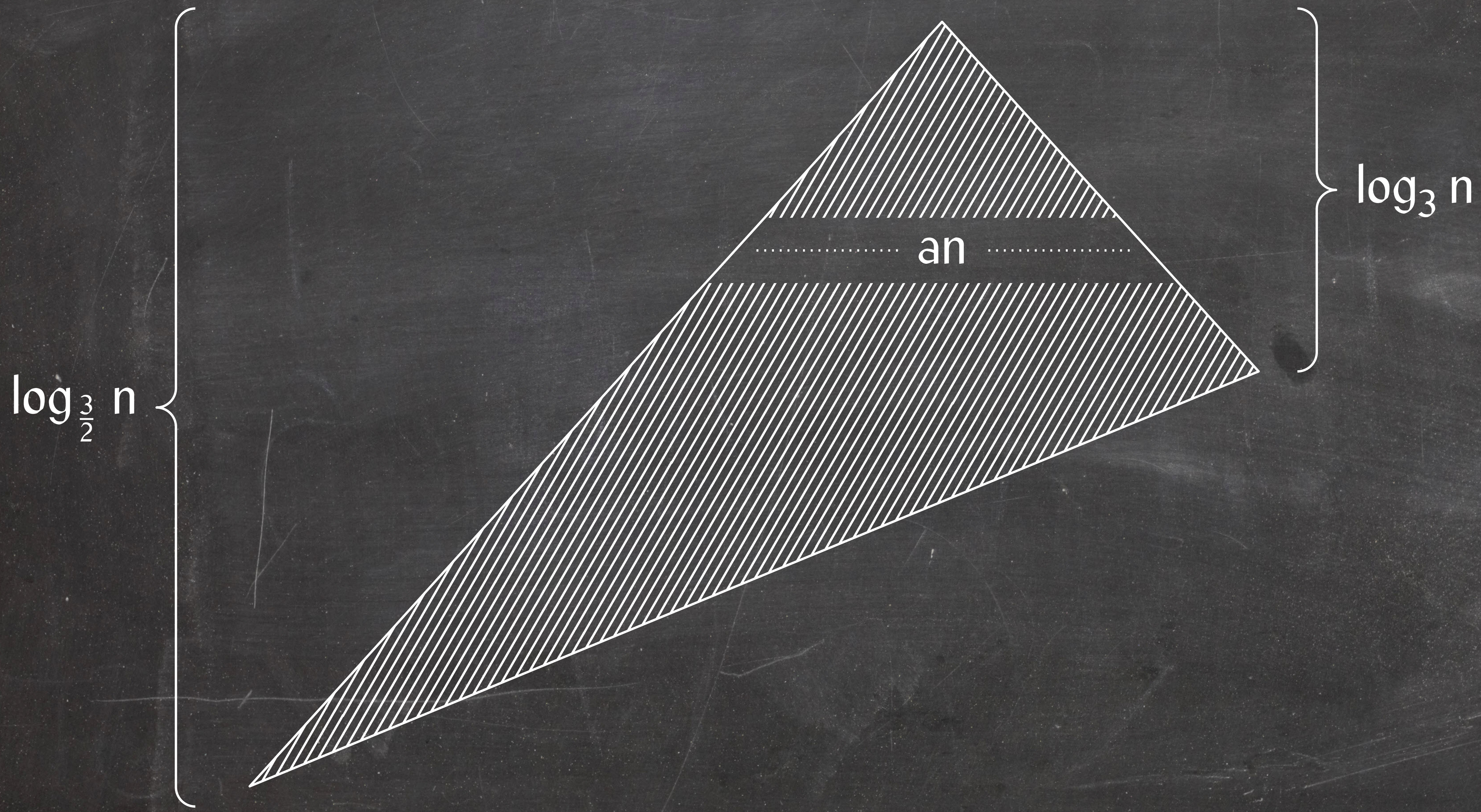
A Less Obvious Recursion Tree

Recurrence: $T(n) = T(2n/3) + T(n/3) + \Theta(n) \Rightarrow T(n) = T(2n/3) + T(n/3) + an$



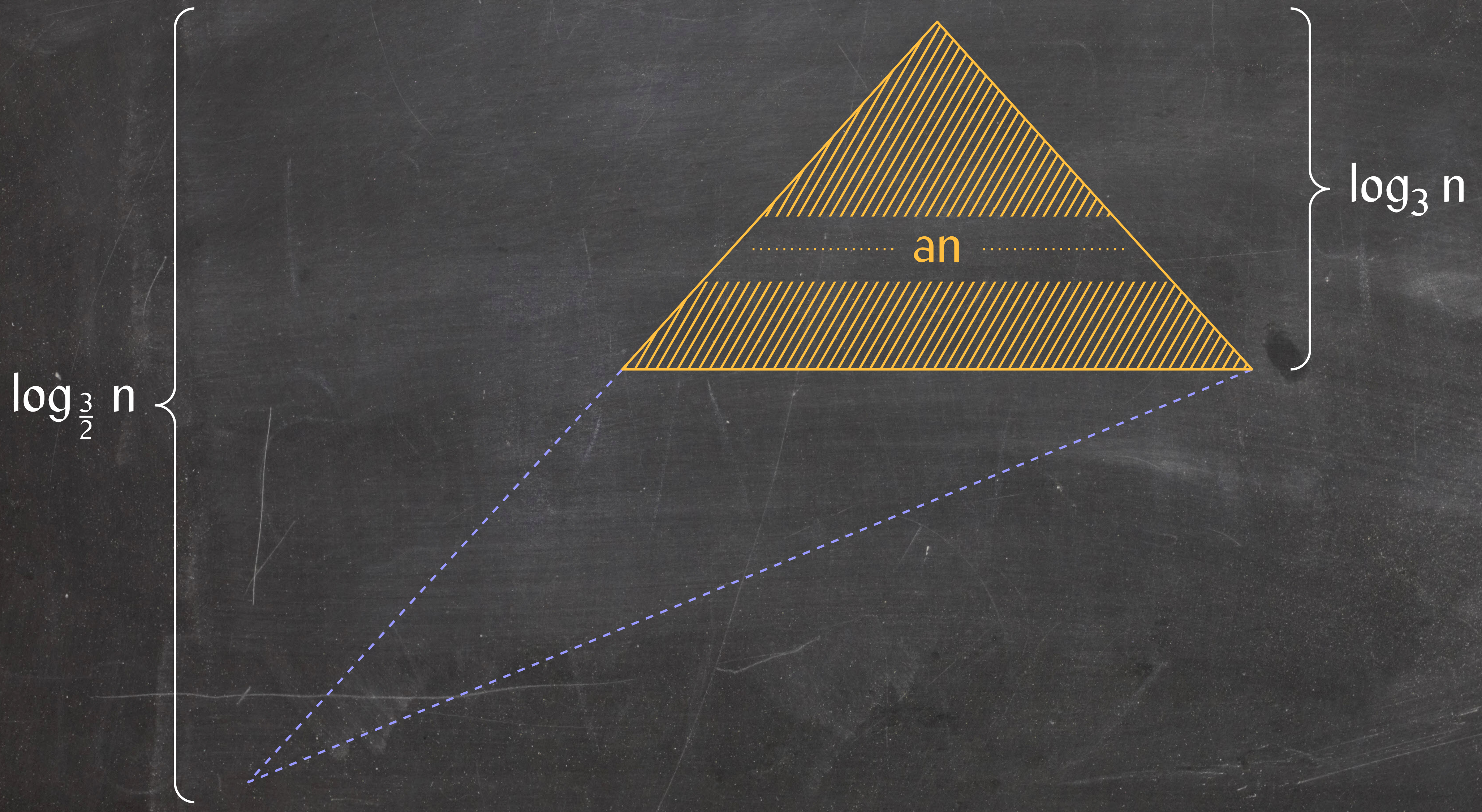
A Less Obvious Recursion Tree

Recurrence: $T(n) = T(2n/3) + T(n/3) + \Theta(n) \Rightarrow T(n) = T(2n/3) + T(n/3) + an$



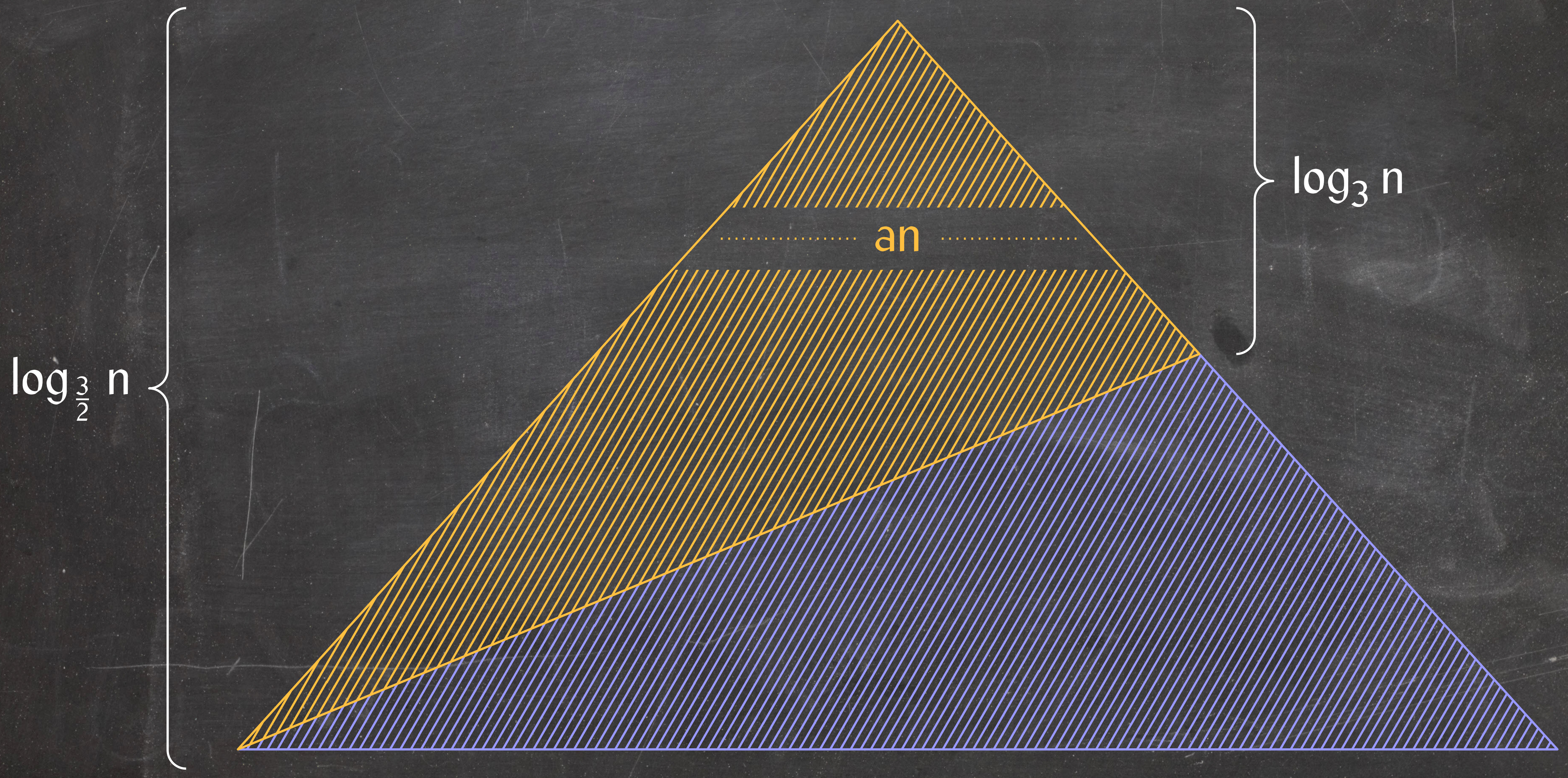
A Less Obvious Recursion Tree

Recurrence: $T(n) = T(2n/3) + T(n/3) + \Theta(n) \Rightarrow T(n) = T(2n/3) + T(n/3) + an$



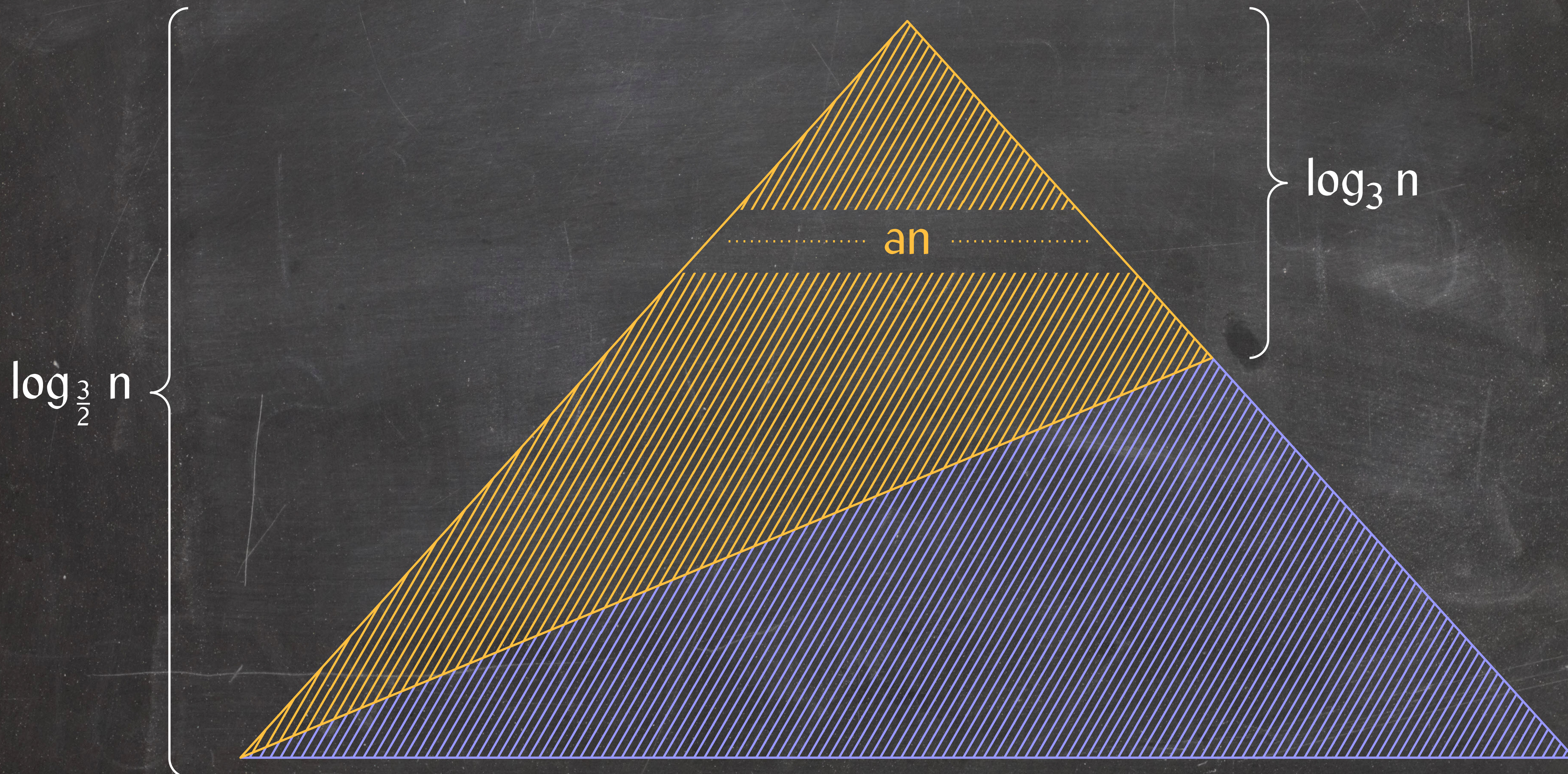
A Less Obvious Recursion Tree

Recurrence: $T(n) = T(2n/3) + T(n/3) + \Theta(n) \Rightarrow T(n) = T(2n/3) + T(n/3) + an$



A Less Obvious Recursion Tree

Recurrence: $T(n) = T(2n/3) + T(n/3) + \Theta(n) \Rightarrow T(n) = T(2n/3) + T(n/3) + an$



Solution: $T(n) \in \Theta(n \lg n)$

Sometimes Only Substitution Will Do

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

Lower bound: $T(n) \in \Omega(n^{1+\log_2(7/6)}) \approx \Omega(n^{1.22})$

Upper bound: $T(n) \in O(n^{1+\log_{3/2}(7/6)}) \approx O(n^{1.38})$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 1: Merge Sort again

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

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 1: Merge Sort again

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

$$a = 2 \qquad b = 2 \qquad f(n) \in \Theta(n)$$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 1: Merge Sort again

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

$$a = 2 \qquad b = 2 \qquad f(n) \in \Theta(n) = \Theta(n^{\log_2 2})$$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 1: Merge Sort again

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

$$a = 2 \qquad b = 2 \qquad f(n) \in \Theta(n) = \Theta(n^{\log_2 2})$$

$$T(n) \in \Theta(n \lg n)$$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 2: Matrix multiplication

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

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 2: Matrix multiplication

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

$a = 7 \quad b = 2 \quad f(n) \in \Theta(n^2)$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 2: Matrix multiplication

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

$$a = 7 \qquad b = 2 \qquad f(n) \in \Theta(n^2) \subseteq O(n^{\log_2 7 - \epsilon}) \text{ for all } 0 < \epsilon \leq \log_2 7 - 2$$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 2: Matrix multiplication

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

$$a = 7 \qquad b = 2 \qquad f(n) \in \Theta(n^2) \subseteq O(n^{\log_2 7 - \epsilon}) \text{ for all } 0 < \epsilon \leq \log_2 7 - 2$$

$$T(n) \in \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 3:

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

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 3:

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

$$a = 1 \qquad b = 2$$

$f(n) = n$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 3:

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

$$a = 1 \qquad b = 2$$

$$f(n) = n \in \Omega(n^{\log_2 1 + \epsilon}) \text{ for all } 0 < \epsilon \leq 1$$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 3:

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

$$a = 1 \qquad b = 2$$

$$f(n) = n \in \Omega(n^{\log_2 1 + \epsilon}) \text{ for all } 0 < \epsilon \leq 1 \qquad f(n/2) = n/2 \leq f(n)/2$$

Master Theorem

Master Theorem: Let $a \geq 1$ and $b > 1$, let $f(n)$ be a positive function and let $T(n)$ be given by the following recurrence:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n).$$

(i) If $f(n) \in O(n^{\log_b a - \epsilon})$, for some $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$.

(ii) If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \lg n)$.

(iii) If $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq cf(n)$, for some $\epsilon > 0$ and $c < 1$ and for all $n \geq n_0$, then $T(n) \in \Theta(f(n))$.

Example 3:

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

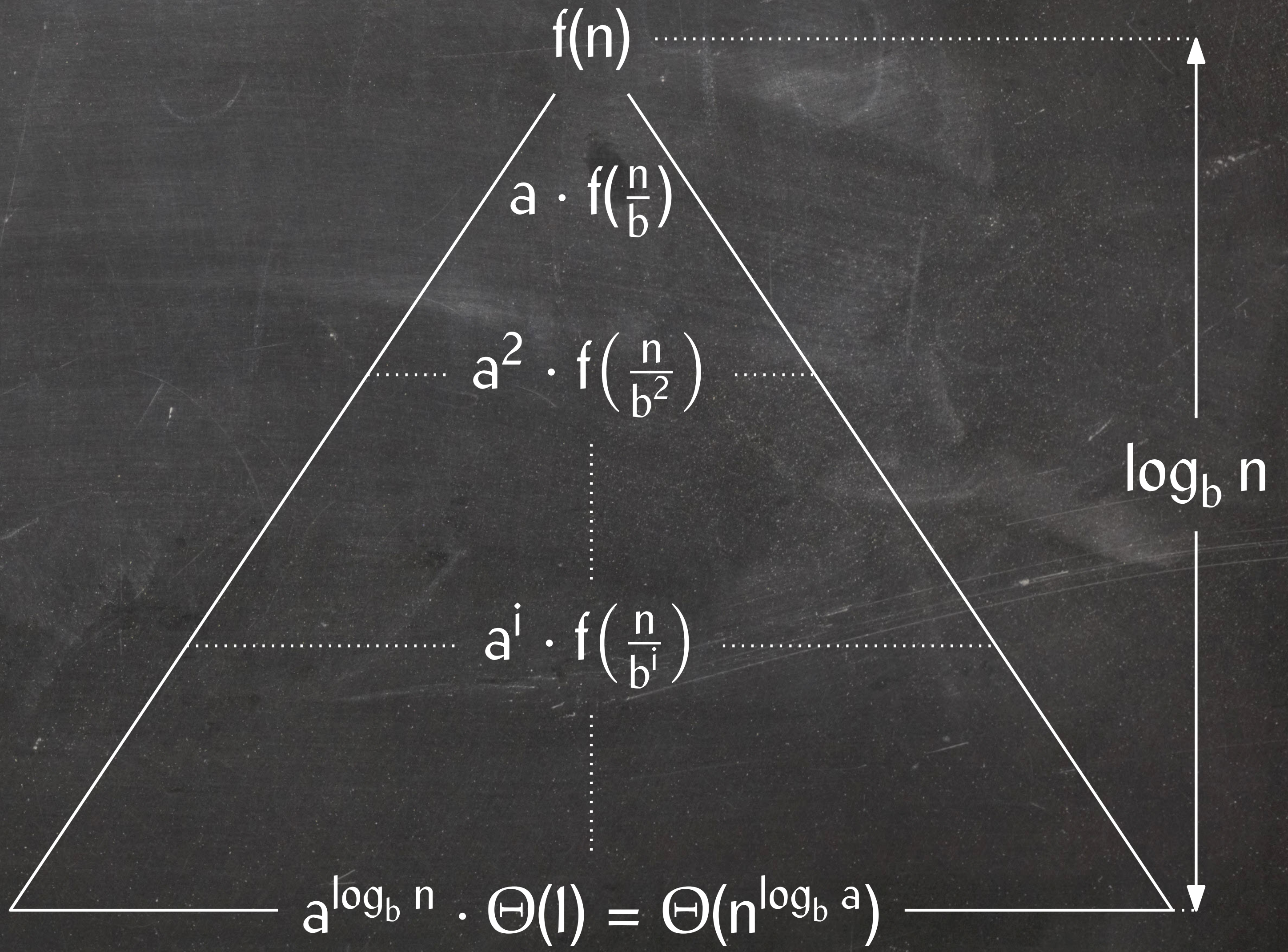
$$a = 1 \qquad b = 2$$

$$f(n) = n \in \Omega(n^{\log_2 1 + \epsilon}) \text{ for all } 0 < \epsilon \leq 1 \qquad f(n/2) = n/2 \leq f(n)/2$$

$$T(n) \in \Theta(n)$$

Master Theorem: Proof

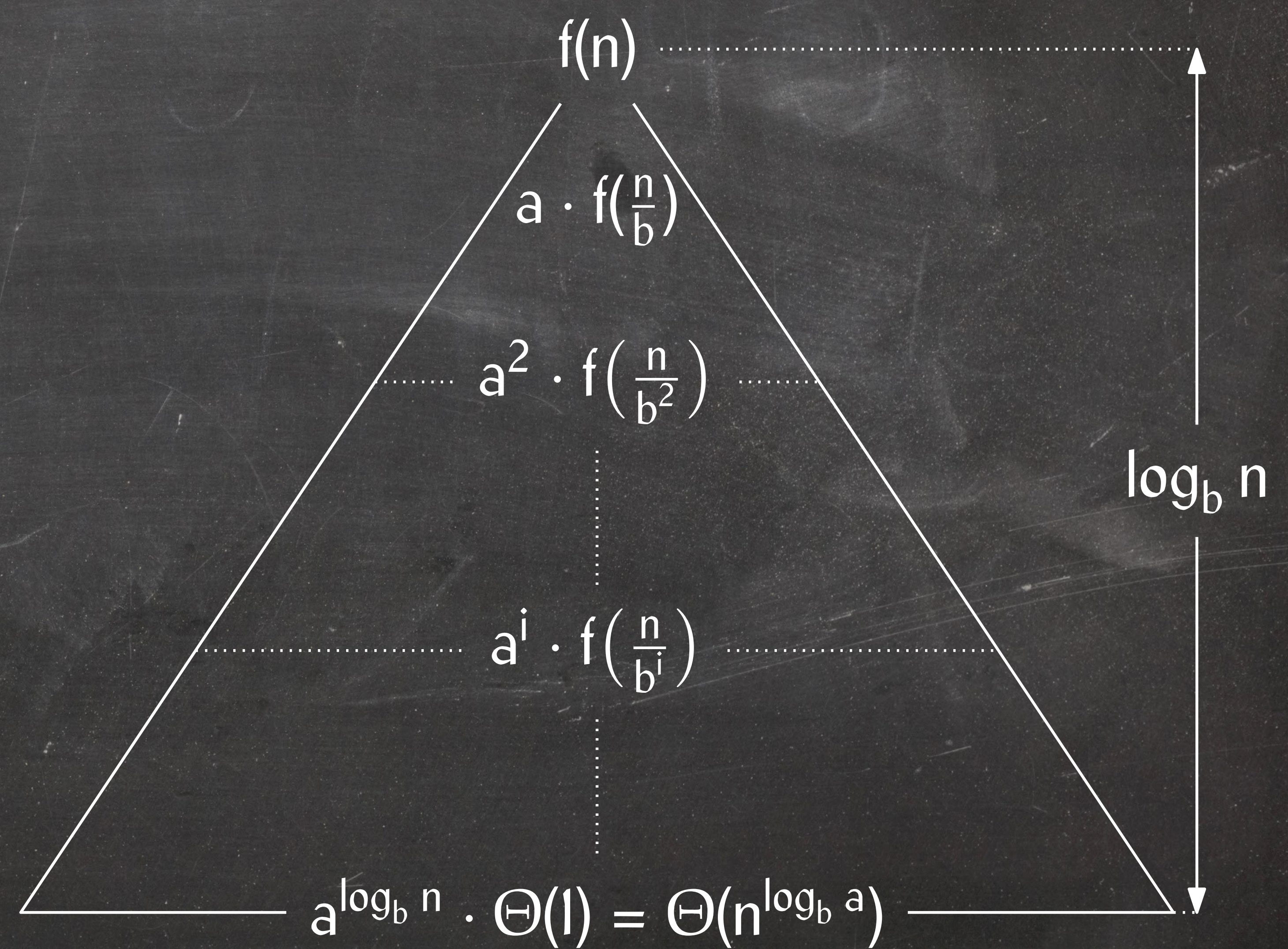
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$



Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 1: $f(n) \in O(n^{\log_b a - \epsilon})$

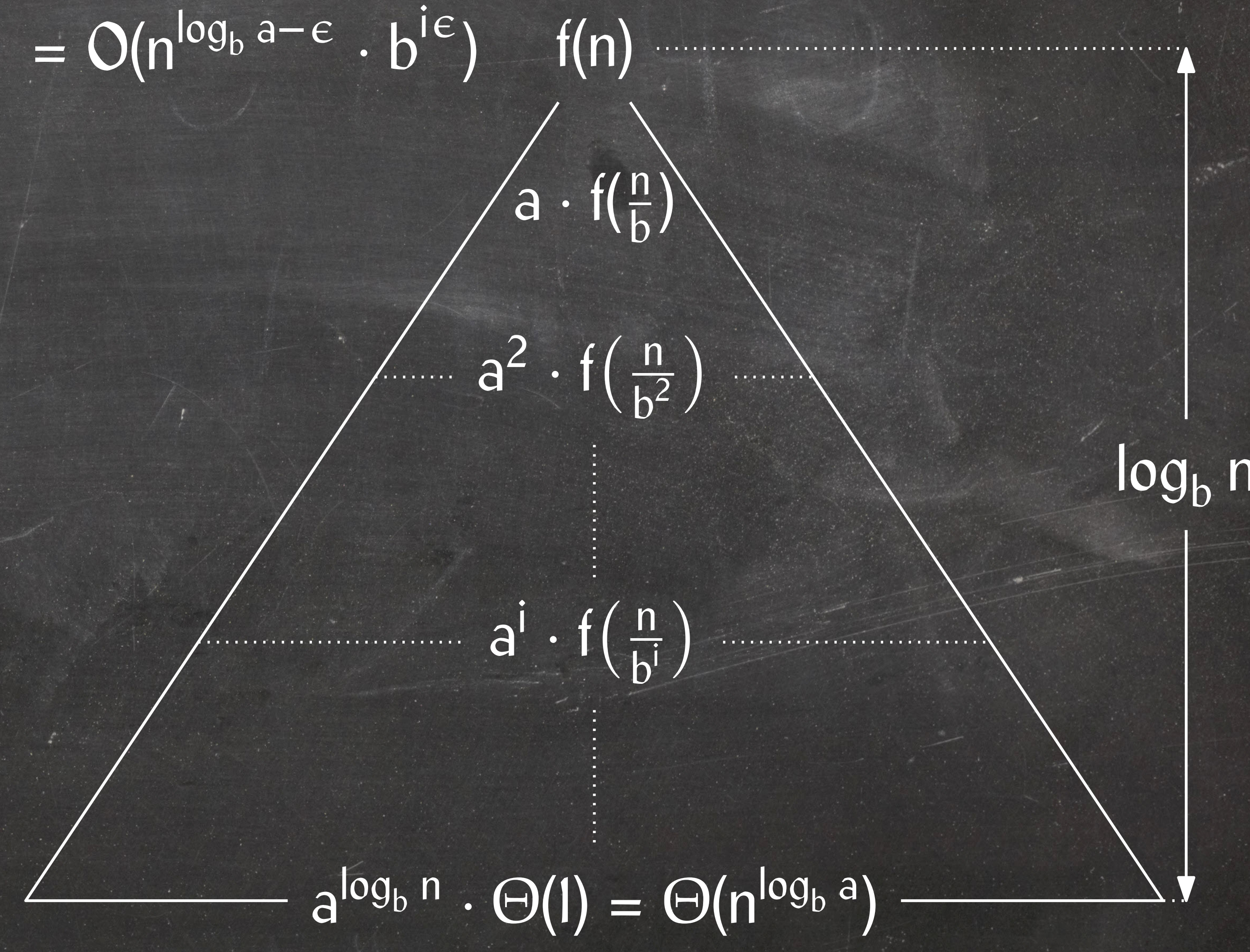


Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 1: $f(n) \in O(n^{\log_b a - \epsilon})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in O\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon}\right) = O(n^{\log_b a - \epsilon} \cdot b^{i\epsilon})$$

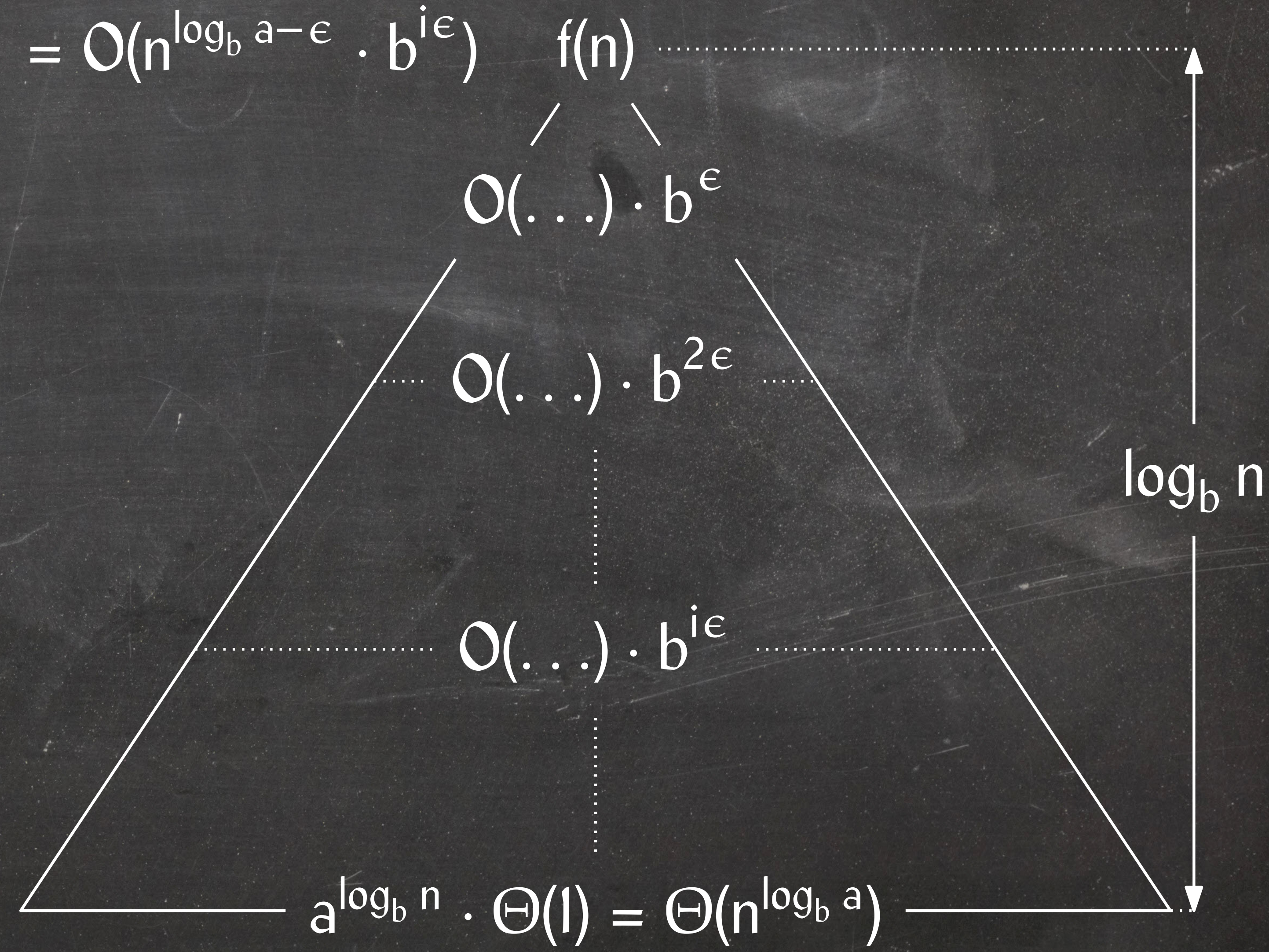


Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 1: $f(n) \in O(n^{\log_b a - \epsilon})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in O\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon}\right) = O(n^{\log_b a - \epsilon} \cdot b^{i\epsilon})$$



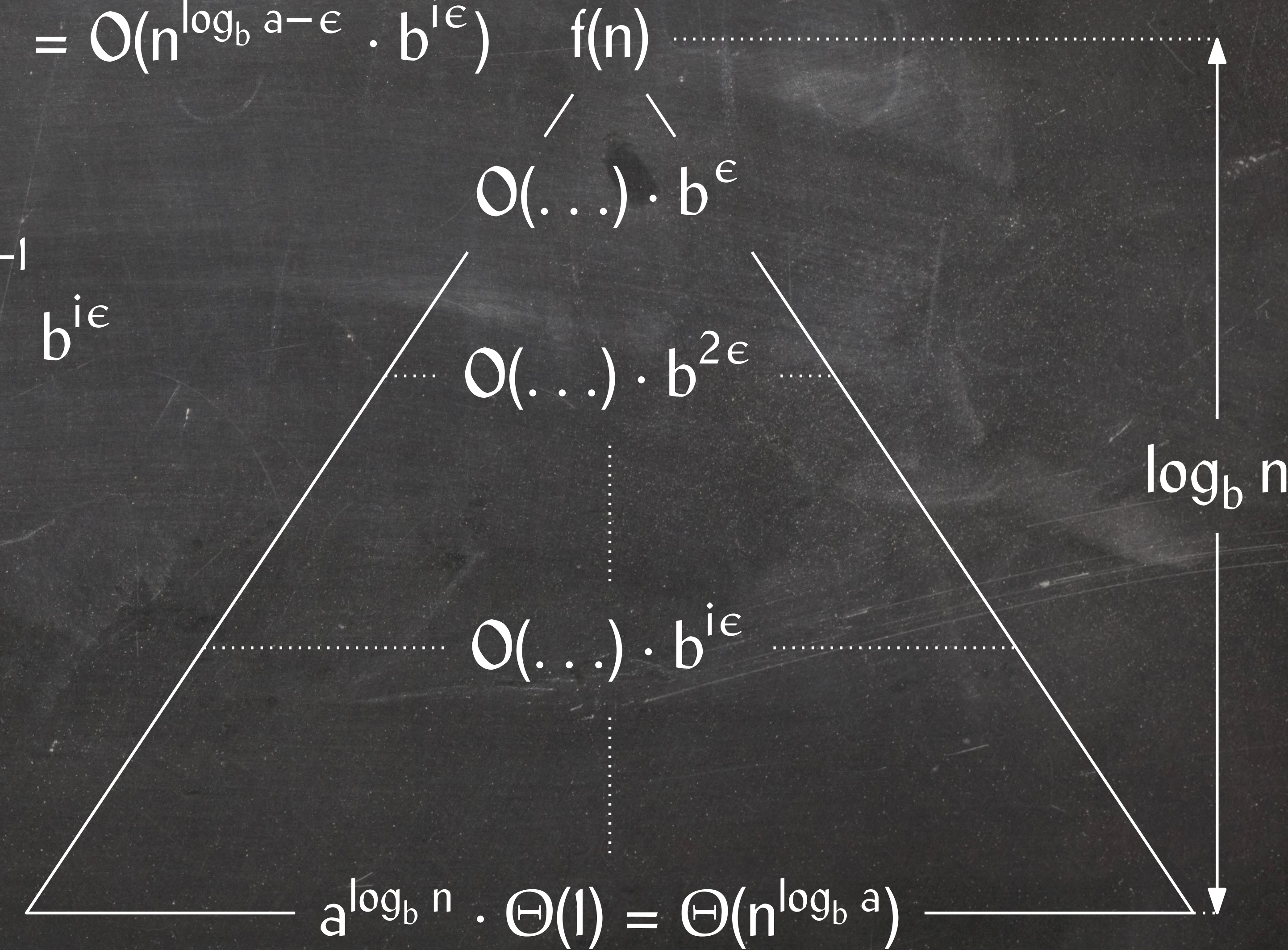
Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 1: $f(n) \in O(n^{\log_b a - \epsilon})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in O\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon}\right) = O(n^{\log_b a - \epsilon} \cdot b^{i\epsilon})$$

$$T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot \sum_{i=1}^{\log_b n - 1} b^{i\epsilon}$$



Master Theorem: Proof

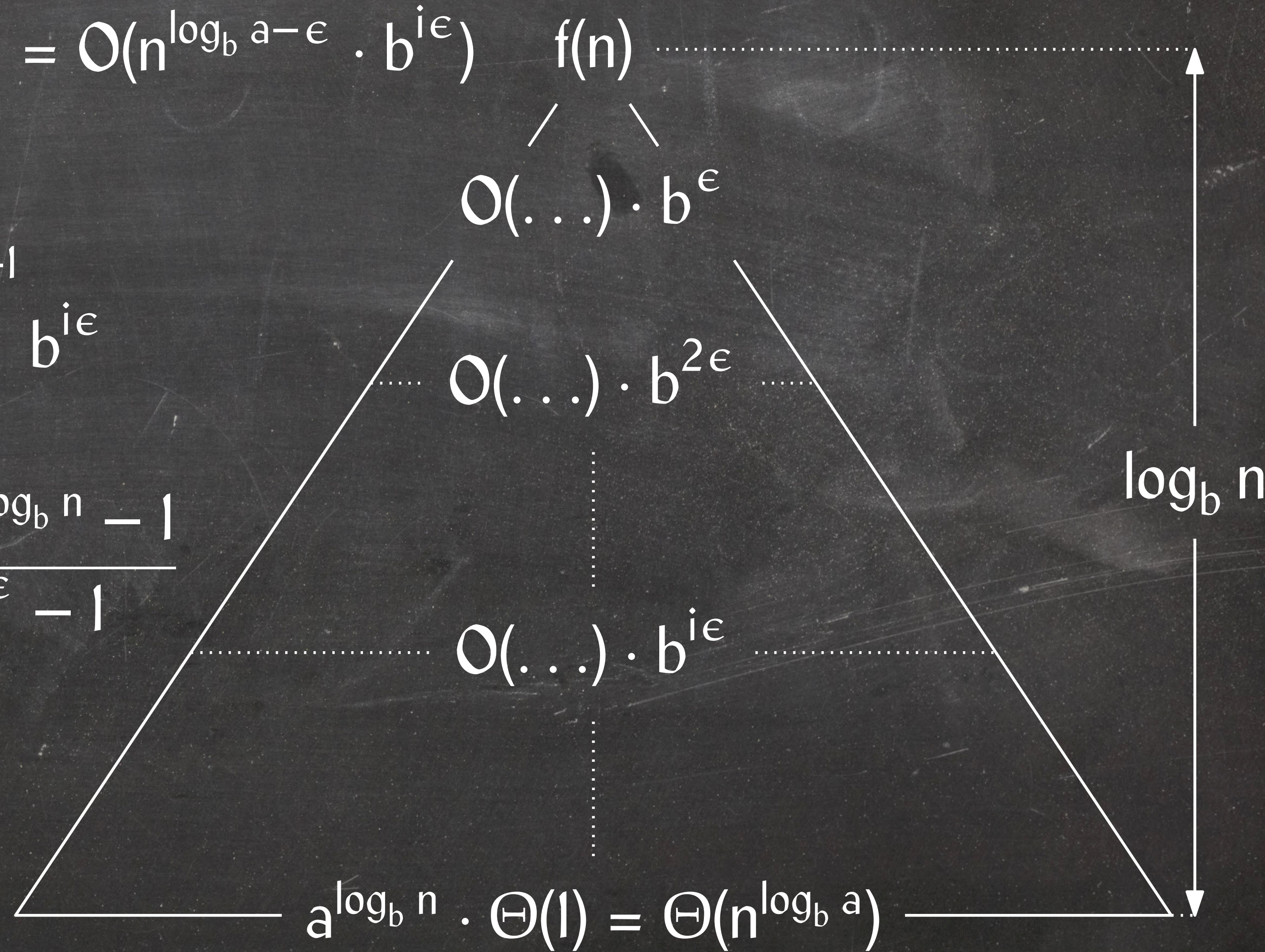
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 1: $f(n) \in O(n^{\log_b a - \epsilon})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in O\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon}\right) = O(n^{\log_b a - \epsilon} \cdot b^{i\epsilon})$$

$$T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot \sum_{i=1}^{\log_b n - 1} b^{i\epsilon}$$

$$= \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot \frac{(b^{\epsilon})^{\log_b n} - 1}{b^{\epsilon} - 1}$$



Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

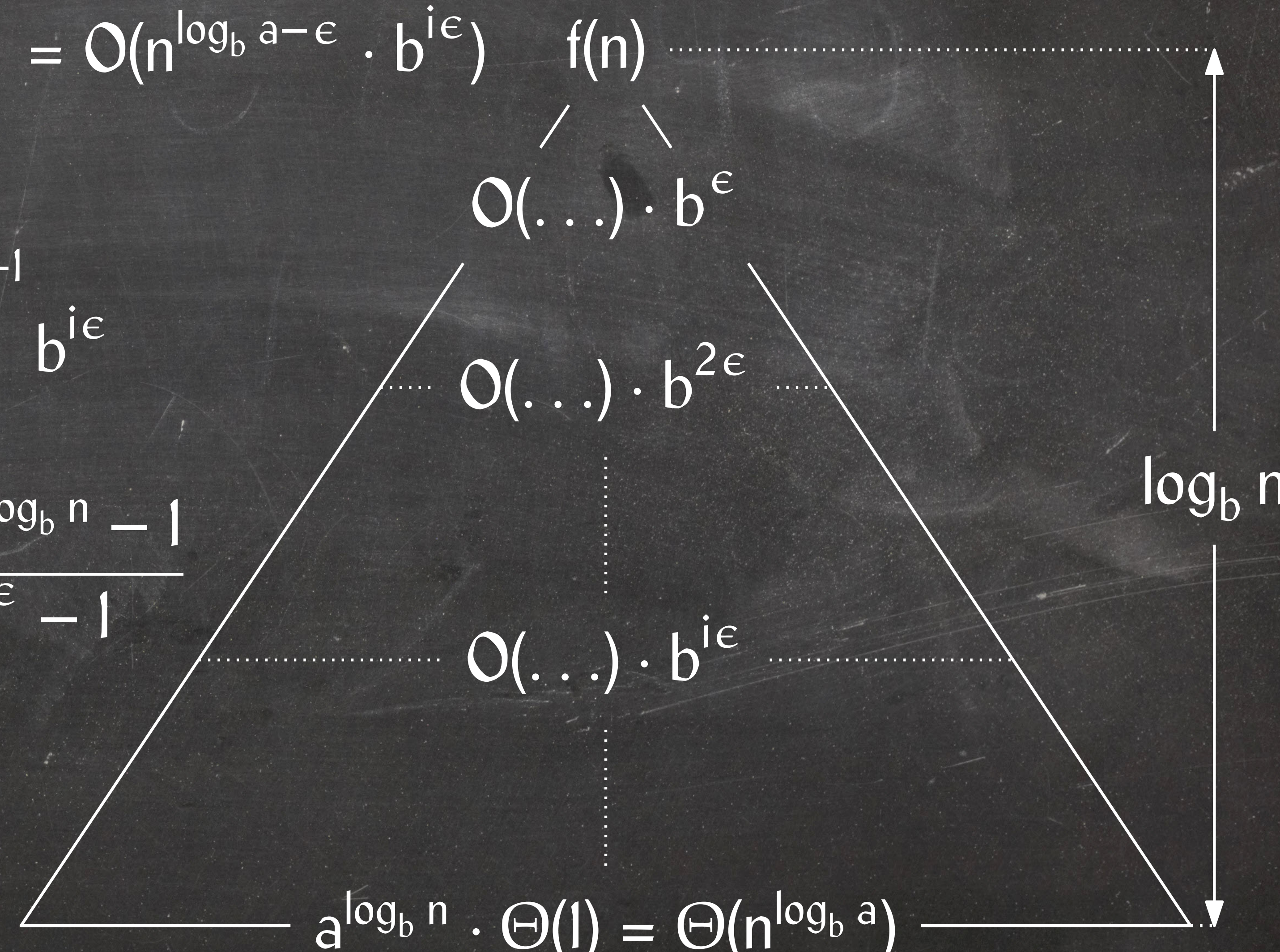
Case 1: $f(n) \in O(n^{\log_b a - \epsilon})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in O\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon}\right) = O(n^{\log_b a - \epsilon} \cdot b^{i\epsilon})$$

$$T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot \sum_{i=1}^{\log_b n - 1} b^{i\epsilon}$$

$$= \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot \frac{(b^{\epsilon})^{\log_b n} - 1}{b^{\epsilon} - 1}$$

$$= \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot n^{\epsilon}$$



Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 1: $f(n) \in O(n^{\log_b a - \epsilon})$

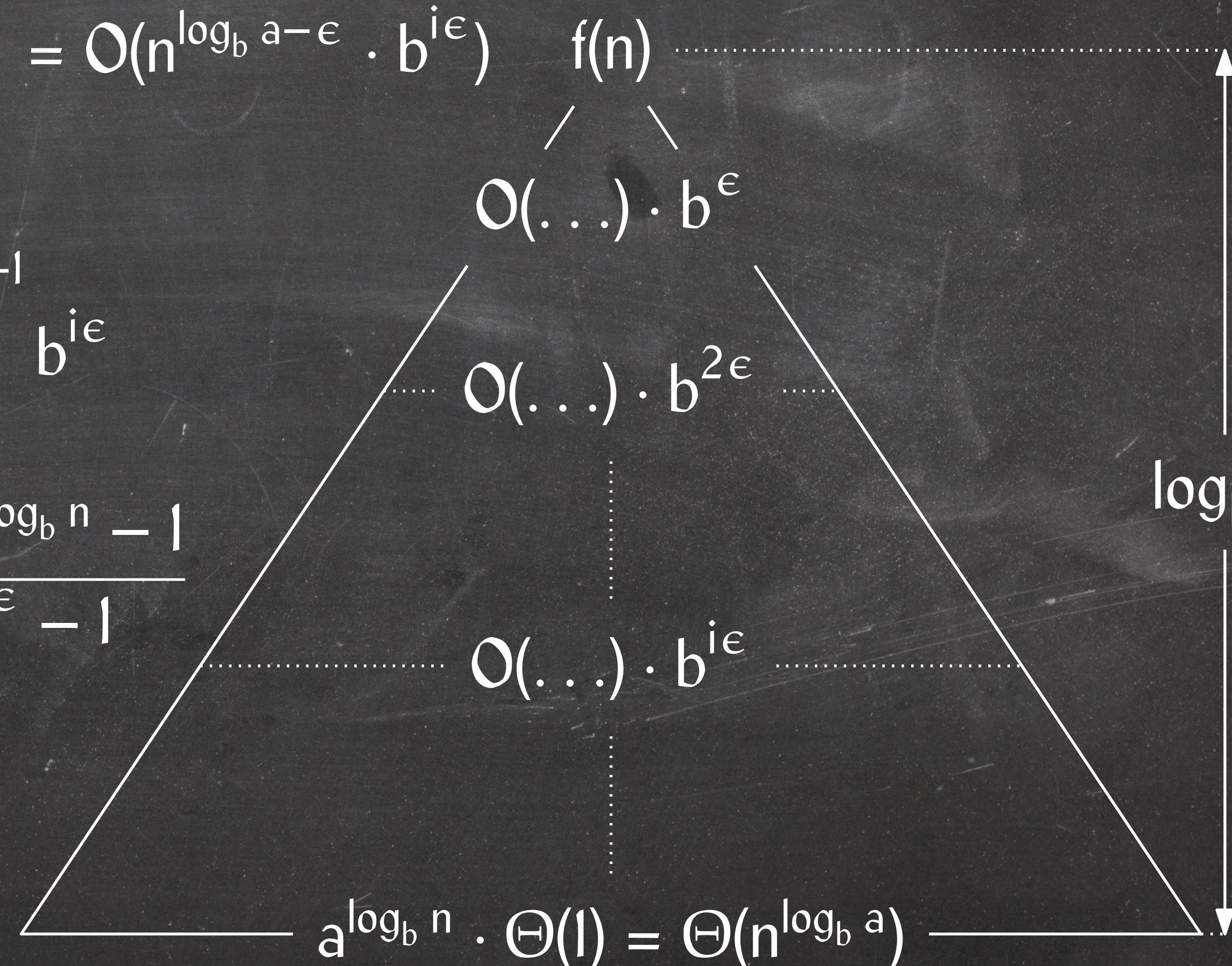
$$a^i \cdot f\left(\frac{n}{b^i}\right) \in O\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a - \epsilon}\right) = O(n^{\log_b a - \epsilon} \cdot b^{i\epsilon})$$

$$T(n) = \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot \sum_{i=1}^{\log_b n - 1} b^{i\epsilon}$$

$$= \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot \frac{(b^{\epsilon})^{\log_b n} - 1}{b^{\epsilon} - 1}$$

$$= \Theta(n^{\log_b a}) + O(n^{\log_b a - \epsilon}) \cdot n^{\epsilon}$$

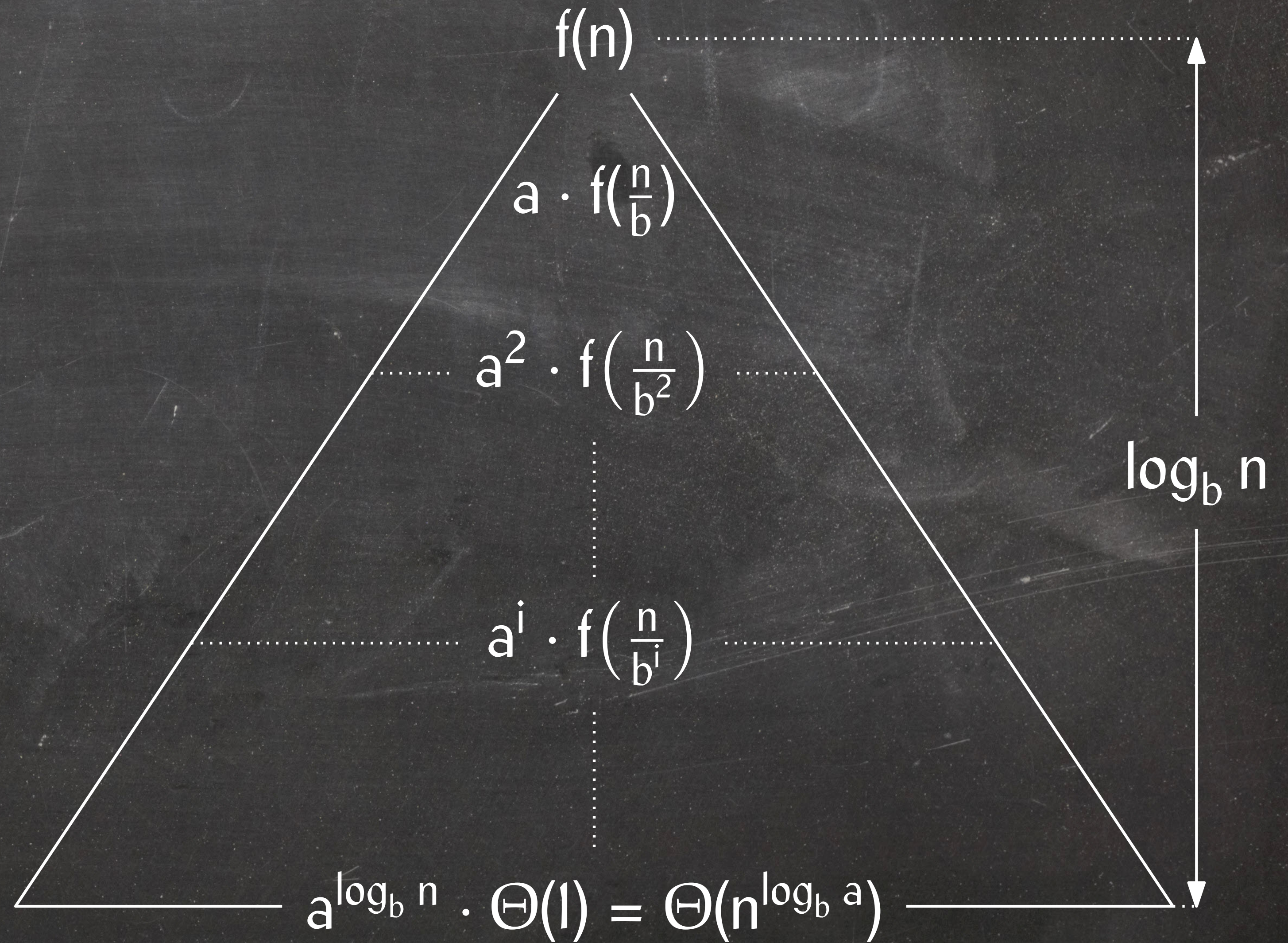
$$= \Theta(n^{\log_b a})$$



Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 2: $f(n) \in \Theta(n^{\log_b a})$

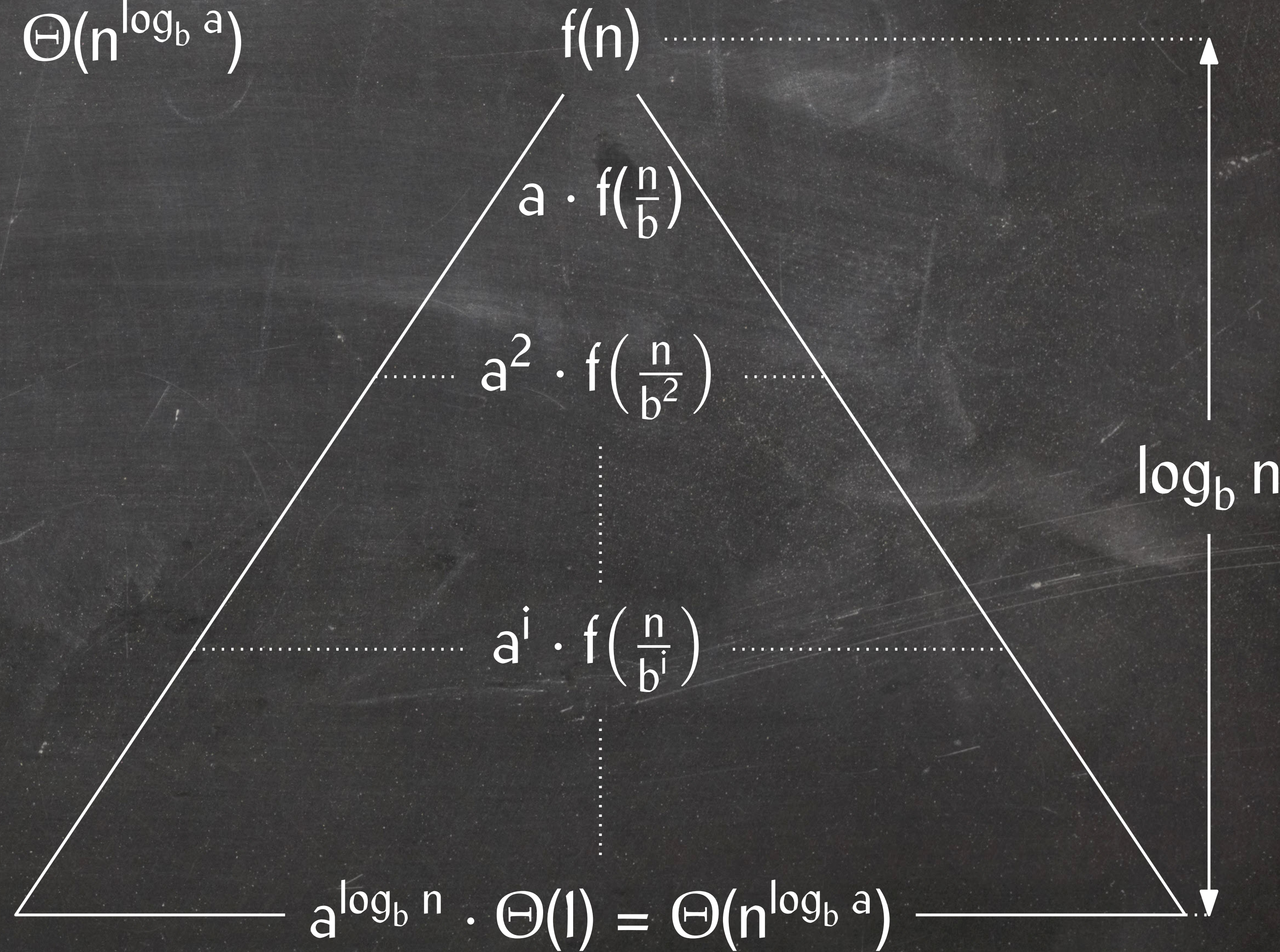


Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 2: $f(n) \in \Theta(n^{\log_b a})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in \Theta\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a}\right) = \Theta(n^{\log_b a})$$

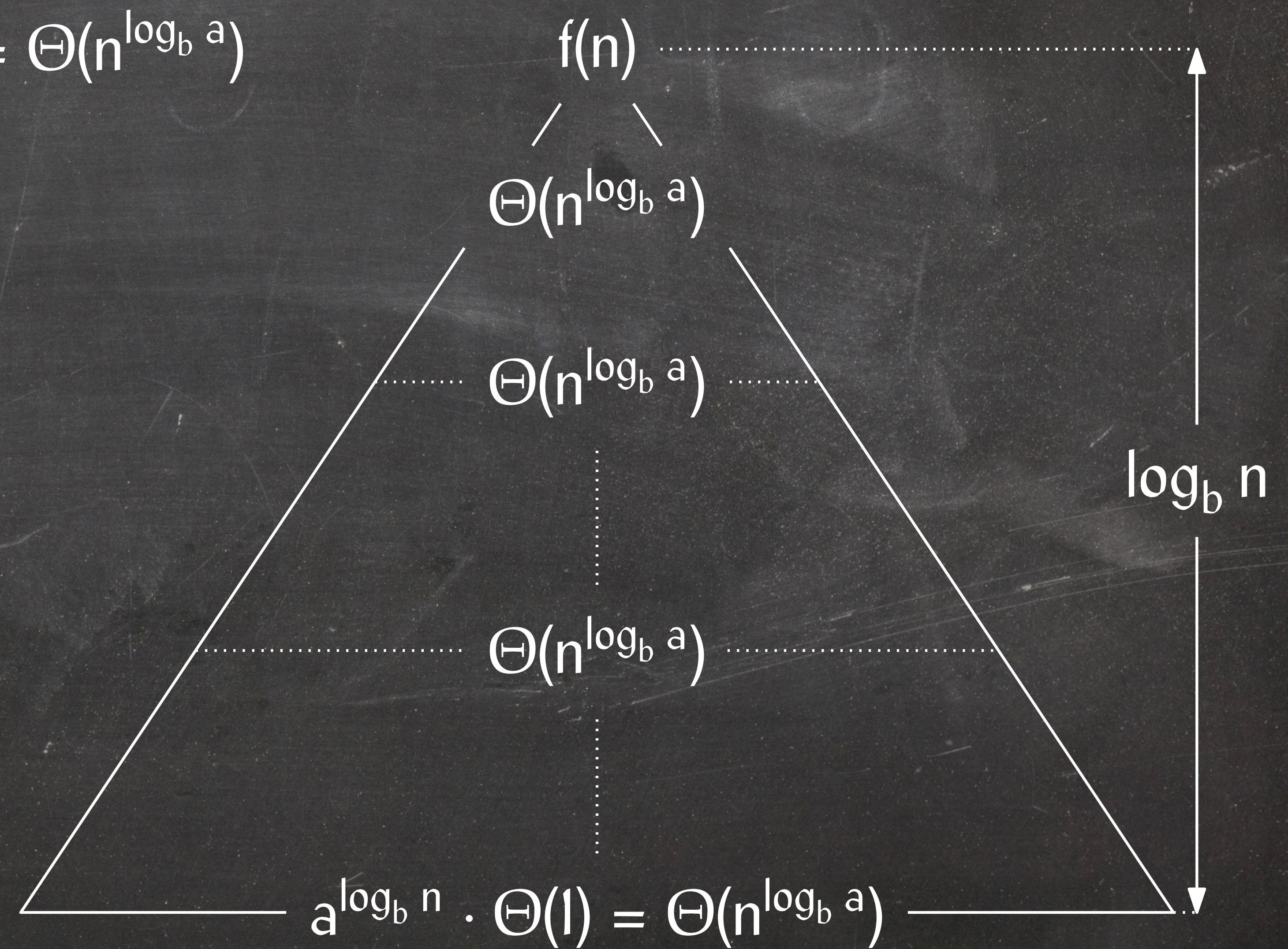


Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 2: $f(n) \in \Theta(n^{\log_b a})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in \Theta\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a}\right) = \Theta(n^{\log_b a})$$



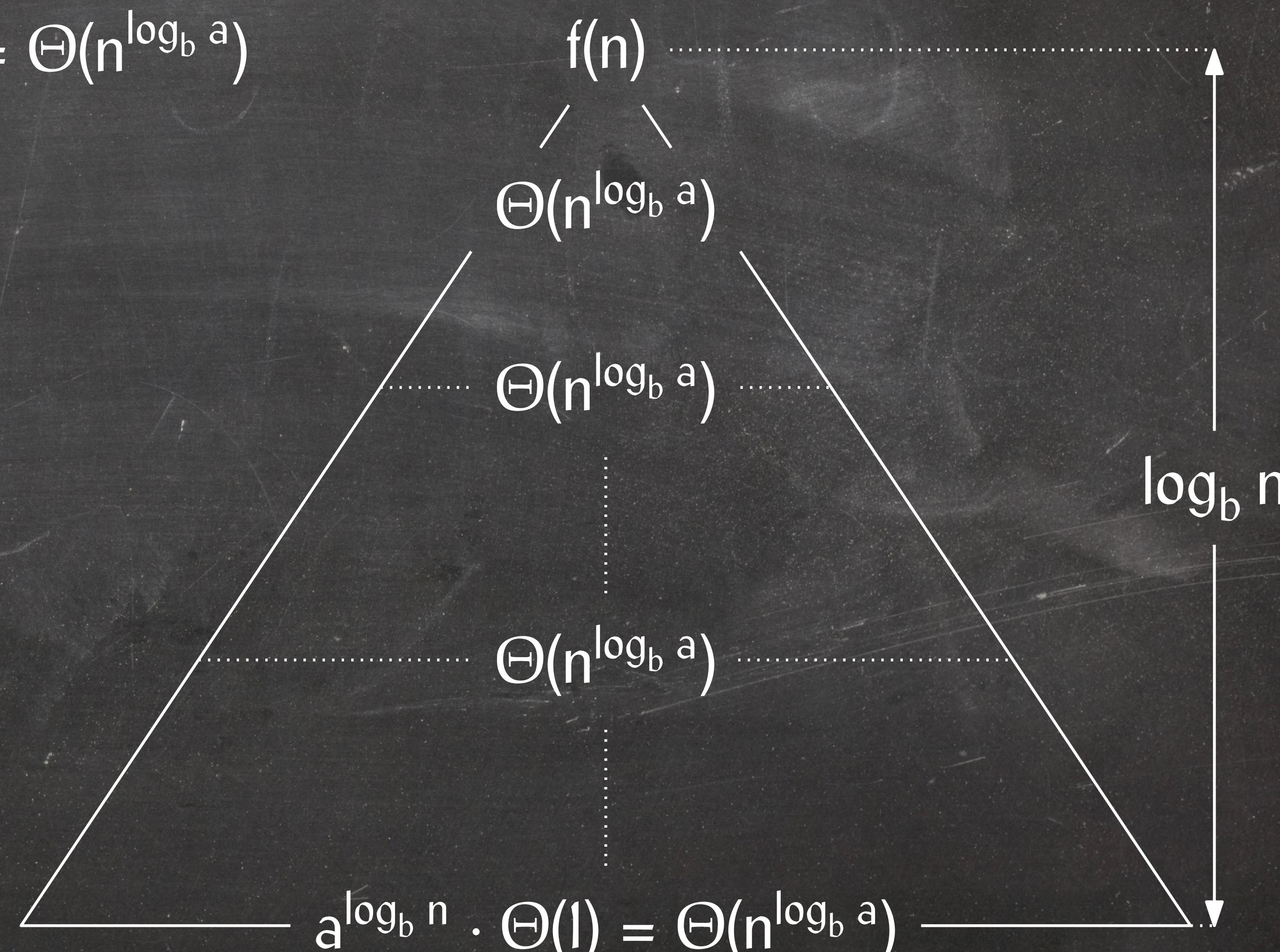
Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 2: $f(n) \in \Theta(n^{\log_b a})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in \Theta\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a}\right) = \Theta(n^{\log_b a})$$

$$T(n) = \Theta(n^{\log_b a}) \cdot \log_b n$$



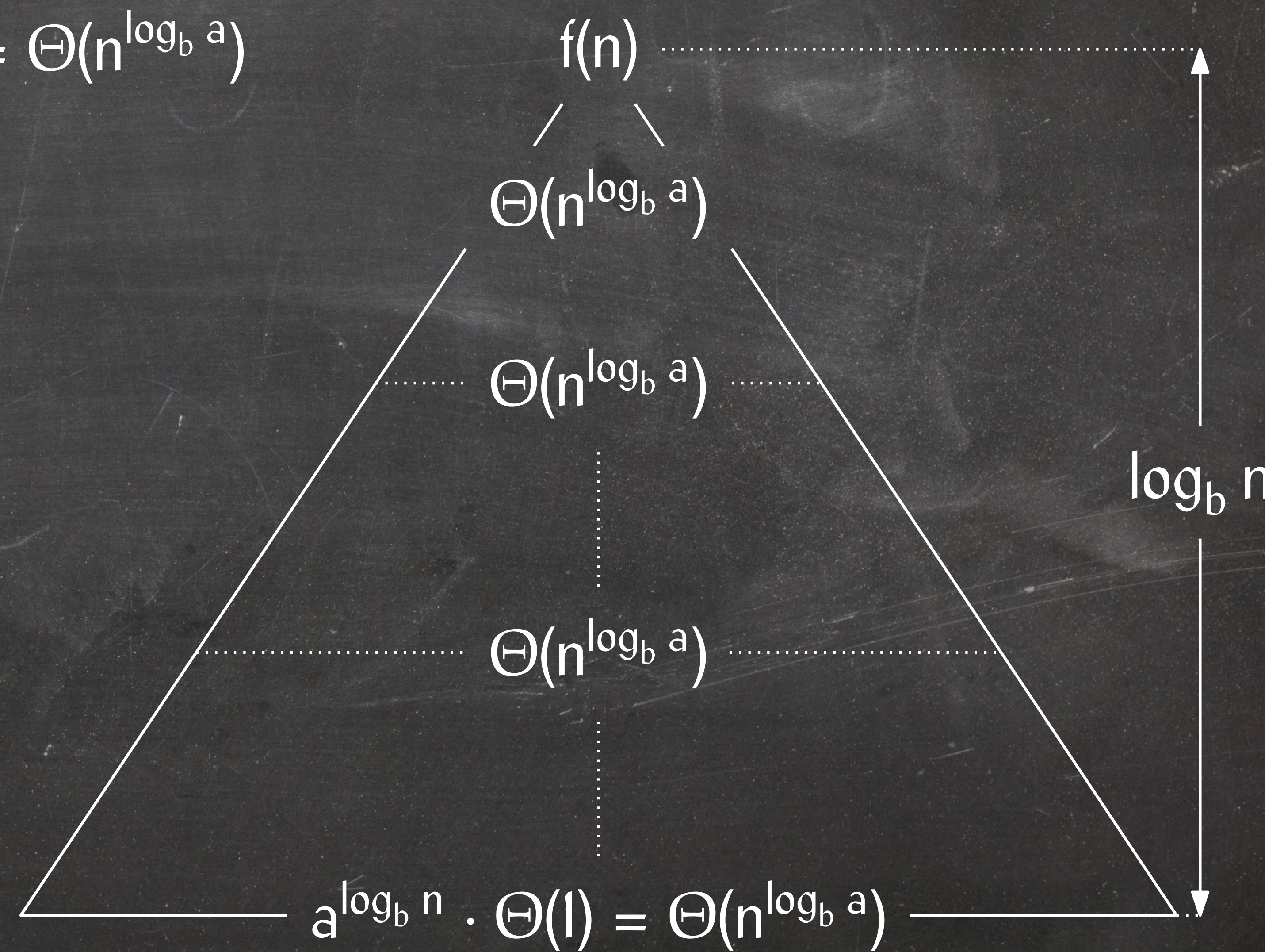
Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 2: $f(n) \in \Theta(n^{\log_b a})$

$$a^i \cdot f\left(\frac{n}{b^i}\right) \in \Theta\left(a^i \cdot \left(\frac{n}{b^i}\right)^{\log_b a}\right) = \Theta(n^{\log_b a})$$

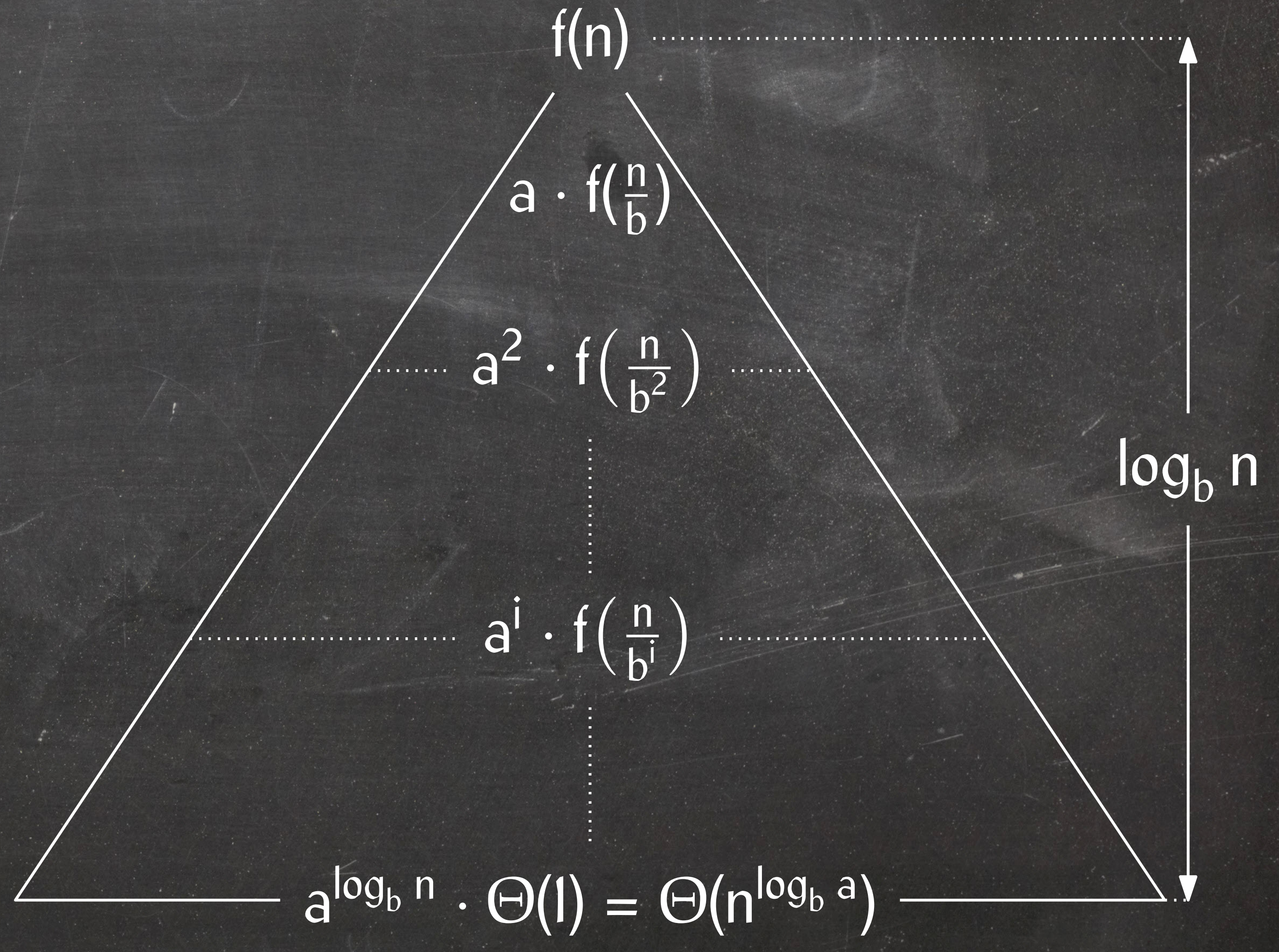
$$T(n) = \Theta(n^{\log_b a}) \cdot \log_b n$$
$$= \Theta(n^{\log_b a} \lg n)$$



Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

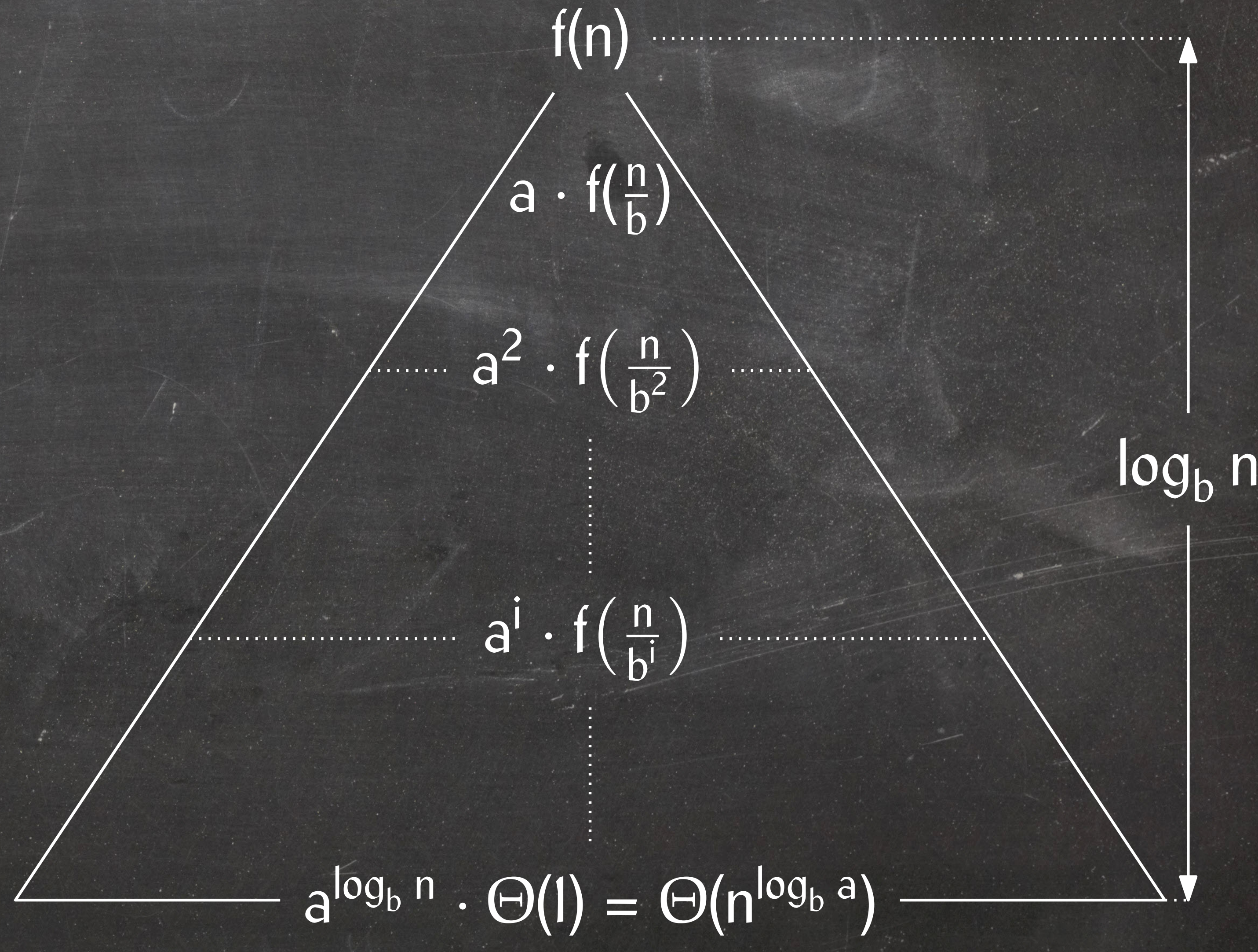


Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$



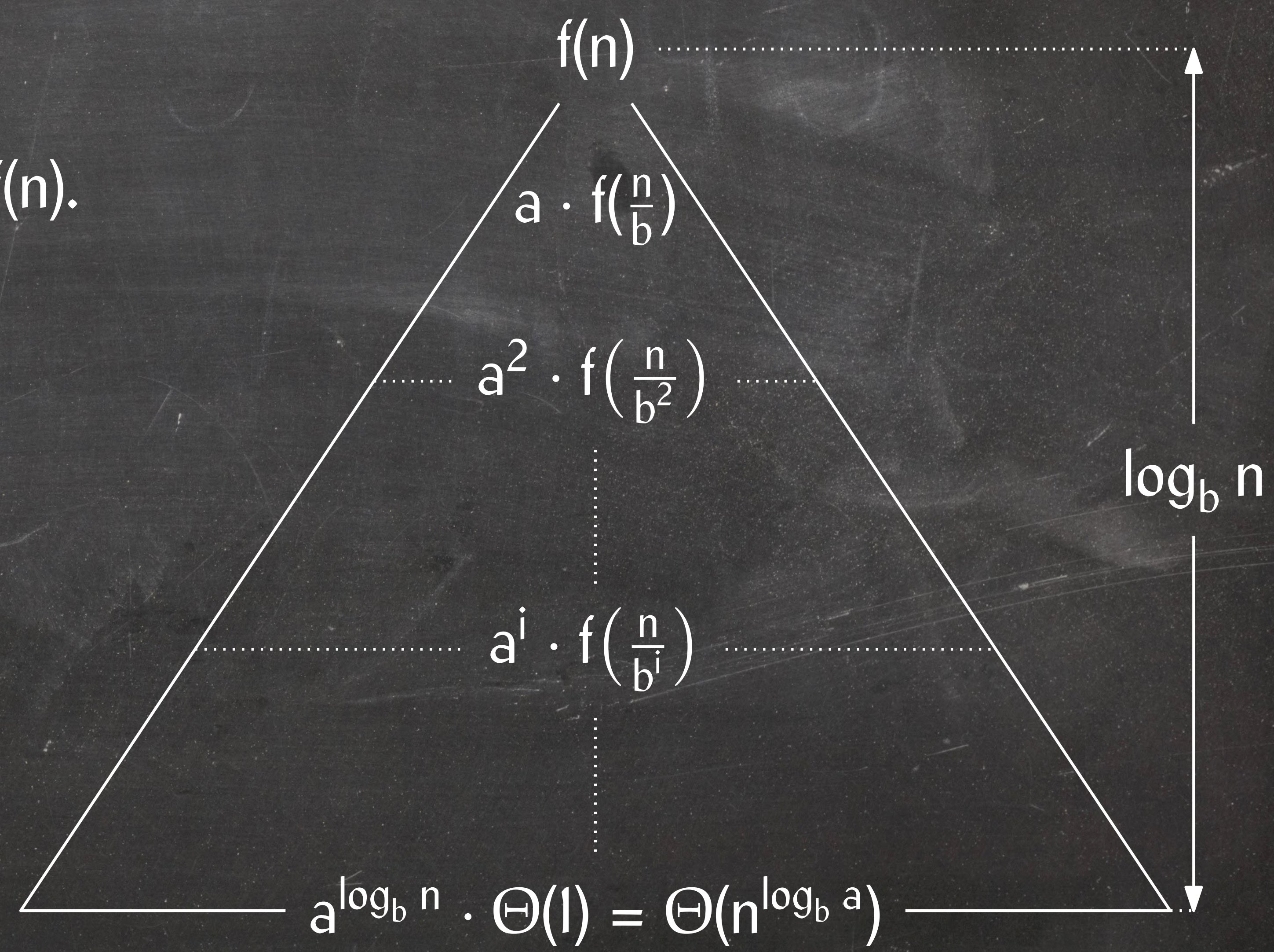
Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

For $i = 0$, $a^0 \cdot f\left(\frac{n}{b^0}\right) = f(n) = c^0 \cdot f(n)$.



Master Theorem: Proof

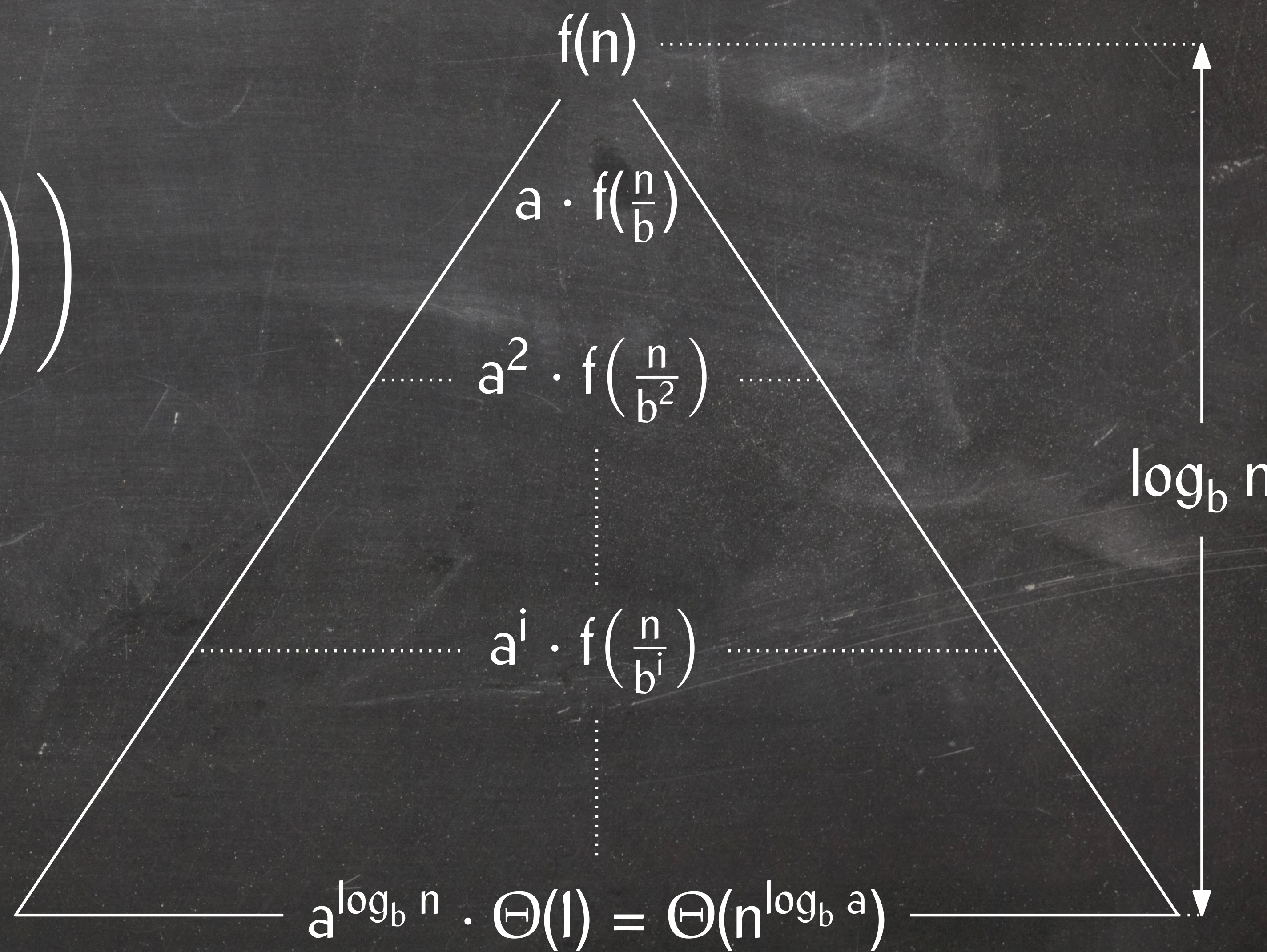
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

For $i > 0$,

$$a^i \cdot f\left(\frac{n}{b^i}\right) = a^{i-1} \cdot \left(a \cdot f\left(\frac{n/b^{i-1}}{b}\right)\right)$$



Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

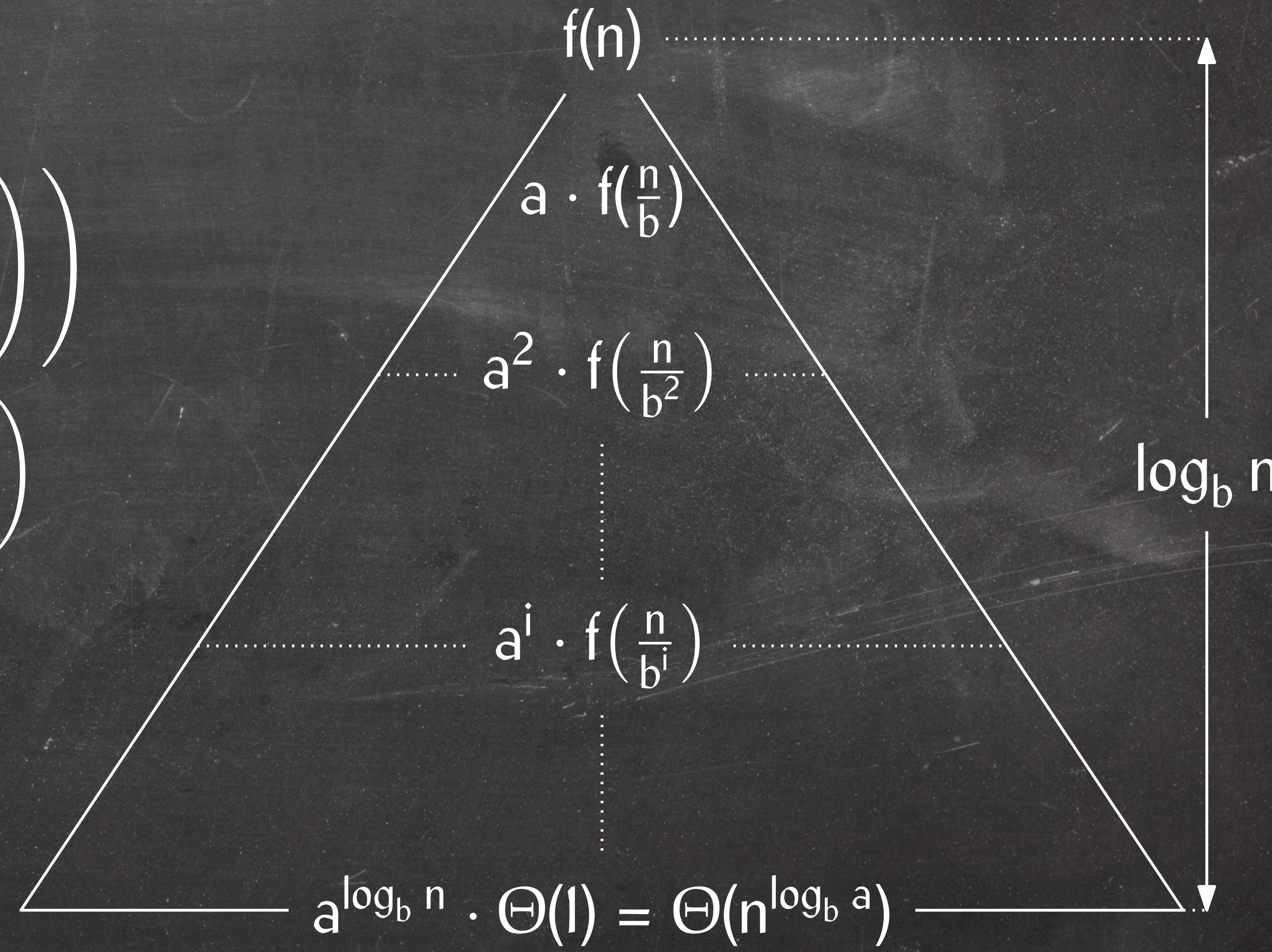
Case 3: $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

For $i > 0$,

$$a^i \cdot f\left(\frac{n}{b^i}\right) = a^{i-1} \cdot \left(a \cdot f\left(\frac{n/b^{i-1}}{b}\right)\right)$$

$$\leq a^{i-1} \cdot \left(c \cdot f\left(\frac{n}{b^{i-1}}\right)\right)$$



Master Theorem: Proof

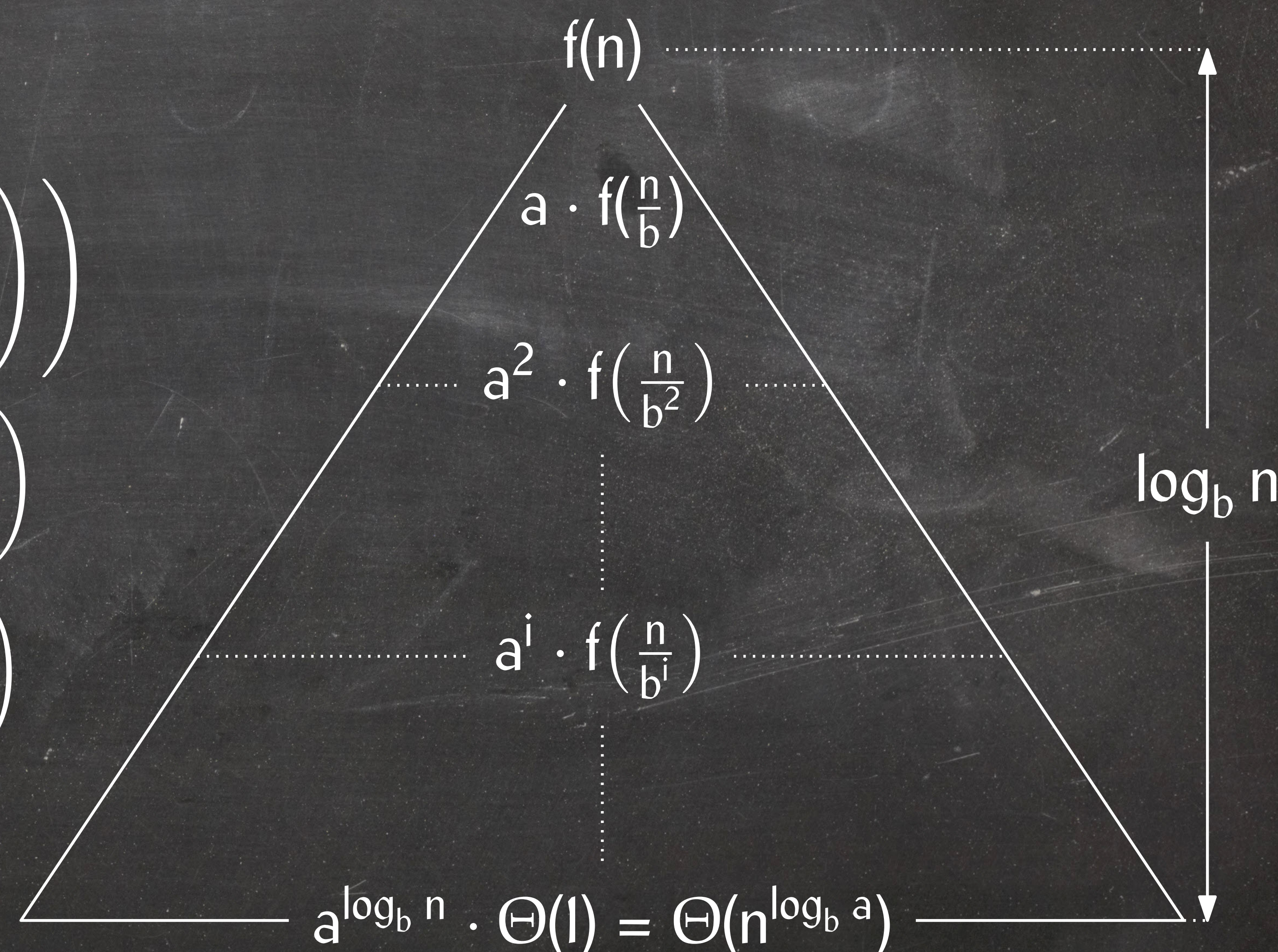
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

For $i > 0$,

$$\begin{aligned} a^i \cdot f\left(\frac{n}{b^i}\right) &= a^{i-1} \cdot \left(a \cdot f\left(\frac{n/b^{i-1}}{b}\right)\right) \\ &\leq a^{i-1} \cdot \left(c \cdot f\left(\frac{n}{b^{i-1}}\right)\right) \\ &= c \cdot \left(a^{i-1} \cdot f\left(\frac{n}{b^{i-1}}\right)\right) \end{aligned}$$



Master Theorem: Proof

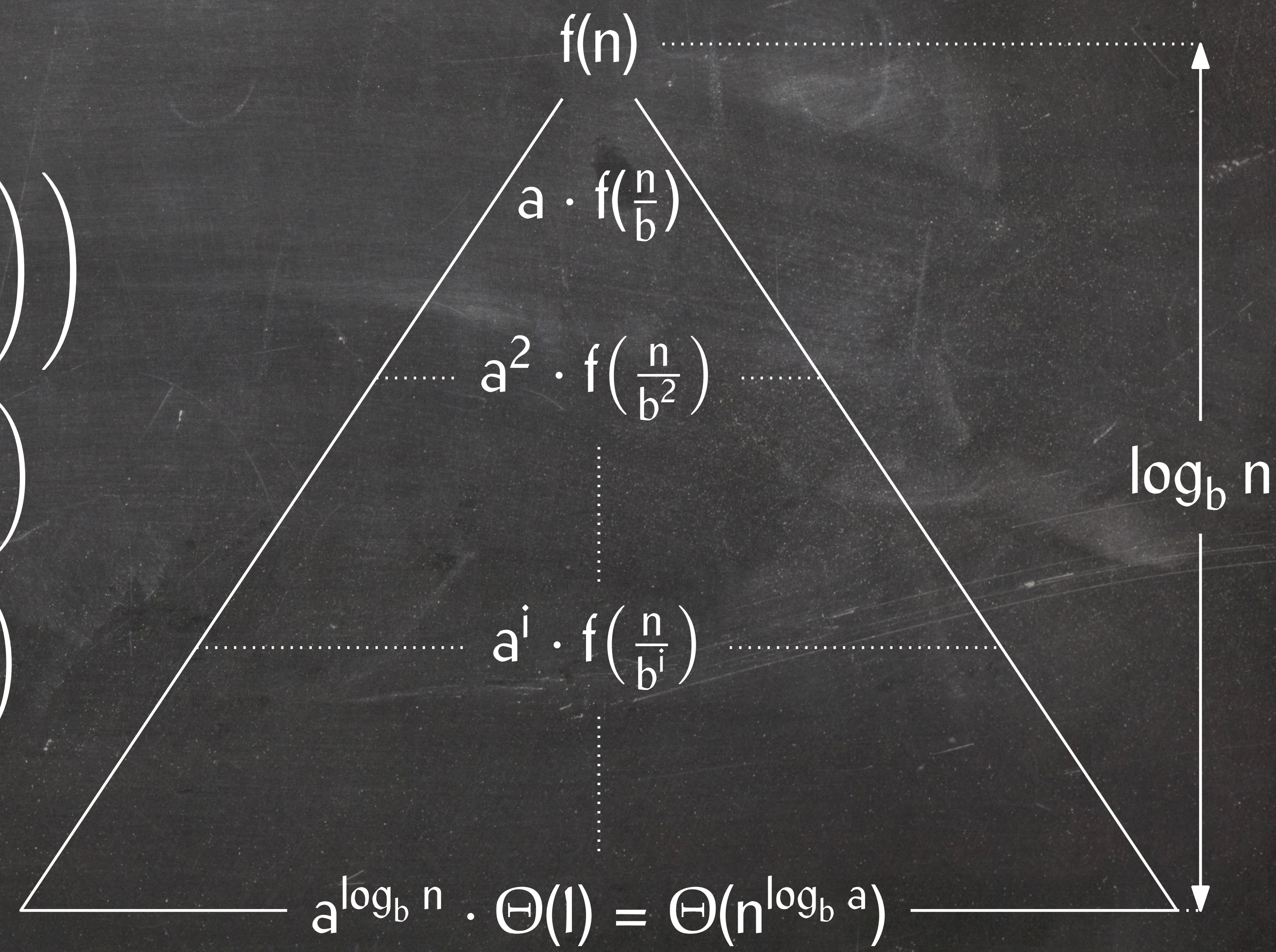
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

For $i > 0$,

$$\begin{aligned}
a^i \cdot f\left(\frac{n}{b^i}\right) &= a^{i-1} \cdot \left(a \cdot f\left(\frac{n/b^{i-1}}{b}\right)\right) \\
&\leq a^{i-1} \cdot \left(c \cdot f\left(\frac{n}{b^{i-1}}\right)\right) \\
&= c \cdot \left(a^{i-1} \cdot f\left(\frac{n}{b^{i-1}}\right)\right) \\
&\leq c \cdot \left(c^{i-1} \cdot f(n)\right)
\end{aligned}$$



Master Theorem: Proof

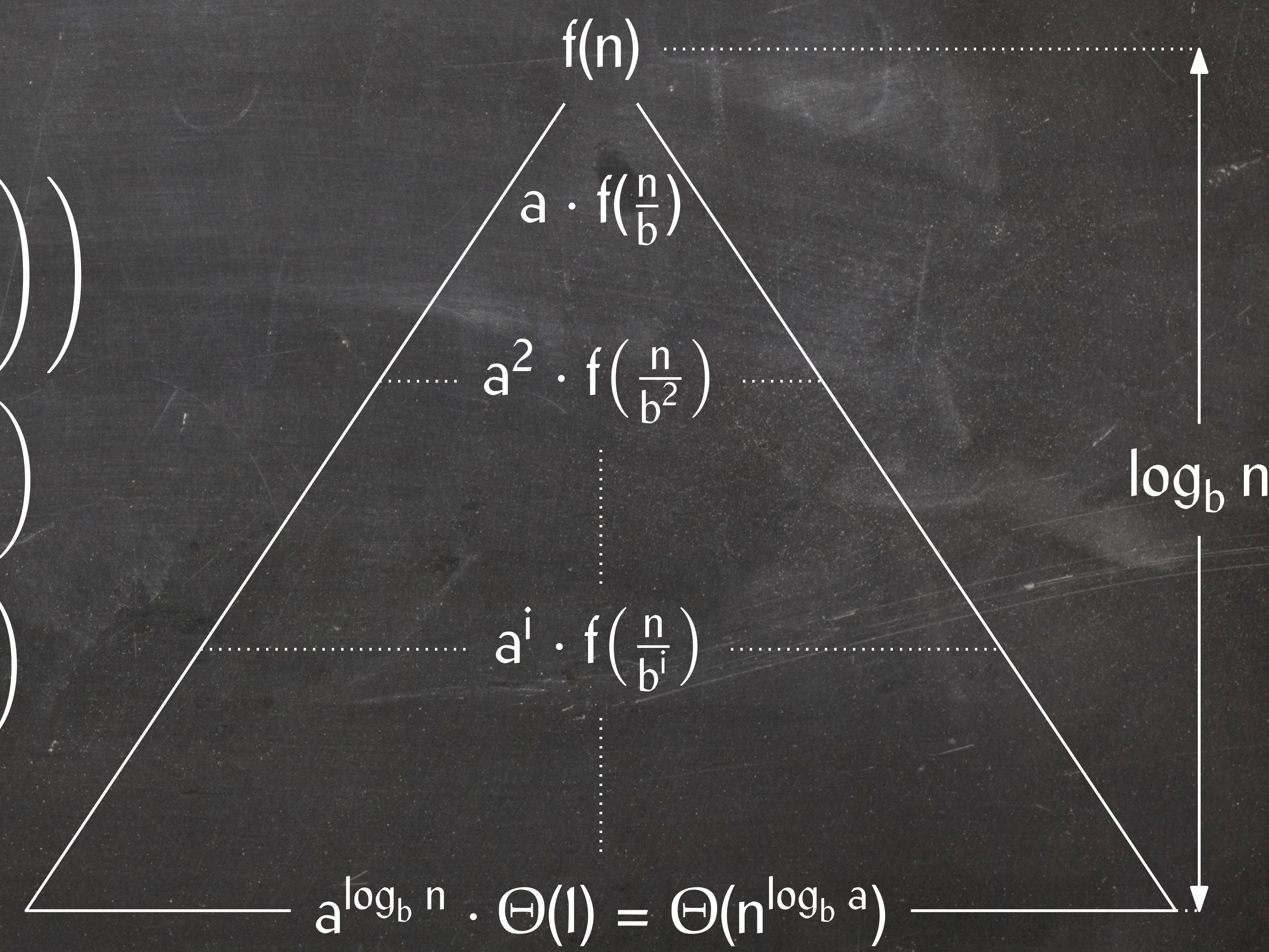
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \le c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \le c^i \cdot f(n)$

For $i > 0$,

$$\begin{aligned}
a^i \cdot f\left(\frac{n}{b^i}\right) &= a^{i-1} \cdot \left(a \cdot f\left(\frac{n/b^{i-1}}{b}\right)\right) \\
&\le a^{i-1} \cdot \left(c \cdot f\left(\frac{n}{b^{i-1}}\right)\right) \\
&= c \cdot \left(a^{i-1} \cdot f\left(\frac{n}{b^{i-1}}\right)\right) \\
&\le c \cdot (c^{i-1} \cdot f(n)) \\
&= c^i \cdot f(n)
\end{aligned}$$

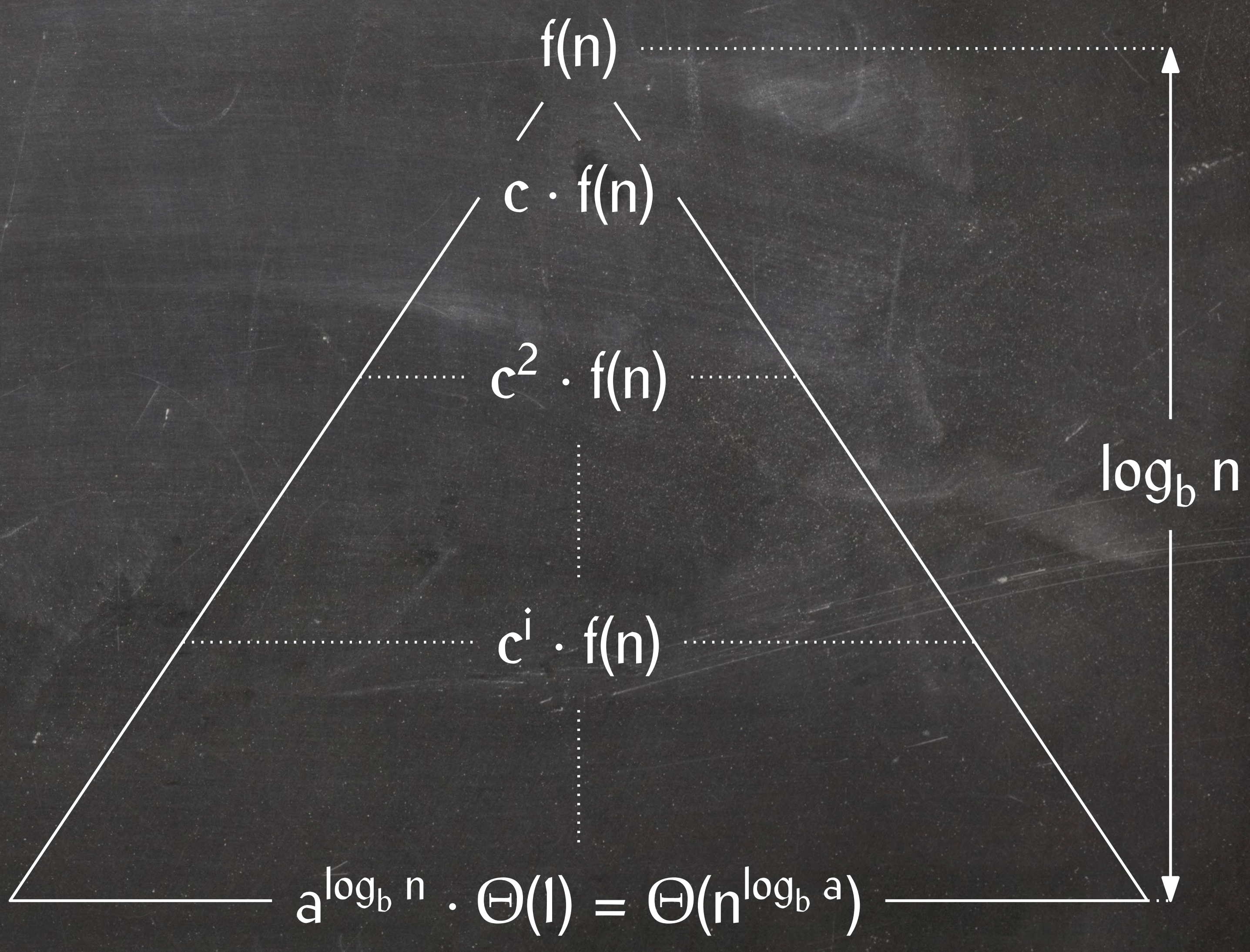


Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$



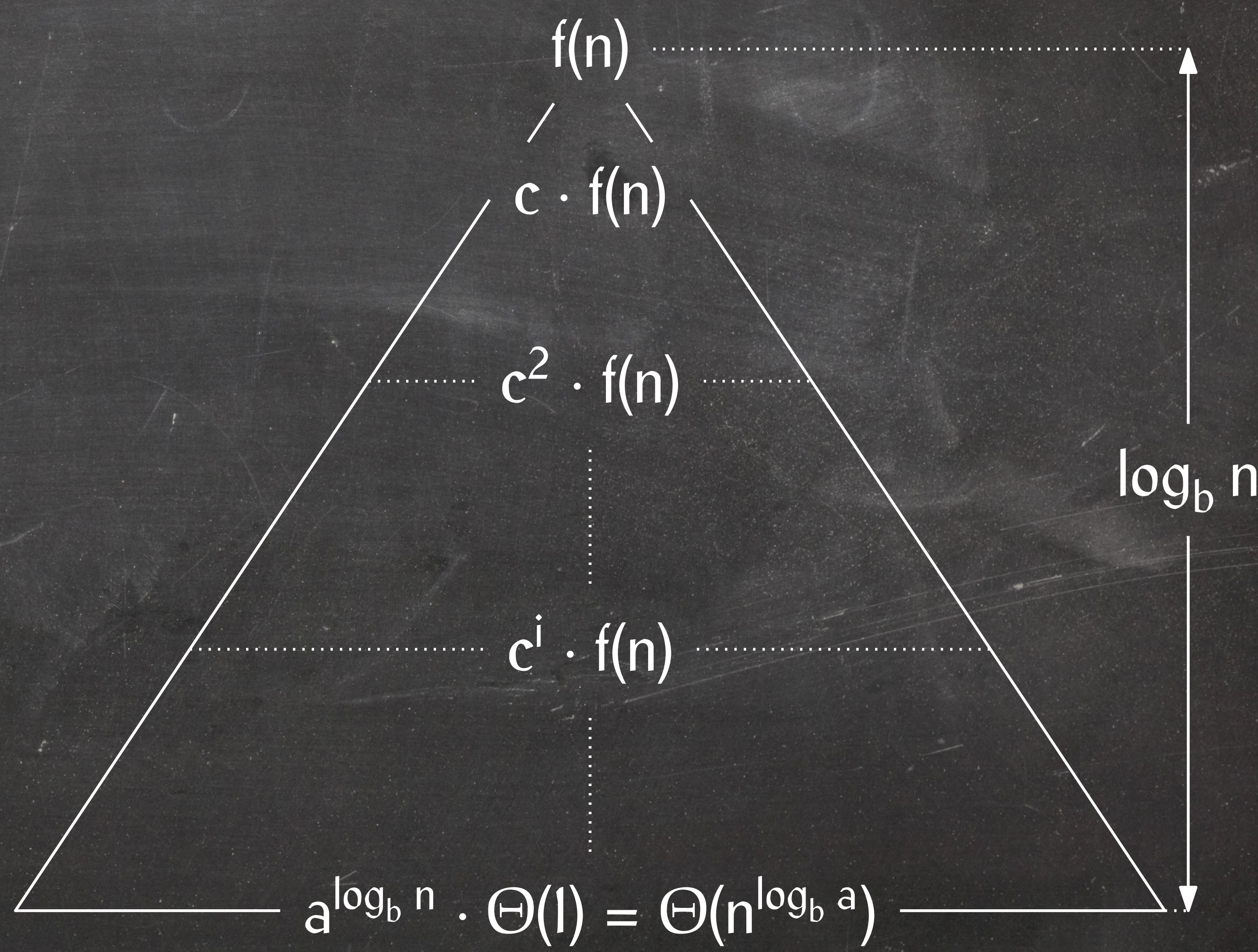
Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

$T(n) \in \Omega(n^{\log_b a} + f(n)) = \Omega(f(n))$



Master Theorem: Proof

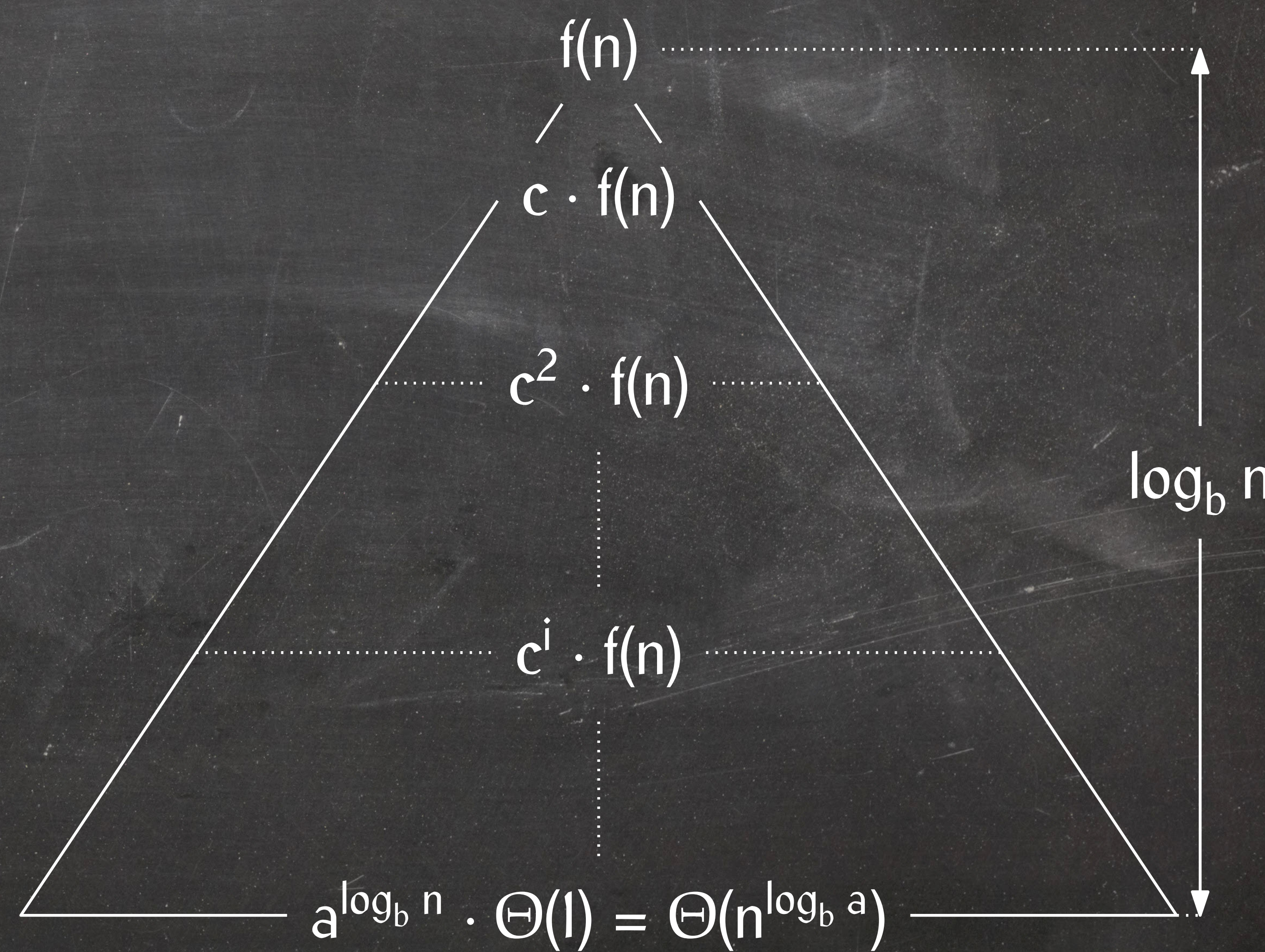
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

$$T(n) \in \Omega(n^{\log_b a} + f(n)) = \Omega(f(n))$$

$$T(n) \in O\left(n^{\log_b a} + \sum_{i=0}^{\infty} c^i \cdot f(n)\right)$$



Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

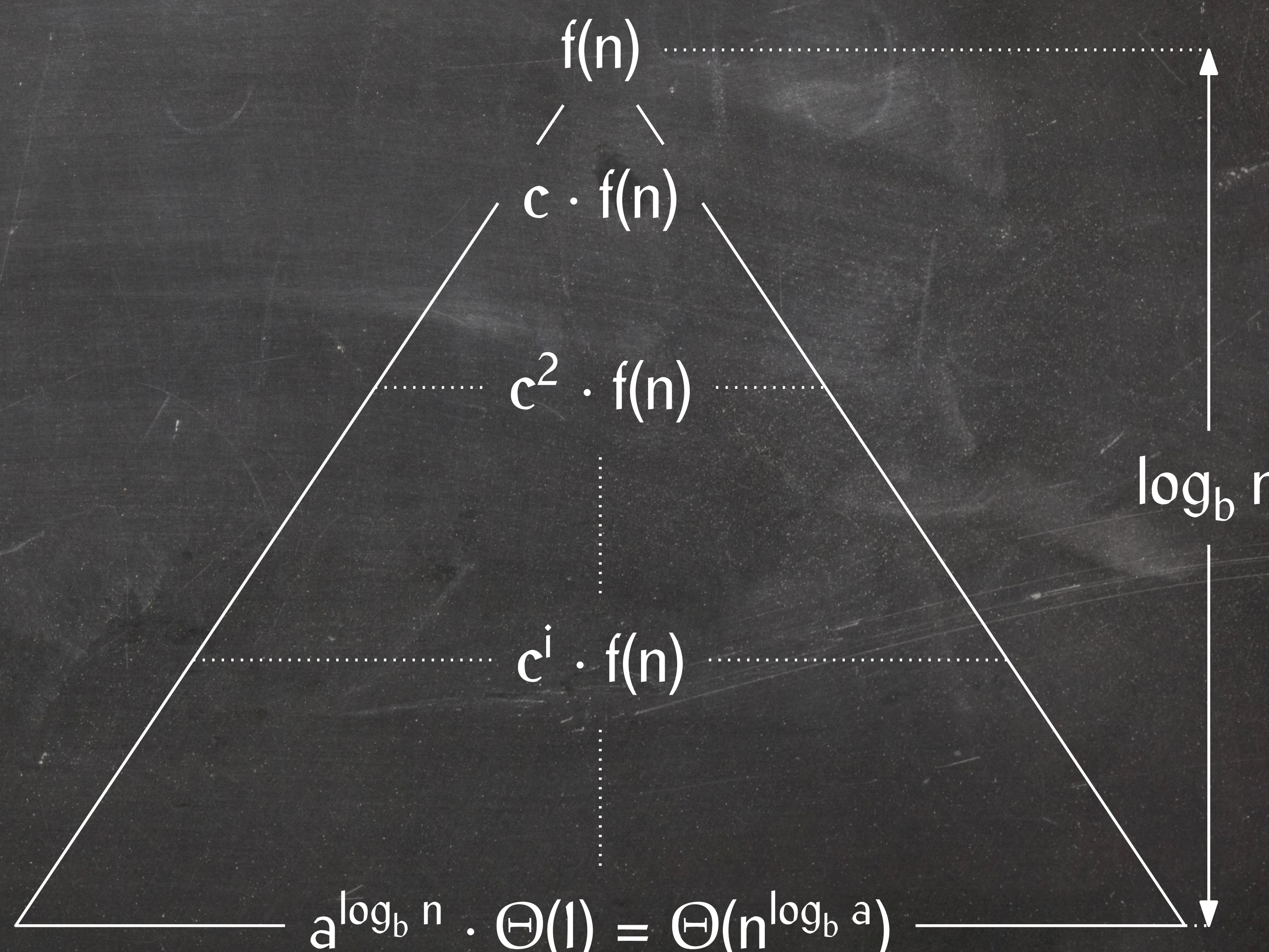
Case 3: $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

$$T(n) \in \Omega(n^{\log_b a} + f(n)) = \Omega(f(n))$$

$$T(n) \in O\left(n^{\log_b a} + \sum_{i=0}^{\infty} c^i \cdot f(n)\right)$$

$$= O\left(n^{\log_b a} + f(n) \cdot \sum_{i=0}^{\infty} c^i\right)$$



Master Theorem: Proof

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a + \epsilon})$ and $a \cdot f(n/b) \leq c \cdot f(n)$ for some $c < 1$

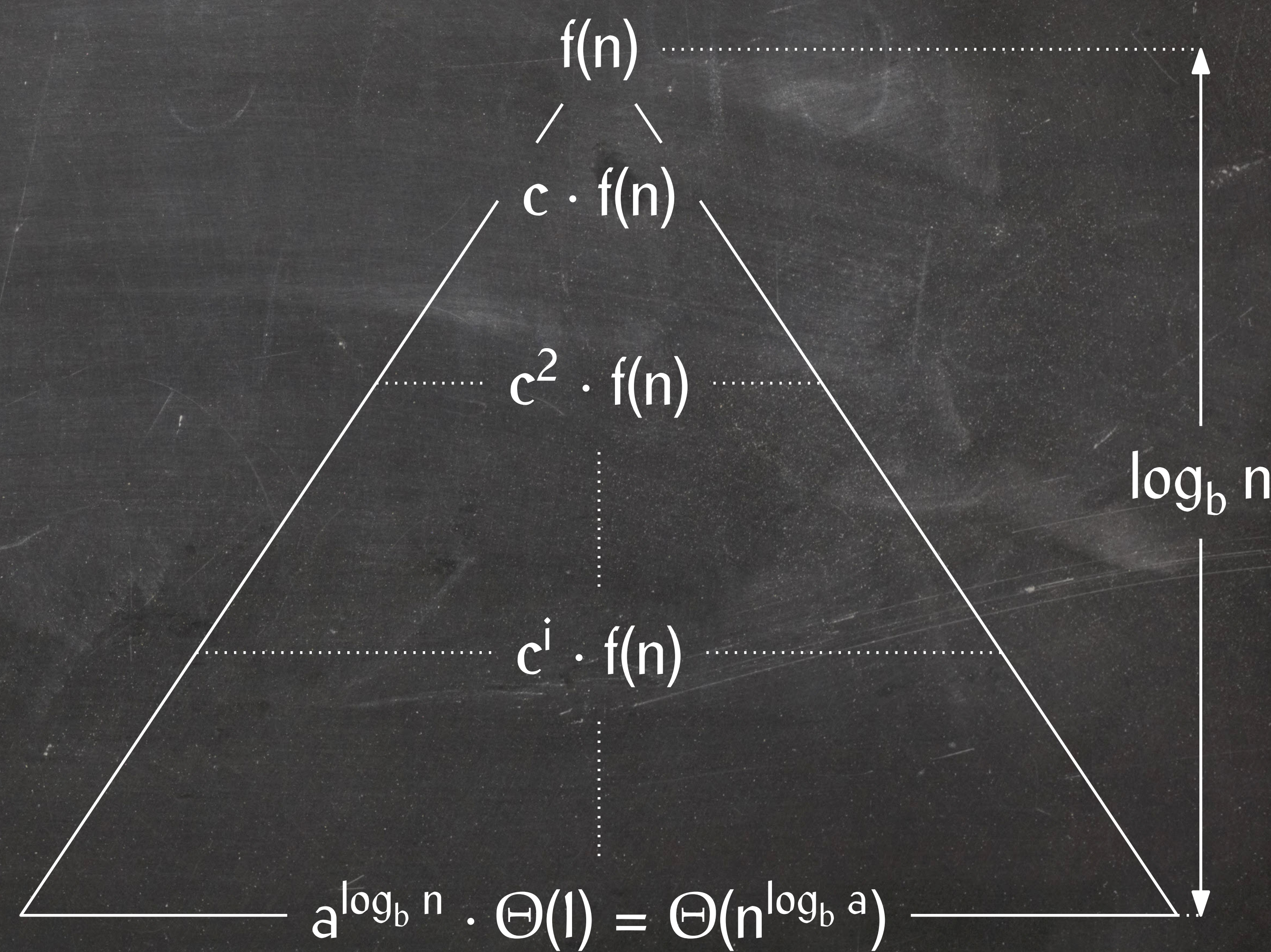
Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \leq c^i \cdot f(n)$

$$T(n) \in \Omega(n^{\log_b a} + f(n)) = \Omega(f(n))$$

$$T(n) \in O\left(n^{\log_b a} + \sum_{i=0}^{\infty} c^i \cdot f(n)\right)$$

$$= O\left(n^{\log_b a} + f(n) \cdot \sum_{i=0}^{\infty} c^i\right)$$

$$= O\left(n^{\log_b a} + f(n) \cdot \frac{1}{1-c}\right)$$



Master Theorem: Proof

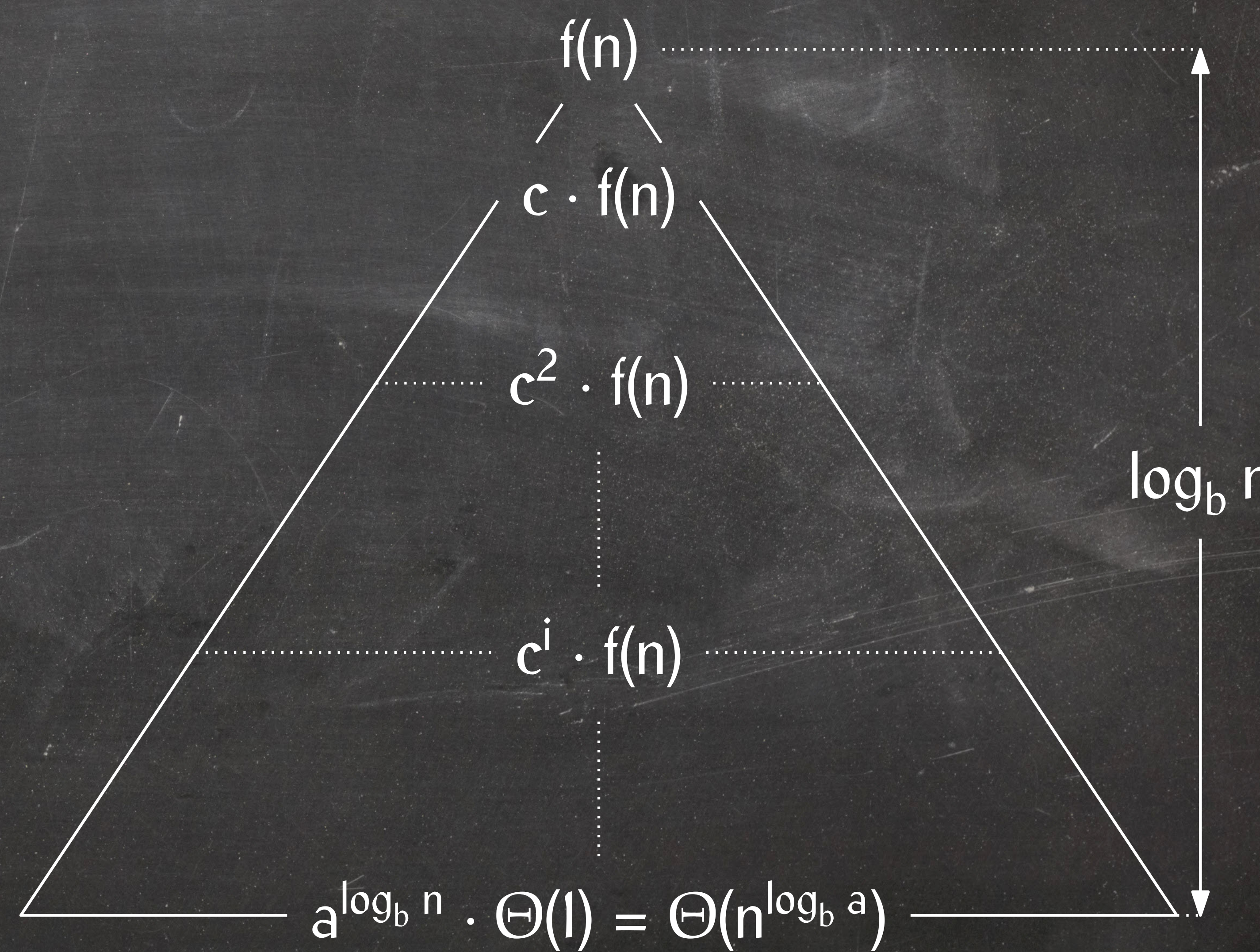
$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Case 3: $f(n) \in \Omega(n^{\log_b a+\epsilon})$ and $a \cdot f(n/b) \le c \cdot f(n)$ for some $c < 1$

Claim: $a^i \cdot f\left(\frac{n}{b^i}\right) \le c^i \cdot f(n)$

$$T(n) \in \Omega(n^{\log_b a} + f(n)) = \Omega(f(n))$$

$$\begin{aligned} T(n) \in O\left(n^{\log_b a} + \sum_{i=0}^{\infty} c^i \cdot f(n)\right) &= O\left(n^{\log_b a} + f(n) \cdot \sum_{i=0}^{\infty} c^i\right) \\ &= O\left(n^{\log_b a} + f(n) \cdot \frac{1}{1-c}\right) \\ &= O(f(n)) \end{aligned}$$



Quick Sort

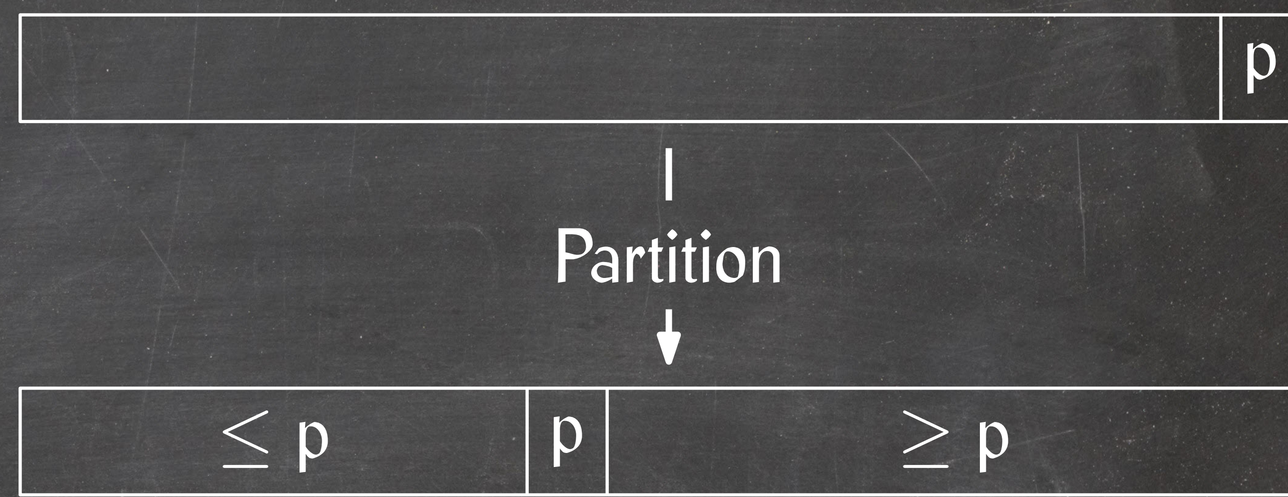
QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```

Quick Sort

QuickSort(A, ℓ, r)

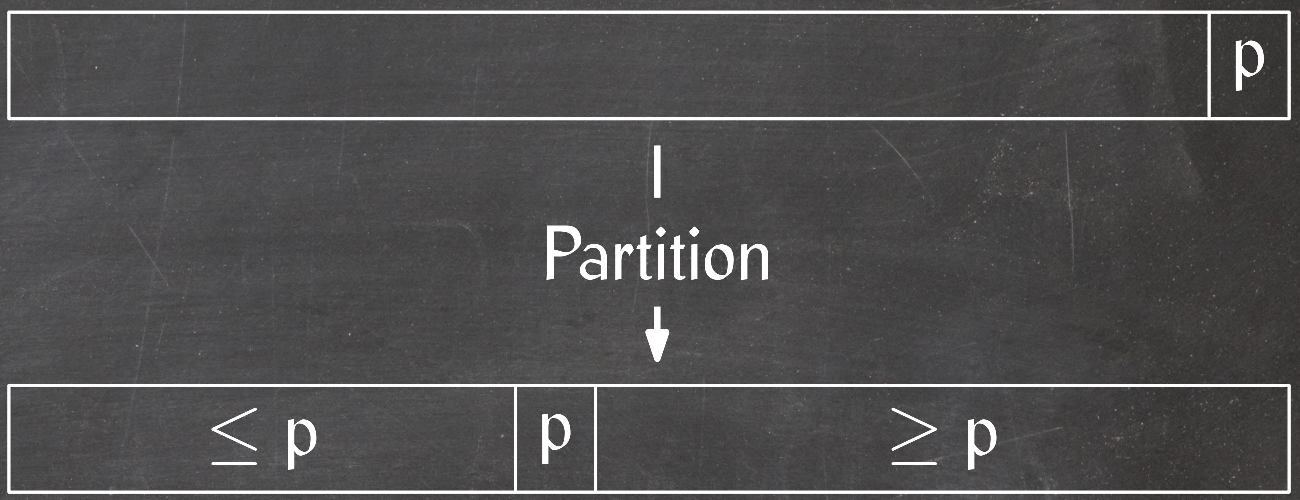
```
if r ≤ ℓ
    then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```



Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```

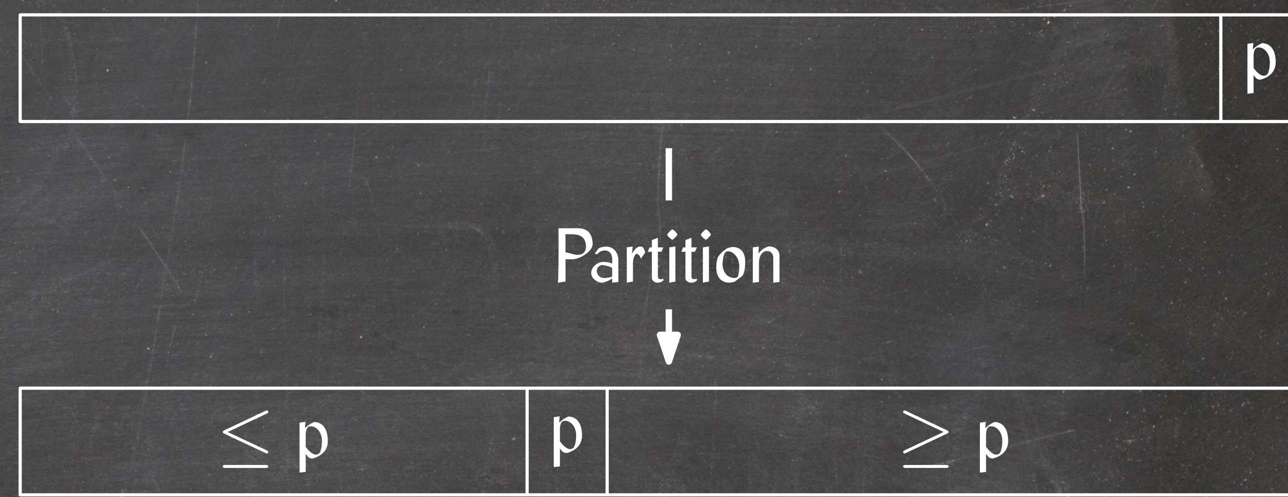


Worst case:

Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```



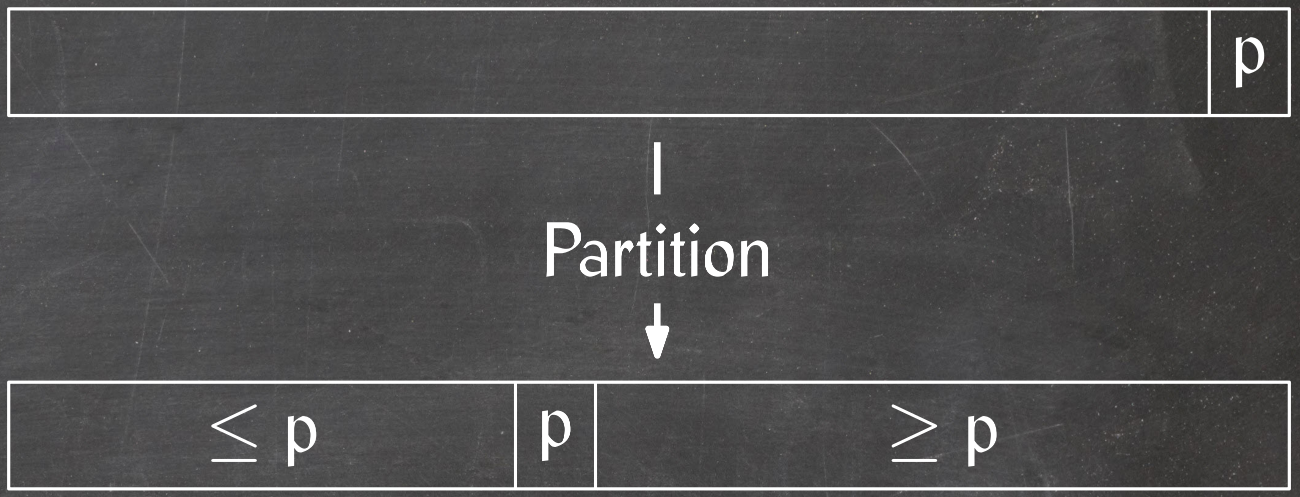
Worst case:

$$T(n) = \Theta(n) + T(n-1)$$

Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```



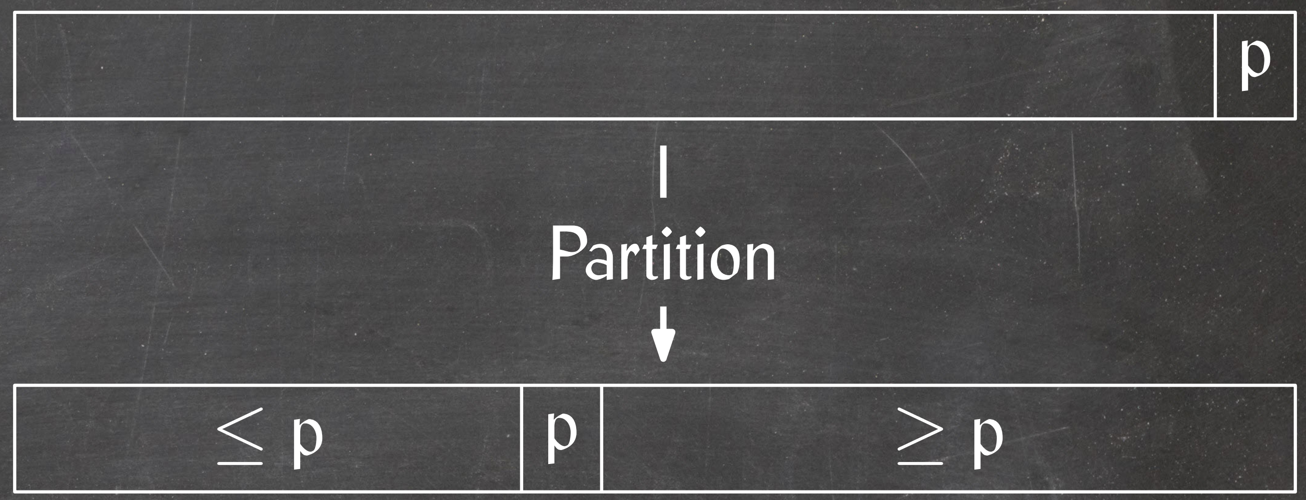
Worst case:

$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
   then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```



Worst case:

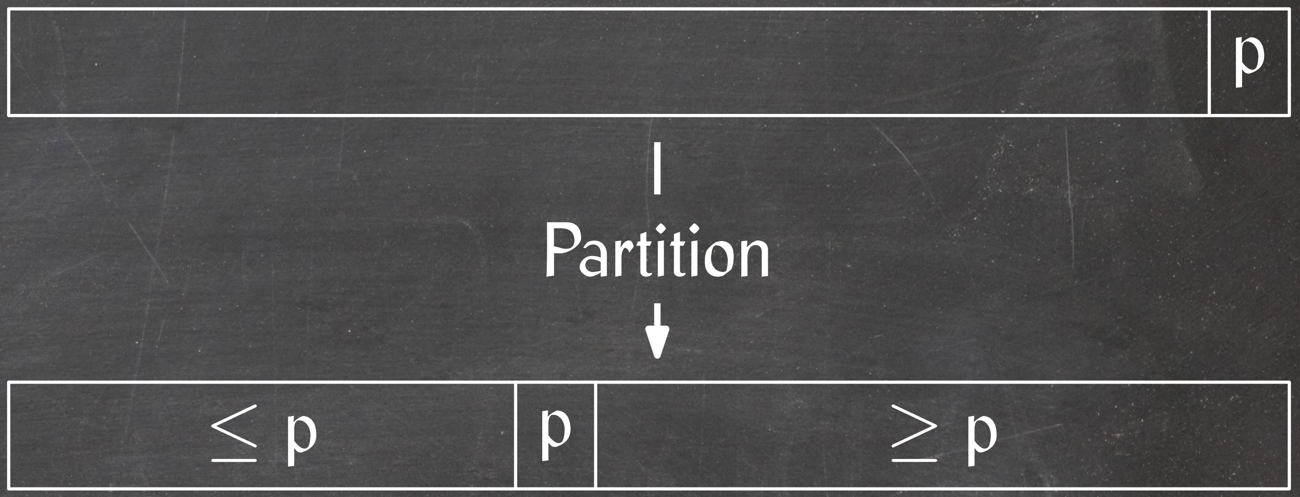
$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

Best case:

Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```



Worst case:

$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

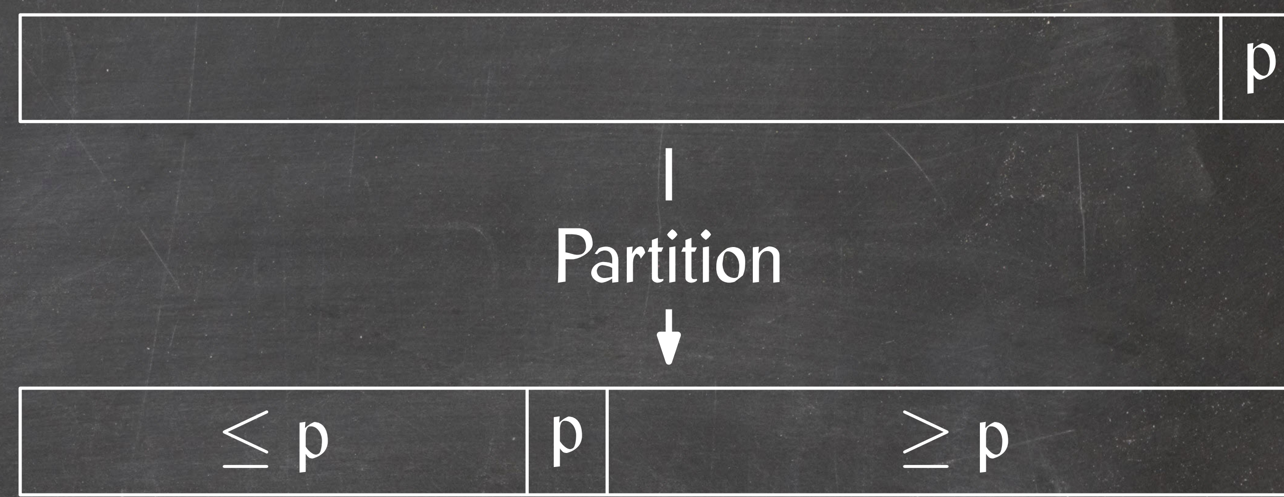
Best case:

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

Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```



Worst case:

$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

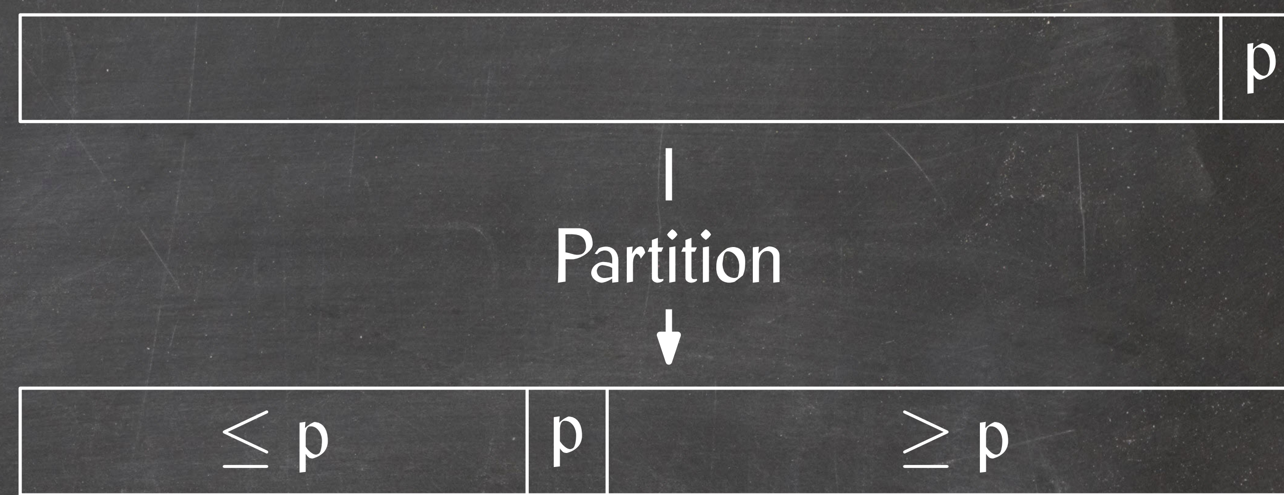
Best case:

$$T(n) = \Theta(n) + 2T(\lfloor n/2 \rfloor) = \Theta(n \lg n)$$

Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
m = Partition(A, ℓ, r)
QuickSort(A, ℓ, m − 1)
QuickSort(A, m + 1, r)
```



Worst case:

$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

Best case:

$$T(n) = \Theta(n) + 2T(\lfloor n/2 \rfloor) = \Theta(n \lg n)$$

Average case:

$$T(n) = \Theta(n \lg n)$$

Two Partitioning Algorithms

HoarePartition(A, l, r)

```
x = A[r]
i = l − 1
j = r + 1
while True
    do repeat i = i + 1
           until A[i] ≥ x
       repeat j = j − 1
           until A[j] ≤ x
       if i < j
           then swap A[i] and A[j]
           else return j
```

Loop invariants:

Two Partitioning Algorithms

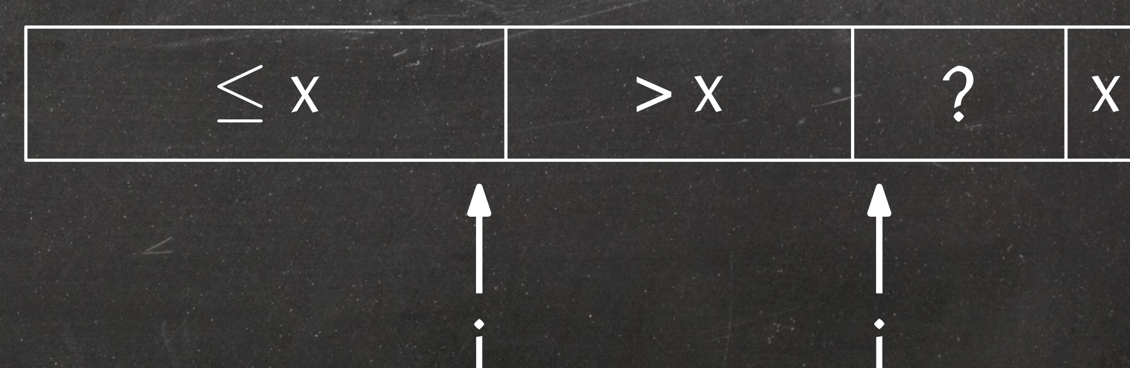
HoarePartition(A, l, r)

```
x = A[r]
i = l − 1
j = r + 1
while True
   do repeat i = i + 1
         until A[i] ≥ x
         repeat j = j − 1
         until A[j] ≤ x
         if i < j
            then swap A[i] and A[j]
            else  return j
```

LomutoPartition(A, l, r)

```
i = l − 1
for j = l to r − 1
   do if A[j] ≤ A[r]
         then i = i + 1
              swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

Loop invariants:



Two Partitioning Algorithms

HoarePartition(A, l, r)

```
x = A[r]
i = l − 1
j = r + 1
while True
   do repeat i = i + 1
          until A[i] ≥ x
          repeat j = j − 1
          until A[j] ≤ x
          if i < j
             then swap A[i] and A[j]
             else  return j
```

LomutoPartition(A, l, r)

```
i = l − 1
for j = l to r − 1
   do if A[j] ≤ A[r]
         then i = i + 1
               swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

HoarePartition is more efficient in practice.

LomutoPartition has some properties that make average-case analysis easier.

HoarePartition is more convenient for worst-case Quick Sort.

Two Partitioning Algorithms

HoarePartition(A, l, r, x)

```
i = l − 1
j = r + 1
while True
    do repeat i = i + 1
          until A[i] ≥ x
          repeat j = j − 1
          until A[j] ≤ x
          if i < j
              then swap A[i] and A[j]
              else  return j
```

LomutoPartition(A, l, r)

```
i = l − 1
for j = l to r − 1
    do if A[j] ≤ A[r]
           then i = i + 1
                swap A[i] and A[j]
swap A[i + 1] and A[r]
return i + 1
```

HoarePartition is more efficient in practice.

LomutoPartition has some properties that make average-case analysis easier.

HoarePartition is more convenient for worst-case Quick Sort.

Selection

Goal: Given an unsorted array A and an integer $1 \leq k \leq n$, find the kth smallest element in A.

8	17	5	43	3	12	64	21

$\uparrow$

4th smallest element

Selection

Goal: Given an unsorted array A and an integer $1 \le k \le n$, find the kth smallest element in A.

8	17	5	43	3	12	64	21

4th smallest element

First idea: Sort the array and then return the element in the kth position.

$\Rightarrow$ $O(n \lg n)$ time

Selection

Goal: Given an unsorted array A and an integer $1 \le k \le n$, find the kth smallest element in A.

8	17	5	43	3	12	64	21

4th smallest element

First idea: Sort the array and then return the element in the kth position.

$\Rightarrow$ $O(n \lg n)$ time

We can find the minimum $(k = 1)$ or the maximum $(k = n)$ in $O(n)$ time!

Selection

Goal: Given an unsorted array A and an integer $1 \leq k \leq n$, find the kth smallest element in A.

8	17	5	43	3	12	64	21

4th smallest element

First idea: Sort the array and then return the element in the kth position.

$\Rightarrow$ $O(n \lg n)$ time

We can find the minimum ($k = 1$) or the maximum ($k = n$) in $O(n)$ time!

It would be nice if we were able to find the kth smallest element, for any k, in $O(n)$ time.

The Sorting Idea Isn't Half Bad, But ...

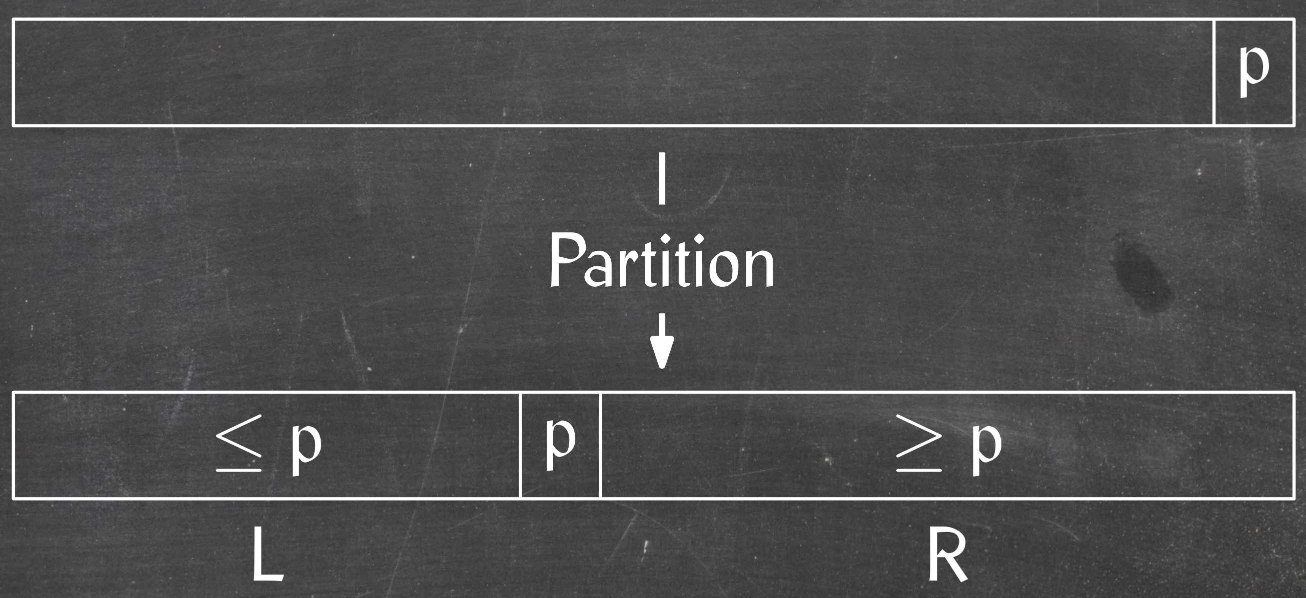
To find the kth smallest element, we don't need to sort the input completely.

We only need to verify that there are exactly $k - 1$ elements smaller than the element we return.

The Sorting Idea Isn't Half Bad, But …

To find the kth smallest element, we don't need to sort the input completely.

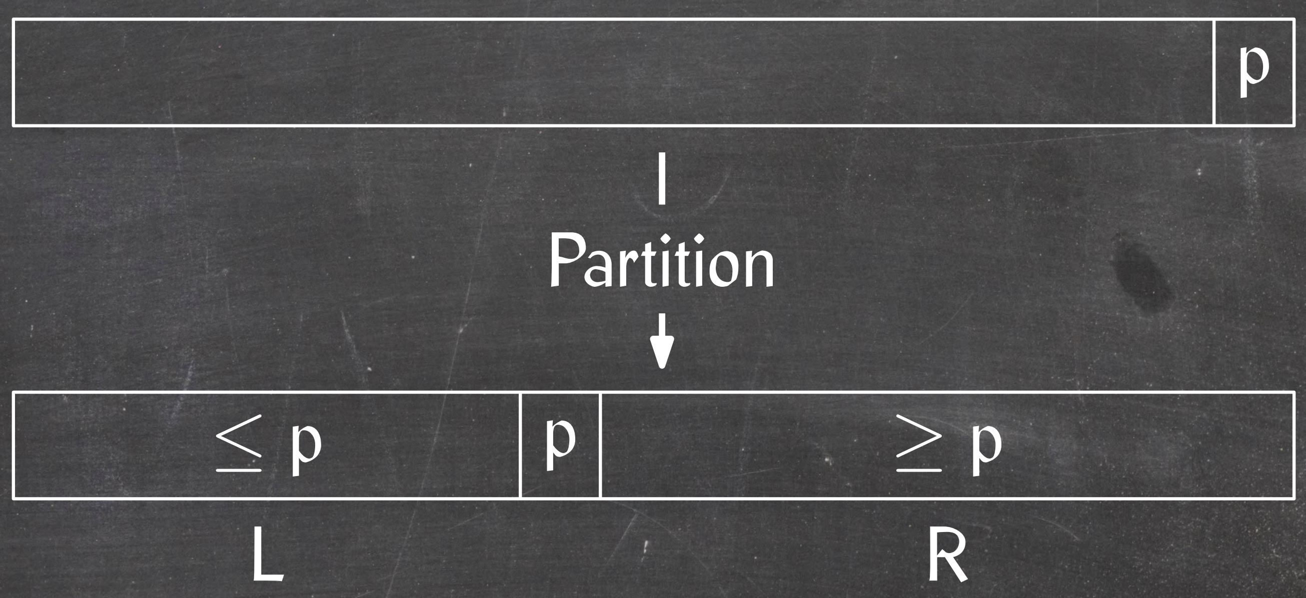
We only need to verify that there are exactly $k - 1$ elements smaller than the element we return.



The Sorting Idea Isn't Half Bad, But ...

To find the kth smallest element, we don't need to sort the input completely.

We only need to verify that there are exactly $k - 1$ elements smaller than the element we return.

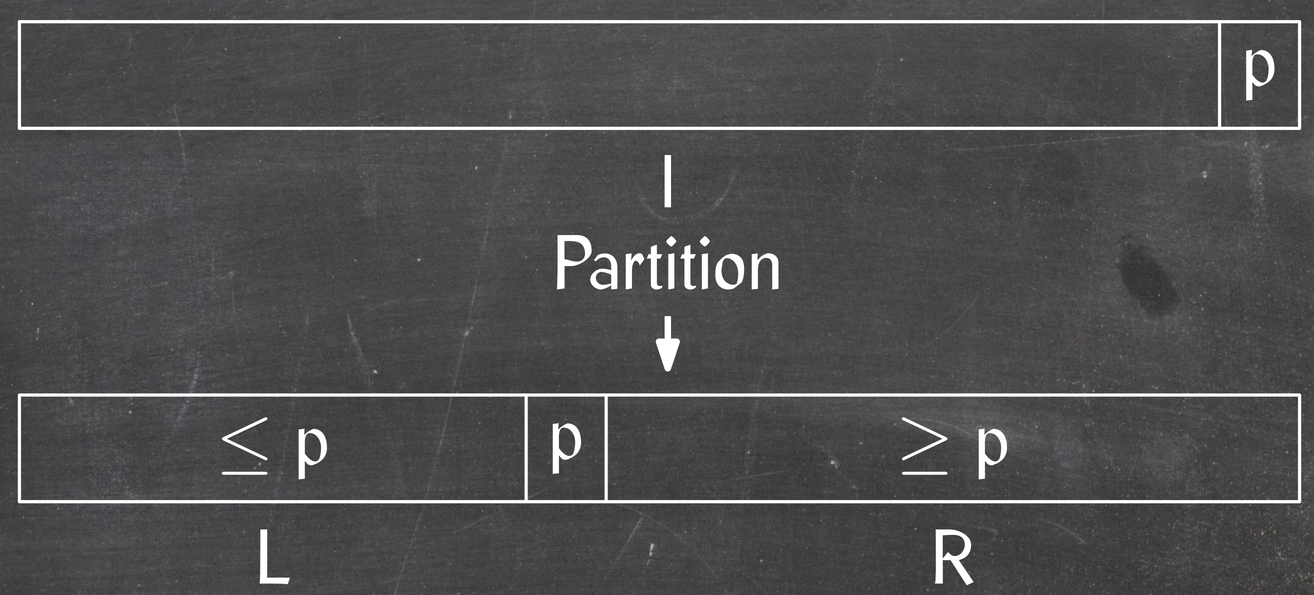


If $|L| = k - 1$, then p is the kth smallest element.

The Sorting Idea Isn't Half Bad, But …

To find the kth smallest element, we don't need to sort the input completely.

We only need to verify that there are exactly $k - 1$ elements smaller than the element we return.



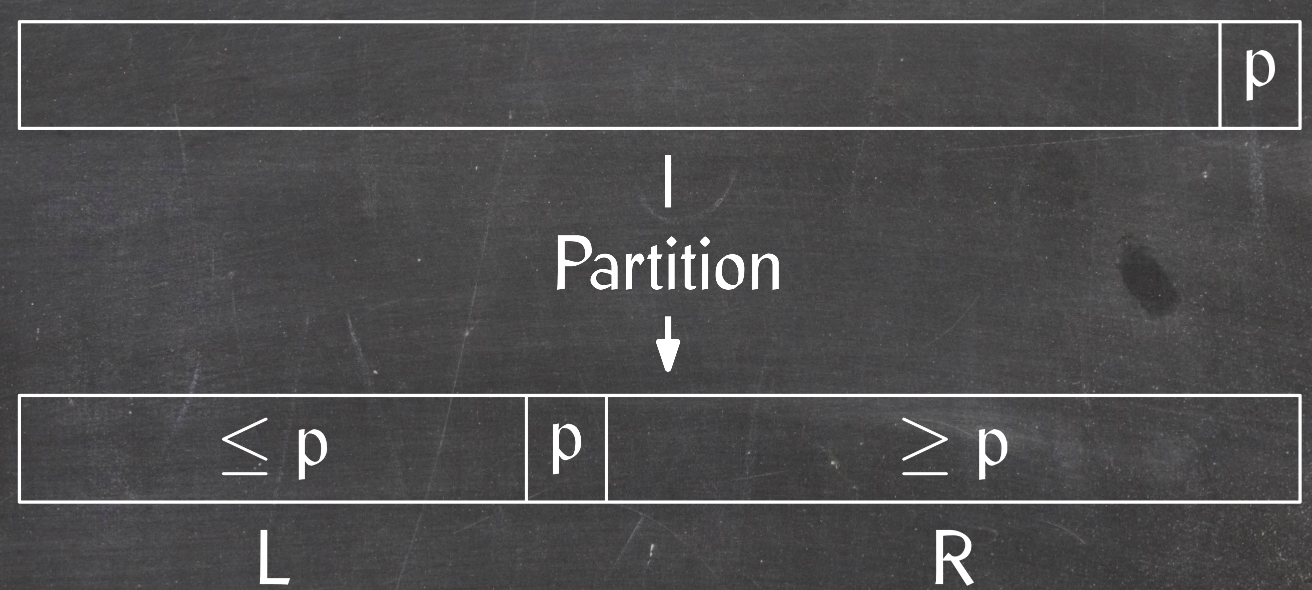
If $|L| = k - 1$, then p is the kth smallest element.

If $|L| \geq k$, then the kth smallest element in L is the kth smallest element in A.

The Sorting Idea Isn't Half Bad, But …

To find the kth smallest element, we don't need to sort the input completely.

We only need to verify that there are exactly $k - 1$ elements smaller than the element we return.



If $|L| = k - 1$, then p is the kth smallest element.

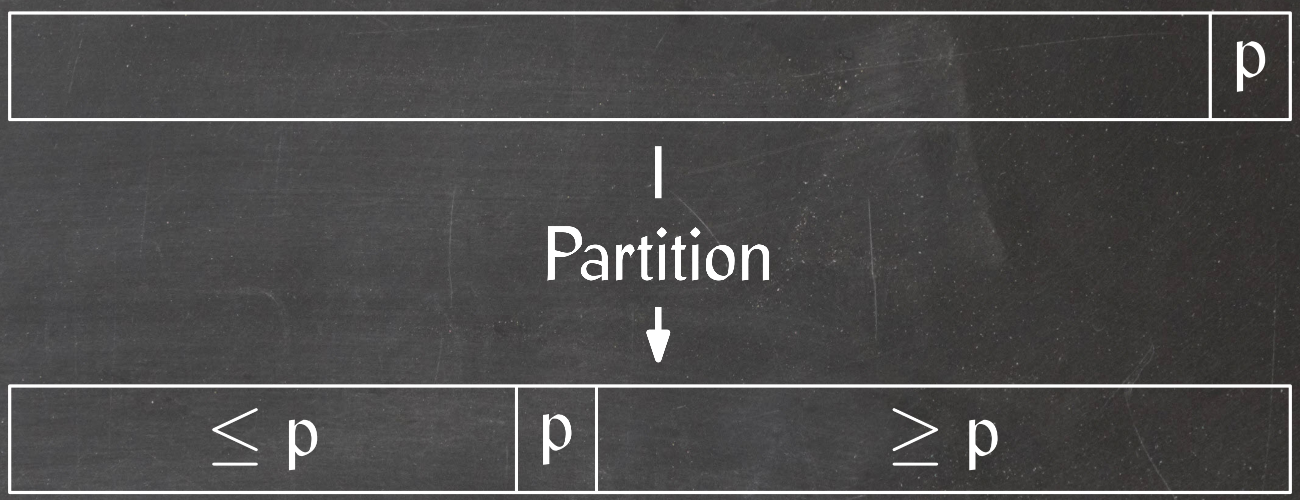
If $|L| \geq k$, then the kth smallest element in L is the kth smallest element in A.

If $|L| < k - 1$, then the $(k - |L| + 1)$st element in R is the kth smallest element in A.

Quick Select

QuickSelect(A, ℓ, r, k)

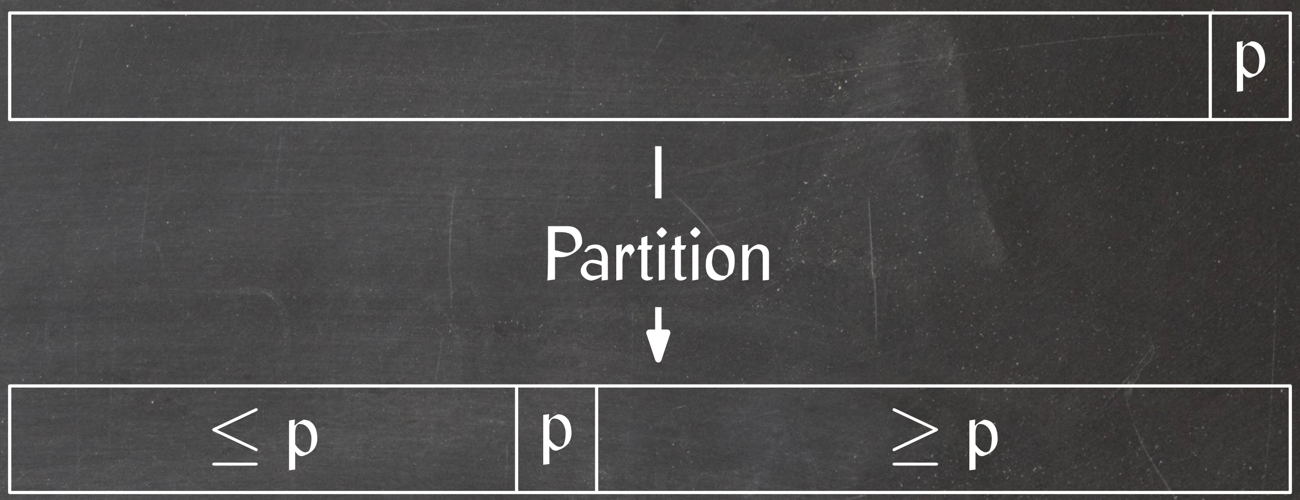
```
if r ≤ ℓ
    then return A[ℓ]
m = Partition(A, ℓ, r)
if m − ℓ = k − 1
    then return A[m]
    else if m − ℓ ≥ k
            then return QuickSelect(A, ℓ, m − 1, k)
            else return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```



Quick Select

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
    then return A[ℓ]
m = Partition(A, ℓ, r)
if m − ℓ = k − 1
    then return A[m]
    else if m − ℓ ≥ k
            then return QuickSelect(A, ℓ, m − 1, k)
            else return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```

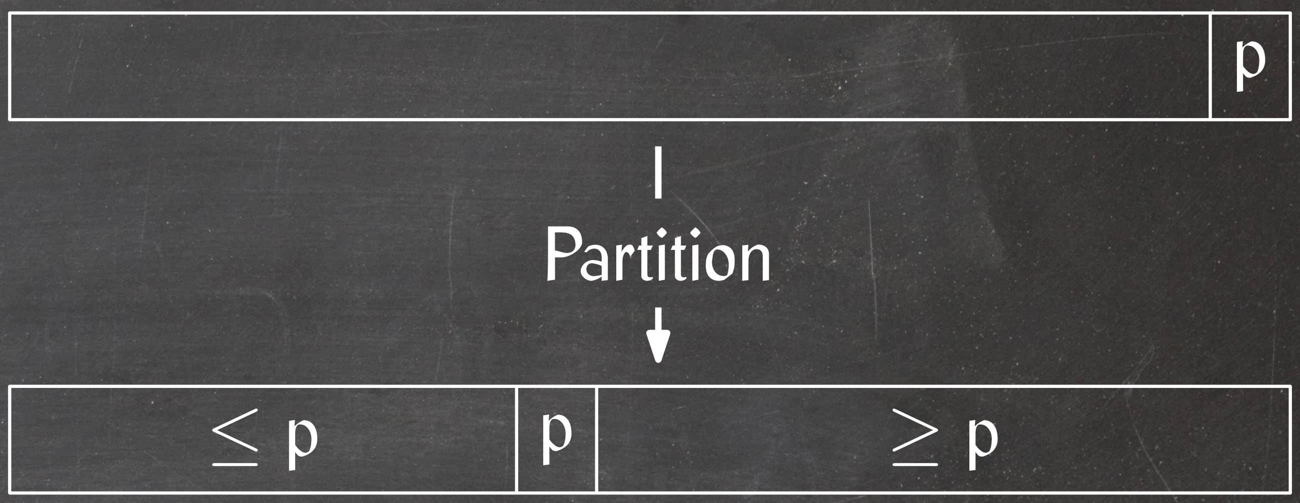


Worst case:

Quick Select

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
    then return A[ℓ]
m = Partition(A, ℓ, r)
if m − ℓ = k − 1
    then return A[m]
    else if m − ℓ ≥ k
            then return QuickSelect(A, ℓ, m − 1, k)
            else return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```



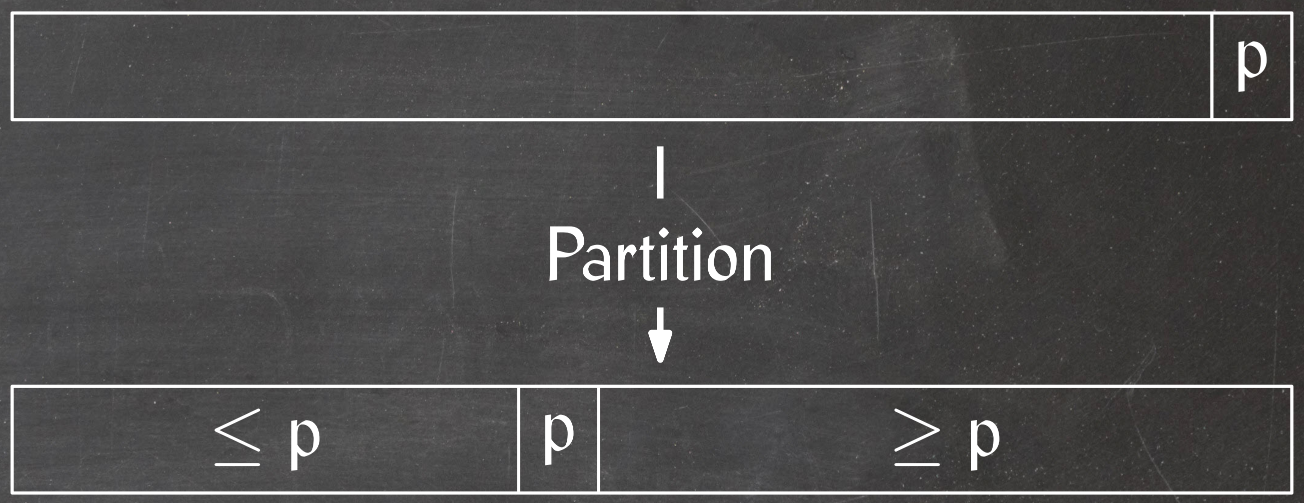
Worst case:

$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

Quick Select

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
   then return A[ℓ]
m = Partition(A, ℓ, r)
if m − ℓ = k − 1
   then return A[m]
   else if m − ℓ ≥ k
           then return QuickSelect(A, ℓ, m − 1, k)
           else return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```



Worst case:

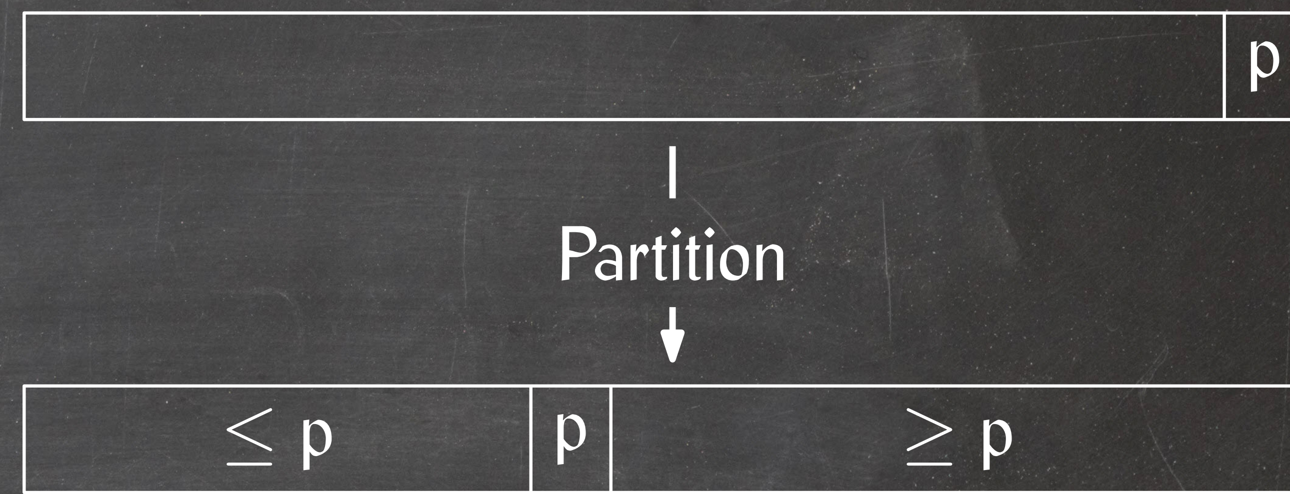
$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

Best case:

Quick Select

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
   then return A[ℓ]
m = Partition(A, ℓ, r)
if m − ℓ = k − 1
   then return A[m]
   else if m − ℓ ≥ k
           then return QuickSelect(A, ℓ, m − 1, k)
           else return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```



Worst case:

$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

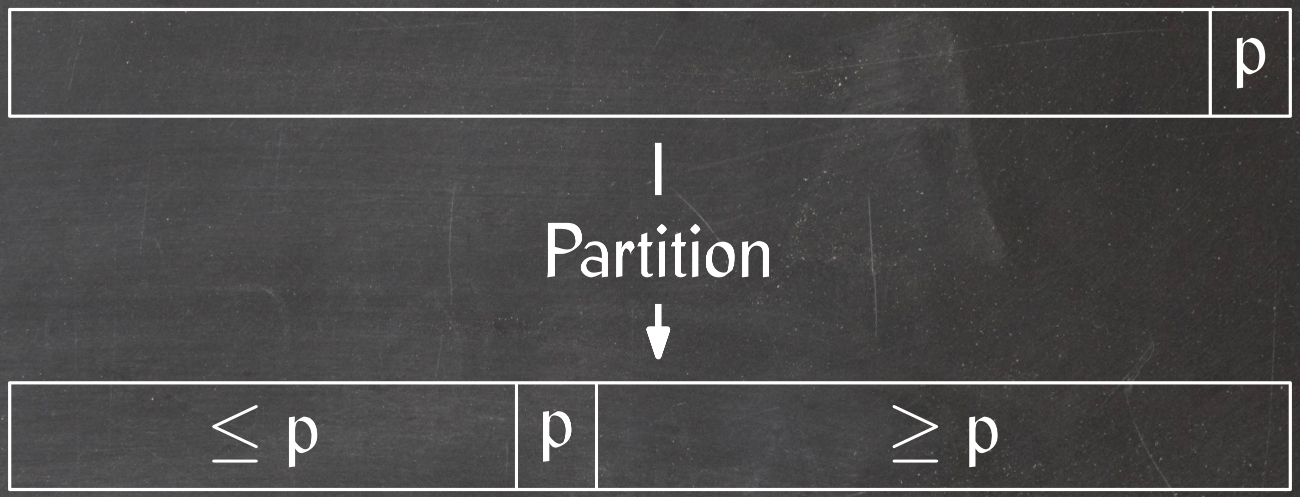
Best case:

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

Quick Select

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
    then return A[ℓ]
m = Partition(A, ℓ, r)
if m − ℓ = k − 1
    then return A[m]
    else if m − ℓ ≥ k
             then return QuickSelect(A, ℓ, m − 1, k)
             else return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```



Worst case:

$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

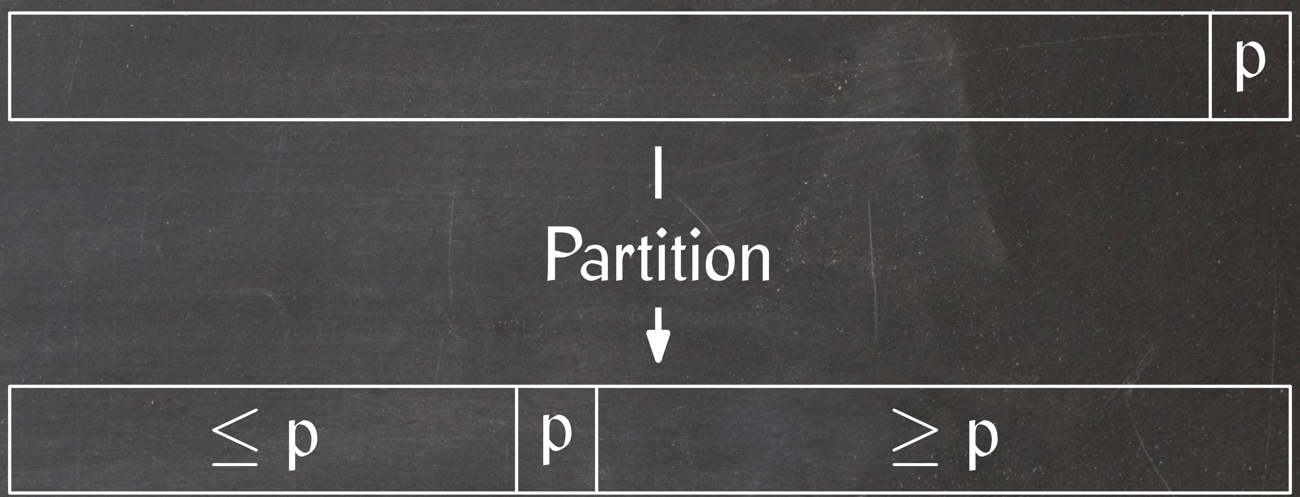
Best case:

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

Quick Select

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
    then return A[ℓ]
m = Partition(A, ℓ, r)
if m − ℓ = k − 1
    then return A[m]
    else if m − ℓ ≥ k
            then return QuickSelect(A, ℓ, m − 1, k)
            else return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```



Worst case:

$$T(n) = \Theta(n) + T(n-1) = \Theta(n^2)$$

Best case:

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

Average case:

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

Worst-Case Selection

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
    then return A[ℓ]
p = FindPivot(A, ℓ, r)
m = HoarePartition(A, ℓ, r, p)
if m − ℓ + 1 ≥ k
    then return QuickSelect(A, ℓ, m, k)
    else  return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```

Worst-Case Selection

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
    then return A[ℓ]
p = FindPivot(A, ℓ, r)
m = HoarePartition(A, ℓ, r, p)
if m − ℓ + 1 ≥ k
    then return QuickSelect(A, ℓ, m, k)
    else return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```

If we could guarantee that p is the median of A[ℓ . . r], then we'd recurse on at most n/2 elements.

Worst-Case Selection

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
    then return A[ℓ]
p = FindPivot(A, ℓ, r)
m = HoarePartition(A, ℓ, r, p)
if m − ℓ + 1 ≥ k
    then return QuickSelect(A, ℓ, m, k)
    else  return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```

If we could guarantee that p is the median of $A[\ell \, .. \, r]$, then we'd recurse on at most n/2 elements.

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

Worst-Case Selection

QuickSelect(A, ℓ, r, k)

```
if r ≤ ℓ
    then return A[ℓ]
p = FindPivot(A, ℓ, r)
m = HoarePartition(A, ℓ, r, p)
if m − ℓ + 1 ≥ k
    then return QuickSelect(A, ℓ, m, k)
    else  return QuickSelect(A, m + 1, r, k − (m + 1 − ℓ))
```

If we could guarantee that p is the median of $A[\ell \,..\, r]$, then we'd recurse on at most $n/2$ elements.

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

Alas, finding the median is selection!

Making Do With An Approximate Median

$$\underbrace{\boxed{\quad \leq p \quad}}_{\geq \epsilon n}\boxed{p}\underbrace{\boxed{\quad \geq p \quad}}_{\geq \epsilon n}$$

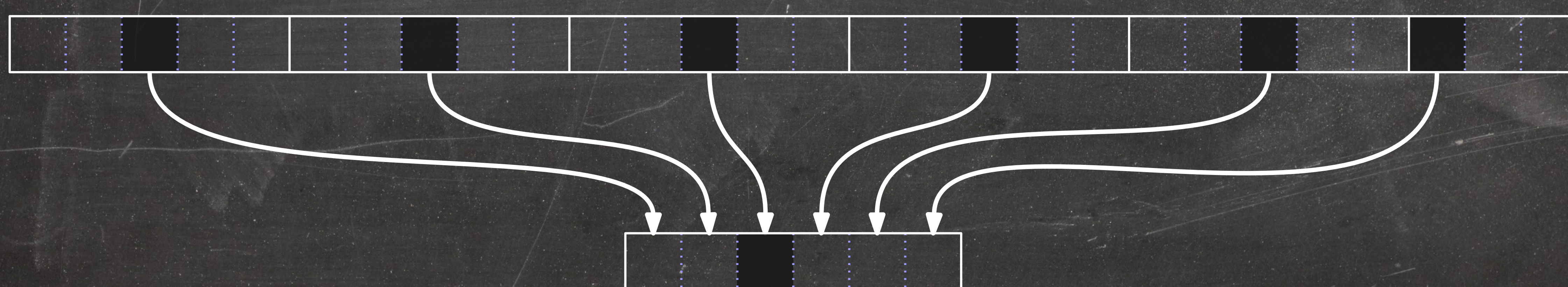
If there are at least ϵn elements smaller than p and at least ϵn elements greater than p, then

$$T(n) \leq \Theta(n) + T((1-\epsilon)n) = \Theta(n).$$

Finding An Approximate Median

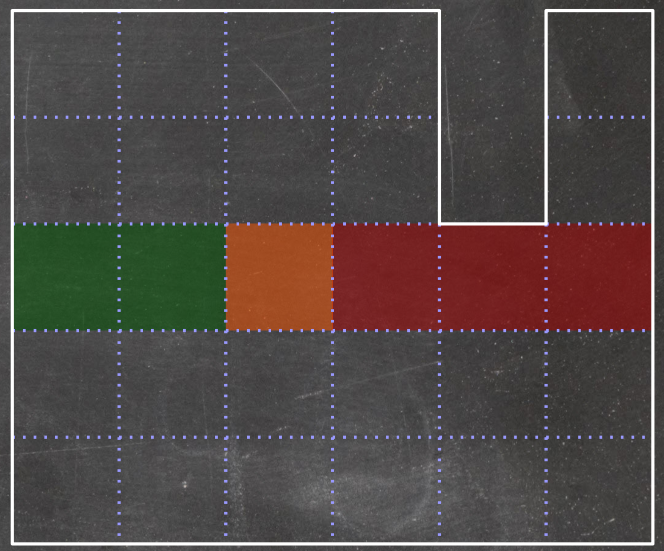
FindPivot(A, ℓ, r)

```
n′ = ⌊(r − ℓ)/5⌋ + 1
for i = 0 to n′ − 1
    do InsertionSort(A, ℓ + 5 · i, min(ℓ + 5 · i + 4, r))
       if ℓ + 5i + 4 ≤ r
          then B[i + 1] = A[ℓ + 5 · i + 2]
          else B[i + 1] = A[ℓ + 5 · i]
return QuickSelect(B, 1, n′, ⌈n′/2⌉)
```



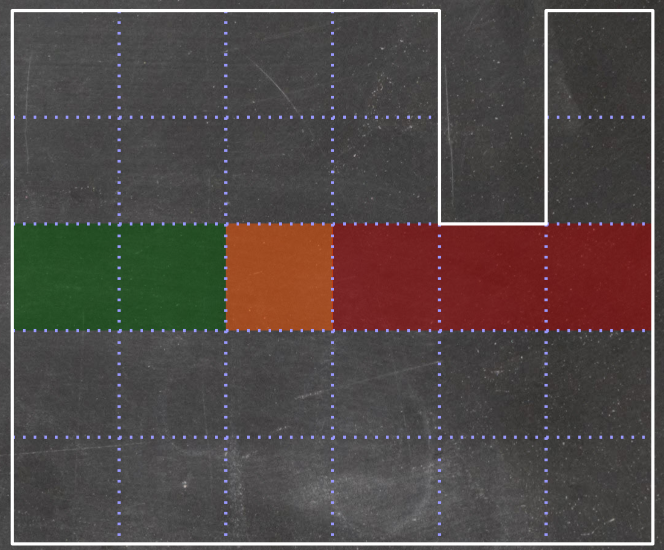
Approximate median

Finding An Approximate Median



Finding An Approximate Median

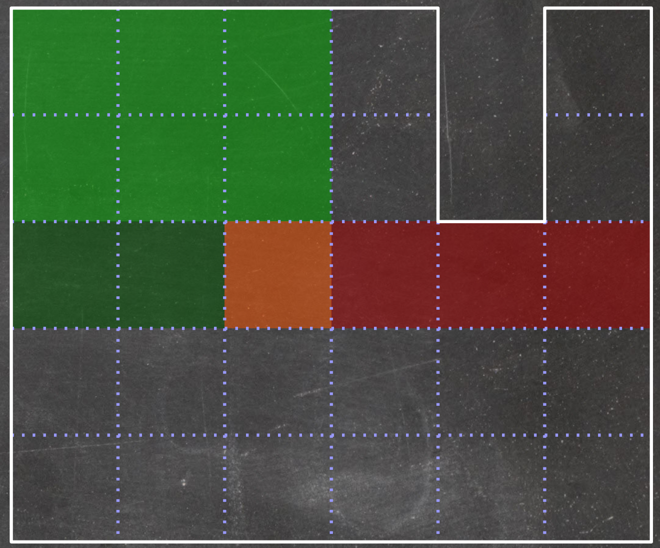
There are at least $\lceil n'/2 \rceil - 1$ medians smaller than the median of medians.



Finding An Approximate Median

There are at least $\lceil n'/2 \rceil - 1$ medians smaller than the median of medians.

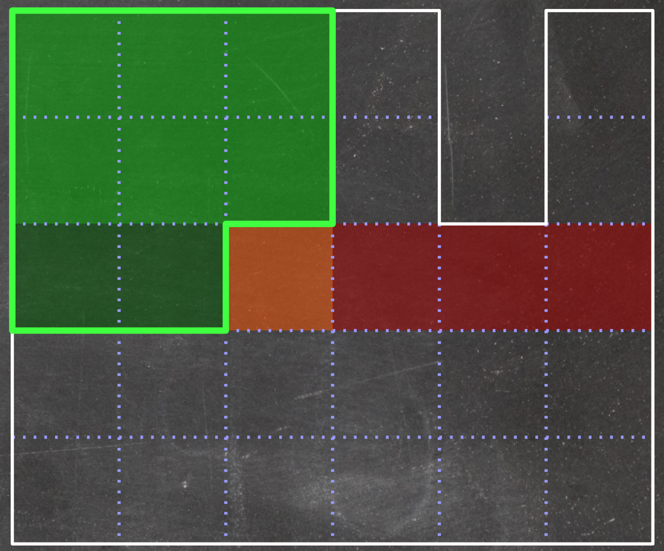
For at least $\lceil n'/2 \rceil - 1$ of the medians, there are two elements in each of their groups that are smaller than the median of medians.



Finding An Approximate Median

There are at least $\lceil n'/2 \rceil - 1$ medians smaller than the median of medians.

For at least $\lceil n'/2 \rceil - 1$ of the medians, there are two elements in each of their groups that are smaller than the median of medians.

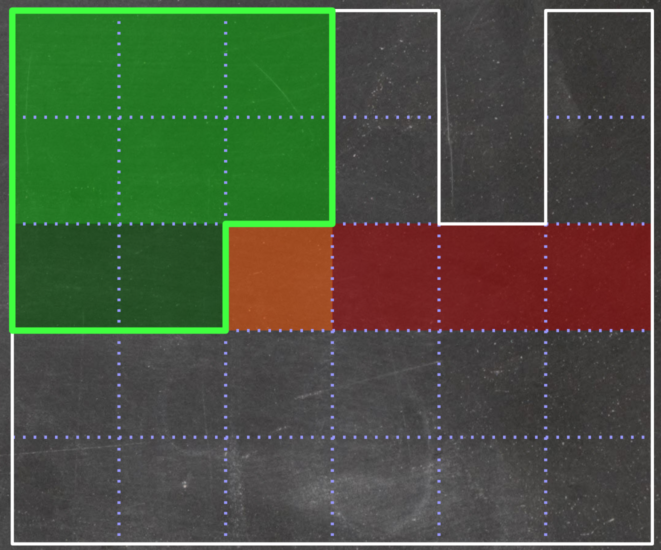


Finding An Approximate Median

There are at least $\lceil n'/2 \rceil - 1$ medians smaller than the median of medians.

For at least $\lceil n'/2 \rceil - 1$ of the medians, there are two elements in each of their groups that are smaller than the median of medians.

$n' = \lceil n/5 \rceil$

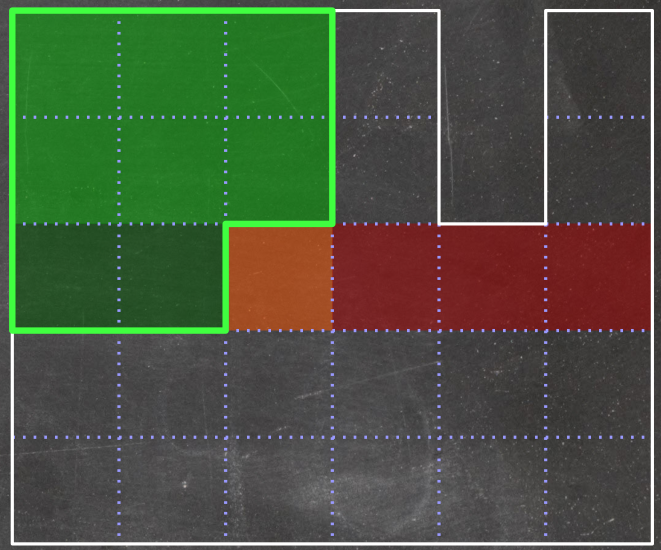


Finding An Approximate Median

There are at least $\lceil n'/2 \rceil - 1$ medians smaller than the median of medians.

For at least $\lceil n'/2 \rceil - 1$ of the medians, there are two elements in each of their groups that are smaller than the median of medians.

$n' = \lceil n/5 \rceil$



Total number of elements smaller than the median of medians:

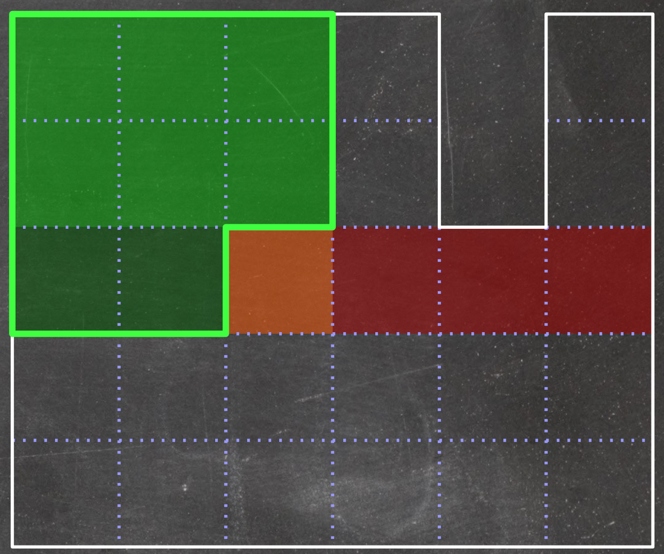
$$3\left(\left\lceil \frac{n'}{2} \right\rceil - 1\right) = 3\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil - 3 \geq \frac{3n}{10} - 3$$

Finding An Approximate Median

There are at least $\lceil n'/2 \rceil - 1$ medians smaller than the median of medians.

For at least $\lceil n'/2 \rceil - 1$ of the medians, there are two elements in each of their groups that are smaller than the median of medians.

$n' = \lceil n/5 \rceil$



Total number of elements smaller than the median of medians:

$$3\left(\left\lceil \frac{n'}{2} \right\rceil - 1\right) = 3\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil - 3 \geq \frac{3n}{10} - 3$$

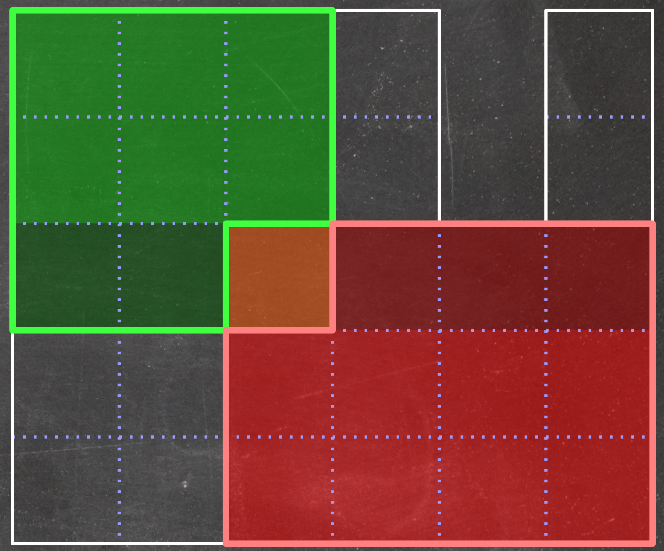
The same analysis holds for counting the number of elements greater than the median of medians.

Finding An Approximate Median

There are at least $\lceil n'/2 \rceil - 1$ medians smaller than the median of medians.

For at least $\lceil n'/2 \rceil - 1$ of the medians, there are two elements in each of their groups that are smaller than the median of medians.

$n' = \lceil n/5 \rceil$



Total number of elements smaller than the median of medians:

$$3\left(\left\lceil \frac{n'}{2} \right\rceil - 1\right) = 3\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil - 3 \geq \frac{3n}{10} - 3$$

The same analysis holds for counting the number of elements greater than the median of medians.

Worst-Case Selection: Analysis

FindPivot (excluding the recursive call to QuickSelect) and Partition take $O(n)$ time.

Worst-Case Selection: Analysis

FindPivot (excluding the recursive call to QuickSelect) and Partition take $O(n)$ time.

FindPivot recurses on $\lceil \frac{n}{5} \rceil < \frac{n}{5} + 1$ elements.

Worst-Case Selection: Analysis

FindPivot (excluding the recursive call to QuickSelect) and Partition take $O(n)$ time.

FindPivot recurses on $\lceil \frac{n}{5} \rceil < \frac{n}{5} + 1$ elements.

QuickSelect itself recurses on at most $\frac{7n}{10} + 3$ elements.

Worst-Case Selection: Analysis

FindPivot (excluding the recursive call to QuickSelect) and Partition take $O(n)$ time.

FindPivot recurses on $\lceil \frac{n}{5} \rceil < \frac{n}{5} + 1$ elements.

QuickSelect itself recurses on at most $\frac{7n}{10} + 3$ elements.

$$T(n) \leq T(\tfrac{n}{5} + 1) + T(\tfrac{7n}{10} + 3) + O(n)$$

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Base case: $(n < 80)$

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Base case: $(n < 80)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Base case: $(n < 80)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.

$\Rightarrow$ $T(n) \leq c' \leq cn$ for c sufficiently large.

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Base case: $(n < 80)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.

$\Rightarrow$ $T(n) \leq c' \leq cn$ for c sufficiently large.

Inductive Step: $(n \geq 80)$

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Base case: $(n < 80)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.
$\Rightarrow$ $T(n) \leq c' \leq cn$ for c sufficiently large.

Inductive Step: $(n \geq 80)$

$$T(n) \leq T\left(\frac{n}{5} + 1\right) + T\left(\frac{7n}{10} + 3\right) + an$$

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Base case: $(n < 80)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.

$\Rightarrow$ $T(n) \leq c' \leq cn$ for c sufficiently large.

Inductive Step: $(n \geq 80)$

$$\begin{aligned} T(n) &\leq T\left(\frac{n}{5} + 1\right) + T\left(\frac{7n}{10} + 3\right) + an \\ &\leq c\left(\frac{n}{5} + 1\right) + c\left(\frac{7n}{10} + 3\right) + an \end{aligned}$$

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Base case: $(n < 80)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.
$\Rightarrow$ $T(n) \leq c' \leq cn$ for c sufficiently large.

Inductive Step: $(n \geq 80)$

$$
\begin{aligned}
T(n) &\leq T\left(\frac{n}{5}+1\right) + T\left(\frac{7n}{10}+3\right) + an \\
&\leq c\left(\frac{n}{5}+1\right) + c\left(\frac{7n}{10}+3\right) + an \\
&\leq c\left(\frac{4n}{20}+\frac{14n}{20}+\frac{n}{20}\right) + an
\end{aligned}
$$

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \le cn$, for some $c > 0$ and all $n \ge 1$.

Base case: $(n < 80)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.
$\Rightarrow$ $T(n) \le c' \le cn$ for c sufficiently large.

Inductive Step: $(n \ge 80)$

$$\begin{aligned} T(n) &\le T\left(\frac{n}{5} + 1\right) + T\left(\frac{7n}{10} + 3\right) + an \\ &\le c\left(\frac{n}{5} + 1\right) + c\left(\frac{7n}{10} + 3\right) + an \\ &\le c\left(\frac{4n}{20} + \frac{14n}{20} + \frac{n}{20}\right) + an \\ &= \left(\frac{19c}{20} + a\right) n \end{aligned}$$

Worst-Case Selection: Analysis

Claim: $T(n) \in O(n)$, that is, $T(n) \leq cn$, for some $c > 0$ and all $n \geq 1$.

Base case: $(n < 80)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.

$\Rightarrow$ $T(n) \leq c' \leq cn$ for c sufficiently large.

Inductive Step: $(n \geq 80)$

$$
\begin{aligned}
T(n) &\leq T\left(\frac{n}{5} + 1\right) + T\left(\frac{7n}{10} + 3\right) + an \\
&\leq c\left(\frac{n}{5} + 1\right) + c\left(\frac{7n}{10} + 3\right) + an \\
&\leq c\left(\frac{4n}{20} + \frac{14n}{20} + \frac{n}{20}\right) + an \\
&= \left(\frac{19c}{20} + a\right) n \\
&\leq cn \quad \forall c \geq 20a
\end{aligned}
$$

Worst-Case Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
p = FindPivot(A, ℓ, r)
m = HoarePartition(A, ℓ, r, p)
return QuickSort(A, ℓ, m)
return QuickSort(A, m + 1, r)
```

Worst-Case Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
p = FindPivot(A, ℓ, r)
m = HoarePartition(A, ℓ, r, p)
return QuickSort(A, ℓ, m)
return QuickSort(A, m + 1, r)
```

$T(n) = \Theta(n) + T(n_1) + T(n_2)$, where $n_1 + n_2 = n$ and $n_1, n_2 \le \frac{7n}{10} + 3$

Worst-Case Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
p = FindPivot(A, ℓ, r)
m = HoarePartition(A, ℓ, r, p)
return QuickSort(A, ℓ, m)
return QuickSort(A, m + 1, r)
```

$$T(n) = \Theta(n) + T(n_1) + T(n_2), \text{ where } n_1 + n_2 = n \text{ and } n_1, n_2 \le \frac{7n}{10} + 3$$

Claim: $T(n) \in O(n \lg n)$, that is, $T(n) \le cn \lg n$, for some $c > 0$ and all $n \ge 2$.

Worst-Case Quick Sort

QuickSort(A, ℓ, r)

```
if r ≤ ℓ
    then return
p = FindPivot(A, ℓ, r)
m = HoarePartition(A, ℓ, r, p)
return QuickSort(A, ℓ, m)
return QuickSort(A, m + 1, r)
```

$$T(n) = \Theta(n) + T(n_l) + T(n_2), \text{ where } n_l + n_2 = n \text{ and } n_l, n_2 \le \frac{7n}{10} + 3$$

Claim: $T(n) \in O(n \lg n)$, that is, $T(n) \le cn \lg n$, for some $c > 0$ and all $n \ge 2$.

Base case: $(n < 30)$

We already observed that the running time is in $O(n^2)$ in the worst case. Since $n \in O(1)$, $n^2 \in O(1)$.

$\Rightarrow$ $T(n) \le c' \le cn$ for c sufficiently large.

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$T(n) \leq T(n_1) + T(n_2) + an$$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$\begin{aligned} T(n) &\leq T(n_1) + T(n_2) + an \\ &\leq cn_1 \lg n_1 + cn_2 \lg n_2 + an \end{aligned}$$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$
\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + an \\
&\leq cn_1 \lg n_1 + cn_2 \lg n_2 + an \\
&\leq cn_1 \lg \left(\frac{7n}{10} + 3\right) + cn_2 \lg \left(\frac{7n}{10} + 3\right) + an
\end{aligned}
$$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$
\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + an \\
&\leq cn_1 \lg n_1 + cn_2 \lg n_2 + an \\
&\leq cn_1 \lg\left(\frac{7n}{10} + 3\right) + cn_2 \lg\left(\frac{7n}{10} + 3\right) + an \\
&= cn \lg\left(\frac{7n}{10} + 3\right) + an
\end{aligned}
$$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + an \\
&\leq cn_1 \lg n_1 + cn_2 \lg n_2 + an \\
&\leq cn_1 \lg \left(\frac{7n}{10} + 3\right) + cn_2 \lg \left(\frac{7n}{10} + 3\right) + an \\
&= cn \lg \left(\frac{7n}{10} + 3\right) + an \\
&\leq cn \lg \left(\frac{7n}{10} + \frac{n}{10}\right) + an
\end{aligned}$$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$
\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + an \\
&\leq cn_1 \lg n_1 + cn_2 \lg n_2 + an \\
&\leq cn_1 \lg \left(\frac{7n}{10} + 3\right) + cn_2 \lg \left(\frac{7n}{10} + 3\right) + an \\
&= cn \lg \left(\frac{7n}{10} + 3\right) + an \\
&\leq cn \lg \left(\frac{7n}{10} + \frac{n}{10}\right) + an \\
&= cn \lg \frac{4n}{5} + an
\end{aligned}
$$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$
\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + an \\
&\leq cn_1 \lg n_1 + cn_2 \lg n_2 + an \\
&\leq cn_1 \lg \left(\frac{7n}{10} + 3\right) + cn_2 \lg \left(\frac{7n}{10} + 3\right) + an \\
&= cn \lg \left(\frac{7n}{10} + 3\right) + an \\
&\leq cn \lg \left(\frac{7n}{10} + \frac{n}{10}\right) + an \\
&= cn \lg \frac{4n}{5} + an \\
&= cn \left(\lg n - \lg \frac{5}{4}\right) + an
\end{aligned}
$$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$
\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + an \\
&\leq cn_1 \lg n_1 + cn_2 \lg n_2 + an \\
&\leq cn_1 \lg \left(\frac{7n}{10} + 3\right) + cn_2 \lg \left(\frac{7n}{10} + 3\right) + an \\
&= cn \lg \left(\frac{7n}{10} + 3\right) + an \\
&\leq cn \lg \left(\frac{7n}{10} + \frac{n}{10}\right) + an \\
&= cn \lg \frac{4n}{5} + an \\
&= cn \left(\lg n - \lg \frac{5}{4}\right) + an \\
&= cn \lg n + \left(a - c \lg \frac{5}{4}\right) n
\end{aligned}
$$

Worst-Case Quick Sort

Inductive Step: $(n \geq 30)$

$$
\begin{aligned}
T(n) &\leq T(n_1) + T(n_2) + an \\
&\leq cn_1 \lg n_1 + cn_2 \lg n_2 + an \\
&\leq cn_1 \lg\left(\frac{7n}{10} + 3\right) + cn_2 \lg\left(\frac{7n}{10} + 3\right) + an \\
&= cn \lg\left(\frac{7n}{10} + 3\right) + an \\
&\leq cn \lg\left(\frac{7n}{10} + \frac{n}{10}\right) + an \\
&= cn \lg \frac{4n}{5} + an \\
&= cn\left(\lg n - \lg \frac{5}{4}\right) + an \\
&= cn \lg n + \left(a - c \lg \frac{5}{4}\right) n \\
&\leq cn \lg n \quad \forall c \geq \frac{a}{\lg(5/4)} \approx 3.1a
\end{aligned}
$$

Matrix Multiplication

We want to compute $C = A \times B$, where $A = (a_{ij})$ and $B = (b_{ij})$ are $n \times n$ matrices and hence $C = (c_{ij})$ is too.

Matrix Multiplication

We want to compute $C = A \times B$, where $A = (a_{ij})$ and $B = (b_{ij})$ are $n \times n$ matrices and hence $C = (c_{ij})$ is too.

Definition:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk} \quad \forall 1 \leq i, k \leq n.$$

Matrix Multiplication

We want to compute $C = A \times B$, where $A = (a_{ij})$ and $B = (b_{ij})$ are $n \times n$ matrices and hence $C = (c_{ij})$ is too.

Definition:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk} \quad \forall\, 1 \le i, k \le n.$$

The naïve algorithm implementing the definition:

MatrixProduct(A, B)

```
C = an n × n array
for i = 1 to n
    do for k = 1 to n
            do C[i, k] = 0
                for j = 1 to n
                    do C[i, k] = C[i, k] + A[i, j] · B[j, k]
return C
```

Matrix Multiplication

We want to compute $C = A \times B$, where $A = (a_{ij})$ and $B = (b_{ij})$ are $n \times n$ matrices and hence $C = (c_{ij})$ is too.

Definition:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk} \quad \forall\, 1 \leq i, k \leq n.$$

The naïve algorithm implementing the definition:

MatrixProduct(A, B)

```
C = an n × n array
for i = 1 to n
    do for k = 1 to n
           do C[i, k] = 0
              for j = 1 to n
                  do C[i, k] = C[i, k] + A[i, j] · B[j, k]
return C
```

Cost: $\Theta(n^3)$

Matrix Multiplication: Divide and Conquer

For simplicity, assume $n = 2^t$ for some integer t.

Matrix Multiplication: Divide and Conquer

For simplicity, assume $n = 2^t$ for some integer t.

Base case: $t = 0$

$$c_{11} = a_{11} \cdot b_{11}$$

Matrix Multiplication: Divide and Conquer

For simplicity, assume $n = 2^t$ for some integer t.

Base case: $t = 0$

$$c_{11} = a_{11} \cdot b_{11}$$

Inductive step: $t > 0$

$$\begin{array}{|c|c|}\hline C_{11} & C_{12} \\ \hline C_{21} & C_{22} \\ \hline\end{array} = \begin{array}{|c|c|}\hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline\end{array} \times \begin{array}{|c|c|}\hline B_{11} & B_{12} \\ \hline B_{21} & B_{22} \\ \hline\end{array}$$

For $1 \leq i, k \leq 2$,

$$C_{ik} = \sum_{j=1}^{2} (A_{ij} \times B_{jk}).$$

Matrix Multiplication: Divide and Conquer

MatrixProductDNC$(A, B, C, i_l, i_u, j_l, j_u, k_l, k_u)$

if $i_l = i_u$
    then $C[i_l, k_l] = C[i_l, k_l] + A[i_l, j_l] \times B[j_l, k_l]$
    else $i_m = (i_l + i_u - 1)/2$
          $j_m = (j_l + j_u - 1)/2$
          $k_m = (k_l + k_u - 1)/2$
          MatrixProductDNC$(A, B, C, i_l, i_m, j_l, j_m, k_l, k_m)$
          MatrixProductDNC$(A, B, C, i_l, i_m, j_m + 1, j_u, k_l, k_m)$
          MatrixProductDNC$(A, B, C, i_l, i_m, j_l, j_m, k_m + 1, k_u)$
          MatrixProductDNC$(A, B, C, i_l, i_m, j_m + 1, j_u, k_m + 1, k_u)$
          MatrixProductDNC$(A, B, C, i_m + 1, i_u, j_l, j_m, k_l, k_m)$
          MatrixProductDNC$(A, B, C, i_m + 1, i_u, j_m + 1, j_u, k_l, k_m)$
          MatrixProductDNC$(A, B, C, i_m + 1, i_u, j_l, j_m, k_m + 1, k_u)$
          MatrixProductDNC$(A, B, C, i_m + 1, i_u, j_m + 1, j_u, k_m + 1, k_u)$

Matrix Multiplication: Divide and Conquer

MatrixProductDNC($A, B, C, i_l, i_u, j_l, j_u, k_l, k_u$)

if $i_l = i_u$
    then $C[i_l, k_l] = C[i_l, k_l] + A[i_l, j_l] \times B[j_l, k_l]$
    else $i_m = (i_l + i_u - 1)/2$
        $j_m = (j_l + j_u - 1)/2$
        $k_m = (k_l + k_u - 1)/2$
        MatrixProductDNC($A, B, C, i_l, i_m, j_l, j_m, k_l, k_m$)
        MatrixProductDNC($A, B, C, i_l, i_m, j_m + 1, j_u, k_l, k_m$)
        MatrixProductDNC($A, B, C, i_l, i_m, j_l, j_m, k_m + 1, k_u$)
        MatrixProductDNC($A, B, C, i_l, i_m, j_m + 1, j_u, k_m + 1, k_u$)
        MatrixProductDNC($A, B, C, i_m + 1, i_u, j_l, j_m, k_l, k_m$)
        MatrixProductDNC($A, B, C, i_m + 1, i_u, j_m + 1, j_u, k_l, k_m$)
        MatrixProductDNC($A, B, C, i_m + 1, i_u, j_l, j_m, k_m + 1, k_u$)
        MatrixProductDNC($A, B, C, i_m + 1, i_u, j_m + 1, j_u, k_m + 1, k_u$)

Cost: $T(n) = 8T(n/2) + \Theta(1)$

Matrix Multiplication: Divide and Conquer

MatrixProductDNC(A, B, C, i_l, i_u, j_l, j_u, k_l, k_u)

```
if i_l = i_u
    then C[i_l, k_l] = C[i_l, k_l] + A[i_l, j_l] × B[j_l, k_l]
    else i_m = (i_l + i_u − 1)/2
         j_m = (j_l + j_u − 1)/2
         k_m = (k_l + k_u − 1)/2
         MatrixProductDNC(A, B, C, i_l,     i_m, j_l,     j_m, k_l,     k_m)
         MatrixProductDNC(A, B, C, i_l,     i_m, j_m + 1, j_u, k_l,     k_m)
         MatrixProductDNC(A, B, C, i_l,     i_m, j_l,     j_m, k_m + 1, k_u)
         MatrixProductDNC(A, B, C, i_l,     i_m, j_m + 1, j_u, k_m + 1, k_u)
         MatrixProductDNC(A, B, C, i_m + 1, i_u, j_l,     j_m, k_l,     k_m)
         MatrixProductDNC(A, B, C, i_m + 1, i_u, j_m + 1, j_u, k_l,     k_m)
         MatrixProductDNC(A, B, C, i_m + 1, i_u, j_l,     j_m, k_m + 1, k_u)
         MatrixProductDNC(A, B, C, i_m + 1, i_u, j_m + 1, j_u, k_m + 1, k_u)
```

Cost: $T(n) = 8T(n/2) + \Theta(1) \in \Theta\left(n^3\right)$

Matrix Multiplication: Strassen's Algorithm

Goal:

$$T(n) = 7T(n/2) + \Theta\left(n^2\right) \in \Theta\left(n^{\lg 7}\right) \approx \Theta\left(n^{2.81}\right)$$

Matrix Multiplication: Strassen's Algorithm

Goal:

$$T(n) = 7T(n/2) + \Theta\left(n^2\right) \in \Theta\left(n^{\lg 7}\right) \approx \Theta\left(n^{2.81}\right)$$

Idea:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$$

Matrix Multiplication: Strassen's Algorithm

Goal:

$$T(n) = 7T(n/2) + \Theta\left(n^2\right) \in \Theta\left(n^{\lg 7}\right) \approx \Theta\left(n^{2.81}\right)$$

Idea:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \boxed{A_{11}\ A_{12}\ A_{21}\ A_{22}} \times \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix}$$

Matrix Multiplication: Strassen's Algorithm

Goal:

$$T(n) = 7T(n/2) + \Theta\left(n^2\right) \in \Theta\left(n^{\lg 7}\right) \approx \Theta\left(n^{2.81}\right)$$

Idea:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \boxed{A_{11}\ A_{12}\ A_{21}\ A_{22}} \times \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix}$$

$$M = A_{11} \times B_{11} + A_{11} \times B_{12} + A_{21} \times B_{11} + A_{21} \times B_{12}$$

Matrix Multiplication: Strassen's Algorithm

Goal:

$$T(n) = 7T(n/2) + \Theta\left(n^2\right) \in \Theta\left(n^{\lg 7}\right) \approx \Theta\left(n^{2.81}\right)$$

Idea:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} A_{11} & A_{12} & A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix}$$

$$M = A_{11} \times B_{11} + A_{11} \times B_{12} + A_{21} \times B_{11} + A_{21} \times B_{12}$$

$$= \begin{bmatrix} A_{11} & A_{12} & A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} 1 & & 1 & \\ & & & \\ 1 & & 1 & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix}$$

Matrix Multiplication: Strassen's Algorithm

Goal:

$$T(n) = 7T(n/2) + \Theta(n^2) \in \Theta\left(n^{\lg 7}\right) \approx \Theta(n^{2.81})$$

Idea:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} A_{11} & A_{12} & A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix}$$

$$\begin{aligned} M &= A_{11} \times B_{11} + A_{11} \times B_{12} + A_{21} \times B_{11} + A_{21} \times B_{12} \\ &= \begin{bmatrix} A_{11} & A_{12} & A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} 1 & & 1 & \\ & & & \\ 1 & & 1 & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix} \\ &= (A_{11} + A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

Matrix Multiplication: Strassen's Algorithm

Goal:

$$T(n) = 7T(n/2) + \Theta\left(n^2\right) \in \Theta\left(n^{\lg 7}\right) \approx \Theta\left(n^{2.81}\right)$$

Idea:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} A_{11} & A_{12} & A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix}$$

2 inconveniently placed ones
= 2 multiplications

$$\begin{aligned} M &= A_{11} \times B_{11} + A_{11} \times B_{12} + A_{21} \times B_{11} + A_{21} \times B_{12} \\ &= \begin{bmatrix} A_{11} & A_{12} & A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} 1 & & 1 & \\ & & & \\ 1 & & 1 & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix} \\ &= (A_{11} + A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

Matrix Multiplication: Strassen's Algorithm

Goal:

$$T(n) = 7T(n/2) + \Theta\left(n^2\right) \in \Theta\left(n^{\lg 7}\right) \approx \Theta\left(n^{2.81}\right)$$

Idea:

$$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \boxed{\begin{matrix} A_{11} & A_{12} & A_{21} & A_{22} \end{matrix}} \times \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix}$$

$$M = A_{11} \times B_{11} + A_{11} \times B_{12} + A_{21} \times B_{11} + A_{21} \times B_{12}$$

$$= \boxed{\begin{matrix} A_{11} & A_{12} & A_{21} & A_{22} \end{matrix}} \times \begin{bmatrix} 1 & & 1 & \\ & & & \\ 1 & & 1 & \\ & & & \end{bmatrix} \times \begin{bmatrix} B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{bmatrix}$$

$$= (A_{11} + A_{21}) \times (B_{11} + B_{12})$$

2 inconveniently placed ones
= 2 multiplications

4 conveniently placed ones
= 1 multiplication

Matrix Multiplication: Strassen's Alg.

$$M_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})$$
$$M_2 = A_{11} \times (B_{12} + B_{22})$$
$$M_3 = A_{22} \times (B_{11} + B_{21})$$
$$M_4 = (A_{11} - A_{12}) \times B_{22}$$
$$M_5 = (A_{22} - A_{21}) \times B_{11}$$
$$M_6 = (A_{11} + A_{21}) \times (B_{11} - B_{12})$$
$$M_7 = (A_{12} + A_{22}) \times (B_{21} - B_{22})$$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1			1
A_{12}				
A_{21}				
A_{22}	1			1

M_1

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}			1	1
A_{12}				
A_{21}				
A_{22}				

M_2

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}				
A_{22}	1	1		

M_3

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				1
A_{12}				-1
A_{21}				
A_{22}				

M_4

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}	-1			
A_{22}	1			

M_5

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1		-1	
A_{12}				
A_{21}	1		-1	
A_{22}				

M_6

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}		1		-1
A_{21}				
A_{22}		1		-1

M_7

Matrix Multiplication: Strassen's Alg.

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1			1
A_{12}				
A_{21}				
A_{22}	1			1

M_1

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}			1	1
A_{12}				
A_{21}				
A_{22}				

M_2

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}				
A_{22}	1	1		

M_3

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				1
A_{12}				-1
A_{21}				
A_{22}				

M_4

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}	-1			
A_{22}	1			

M_5

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1		-1	
A_{12}				
A_{21}	1		-1	
A_{22}				

M_6

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}		1		-1
A_{21}				
A_{22}		1		-1

M_7

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1			
A_{12}		1		
A_{21}				
A_{22}				

C_{11}

Matrix Multiplication: Strassen's Alg.

$$C_{11} = M_1 + M_7 - M_4 - M_3$$

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & 1 & & & 1 \\ A_{12} & & & & \\ A_{21} & & & & \\ A_{22} & 1 & & & 1 \\ \hline \end{array}$$

M_1

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & & & 1 & 1 \\ A_{12} & & & & \\ A_{21} & & & & \\ A_{22} & & & & \\ \hline \end{array}$$

M_2

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & & & & \\ A_{12} & & & & \\ A_{21} & & & & \\ A_{22} & 1 & 1 & & \\ \hline \end{array}$$

M_3

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & & & & 1 \\ A_{12} & & & & -1 \\ A_{21} & & & & \\ A_{22} & & & & \\ \hline \end{array}$$

M_4

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & & & & \\ A_{12} & & & & \\ A_{21} & -1 & & & \\ A_{22} & 1 & & & \\ \hline \end{array}$$

M_5

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & 1 & & -1 & \\ A_{12} & & & & \\ A_{21} & 1 & & -1 & \\ A_{22} & & & & \\ \hline \end{array}$$

M_6

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & & & & \\ A_{12} & & 1 & & -1 \\ A_{21} & & & & \\ A_{22} & & 1 & & -1 \\ \hline \end{array}$$

M_7

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & 1 & & & \\ A_{12} & & 1 & & \\ A_{21} & & & & \\ A_{22} & & & & \\ \hline \end{array}$$

C_{11}

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & 1 & & & 1 \\ A_{12} & & & & \\ A_{21} & & & & \\ A_{22} & 1 & & & 1 \\ \hline \end{array}$$

M_1

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & & & & \\ A_{12} & & 1 & & -1 \\ A_{21} & & & & \\ A_{22} & & 1 & & -1 \\ \hline \end{array}$$

M_7

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & & & & -1 \\ A_{12} & & & & 1 \\ A_{21} & & & & \\ A_{22} & & & & \\ \hline \end{array}$$

$-M_4$

$$\begin{array}{c|cccc|} & B_{11} & B_{21} & B_{12} & B_{22} \\ \hline A_{11} & & & & \\ A_{12} & & & & \\ A_{21} & & & & \\ A_{22} & -1 & -1 & & \\ \hline \end{array}$$

$-M_3$

Matrix Multiplication: Strassen's Alg.

$$C_{11} = M_1 + M_7 - M_4 - M_3$$

$$C_{12} = M_2 - M_4$$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}			1	
A_{12}				1
A_{21}				
A_{22}				

C_{12}

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}			1	1
A_{12}				
A_{21}				
A_{22}				

M_2

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				-1
A_{12}				1
A_{21}				
A_{22}				

$-M_4$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1			1
A_{12}				
A_{21}				
A_{22}	1			1

M_1

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}			1	1
A_{12}				
A_{21}				
A_{22}				

M_2

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}				
A_{22}	1	1		

M_3

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				1
A_{12}				-1
A_{21}				
A_{22}				

M_4

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}	-1			
A_{22}	1			

M_5

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1		-1	
A_{12}				
A_{21}	1		-1	
A_{22}				

M_6

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}		1		-1
A_{21}				
A_{22}		1		-1

M_7

Matrix Multiplication: Strassen's Alg.

$$C_{11} = M_1 + M_7 - M_4 - M_3$$

$$C_{12} = M_2 - M_4$$

$$C_{21} = M_3 - M_5$$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}	1			
A_{22}		1		

C_{12}

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}				
A_{22}	1	1		

M_3

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}	1			
A_{22}	-1			

$-M_5$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1			1
A_{12}				
A_{21}				
A_{22}	1			1

M_1

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}			1	1
A_{12}				
A_{21}				
A_{22}				

M_2

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}				
A_{22}	1	1		

M_3

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				1
A_{12}				-1
A_{21}				
A_{22}				

M_4

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}	-1			
A_{22}	1			

M_5

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1		-1	
A_{12}				
A_{21}	1		-1	
A_{22}				

M_6

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}		1		-1
A_{21}				
A_{22}		1		-1

M_7

Matrix Multiplication: Strassen's Alg.

$$C_{11} = M_1 + M_7 - M_4 - M_3$$

$$C_{12} = M_2 - M_4$$

$$C_{21} = M_3 - M_5$$

$$C_{22} = M_1 - M_6 - M_2 - M_5$$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1			1
A_{12}				
A_{21}				
A_{22}	1			1

M_1

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}			1	1
A_{12}				
A_{21}				
A_{22}				

M_2

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}				
A_{22}	1	1		

M_3

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				1
A_{12}				-1
A_{21}				
A_{22}				

M_4

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}	-1			
A_{22}	1			

M_5

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}			1	
A_{22}				1

C_{22}

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1			1
A_{12}				
A_{21}				
A_{22}	1			1

M_1

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	-1		1	
A_{12}				
A_{21}	-1		1	
A_{22}				

$-M_6$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}			-1	-1
A_{12}				
A_{21}				
A_{22}				

$-M_2$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}				
A_{21}	1			
A_{22}	-1			

$-M_5$

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}	1		-1	
A_{12}				
A_{21}	1		-1	
A_{22}				

M_6

	B_{11}	B_{21}	B_{12}	B_{22}
A_{11}				
A_{12}		1		-1
A_{21}				
A_{22}		1		-1

M_7

Matrix Multiplication: Strassen's Algorithm

Strassen(A, B)

let $n \times n$ be the dimension of A and B
if $n = 1$
    then return $A[1,1] \cdot B[1,1]$
    else partition A into submatrices $A_{11}, A_{12}, A_{21}, A_{22}$
        partition B into submatrices $B_{11}, B_{12}, B_{21}, B_{22}$
        $M_1 = \text{Strassen}(A_{11} + A_{22}, B_{11} + B_{22})$
        $M_2 = \text{Strassen}(A_{11}, B_{12} + B_{22})$
        $M_3 = \text{Strassen}(A_{22}, B_{11} + B_{21})$
        $M_4 = \text{Strassen}(A_{11} - A_{12}, B_{22})$
        $M_5 = \text{Strassen}(A_{22} - A_{21}, B_{11})$
        $M_6 = \text{Strassen}(A_{11} + A_{21}, B_{11} - B_{12})$
        $M_7 = \text{Strassen}(A_{12} + A_{22}, B_{21} - B_{22})$
        $C_{11} = M_1 + M_7 - M_4 - M_3$
        $C_{12} = M_2 - M_4$
        $C_{21} = M_3 - M_5$
        $C_{22} = M_1 - M_6 - M_2 - M_5$
        assemble C from $C_{11}, C_{12}, C_{21}, C_{22}$
        return C

Matrix Multiplication: Strassen's Algorithm

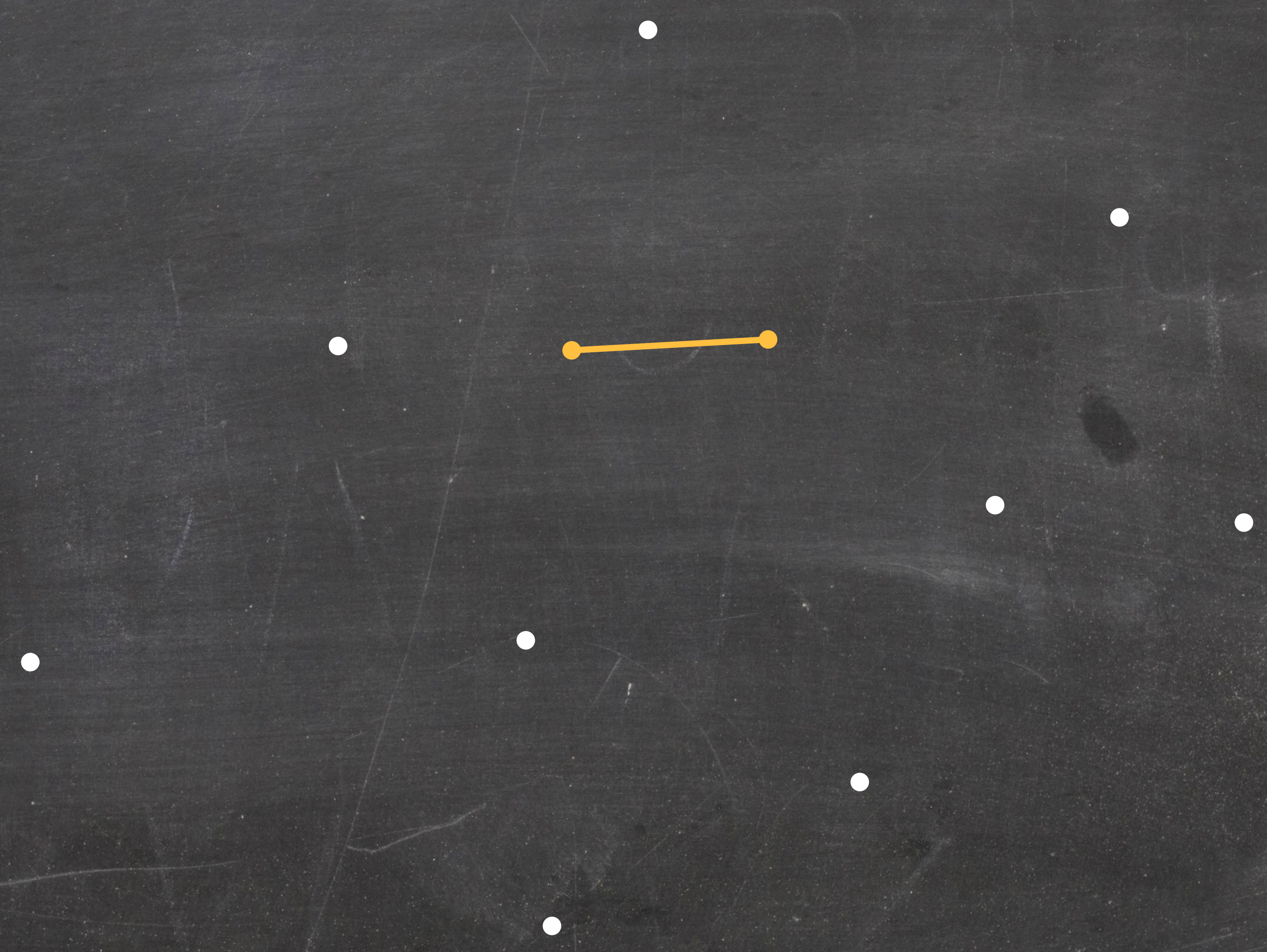
Strassen(A, B)

let $n \times n$ be the dimension of A and B
if $n = 1$
  then return $A[1,1] \cdot B[1,1]$
  else partition A into submatrices $A_{11}, A_{12}, A_{21}, A_{22}$
    partition B into submatrices $B_{11}, B_{12}, B_{21}, B_{22}$
    $M_1 = \text{Strassen}(A_{11} + A_{22}, B_{11} + B_{22})$
    $M_2 = \text{Strassen}(A_{11}, B_{12} + B_{22})$
    $M_3 = \text{Strassen}(A_{22}, B_{11} + B_{21})$
    $M_4 = \text{Strassen}(A_{11} - A_{12}, B_{22})$
    $M_5 = \text{Strassen}(A_{22} - A_{21}, B_{11})$
    $M_6 = \text{Strassen}(A_{11} + A_{21}, B_{11} - B_{12})$
    $M_7 = \text{Strassen}(A_{12} + A_{22}, B_{21} - B_{22})$
    $C_{11} = M_1 + M_7 - M_4 - M_3$
    $C_{12} = M_2 - M_4$
    $C_{21} = M_3 - M_5$
    $C_{22} = M_1 - M_6 - M_2 - M_5$
    assemble C from $C_{11}, C_{12}, C_{21}, C_{22}$
    return C

Cost: $T(n) = 7T(n/2) + \Theta(n^2) \in \Theta\left(n^{\lg 7}\right)$

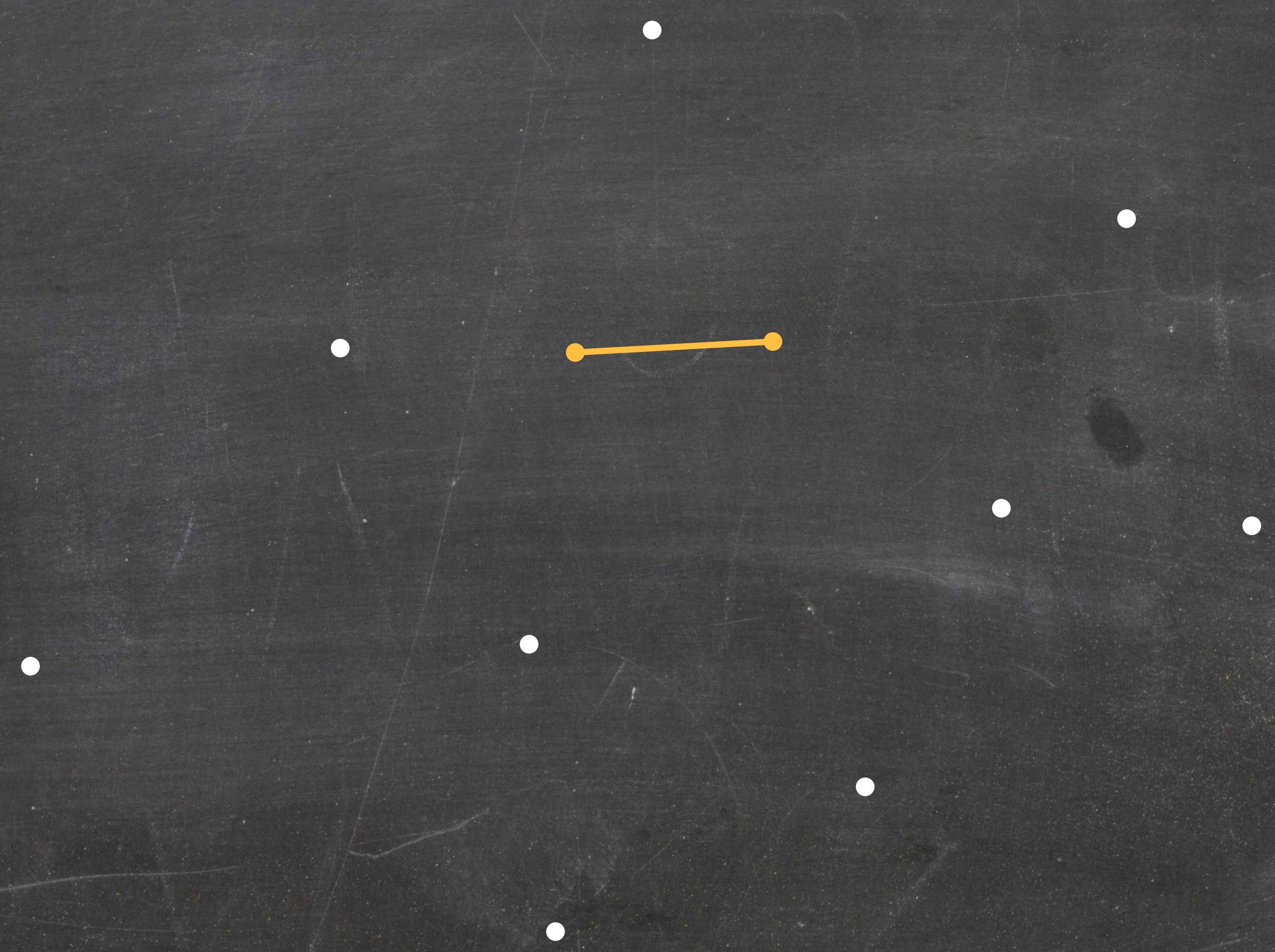
Closest Pair

Given a point set P in the plane, the closest pair is the pair of points $p, q \in P$ that minimizes $\|p - q\|_2$ (the Euclidean distance from p to q).



Closest Pair

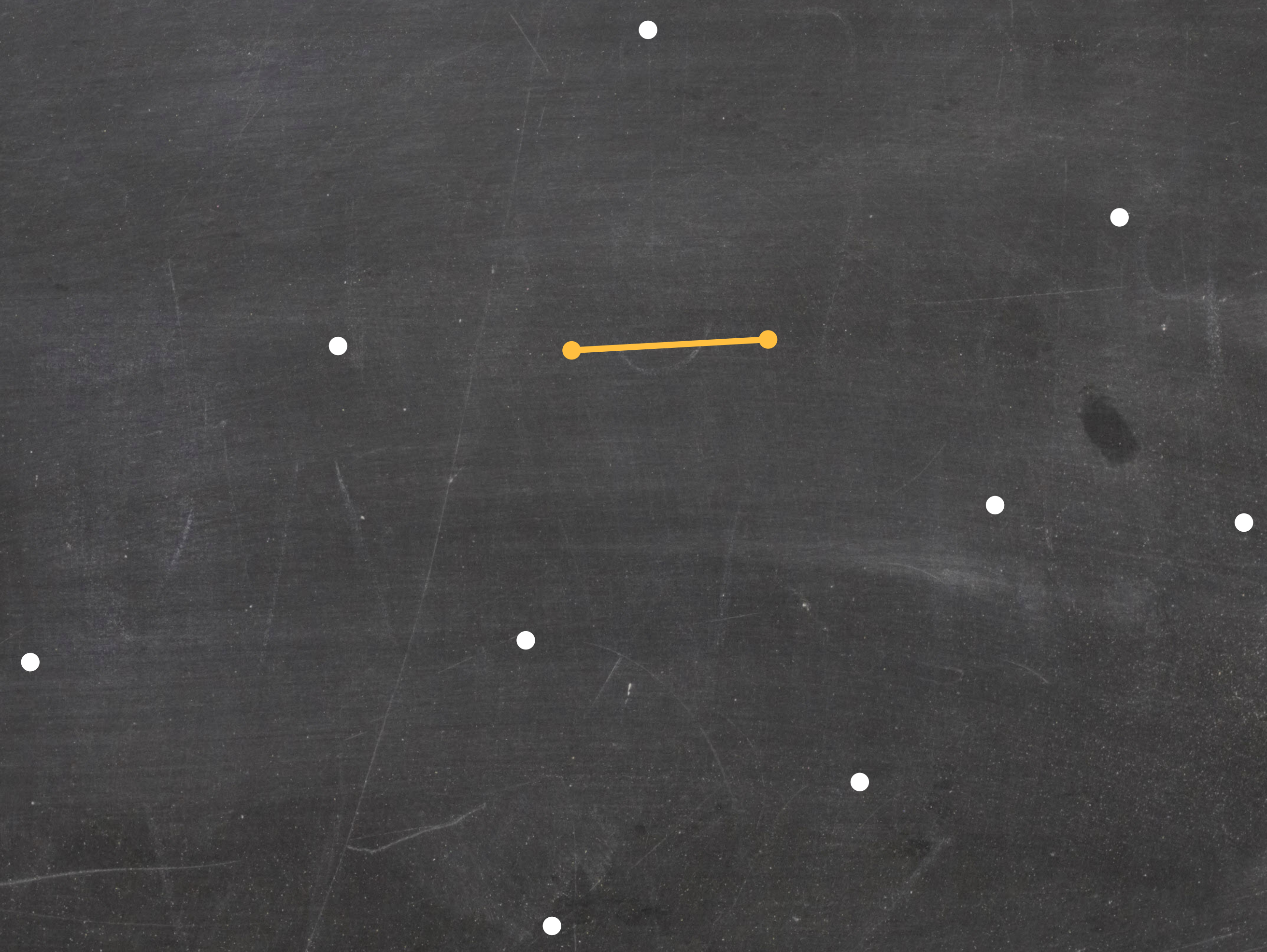
Given a point set P in the plane, the closest pair is the pair of points $p, q \in P$ that minimizes $\|p - q\|_2$ (the Euclidean distance from p to q).



Can be computed in $O(n^2)$ time. How?

Closest Pair

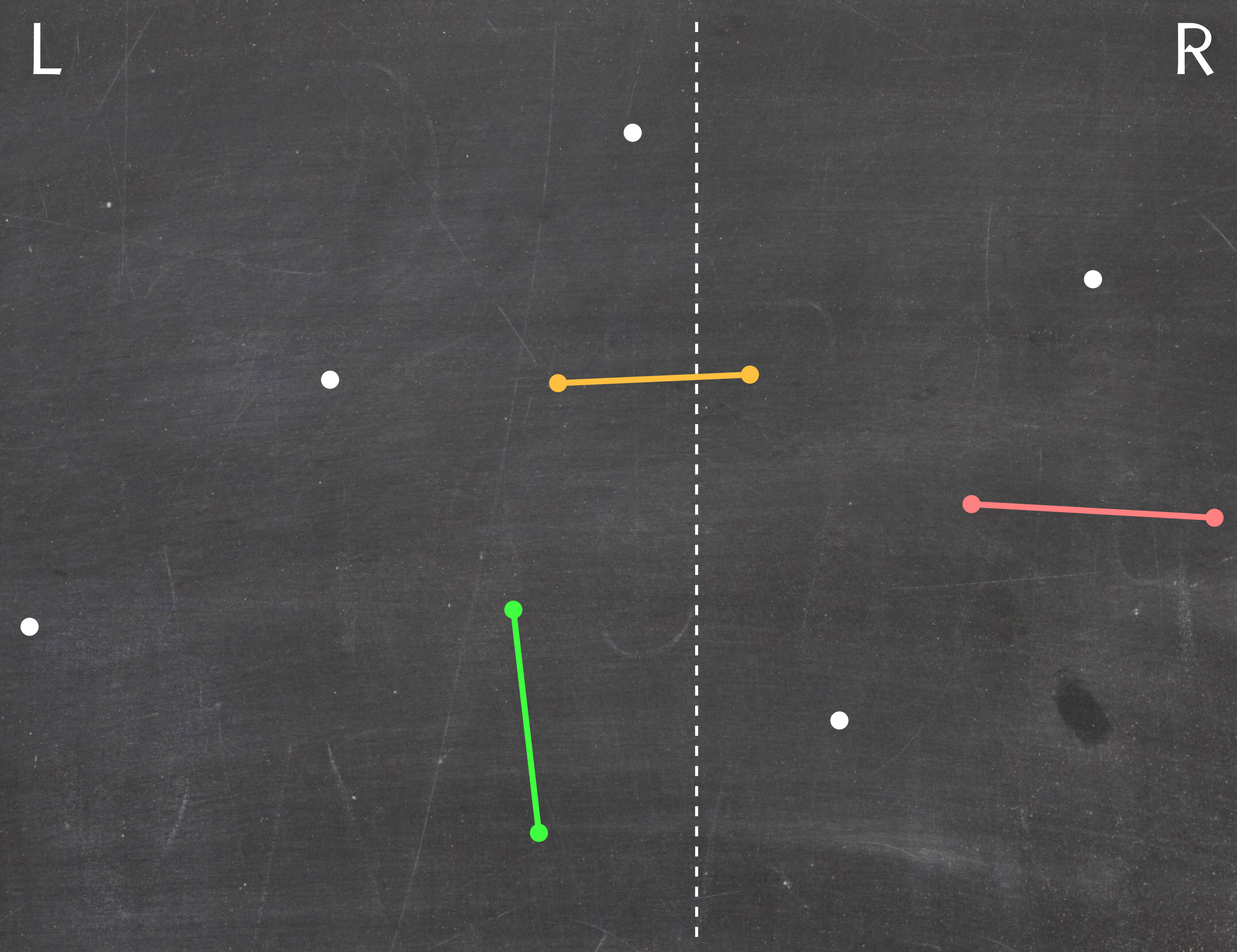
Given a point set P in the plane, the closest pair is the pair of points $p, q \in P$ that minimizes $\|p - q\|_2$ (the Euclidean distance from p to q).



Can be computed in $O(n^2)$ time. How?

Can we do better?

Closest Pair: Divide and Conquer



If we divide the point set into the leftmost $\lceil n/2 \rceil$ points (L) and the rightmost $\lfloor n/2 \rfloor$ points (R), then the closest pair has

- both points in L,
- both points in R or
- one point in L and one point in R.

Closest Pair: Divide and Conquer

ClosestPair(P)

if $|P| \leq 1$
  then return Nothing
Split P into two sets L and R containing the leftmost $\lceil n/2 \rceil$ and the rightmost $\lfloor n/2 \rfloor$ points, respectively.
(p_ℓ, q_ℓ) = ClosestPair(L)
(p_r, q_r) = ClosestPair(R)
(p_m, q_m) = ClosestPairLR(L, R)
return the pair (p_i, q_i), $i \in \{\ell, r, m\}$, that minimizes $\|p_i - q_i\|_2$

Closest Pair: Divide and Conquer

ClosestPair(P)

if $|P| \leq 1$
  then return Nothing
Split P into two sets L and R containing the leftmost $\lceil n/2 \rceil$ and the rightmost $\lfloor n/2 \rfloor$ points, respectively.
(p_ℓ, q_ℓ) = ClosestPair(L)
(p_r, q_r) = ClosestPair(R)
(p_m, q_m) = ClosestPairLR(L, R)
return the pair (p_i, q_i), $i \in \{\ell, r, m\}$, that minimizes $\|p_i - q_i\|_2$

Naïve implementation of ClosestPairLR: $\Theta(n^2)$ time

Closest Pair: Divide and Conquer

ClosestPair(P)

```
if |P| ≤ 1
    then return Nothing
Split P into two sets L and R containing the leftmost ⌈n/2⌉ and the
rightmost ⌊n/2⌋ points, respectively.
(p_ℓ, q_ℓ) = ClosestPair(L)
(p_r, q_r) = ClosestPair(R)
(p_m, q_m) = ClosestPairLR(L, R)
return the pair (p_i, q_i), i ∈ {ℓ, r, m}, that minimizes ||p_i − q_i||_2
```

Naïve implementation of ClosestPairLR: $\Theta(n^2)$ time

Running time: $T(n) = 2T(n/2) + \Theta(n^2)$

Closest Pair: Divide and Conquer

ClosestPair(P)

if $|P| \le 1$
    then return Nothing
Split P into two sets L and R containing the leftmost $\lceil n/2 \rceil$ and the rightmost $\lfloor n/2 \rfloor$ points, respectively.
$(p_\ell, q_\ell) = \text{ClosestPair}(L)$
$(p_r, q_r) = \text{ClosestPair}(R)$
$(p_m, q_m) = \text{ClosestPairLR}(L, R)$
return the pair (p_i, q_i), $i \in \{\ell, r, m\}$, that minimizes $\|p_i - q_i\|_2$

Naïve implementation of ClosestPairLR: $\Theta(n^2)$ time

Running time: $T(n) = 2T(n/2) + \Theta(n^2) \in \Theta(n^2)$

Closest Pair: Divide and Conquer

ClosestPair(P)

```
if |P| ≤ 1
    then return Nothing
Split P into two sets L and R containing the leftmost ⌈n/2⌉ and the
rightmost ⌊n/2⌋ points, respectively.
(p_ℓ, q_ℓ) = ClosestPair(L)
(p_r, q_r) = ClosestPair(R)
(p_m, q_m) = ClosestPairLR(L, R)
return the pair (p_i, q_i), i ∈ {ℓ, r, m}, that minimizes ||p_i − q_i||_2
```

Better implementation of ClosestPairLR: $\Theta(n)$ time

Running time: $T(n) = 2T(n/2) + \Theta(n) \in \Theta(n \lg n)$

Closest Pair: Preprocessing

```
ClosestPair(P)
    Make two copies X and Y of P
    Sort the points in X by their x-coordinates
    Sort the points in Y by their y-coordinates
    return ClosestPairRec(X, Y)
```

Closest Pair: Preprocessing

ClosestPair(P)

```
Make two copies X and Y of P
Sort the points in X by their x-coordinates
Sort the points in Y by their y-coordinates
return ClosestPairRec(X, Y)
```

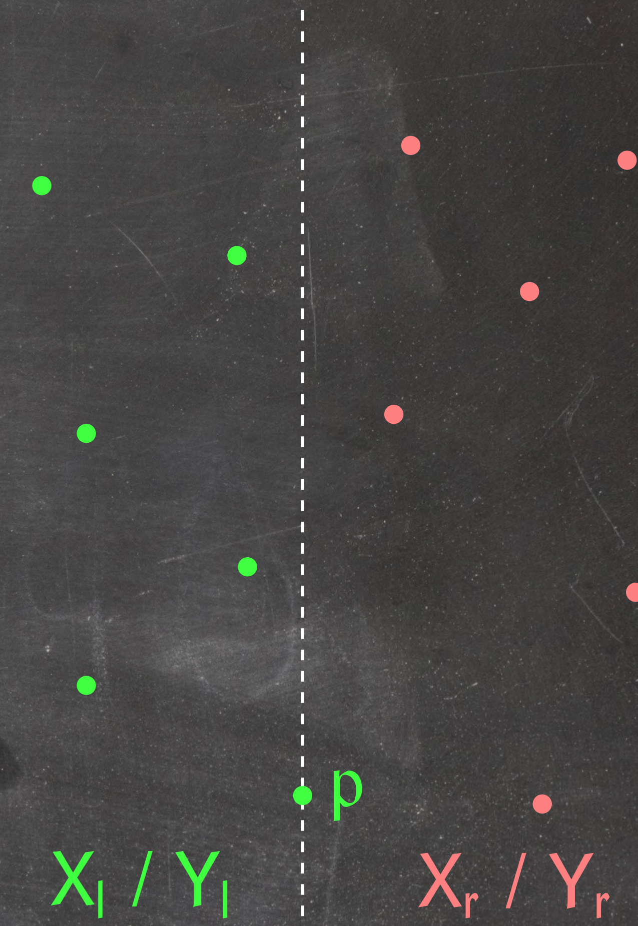
We prove that ClosestPairRec takes $O(n \lg n)$ time.

$\Rightarrow$ The whole algorithm takes $O(n \lg n)$ time.

Closest Pair: Divide and Conquer

ClosestPairRec(X, Y)

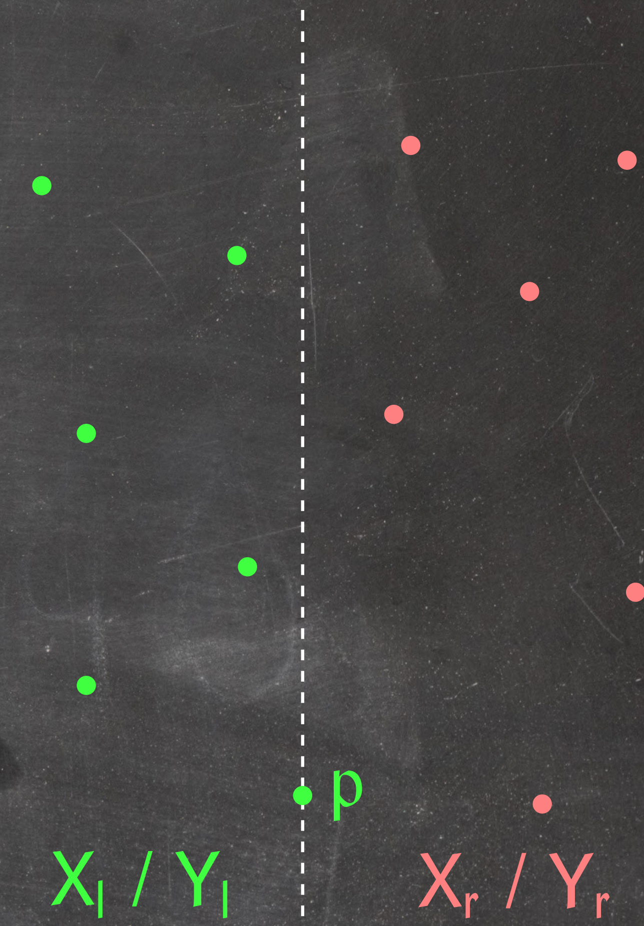
if $|Y| \le 1$
    then return (Nothing, ∞)
p = the middle element in X
X_ℓ = the part of X up to and including p
X_r = the part of X after x
$Y_\ell = \langle q \in Y \mid q.x \le p.x \rangle$
$Y_r = \langle q \in Y \mid q.x > p.x \rangle$
$(pair_\ell, \delta_\ell)$ = ClosestPairRec(X_ℓ, Y_ℓ)
$(pair_r, \delta_r)$ = ClosestPairRec(X_r, Y_r)
$(pair_m, \delta_m)$ = ClosestPairLR$(Y_\ell, Y_r, p.x, \min(\delta_\ell, \delta_r))$
return the pair $(pair_i, \delta_i)$, $i \in \{\ell, r, m\}$, that minimizes δ_i



Closest Pair: Divide and Conquer

ClosestPairRec(X, Y)

if $|Y| \le 1$
    then return (Nothing, ∞)
p = the middle element in X
X_ℓ = the part of X up to and including p
X_r = the part of X after x
$Y_\ell = \langle q \in Y \mid q.x \le p.x \rangle$
$Y_r = \langle q \in Y \mid q.x > p.x \rangle$
$(\text{pair}_\ell, \delta_\ell) = \text{ClosestPairRec}(X_\ell, Y_\ell)$
$(\text{pair}_r, \delta_r) = \text{ClosestPairRec}(X_r, Y_r)$
$(\text{pair}_m, \delta_m) = \text{ClosestPairLR}(Y_\ell, Y_r, p.x, \min(\delta_\ell, \delta_r))$
return the pair $(\text{pair}_i, \delta_i)$, $i \in \{\ell, r, m\}$, that minimizes δ_i



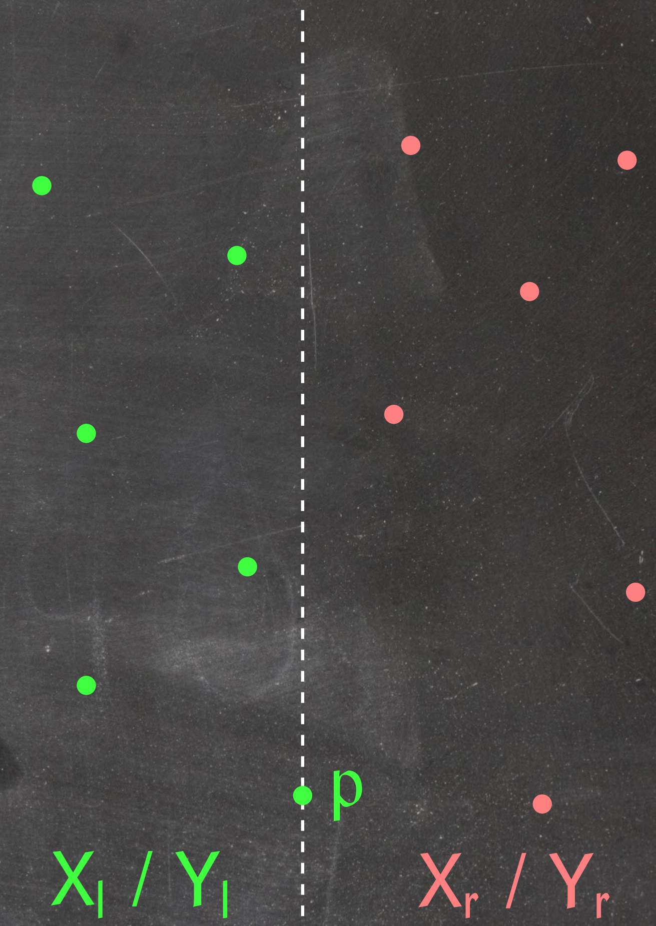
We already have a pair with distance $\delta = \min(\delta_\ell, \delta_r)$.

$\Rightarrow$ only need to look for pairs with distances $< \delta$.

Closest Pair: Divide and Conquer

ClosestPairRec(X, Y)

if $|Y| \le 1$
    then return (Nothing, ∞)
p = the middle element in X
X_ℓ = the part of X up to and including p
X_r = the part of X after x
$Y_\ell = \langle q \in Y \mid q.x \le p.x \rangle$
$Y_r = \langle q \in Y \mid q.x > p.x \rangle$
$(\text{pair}_\ell, \delta_\ell) = \text{ClosestPairRec}(X_\ell, Y_\ell)$
$(\text{pair}_r, \delta_r) = \text{ClosestPairRec}(X_r, Y_r)$
$(\text{pair}_m, \delta_m) = \text{ClosestPairLR}(Y_\ell, Y_r, p.x, \min(\delta_\ell, \delta_r))$
return the pair $(\text{pair}_i, \delta_i)$, $i \in \{\ell, r, m\}$, that minimizes δ_i

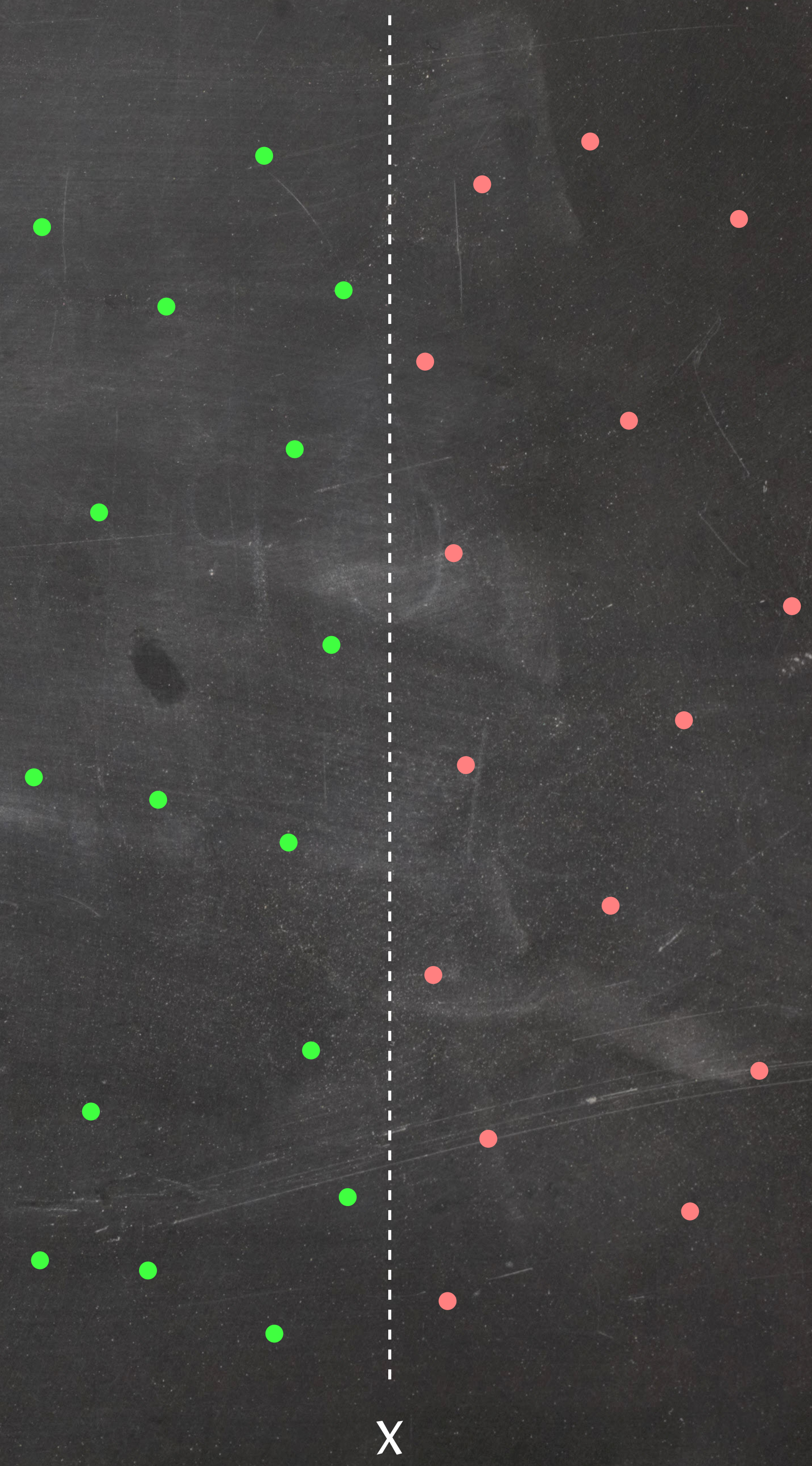


We prove that $\text{ClosestPairLR}(Y_\ell, Y_r, x, \delta)$ takes $O(n)$ time.

$\Rightarrow$ $T(n) = 2T(n/2) + O(n) \in O(n \lg n)$

Closest Pair: One Left, One Right

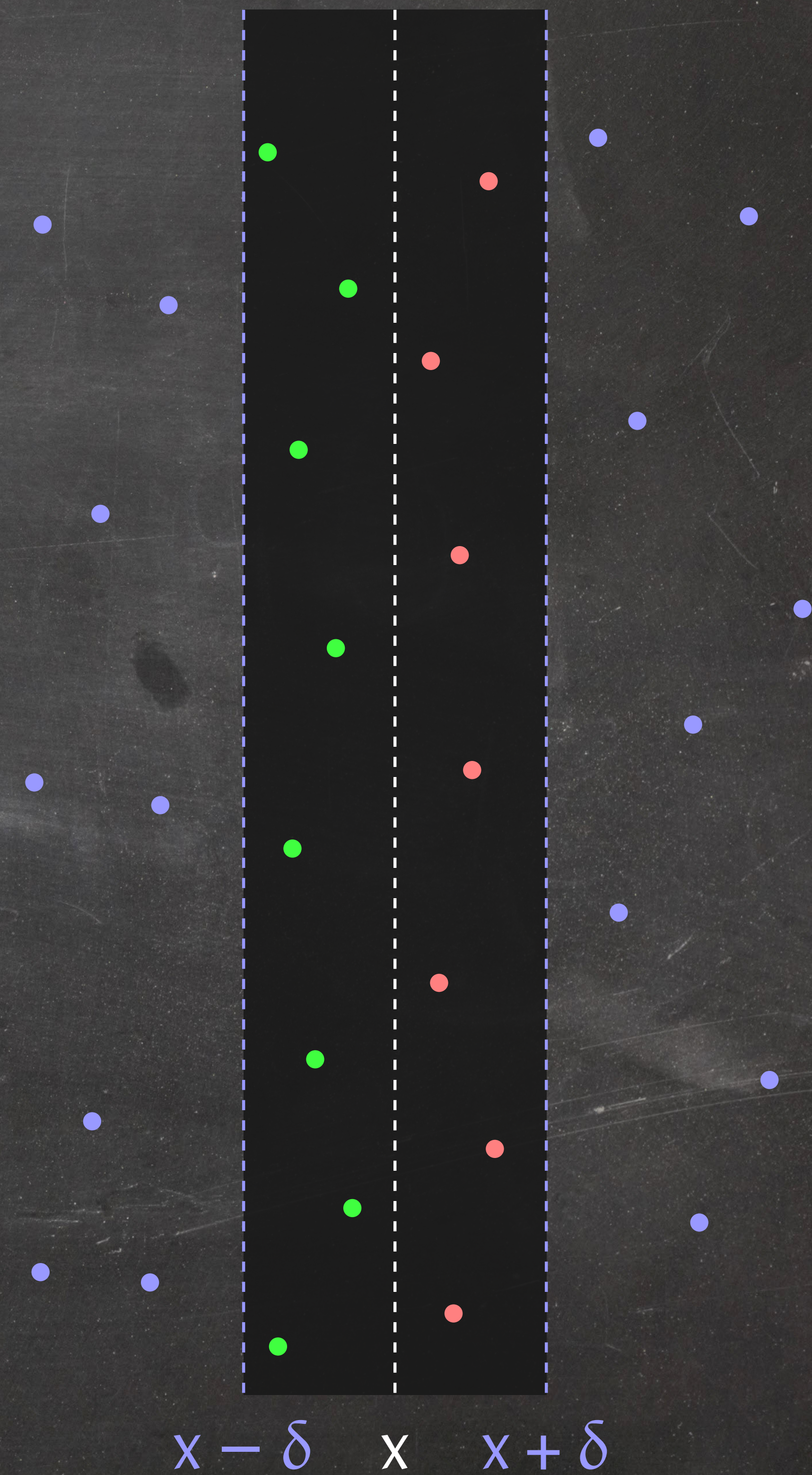
```
ClosestPairLR(Yℓ, Yr, x, δ)

Zℓ = ⟨p ∈ Yℓ | x − p.x ≤ δ⟩
Zr = ⟨p ∈ Yr | p.x − x ≤ δ⟩
pair = Nothing
δ′ = ∞
j = 1
for i = 1 to |Zℓ|
    do while j < |Zr| and Zr[j].y < Zℓ[i].y − δ
           do j = j + 1
       k = j
       while k ≤ |Zr| and Zr[k].y ≤ Zℓ[i].y + δ
           do if ||Zℓ[i] − Zr[k]|| < δ′
                then δ′ = ||Zℓ[i] − Zr[k]||
                     pair = (Zℓ[i], Zr[k])
              k = k + 1
return (pair, δ′)
```



Closest Pair: One Left, One Right

ClosestPairLR(Y_ℓ, Y_r, x, δ)

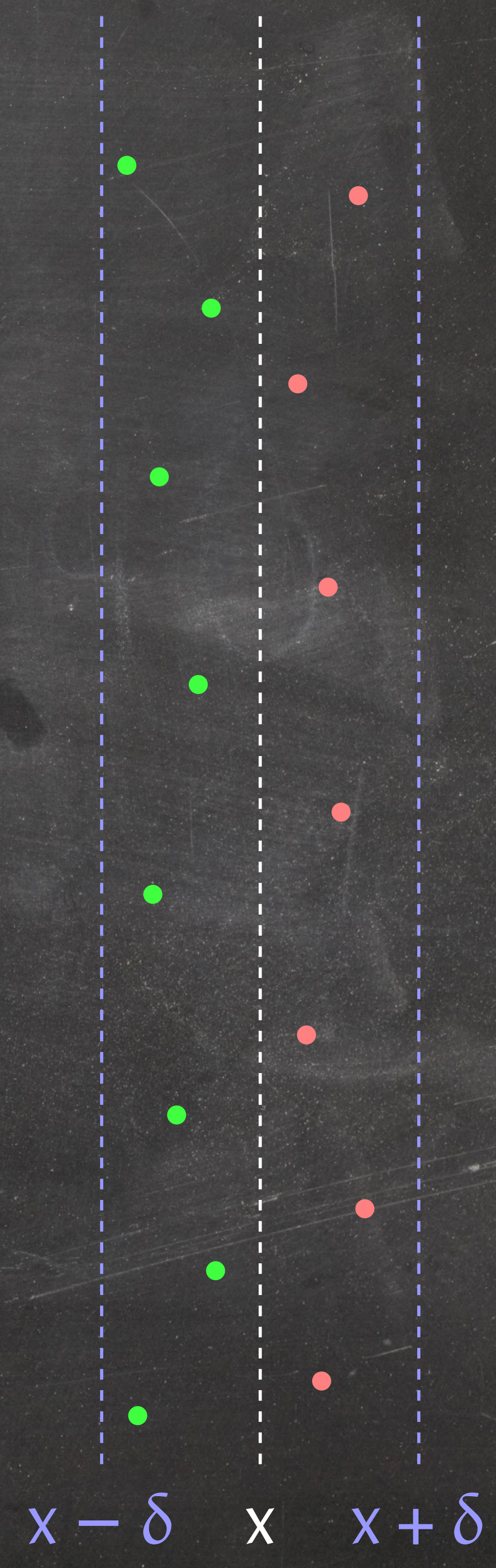
```
Zℓ = ⟨p ∈ Yℓ | x − p.x ≤ δ⟩
Zr = ⟨p ∈ Yr | p.x − x ≤ δ⟩
pair = Nothing
δ′ = ∞
j = 1
for i = 1 to |Zℓ|
    do while j < |Zr| and Zr[j].y < Zℓ[i].y − δ
           do j = j + 1
       k = j
       while k ≤ |Zr| and Zr[k].y ≤ Zℓ[i].y + δ
           do if ||Zℓ[i] − Zr[k]|| < δ′
                  then δ′ = ||Zℓ[i] − Zr[k]||
                       pair = (Zℓ[i], Zr[k])
              k = k + 1
return (pair, δ′)
```



Closest Pair: One Left, One Right

ClosestPairLR(Y_ℓ, Y_r, x, δ)

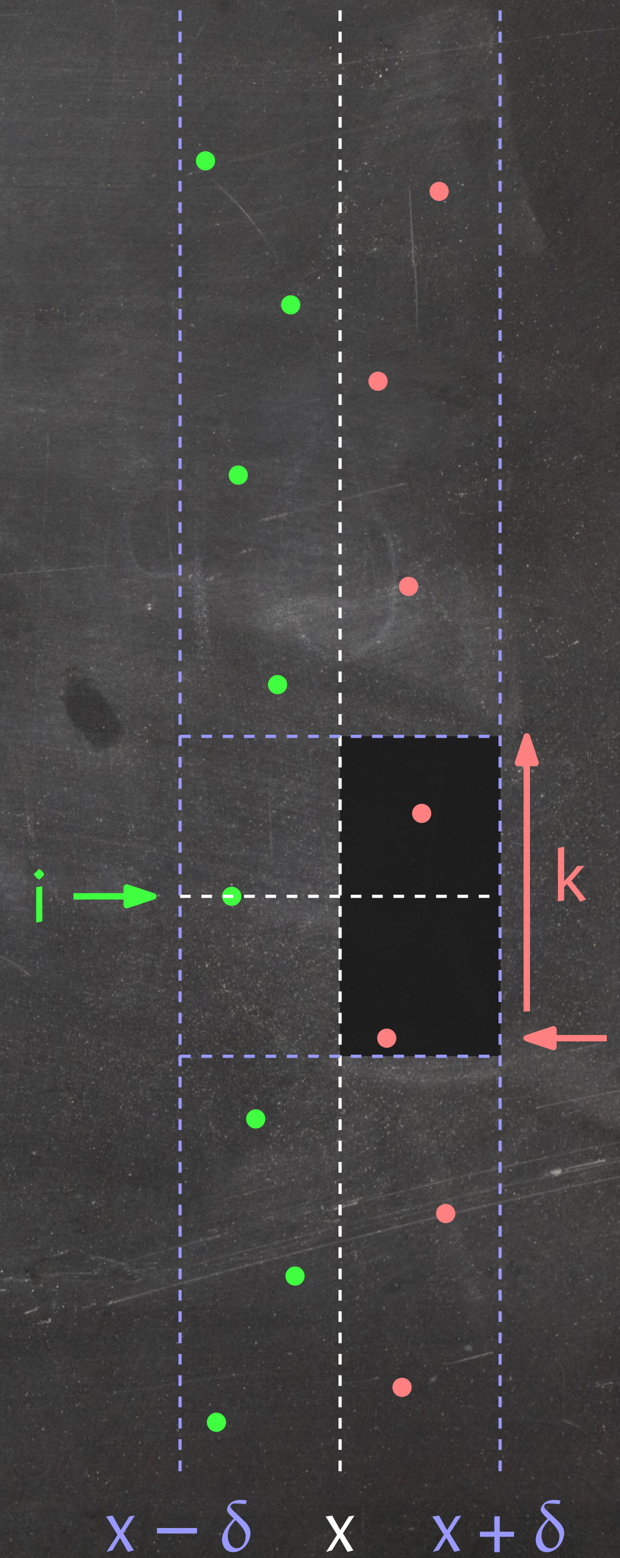
```
Zℓ = ⟨p ∈ Yℓ | x − p.x ≤ δ⟩
Zr = ⟨p ∈ Yr | p.x − x ≤ δ⟩
pair = Nothing
δ′ = ∞
j = 1
for i = 1 to |Zℓ|
    do while j < |Zr| and Zr[j].y < Zℓ[i].y − δ
            do j = j + 1
        k = j
        while k ≤ |Zr| and Zr[k].y ≤ Zℓ[i].y + δ
            do if ‖Zℓ[i] − Zr[k]‖ < δ′
                then δ′ = ‖Zℓ[i] − Zr[k]‖
                    pair = (Zℓ[i], Zr[k])
            k = k + 1
return (pair, δ′)
```



Closest Pair: One Left, One Right

ClosestPairLR(Y_ℓ, Y_r, x, δ)

```
Zℓ = ⟨p ∈ Yℓ | x − p.x ≤ δ⟩
Zr = ⟨p ∈ Yr | p.x − x ≤ δ⟩
pair = Nothing
δ′ = ∞
j = 1
for i = 1 to |Zℓ|
    do while j < |Zr| and Zr[j].y < Zℓ[i].y − δ
            do j = j + 1
        k = j
        while k ≤ |Zr| and Zr[k].y ≤ Zℓ[i].y + δ
            do if ||Zℓ[i] − Zr[k]|| < δ′
                    then δ′ = ||Zℓ[i] − Zr[k]||
                         pair = (Zℓ[i], Zr[k])
                k = k + 1
return (pair, δ′)
```



Closest Pair: One Left, One Right

Lemma: For every point $p \in Z_l$, there exist at most 8 points $q \in Z_r$ such that $|q.y - p.y| \leq \delta$.

Closest Pair: One Left, One Right

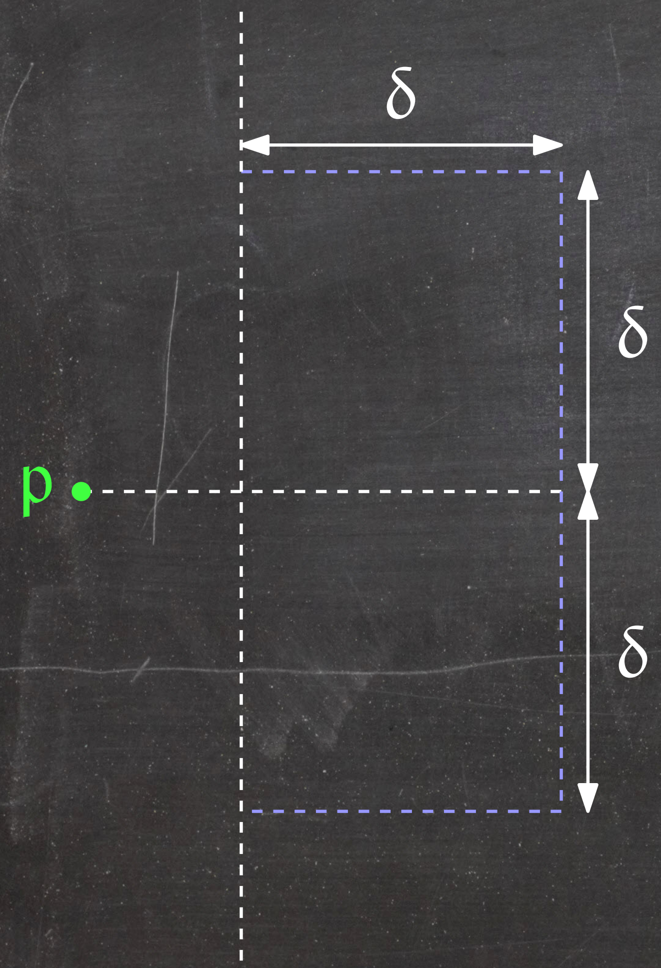
Lemma: For every point $p \in Z_l$, there exist at most 8 points $q \in Z_r$ such that $|q.y - p.y| \leq \delta$.

Corollary: The running time of ClosestPairLR is in $O(n)$.

Closest Pair: One Left, One Right

Lemma: For every point $p \in Z_l$, there exist at most 8 points $q \in Z_r$ such that $|q.y - p.y| \leq \delta$.

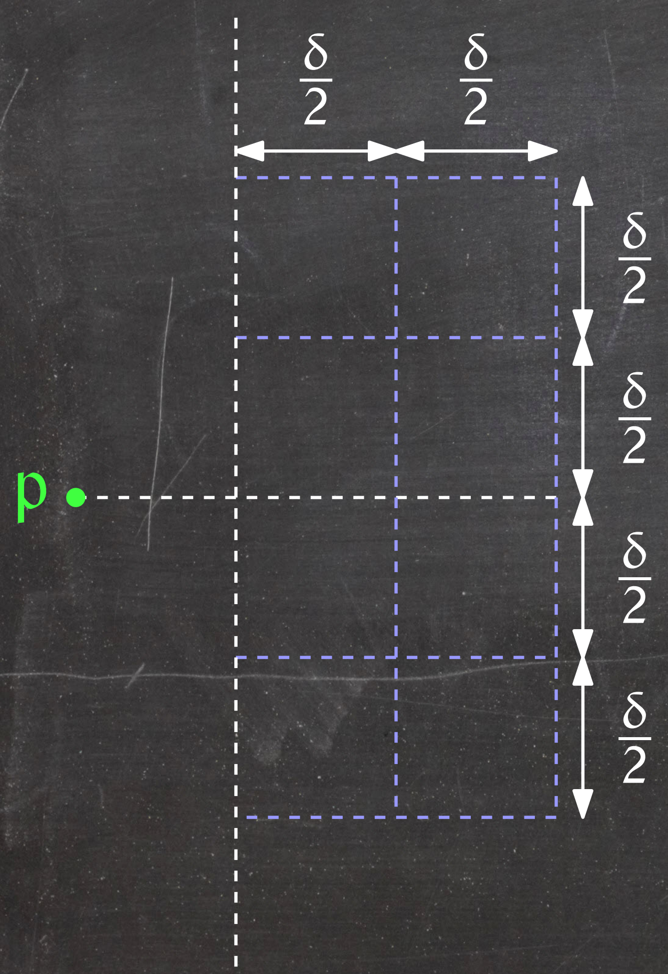
Corollary: The running time of ClosestPairLR is in $O(n)$.



Closest Pair: One Left, One Right

Lemma: For every point $p \in Z_l$, there exist at most 8 points $q \in Z_r$ such that $|q.y - p.y| \leq \delta$.

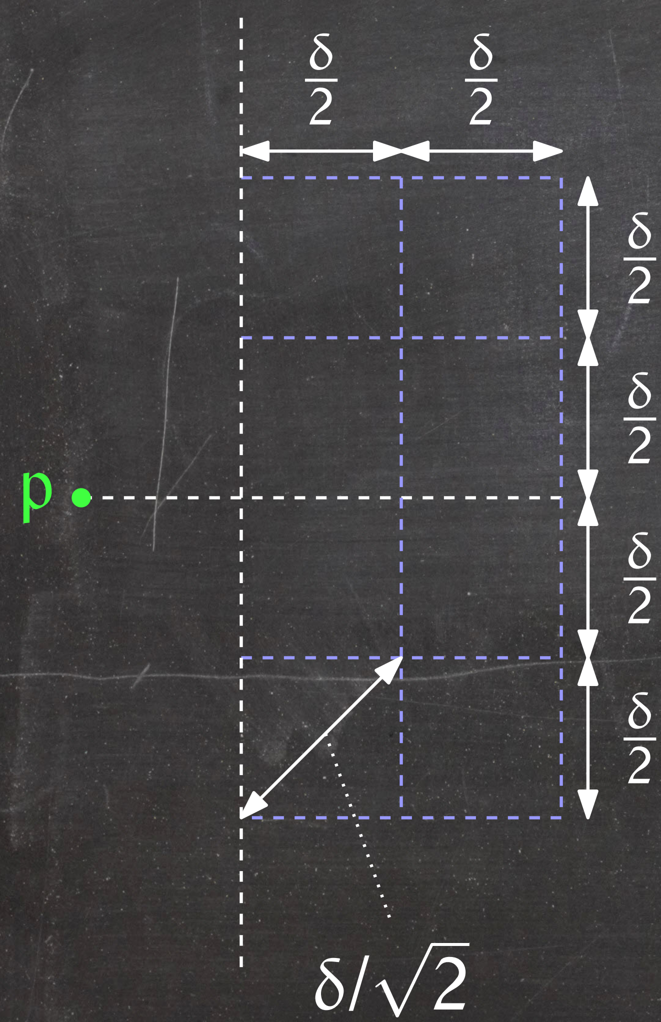
Corollary: The running time of ClosestPairLR is in $O(n)$.



Closest Pair: One Left, One Right

Lemma: For every point $p \in Z_l$, there exist at most 8 points $q \in Z_r$ such that $|q.y - p.y| \leq \delta$.

Corollary: The running time of ClosestPairLR is in $O(n)$.

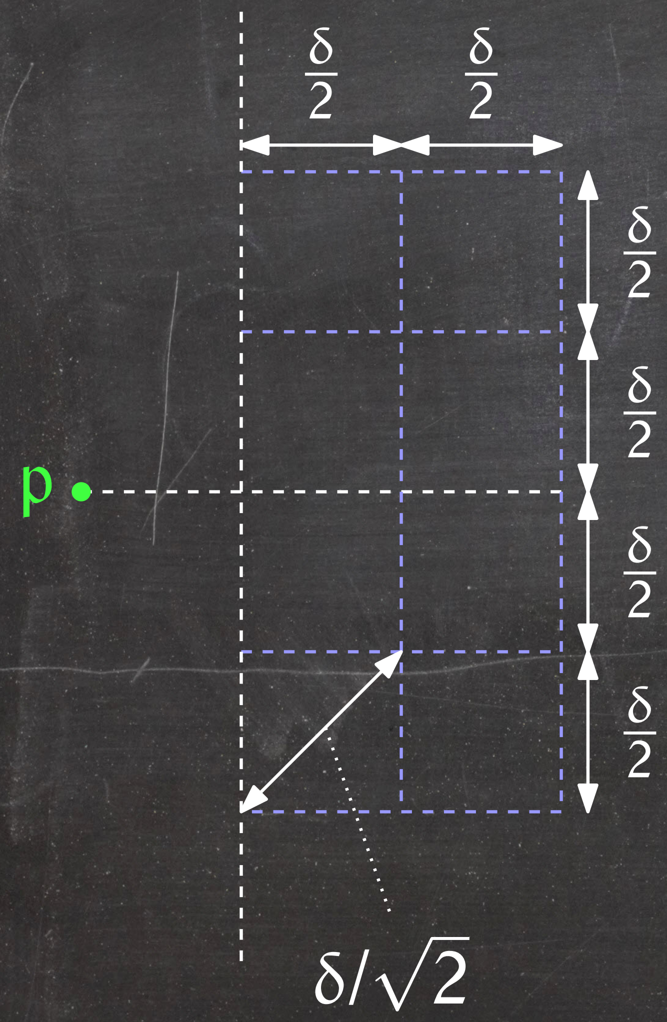


Two points in the same square have distance at most $\delta/\sqrt{2} < \delta$ from each other.

Closest Pair: One Left, One Right

Lemma: For every point $p \in Z_l$, there exist at most 8 points $q \in Z_r$ such that $|q.y - p.y| \leq \delta$.

Corollary: The running time of ClosestPairLR is in $O(n)$.



Two points in the same square have distance at most $\delta/\sqrt{2} < \delta$ from each other.

$\Rightarrow$ At most one point in each of the 8 squares.

Summary

The Divide and Conquer paradigm:

- Divide the input into smaller instances of the same problem.
- Solve these instances recursively.
- Combine the obtained solutions to obtain a solution to the original input.

Divide-and-conquer algorithms always recurse on smaller inputs.

$\Rightarrow$ Natural expression of running time using recurrence relations.
$\Rightarrow$ Natural strategy to prove correctness is induction.

Solving recurrence relations:

- Substitution
- Recursion trees
- Master Theorem