Recursion

Lecture10

Algorithms

- We will see two types of algorithms next

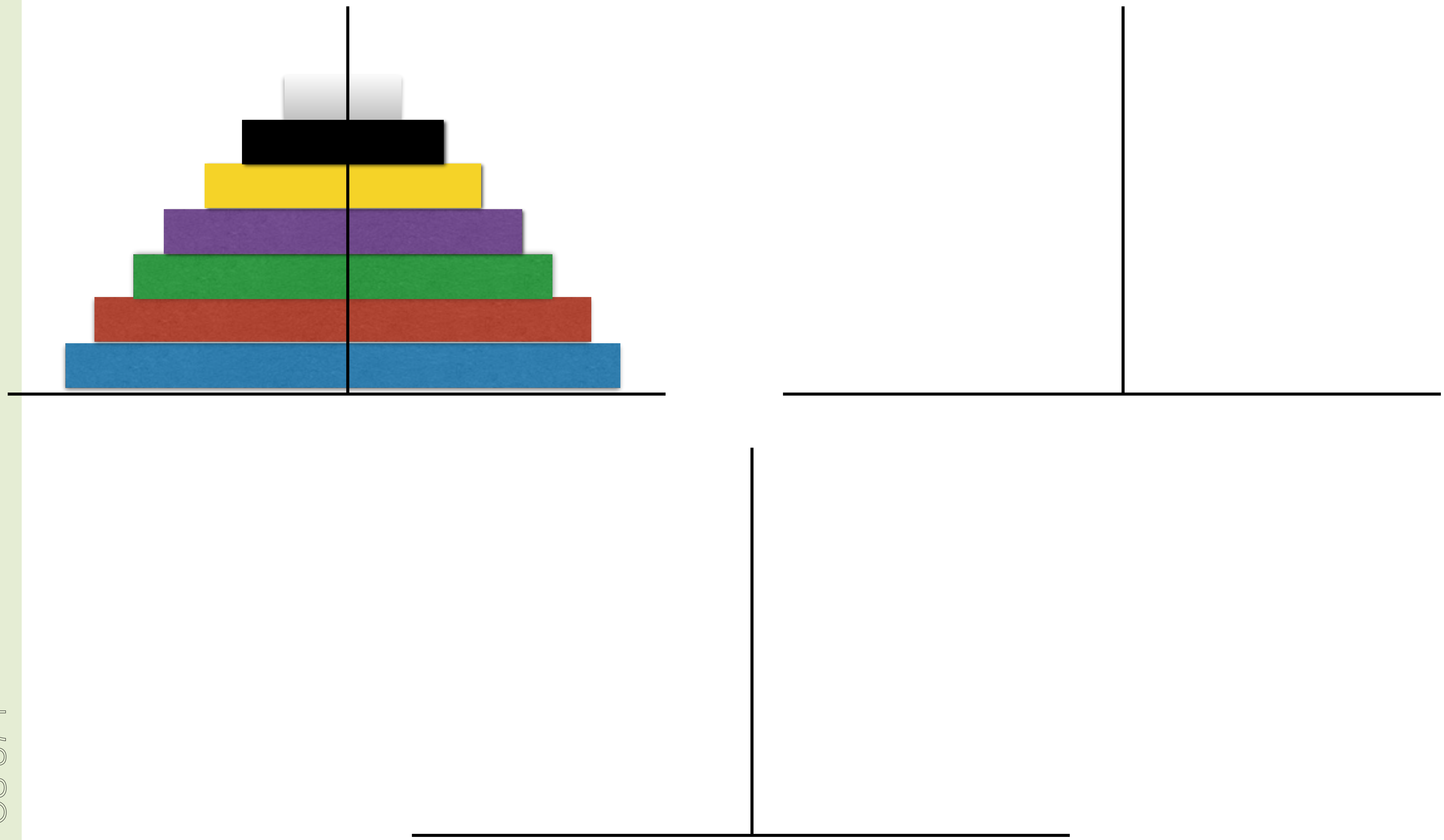
  1. Recursion (e.g. how to build an NFA from RegExp)

  2. Graph Algorithms (later)

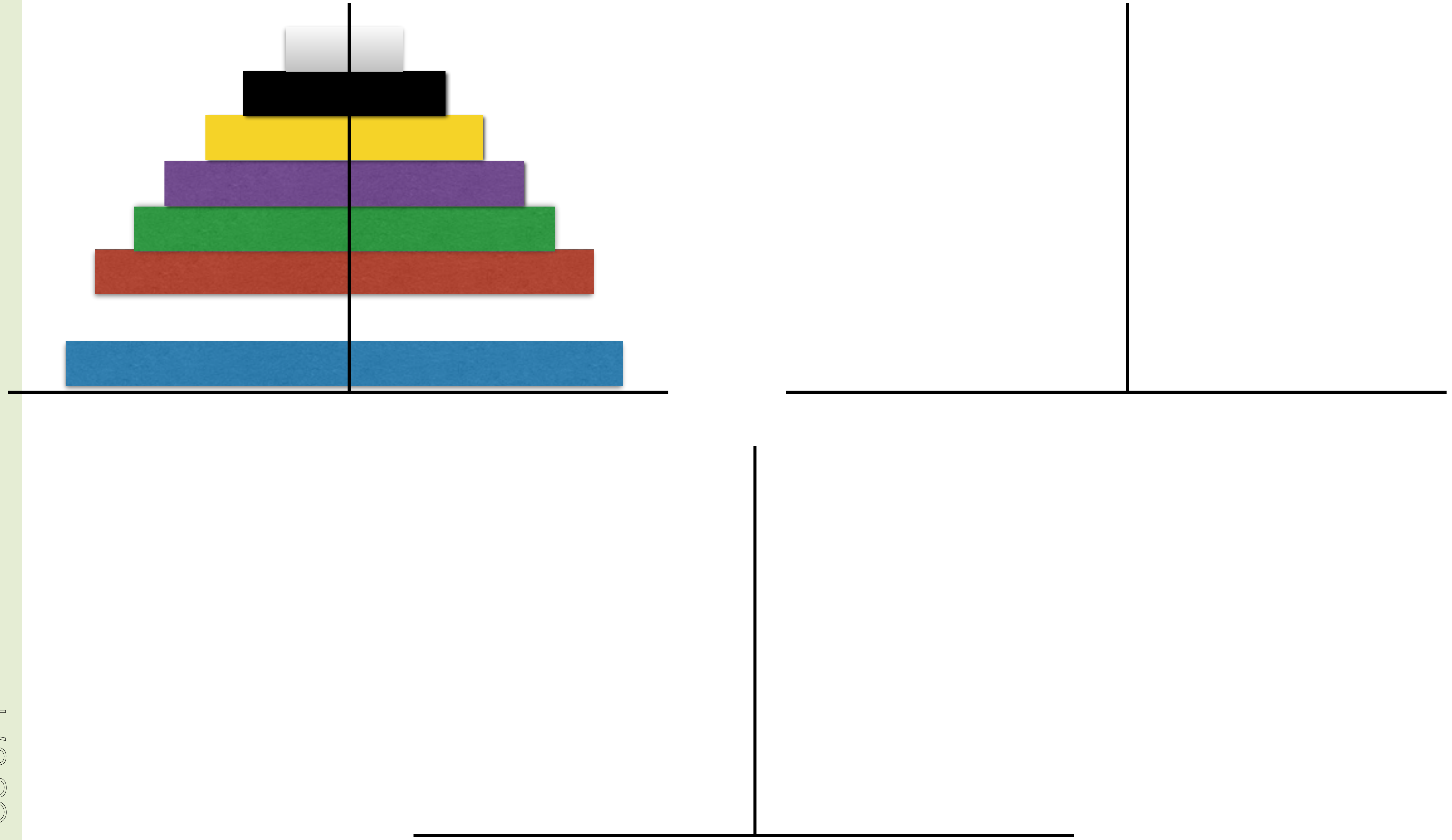
What is Recursion?

Tower of Hanoi: Move the tower from one peg to another without ever putting a larger block on top of a smaller one.

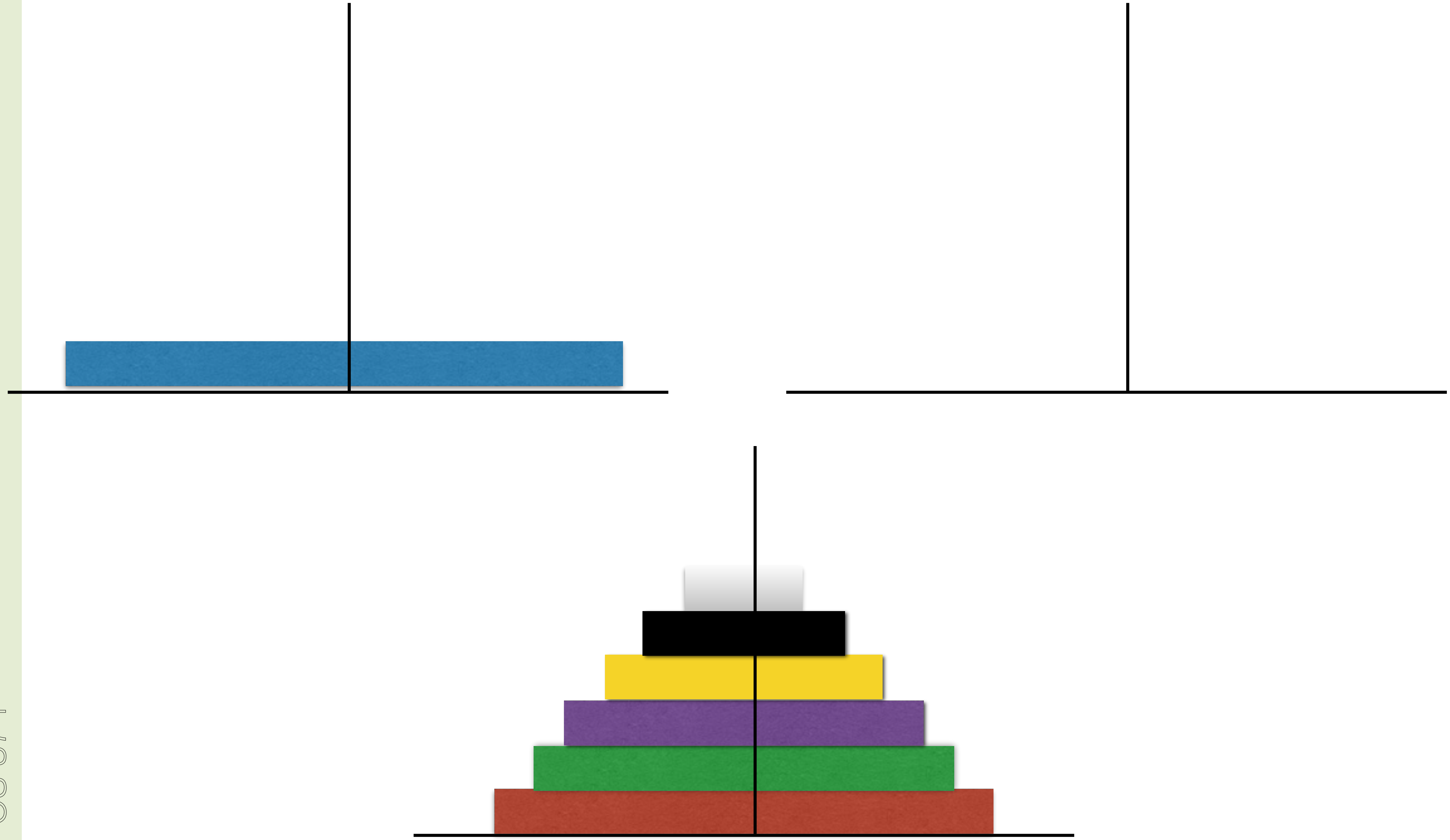
Tower of Hanoi



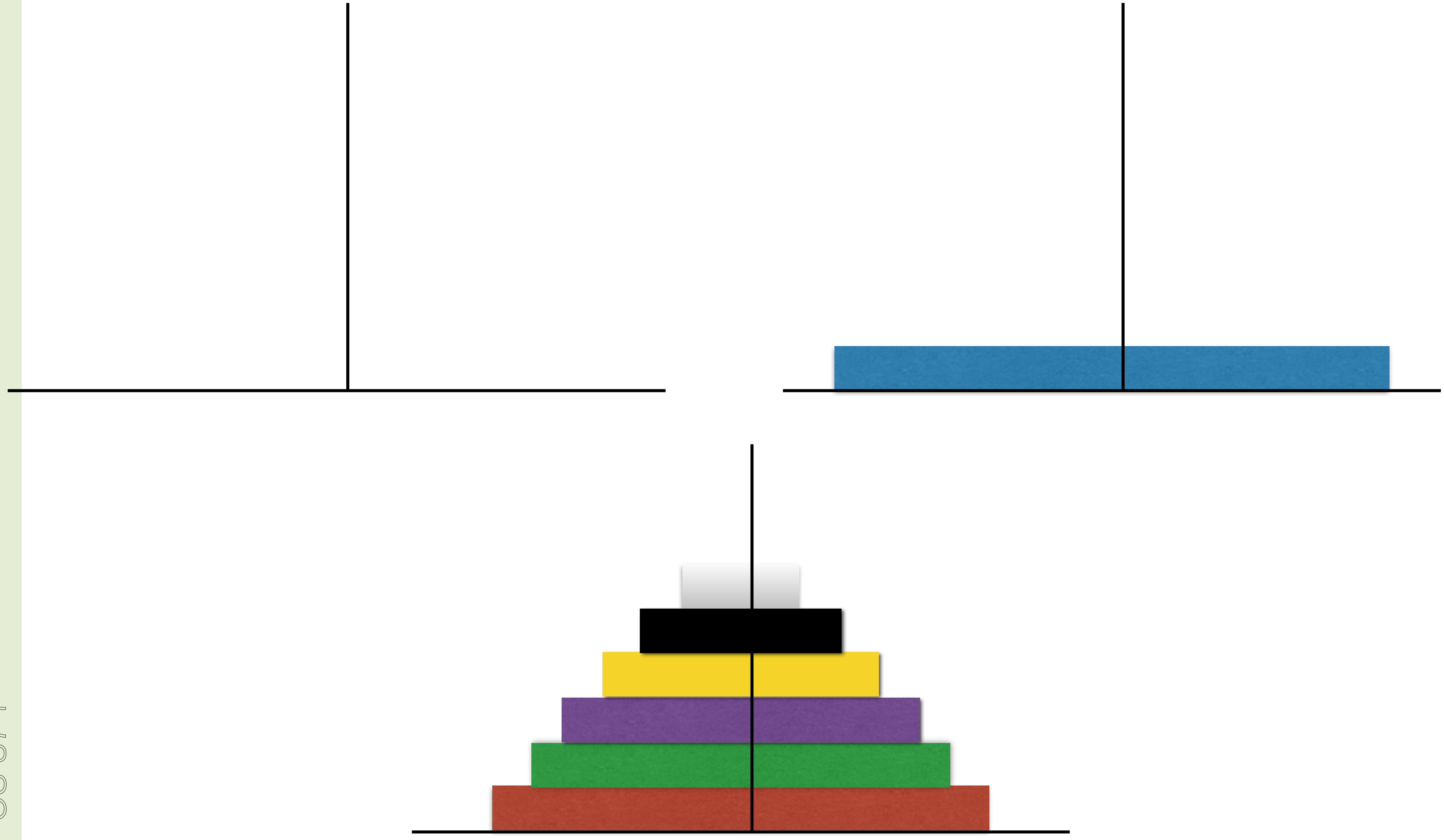
Tower of Hanoi



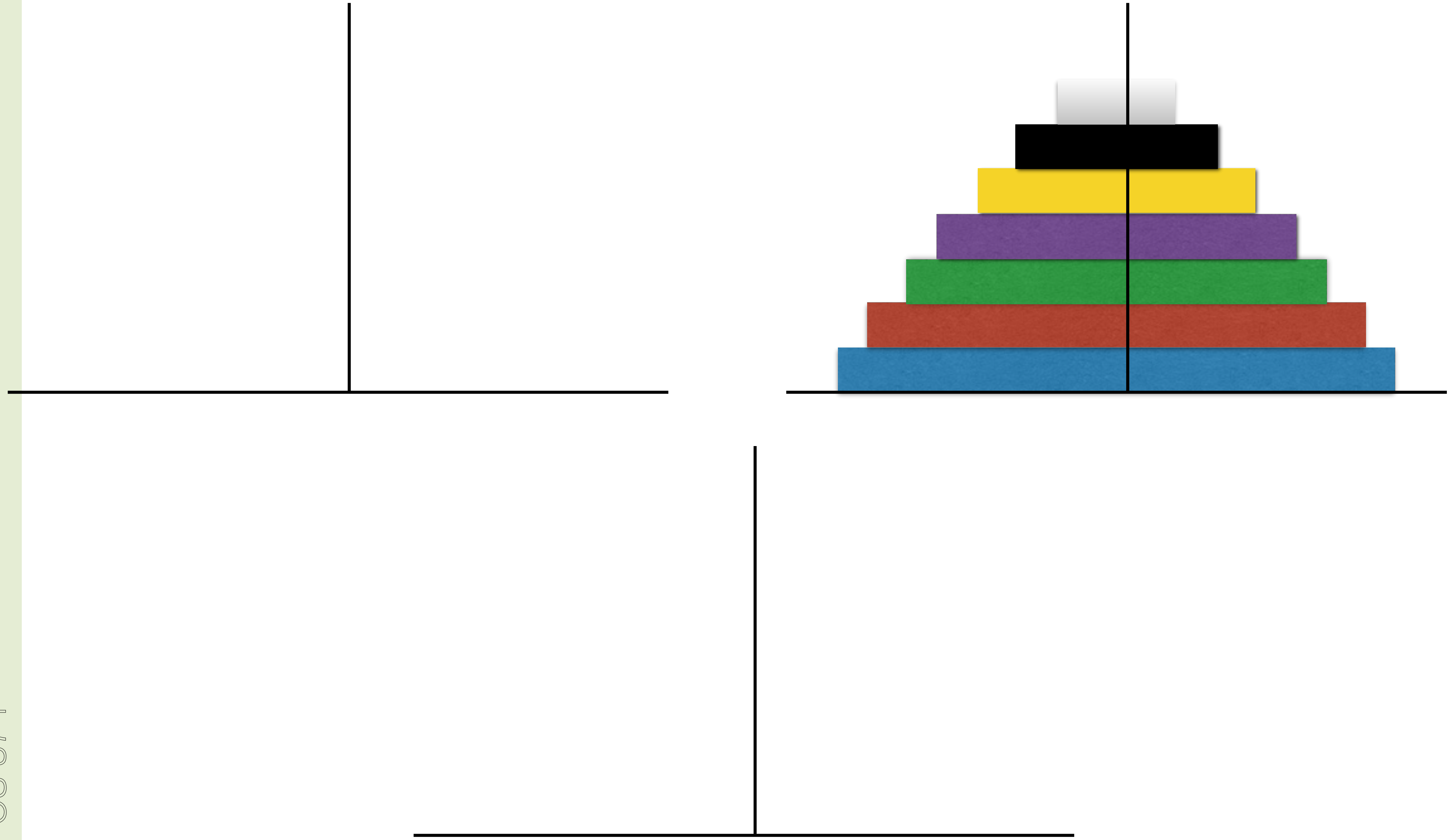
Tower of Hanoi



Tower of Hanoi

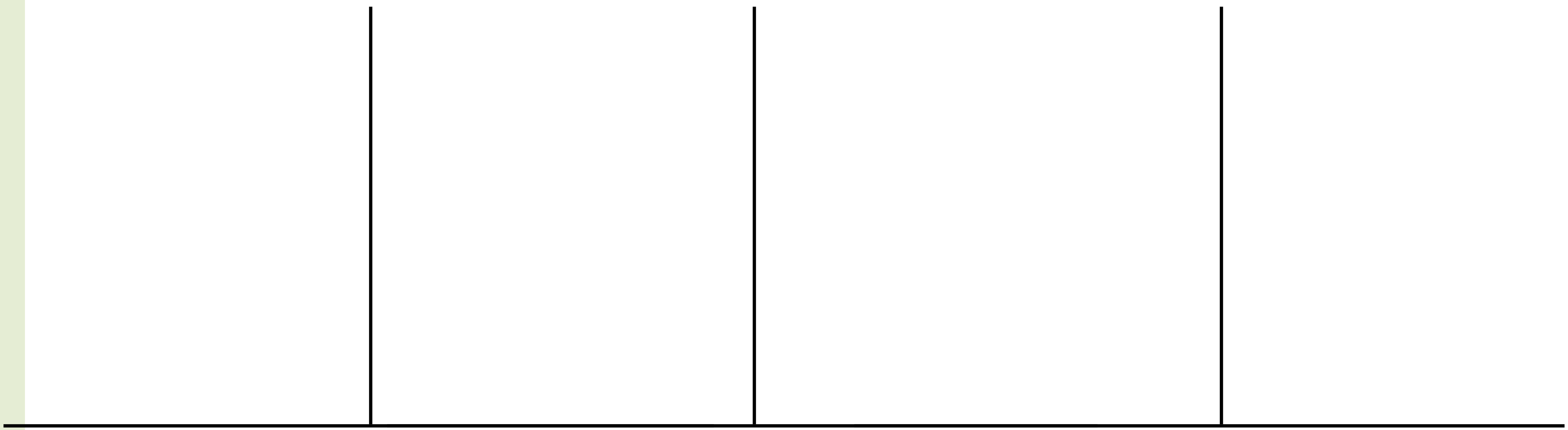


Tower of Hanoi



Base Case?

Need to be careful about when we cannot invoke the induction ferry: Base Cases



Hanoi Algorithm

$\underline{\textsc{Hanoi}(n, src, dst, tmp):}$
    if $n > 0$
        $\textsc{Hanoi}(n-1, src, tmp, dst)$
        move disk n from src to dst
        $\textsc{Hanoi}(n-1, tmp, dst, src)$

Reduction = Delegation

Sometimes hard to delegate.

Reduction = Delegation

Say we want to build a minimal DFA from a regular expression

- Reg Exp → NFA (thompson)
- NFA → DFA (subset)
- DFA → min DFA (Moore)

3 Steps. Not important how any of those work, as long as we are guaranteed they work

Reduction = Delegation

How do you hunt a blue elephant?

- With the blue elephant gun

How do you hunt a red elephant?

- Hold its trunk until it turns blue, then hunt it with the blue elephant gun

How do you hunt a white elephant?

- Embarrass it till it becomes red. Use algorithm for hunting red elephants.

Reduction = Delegation

Sometimes hard to delegate.

Recursion even harder to delegate,you have to trust yourself.

Recursion = Delegation to yourself

Recursion is reduction to smaller instances of the SAME problem, which are solved by magic (or fairies, or inductive hypothesis…)

Sorting

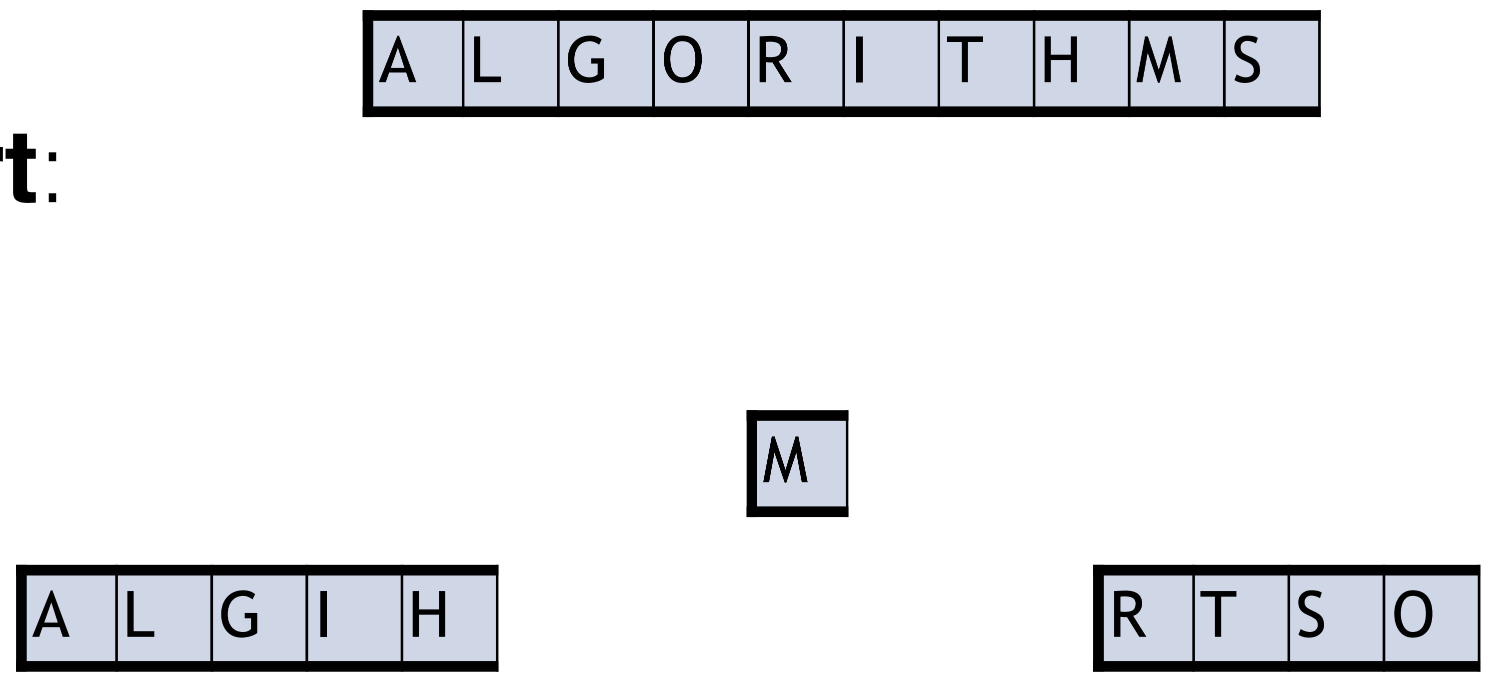
A	L	G	O	R	I	T	H	M	S

Quicksort:

- choose a pivot element from the array
- partition the array into three subarrays: one with elements smaller than pivot, one the pivot itself, one with elements larger than pivot.
- Recursively quick sort the first and last subarray
- How to choose pivot?

Sorting

Quicksort:



Sorting

A	L	G	O	R	I	T	H	M	S

Quicksort:

M

A	G	H	I	L

O	R	S	T

Sorting

A	L	G	O	R	I	T	H	M	S

Quicksort:

A	G	H	I	L	M	O	R	S	T

Sorting

Quicksort:

$\underline{\text{QUICKSORT}(A[1..n])\text{:}}$
  if $(n > 1)$
    Choose a pivot element $A[p]$
    $r \leftarrow \text{PARTITION}(A, p)$
    $\text{QUICKSORT}(A[1..r-1])$
    $\text{QUICKSORT}(A[r+1..n])$

Sorting

Partition (linear time):

<u>PARTITION($A[1..n], p$):</u>

swap $A[p] \leftrightarrow A[n]$
$i \leftarrow 0$
$j \leftarrow n$
while $(i < j)$
  repeat $i \leftarrow i + 1$ until $(i \geq j$ or $A[i] \geq A[n])$
  repeat $j \leftarrow j - 1$ until $(i \geq j$ or $A[j] \leq A[n])$
  if $(i < j)$
    swap $A[i] \leftrightarrow A[j]$
swap $A[i] \leftrightarrow A[n]$
return i

Sorting

A	L	G	O	R	I	T	H	M	S

Mergesort:

- Divide the input array into two subarrays of roughly equal size
- Recursively merge sort each of the subarrays
- Merge the two newly sorted subarrays into a single sorted array

Sorting

A	L	G	O	R	I	T	H	M	S

Mergesort:

A	L	G	O	R

I	T	H	M	S

Sorting

A	L	G	O	R	I	T	H	M	S

Mergesort:

A	G	L	O	R

H	I	M	S	T

Sorting

A	L	G	O	R	I	T	H	M	S

Mergesort:

A	G	L	O	R

H	I	M	S	T

Need to merge the two subarrays.

Sorting

A	L	G	O	R	I	T	H	M	S

Merge:

A	G	L	O	R

H	I	M	S	T

- Compare the first elements of the subarrays
- Write the smallest one in the output array.
- Recursion, now the problem is smaller

Sorting

A	L	G	O	R	I	T	H	M	S

Merge:

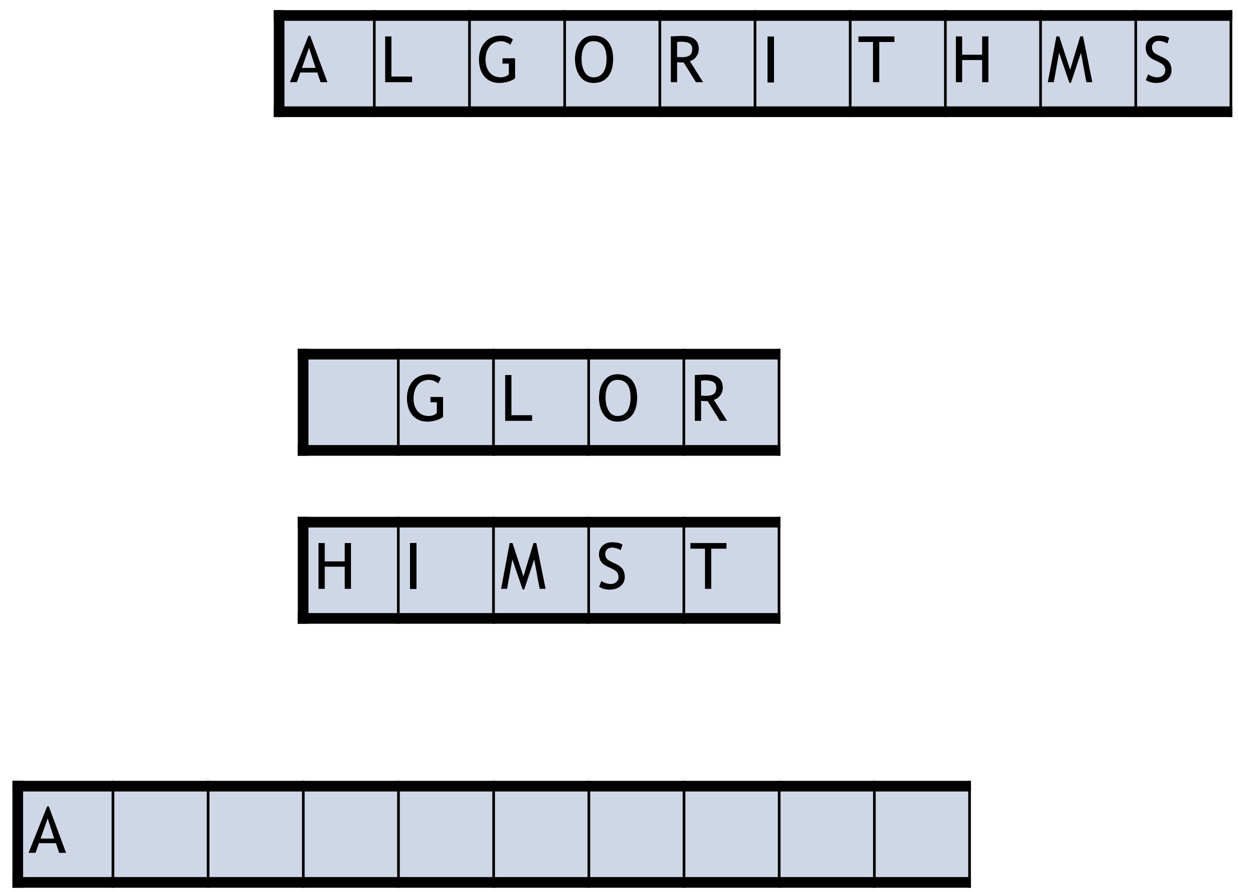
A	G	L	O	R

H	I	M	S	T

One comparison, one recursive call

Sorting

Merge:



One comparison, one recursive call

Sorting

A	L	G	O	R	I	T	H	M	S

Merge:

	G	L	O	R

H	I	M	S	T

A									

Where can this recursion break?

Sorting

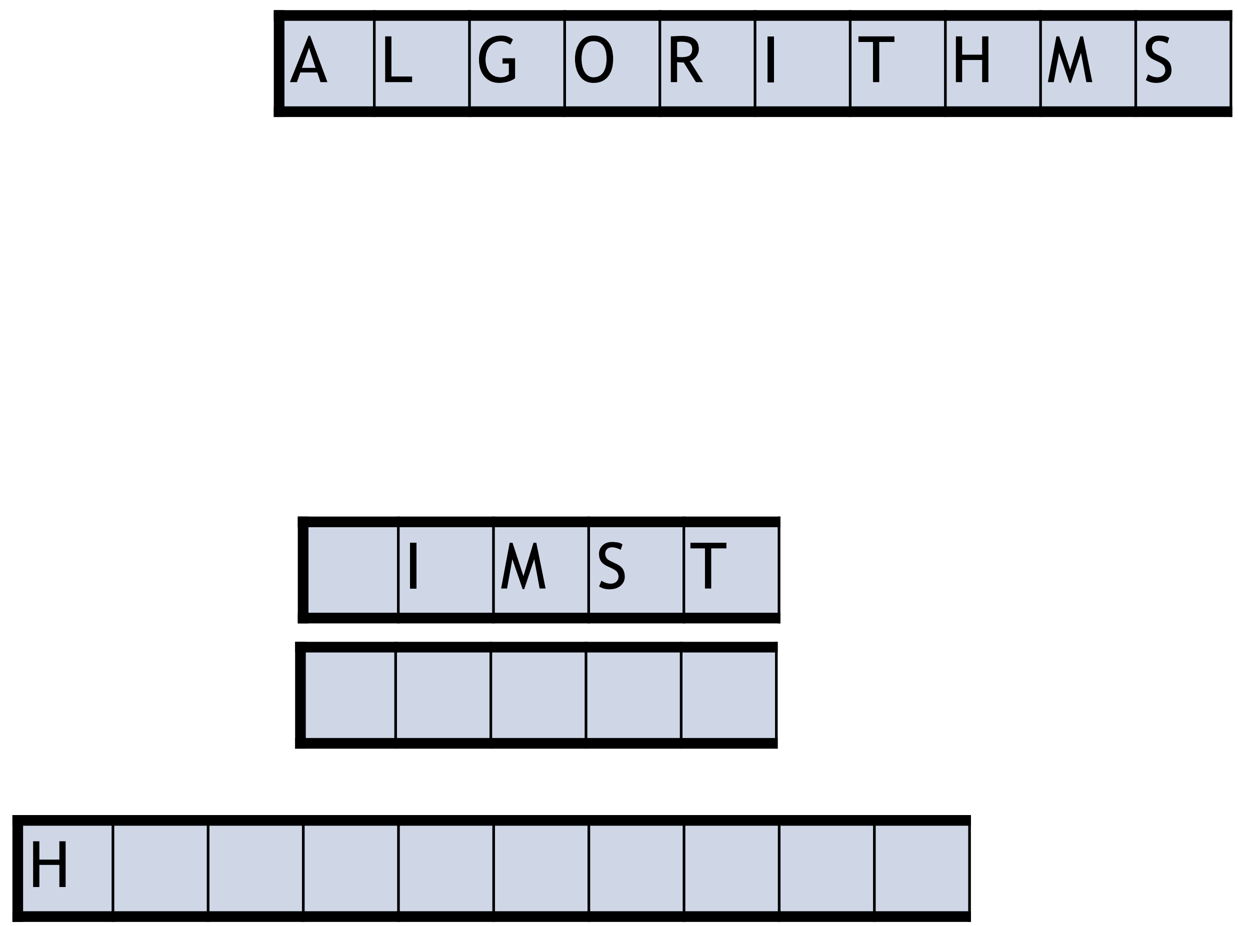
A	L	G	O	R	I	T	H	M	S

Merge:

H	I	M	S	T

Sorting

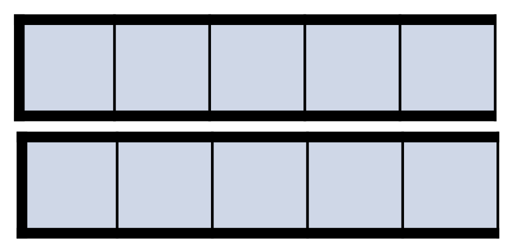
Merge:



Sorting

A	L	G	O	R	I	T	H	M	S

Merge:



Where can this recursion break?

Sorting

Merge:

$$
\begin{array}{l}
\underline{\text{MERGE}(A[1..n], m):} \\
\quad i \leftarrow 1;\ \ j \leftarrow m+1 \\
\quad \text{for } k \leftarrow 1 \text{ to } n \\
\qquad \text{if } j > n \\
\qquad\quad B[k] \leftarrow A[i];\ \ i \leftarrow i+1 \\
\qquad \text{else if } i > m \\
\qquad\quad B[k] \leftarrow A[j];\ \ j \leftarrow j+1 \\
\qquad \text{else if } A[i] < A[j] \\
\qquad\quad B[k] \leftarrow A[i];\ \ i \leftarrow i+1 \\
\qquad \text{else} \\
\qquad\quad B[k] \leftarrow A[j];\ \ j \leftarrow j+1 \\
\\
\quad \text{for } k \leftarrow 1 \text{ to } n \\
\qquad A[k] \leftarrow B[k]
\end{array}
$$

Loop = recursion

- When writing actual code easier to unfold the recursion

- When proving correctness easier to use induction (=recursion)

Sorting

Mergesort:

$$
\begin{array}{l}
\underline{\textsc{MergeSort}(A[1\,..\,n]):}\\
\quad \text{if } n > 1\\
\qquad m \leftarrow \lfloor n/2 \rfloor\\
\qquad \textsc{MergeSort}(A[1\,..\,m])\\
\qquad \textsc{MergeSort}(A[m+1\,..\,n])\\
\qquad \textsc{Merge}(A[1\,..\,n], m)
\end{array}
$$

Base cases:

- When size of arrays to merge is 1
- When size of arrays is less than10 and then brute force
- It doesn’t matter, no need to optimize

Proof of Correctness

- We prove MERGE is correct by induction on $n - k + 1$, which is the total size of the two sorted subarrays $A[i..m]$ and $A[j..n]$ that remain to be merged into $B[k..n]$ when the kth iteration of the main loop begins. There are five cases to consider. Yes, five.
  - If $k > n$, the algorithm correctly merges the two empty subarrays by doing absolutely nothing. (This is the base case of the inductive proof.)
  - If $i \le m$ and $j > n$, the subarray $A[j..n]$ is empty. Because both subarrays are sorted, the smallest element in the union of the two subarrays is $A[i]$. So the assignment $B[k] \leftarrow A[i]$ is correct. The inductive hypothesis implies that the remaining subarrays $A[i+1..m]$ and $A[j..n]$ are correctly merged into $B[k+1..n]$.
  - Similarly, if $i > m$ and $j \le n$, the assignment $B[k] \leftarrow A[j]$ is correct, and The Recursion Fairy correctly merges—sorry, I mean the inductive hypothesis implies that the MERGE algorithm correctly merges—the remaining subarrays $A[i..m]$ and $A[j+1..n]$ into $B[k+1..n]$.
  - If $i \le m$ and $j \le n$ and $A[i] < A[j]$, then the smallest remaining element is $A[i]$. So $B[k]$ is assigned correctly, and the Recursion Fairy correctly merges the rest of the subarrays.
  - Finally, if $i \le m$ and $j \le n$ and $A[i] \ge A[j]$, then the smallest remaining element is $A[j]$. So $B[k]$ is assigned correctly, and the Recursion Fairy correctly does the rest.

Always make sanity check when you design algorithm!

Running time

- Number of fundamental operations as a function of input size n
- If array is sorted, then O(n), but we don't care about best case!
- Worst case running time for this class.
- Maybe different in practice, assumptions

Running time of Quicksort

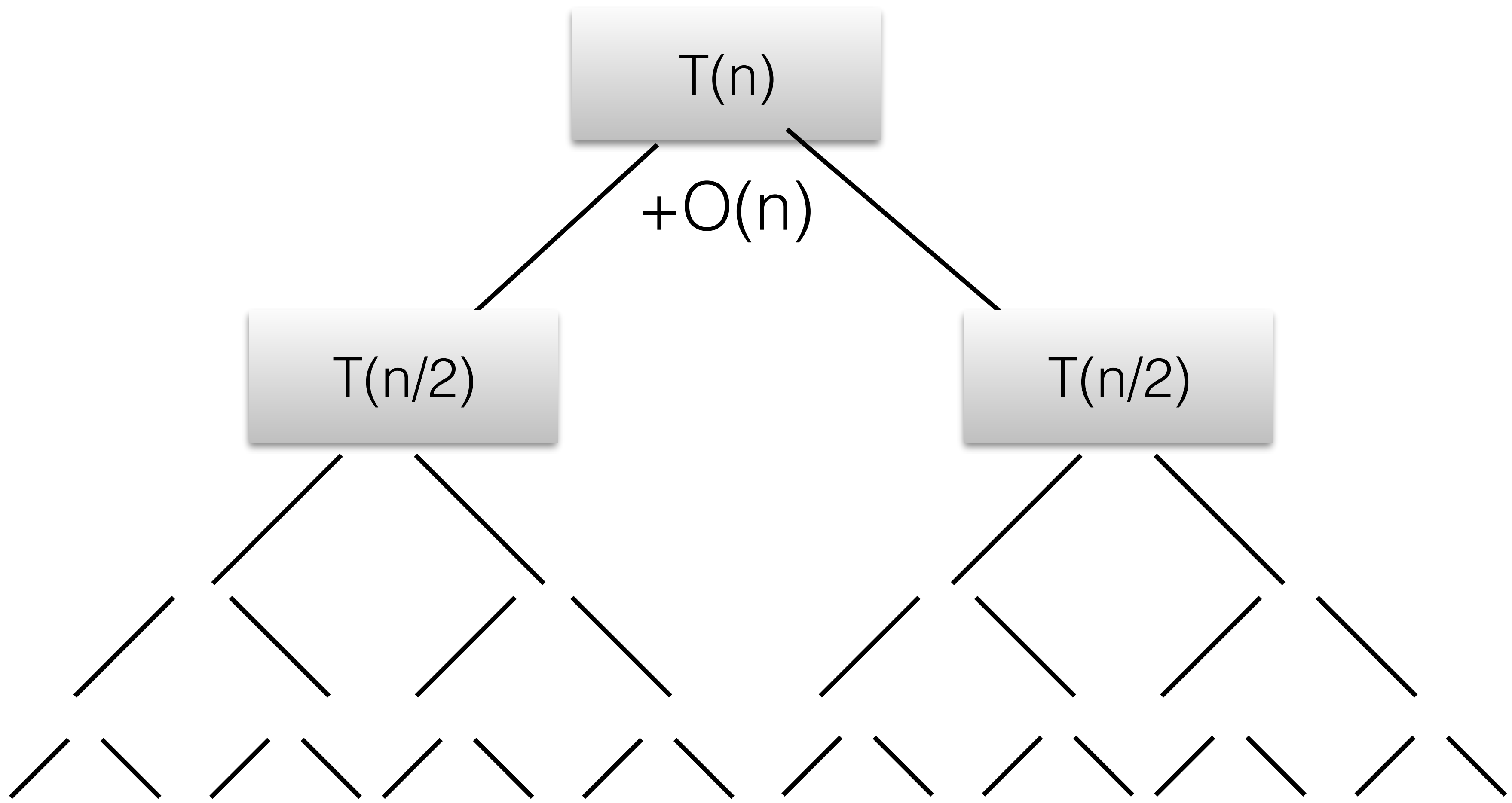
- What is the running time $T(n)$ of quicksort?
- $O(n^2)$ time! (If I choose the smallest pivot)
  - $T(n)=O(n)+T(n-1)$

    $= O(n^2)$

Running time of Mergesort

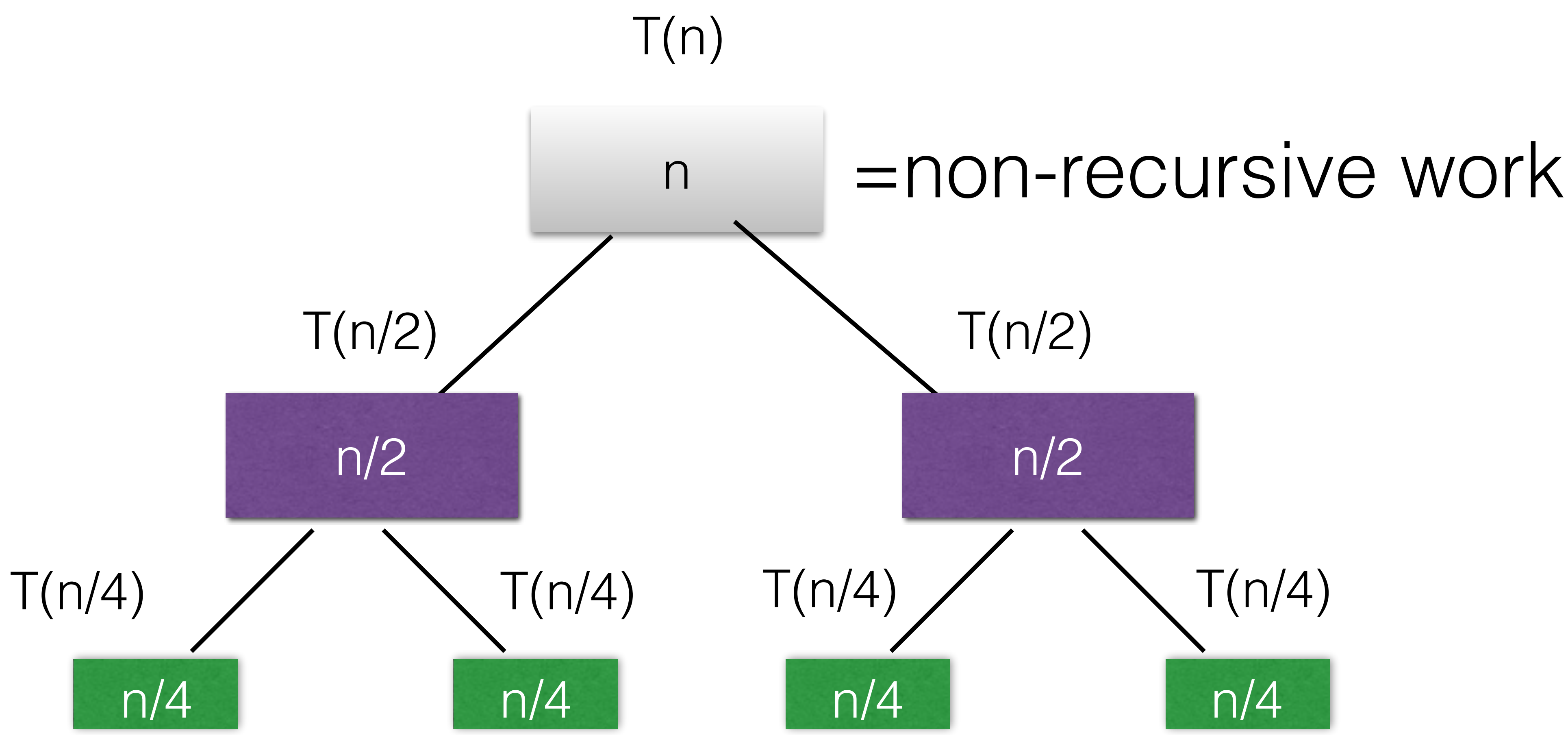
- What is the running time T(n) of mergesort?
- O(nlogn) time!
    - $T(n)=2T(n/2)+O(n)$
    - proof by induction if I know answer
    - recursion tree!

Running time of Mergesort



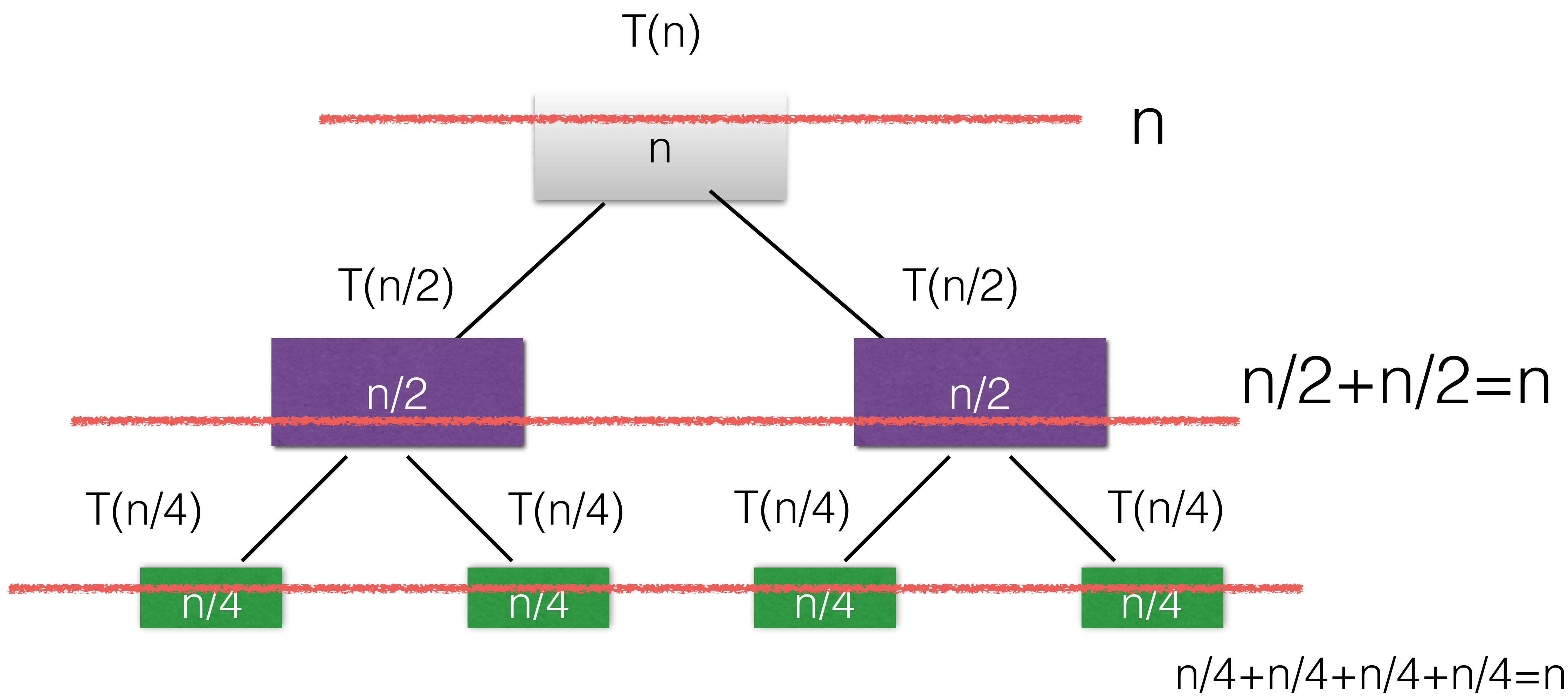
Complete binary tree
every leaf is an array of size 1

Running time of Mergesort



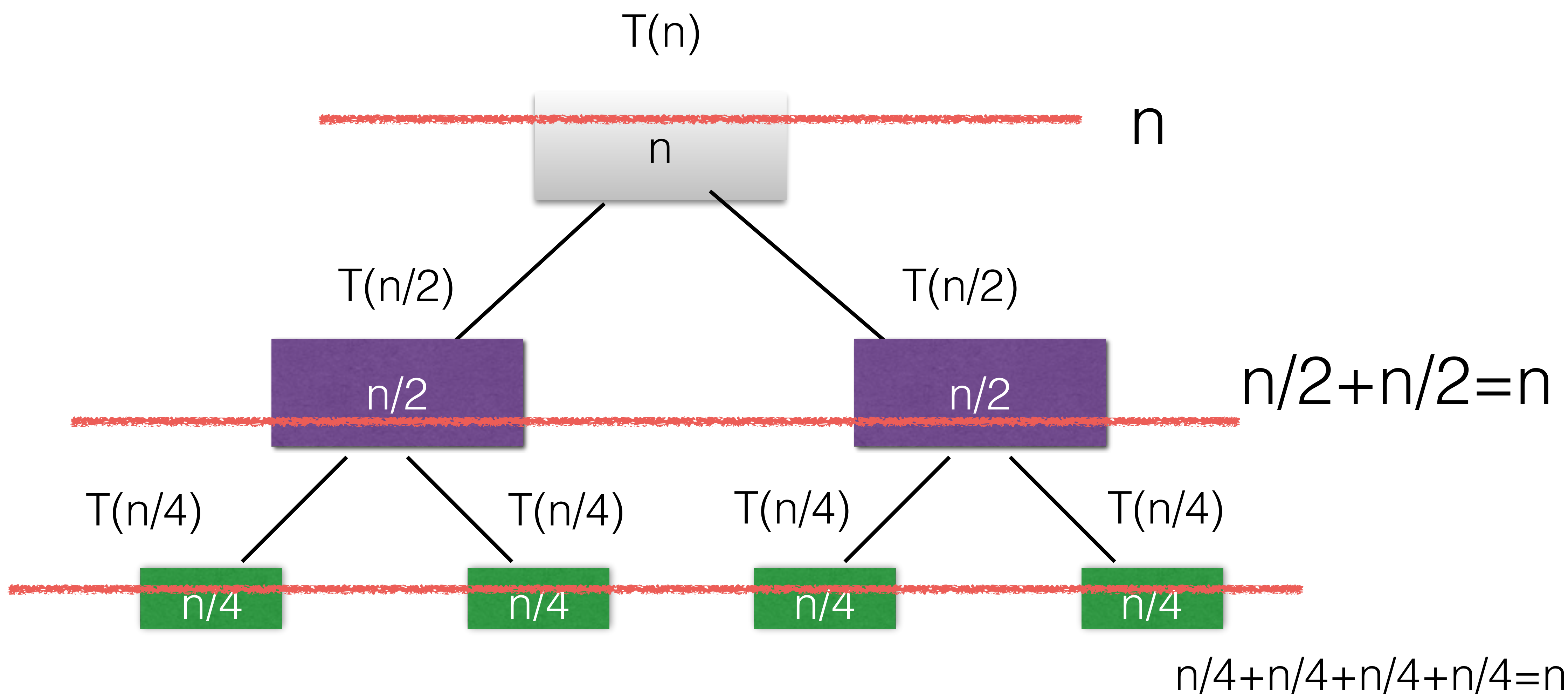
- Leave all the O() till the very end.
- Goal is to sum up all the quantities in all the nodes.

Running time of Mergesort



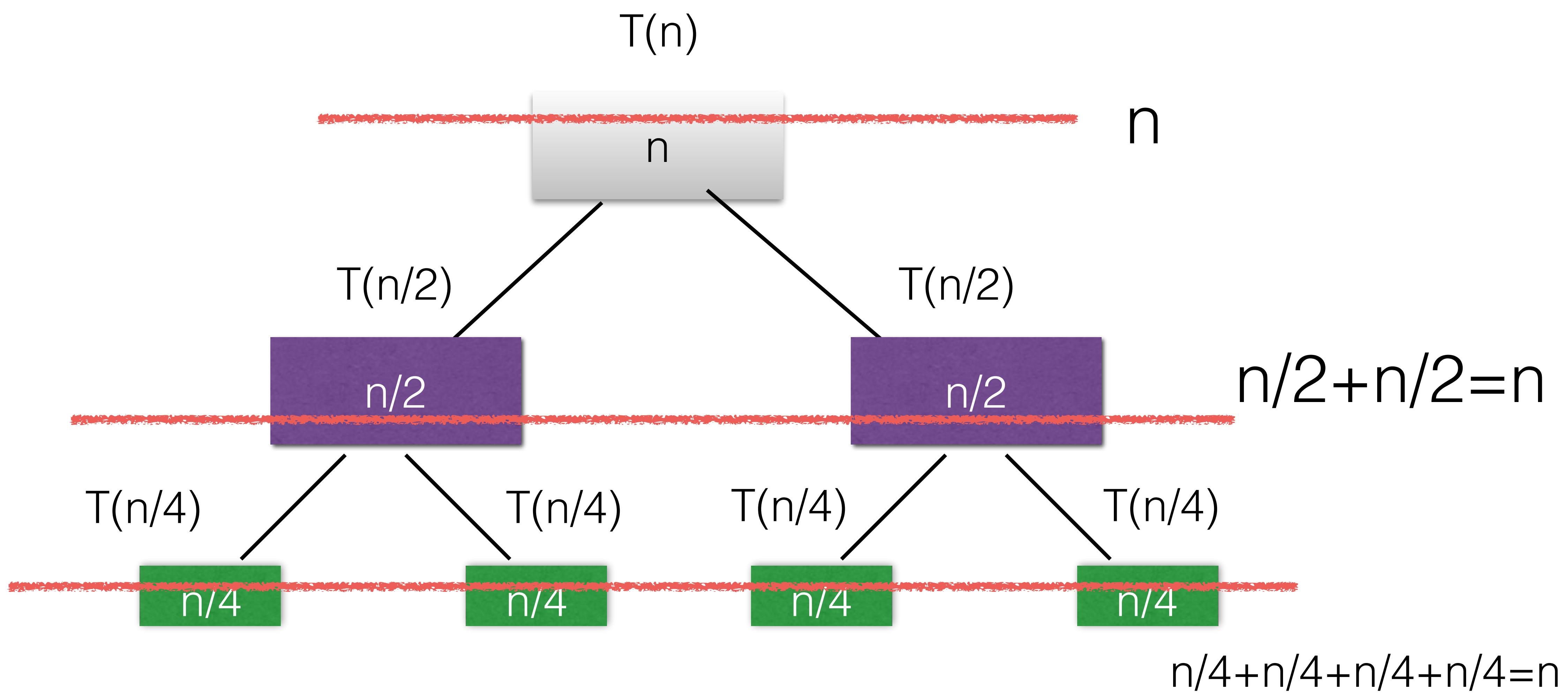
- $T(n)=2T(n/2)+O(n)$
- Solve the recurrence by summing up work at each level

Running time of Mergesort



- T(n)=2T(n/2)+O(n)
- Total amount of work at level k= total amount of work at level k-1 (induction).

Running time of Mergesort



- T(n)=2T(n/2)+O(n)
- Total amount of work = n x (height of the tree) =n logn

Running time of Quicksort, revisited

- Quicksort runs in time O(n log n) in practice.
- Quicksort runs in time O(n log n) on average if the data is randomly permuted
- Quicksort runs in expected time O(n log n) if we randomly permute the data first.