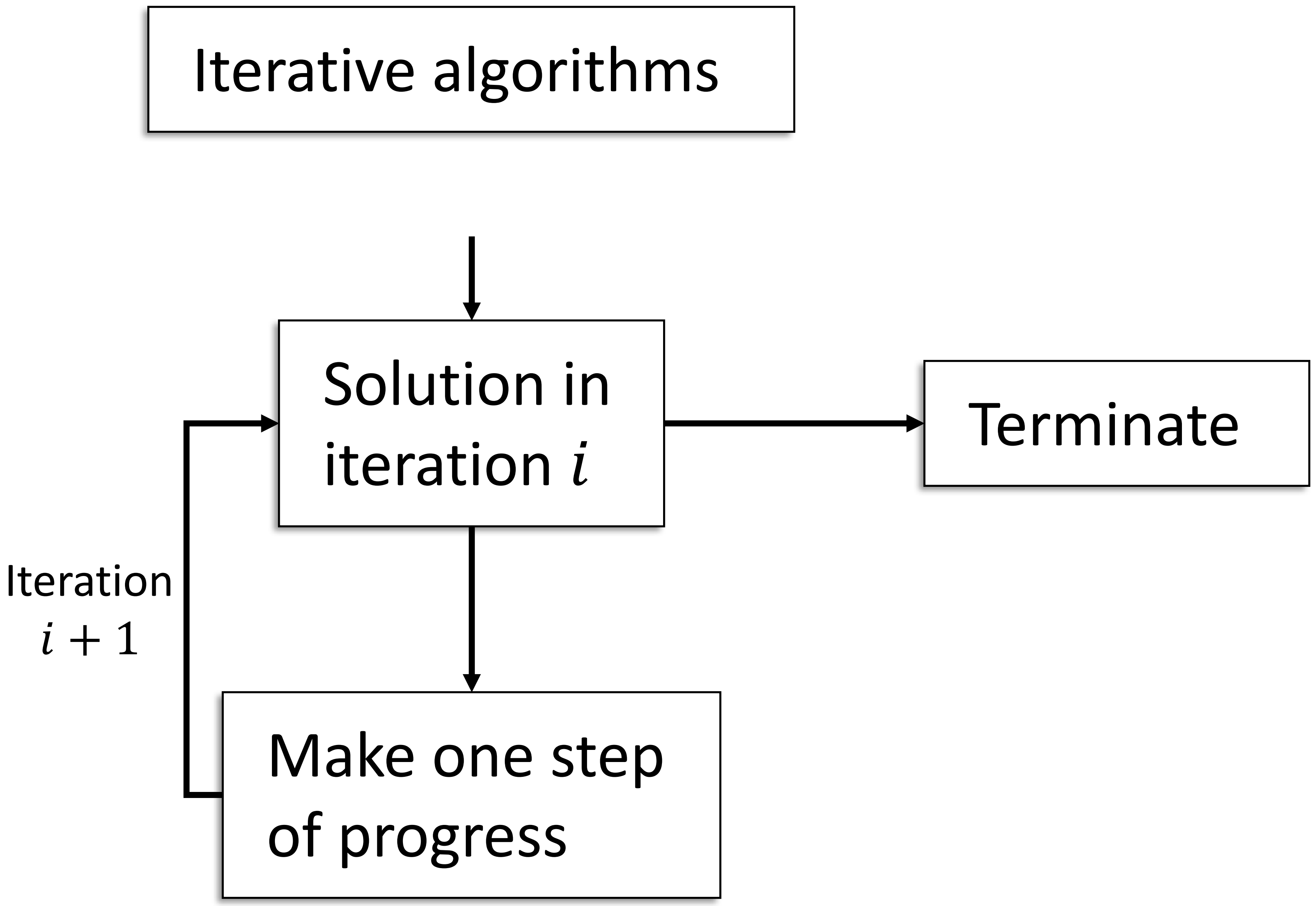
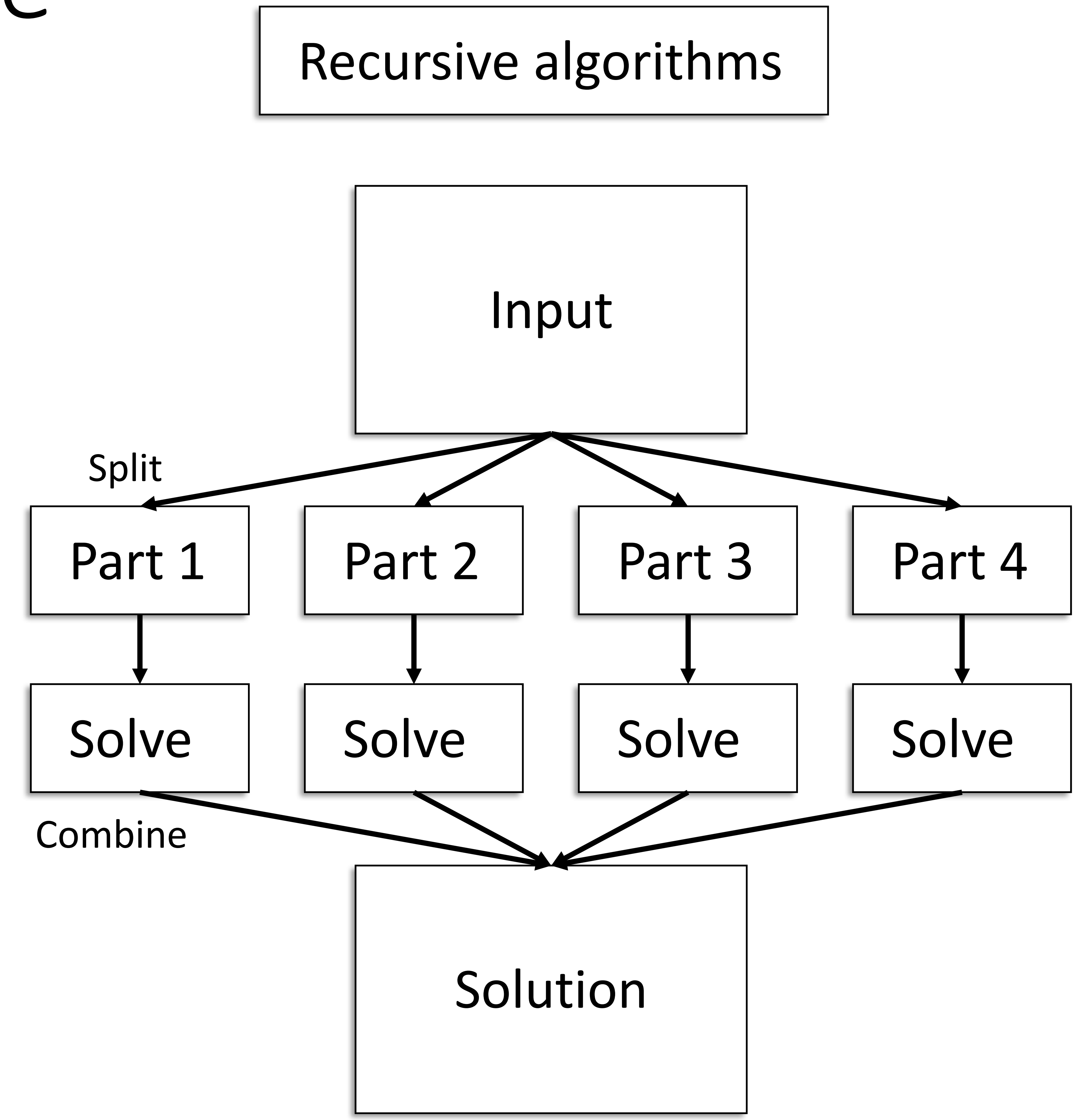
# Recursion

# From Iterative to Recursive



# Outline

- Recursion trees
- Towers of Hanoi
  - Simplify and Delegate!
- Mergesort
  - Divide and Qonquer

# Outline

- Recursion trees
- Towers of Hanoi
  - Simplify and Delegate!
- Mergesort
  - Divide and Qonquer

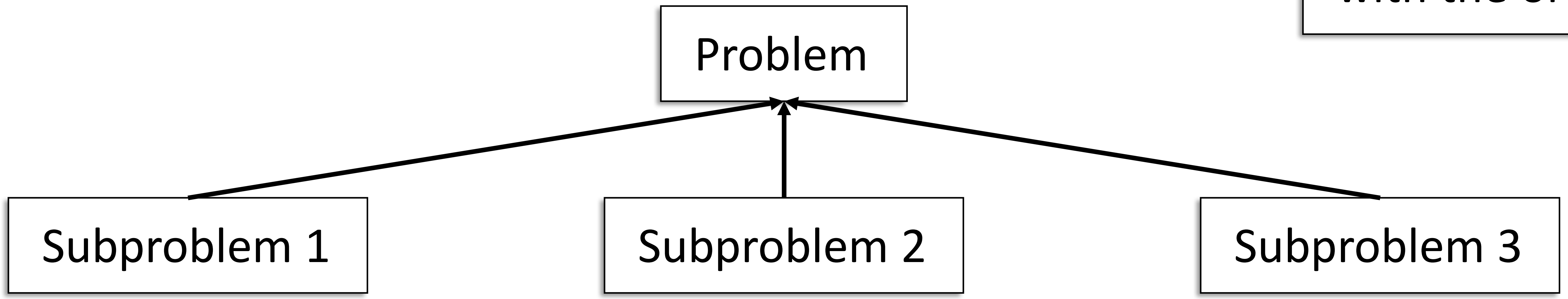
**Learning objectives:**

You are able to

- Explain the recursive approach to algorithm design, i.e., simplify and delegate.
- Describe and analyze recursive algorithms for the Towers of Hanoi problem and sorting

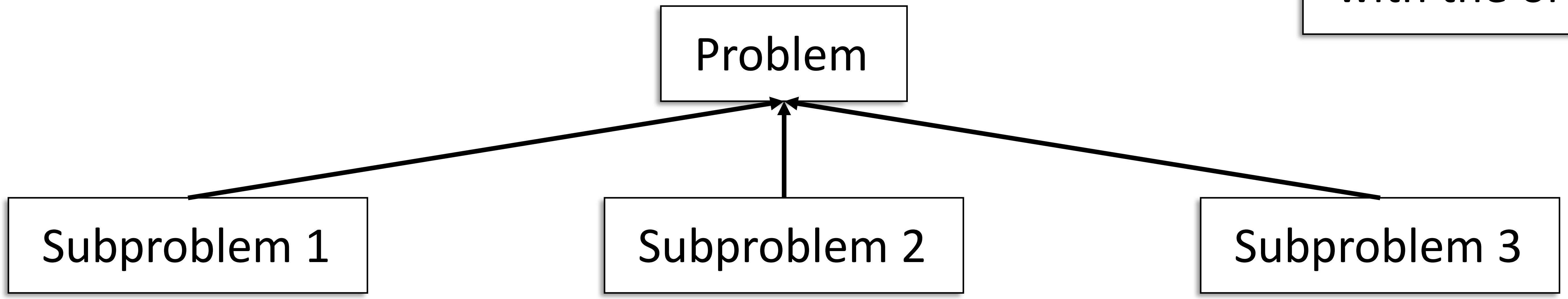
# Recursion Trees

Solutions to subproblems help with the original



# Recursion Trees

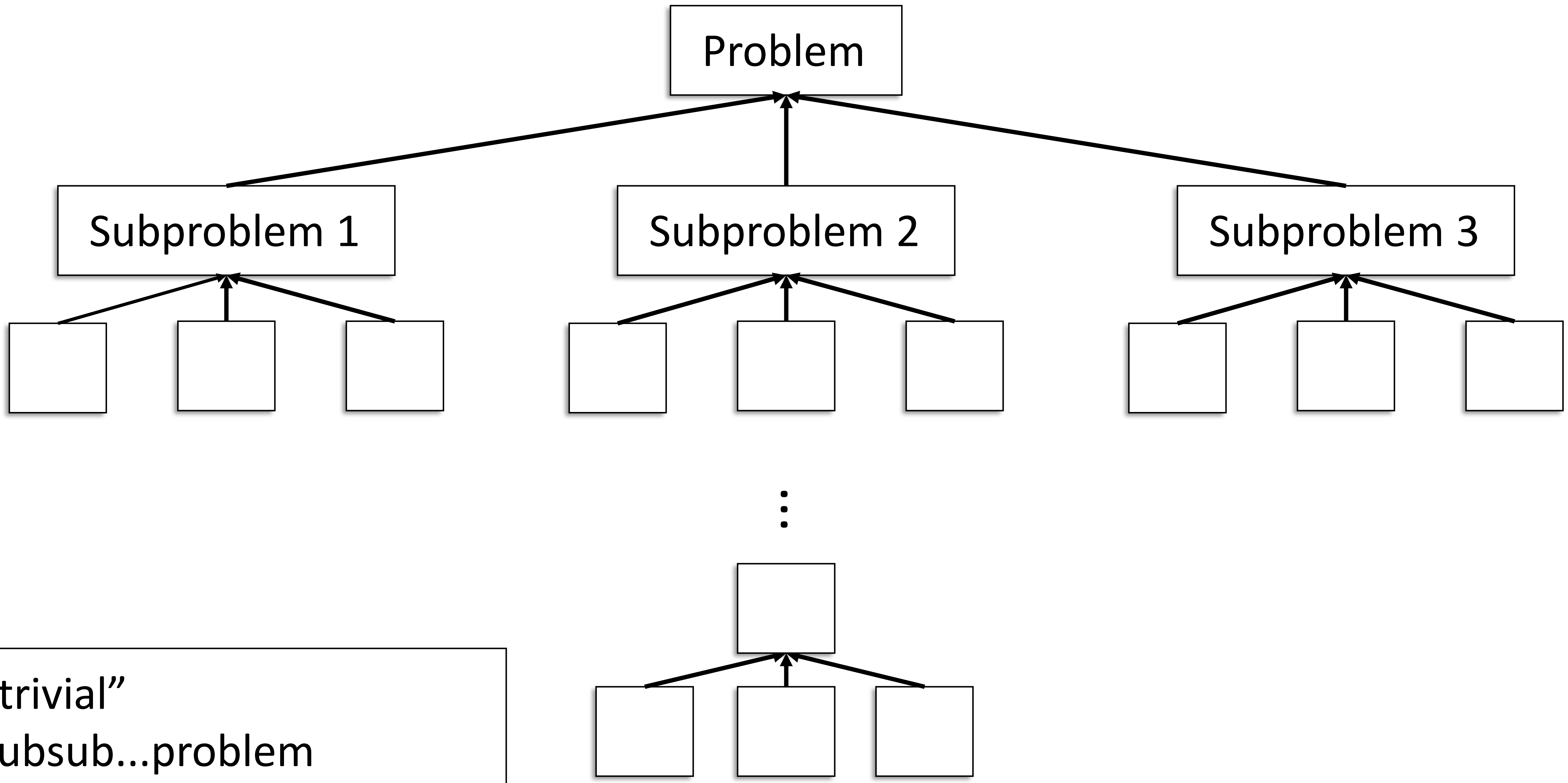
Solutions to subproblems help with the original




**Crucial:**

1) The problem setting does not change – same algorithm works
2) Subproblem is strictly easier
3) Combining subsolutions is easy

# Recursion Trees




"trivial"
subsub...problem

# Outline

- Recursion trees
- Towers of Hanoi
  - Simplify and Delegate!
- Mergesort
  - Divide and conquer

# Towers of Hanoi

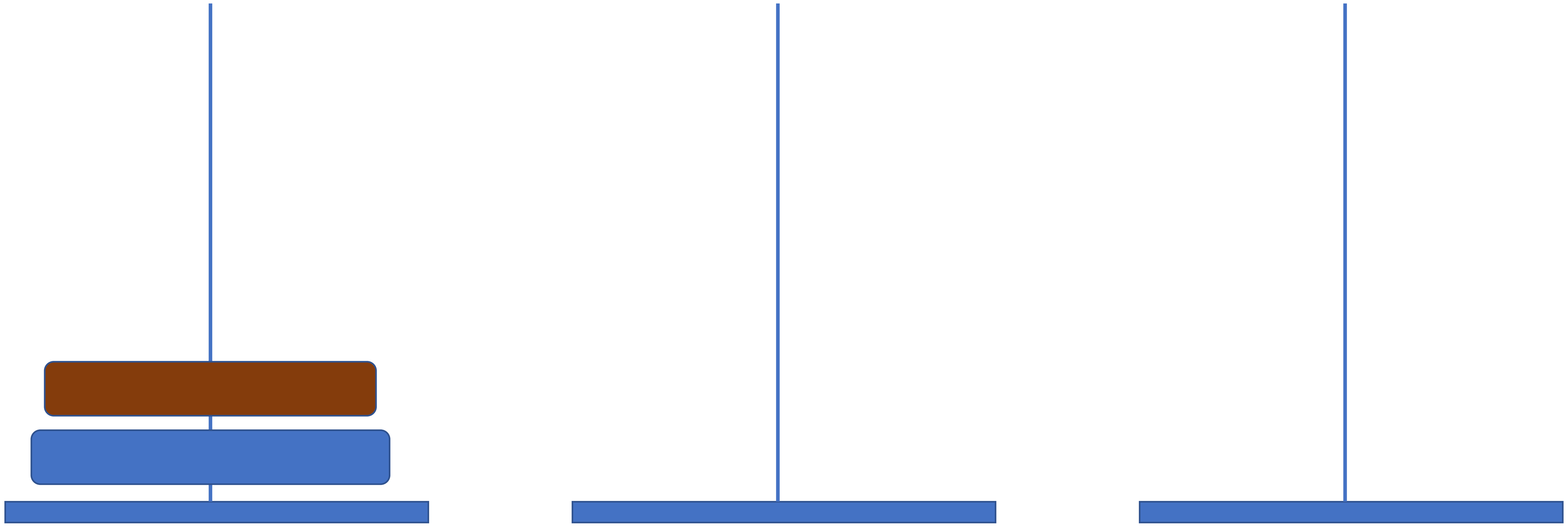
Three towers

First tower has $n$ discs.
The discs are stacked from largest to smallest

**Task:**
Move discs to the last tower.
Move one disc at a time. A disc is not allowed to be on top of a smaller disc.

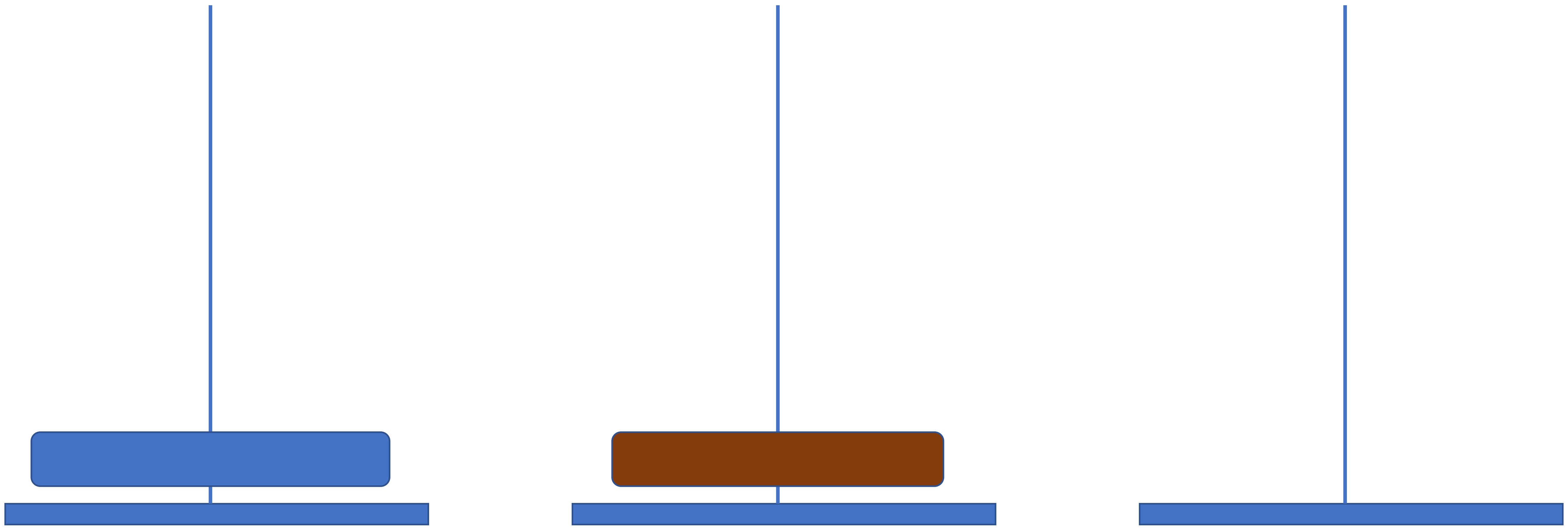
# Towers of Hanoi

The case of two discs

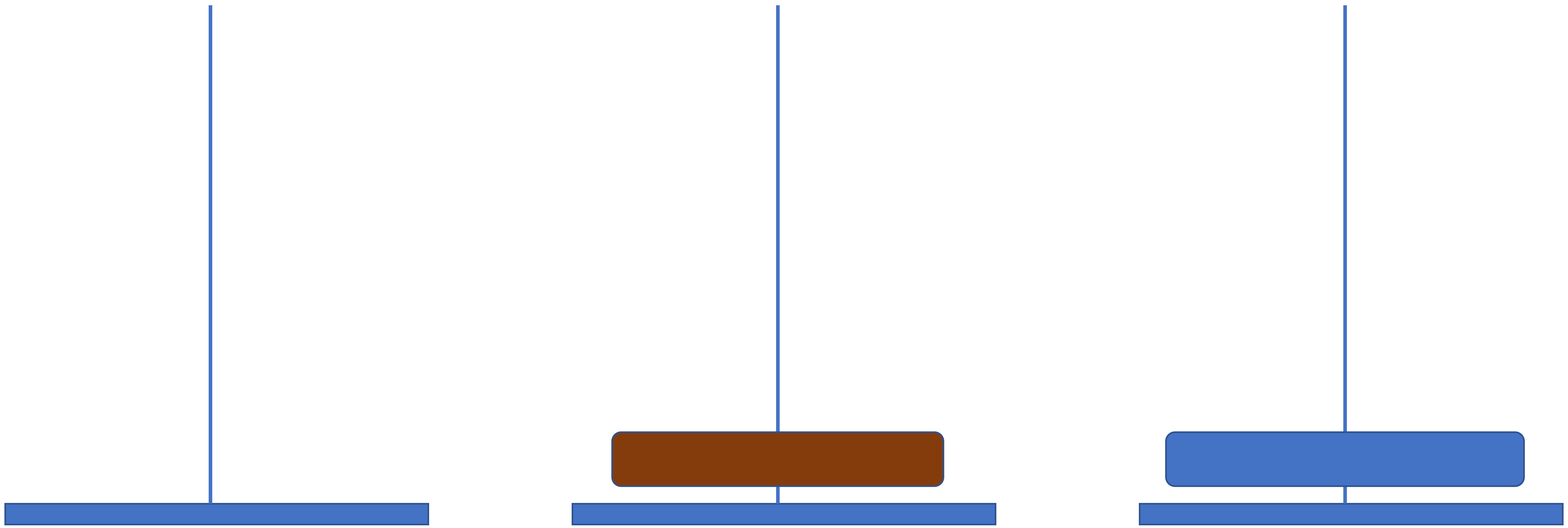


# Towers of Hanoi

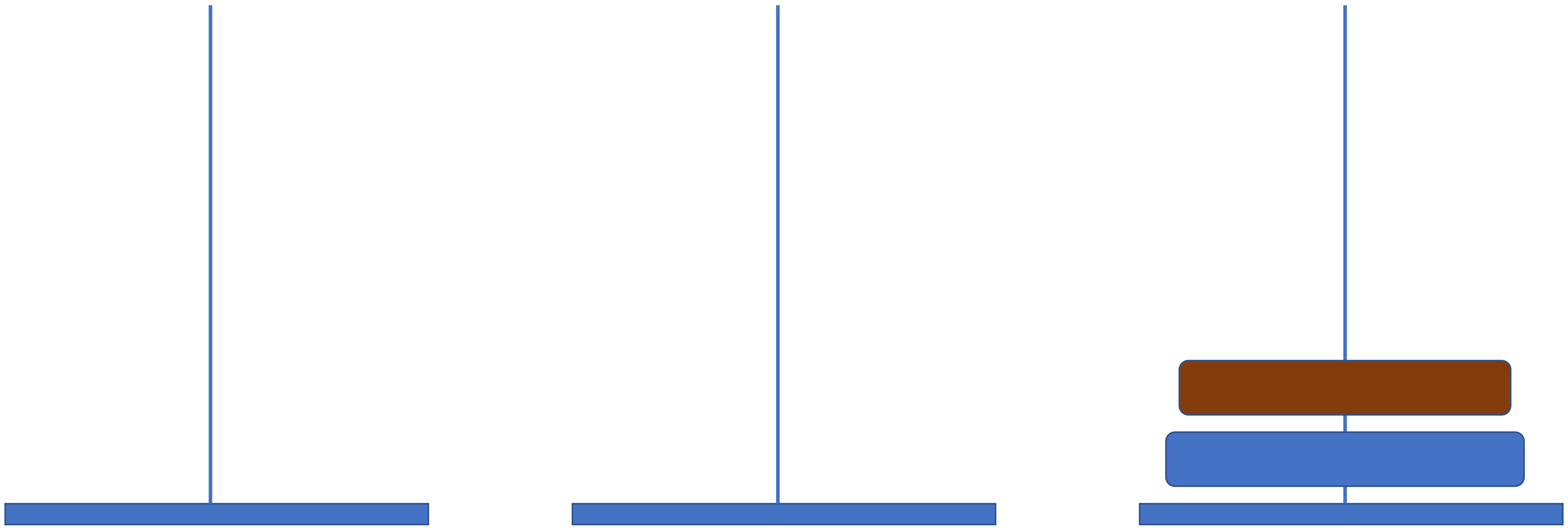
The case of
two discs



# Towers of Hanoi

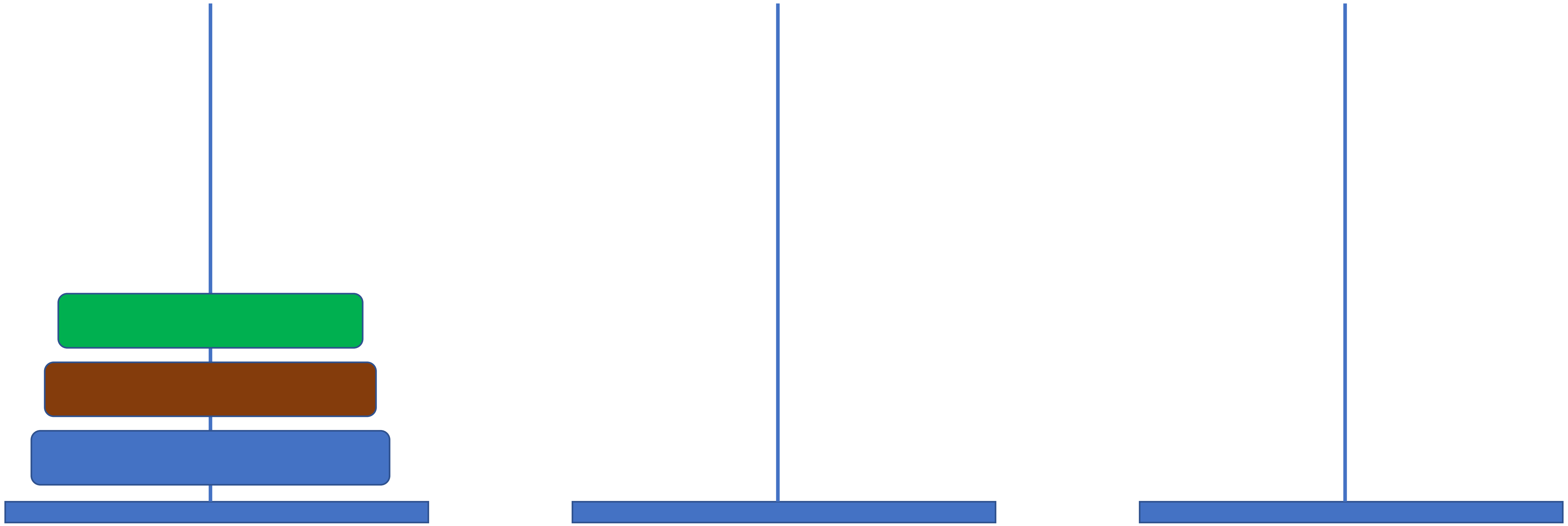


# Towers of Hanoi



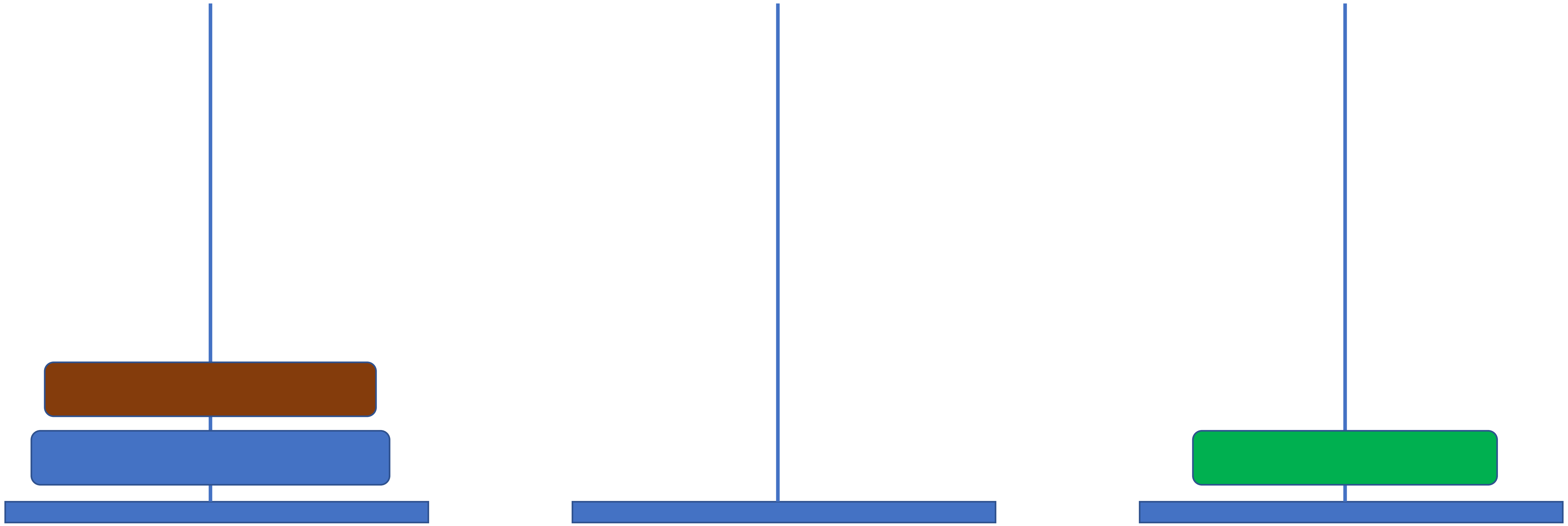
# Towers of Hanoi

The case of three discs



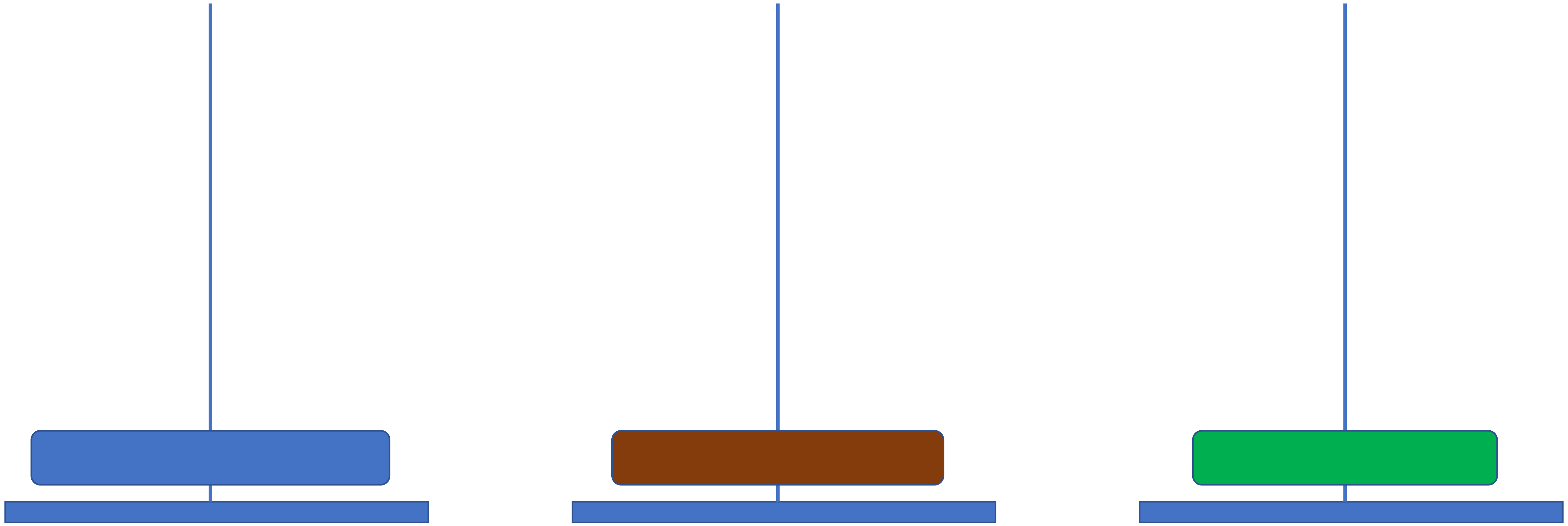
# Towers of Hanoi

The case of three discs



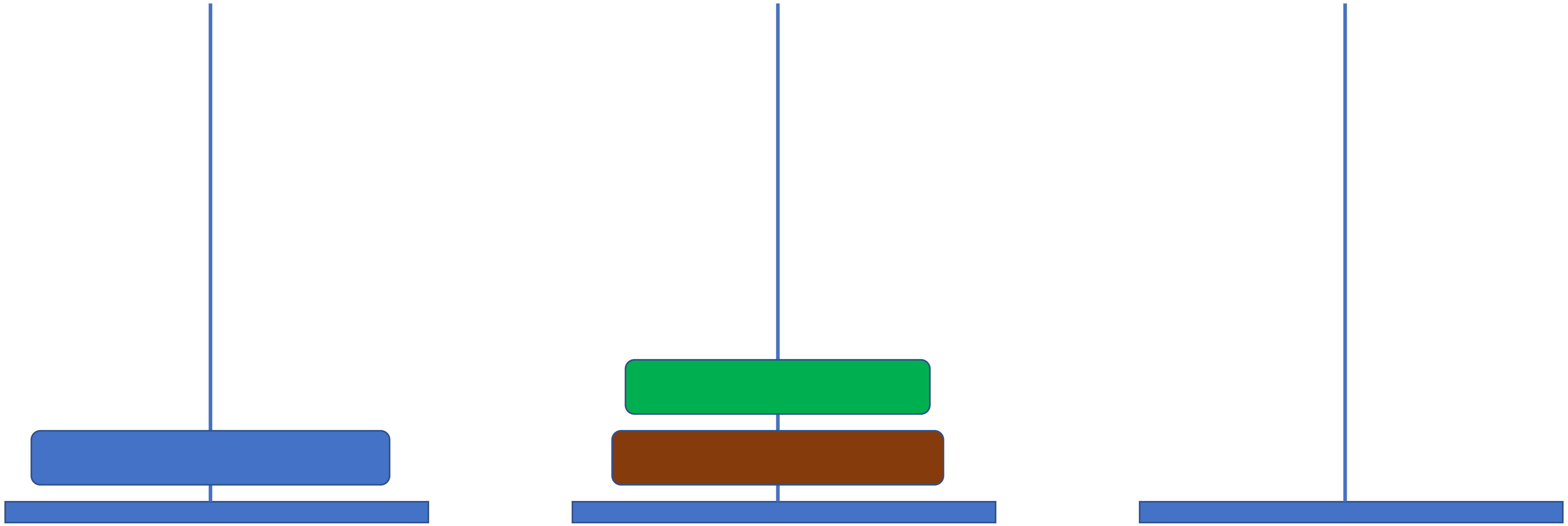
# Towers of Hanoi

The case of
three discs



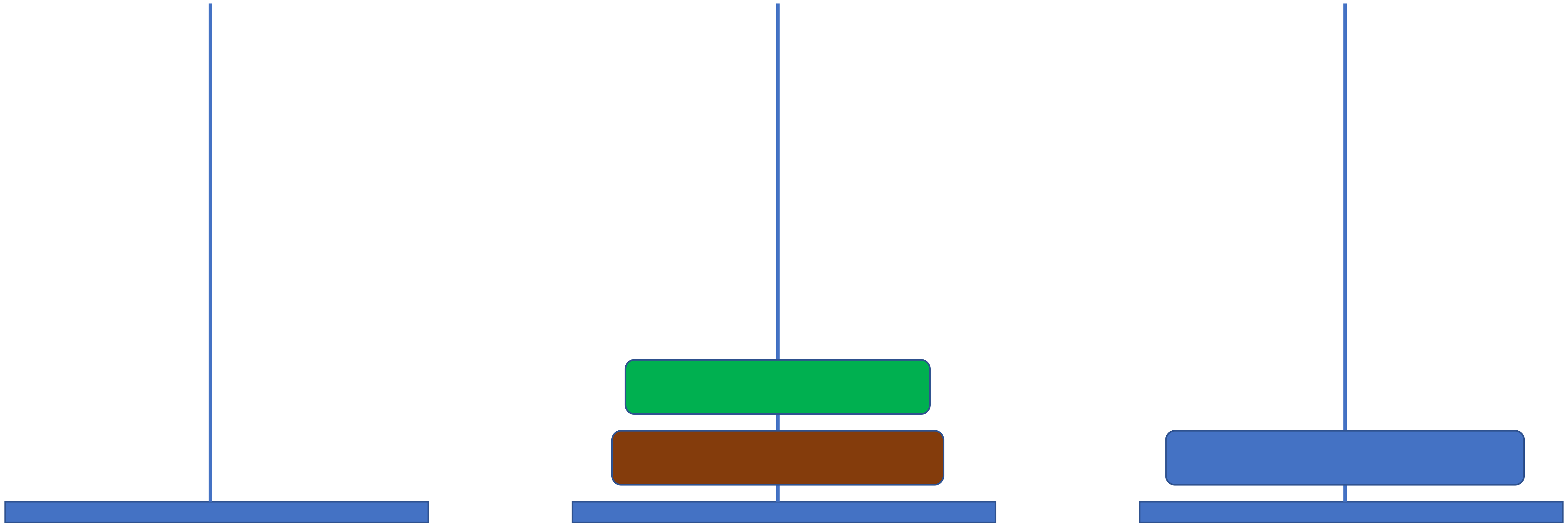
# Towers of Hanoi

The case of three discs



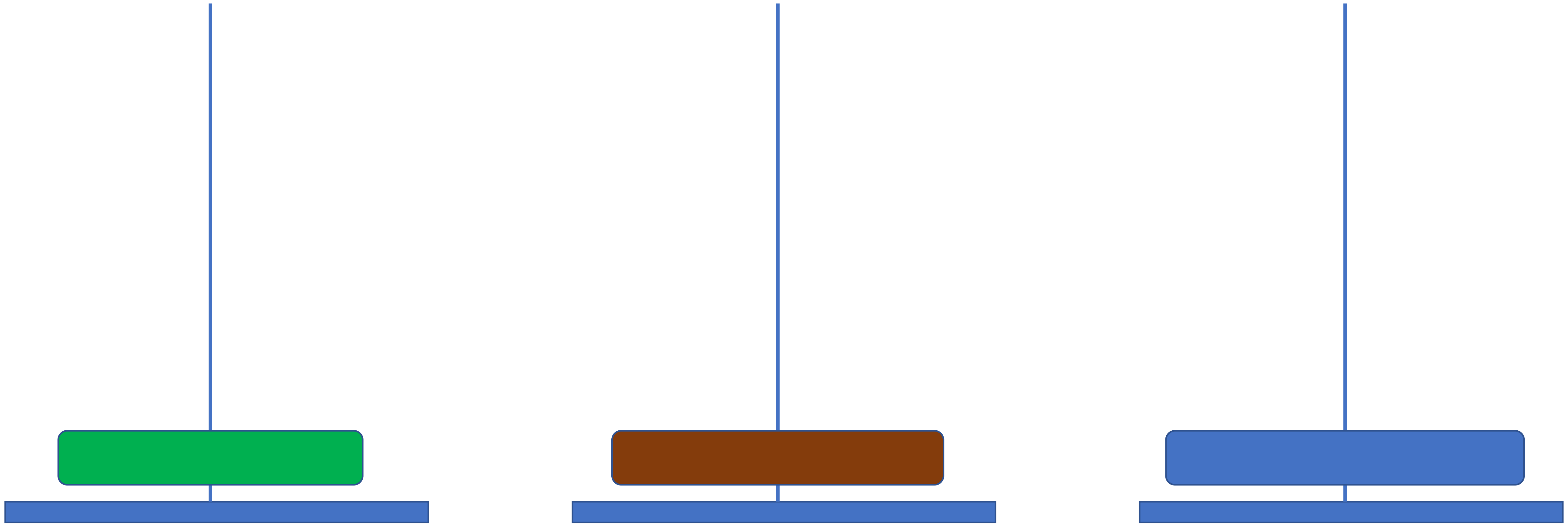
# Towers of Hanoi

The case of three discs



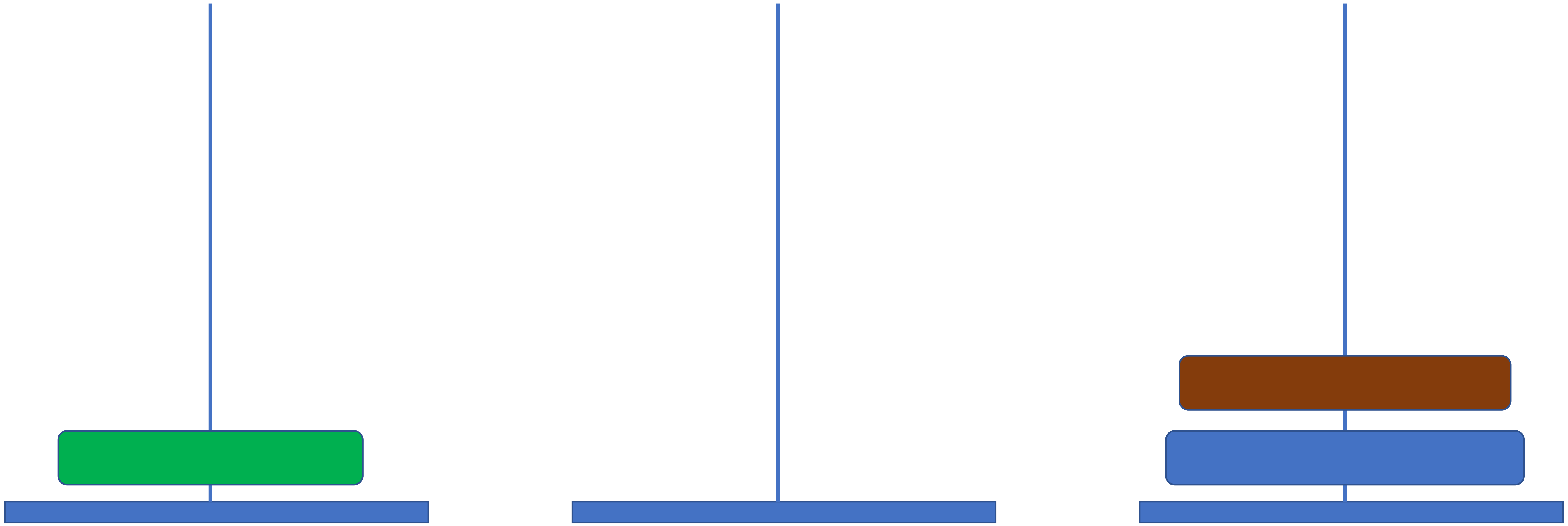
# Towers of Hanoi

The case of three discs



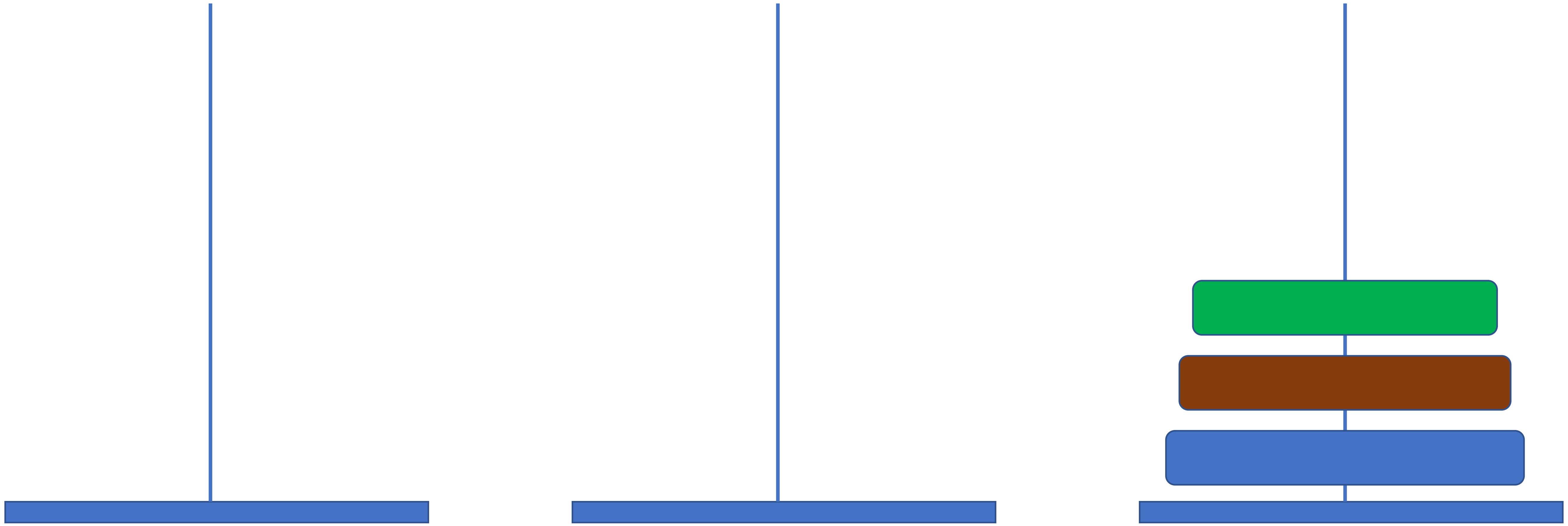
# Towers of Hanoi

The case of three discs



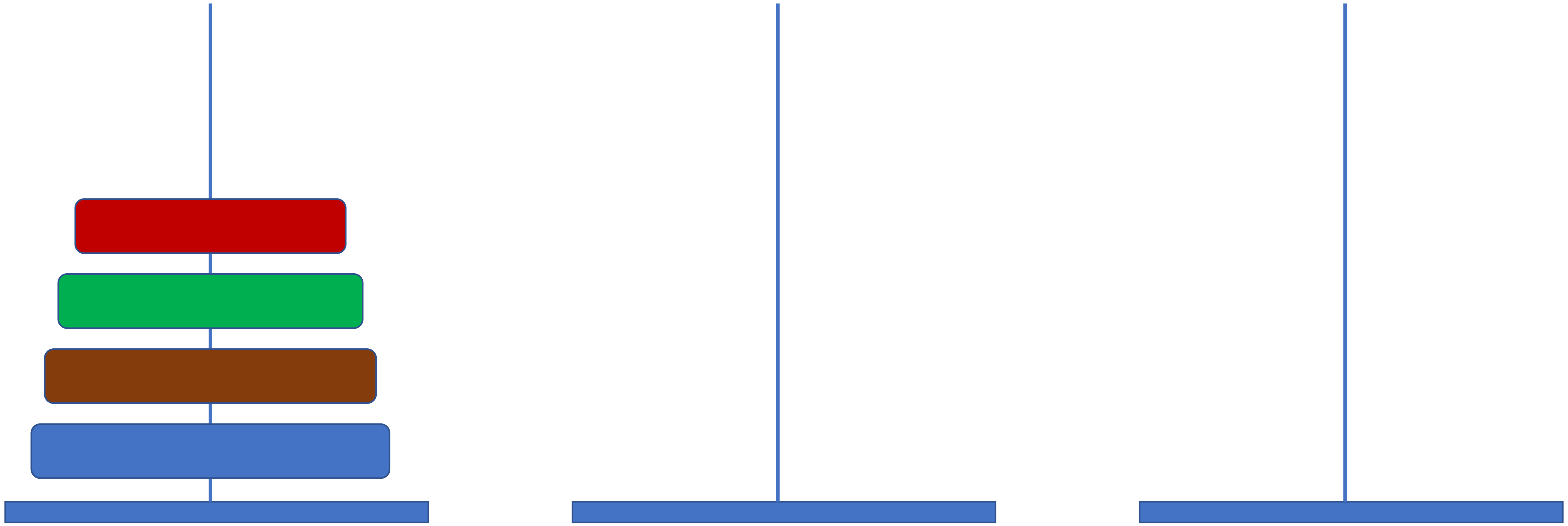
# Towers of Hanoi

The case of three discs



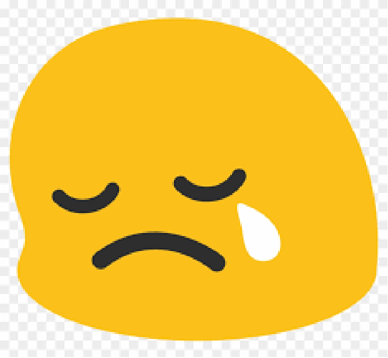
# Towers of Hanoi

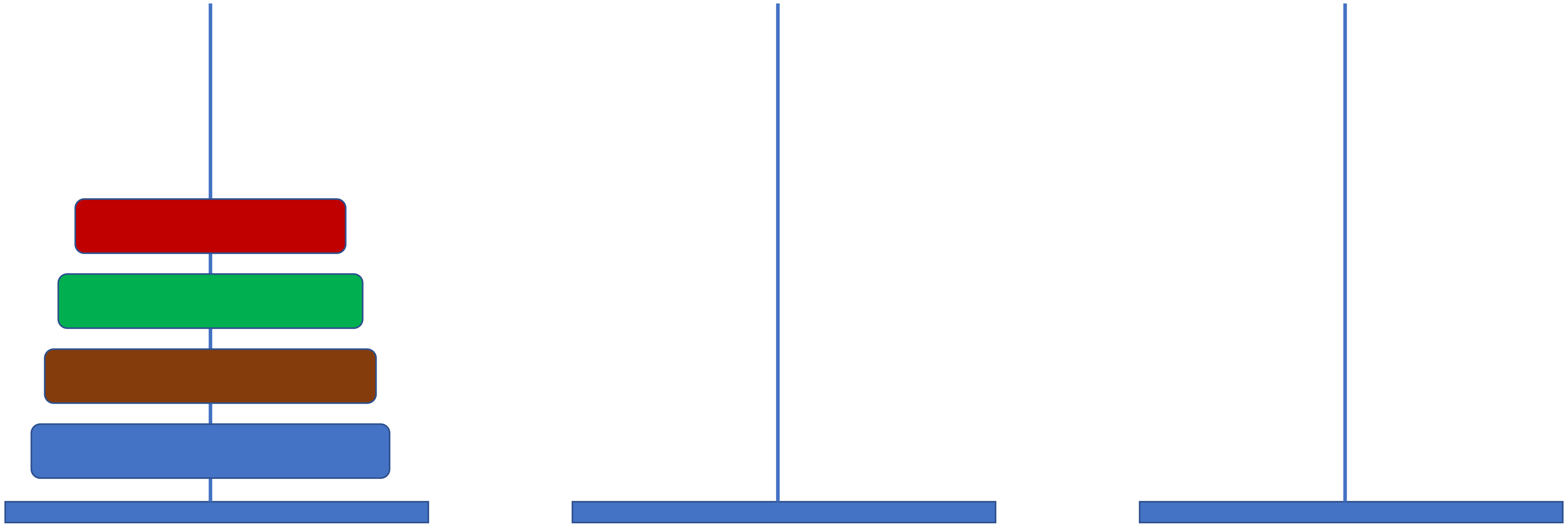
The case of
four discs…?



# Towers of Hanoi

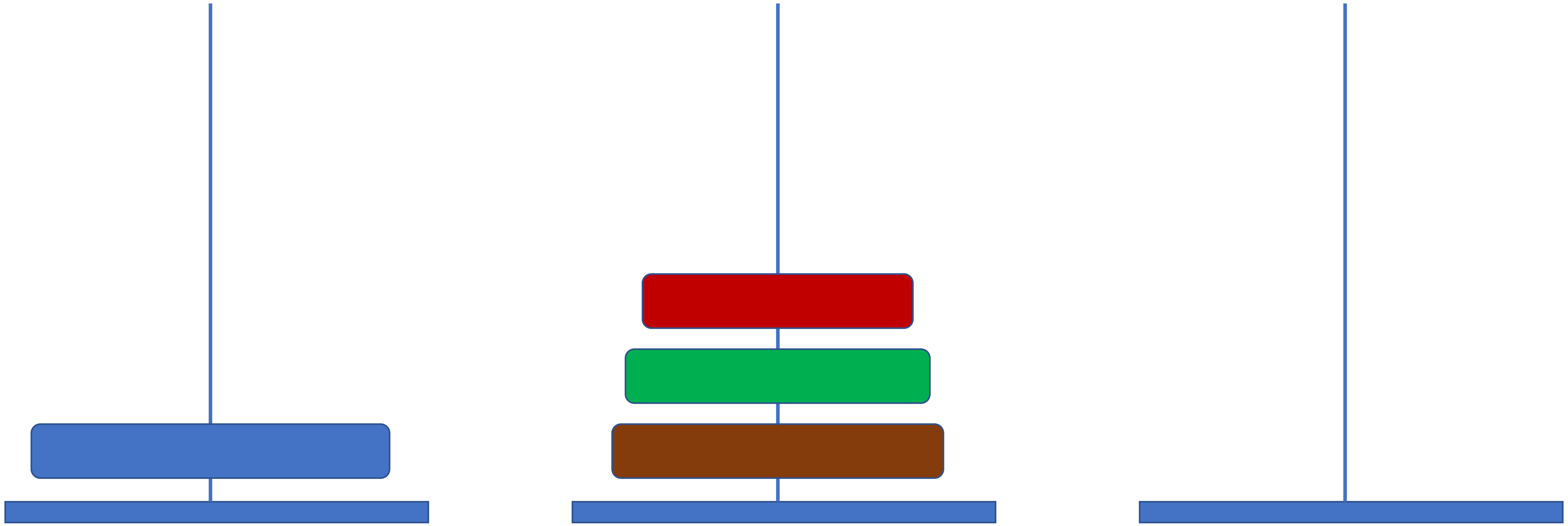
The case of
four discs…?





# Towers of Hanoi

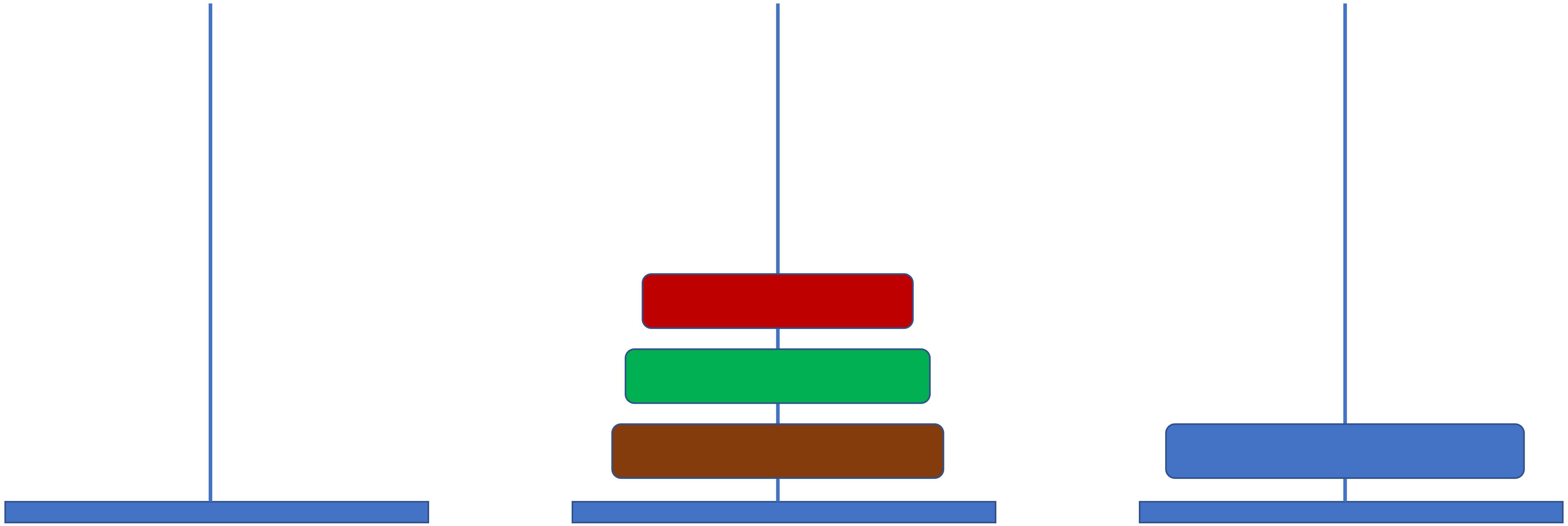
The case of
four discs…?



Move three
to the middle

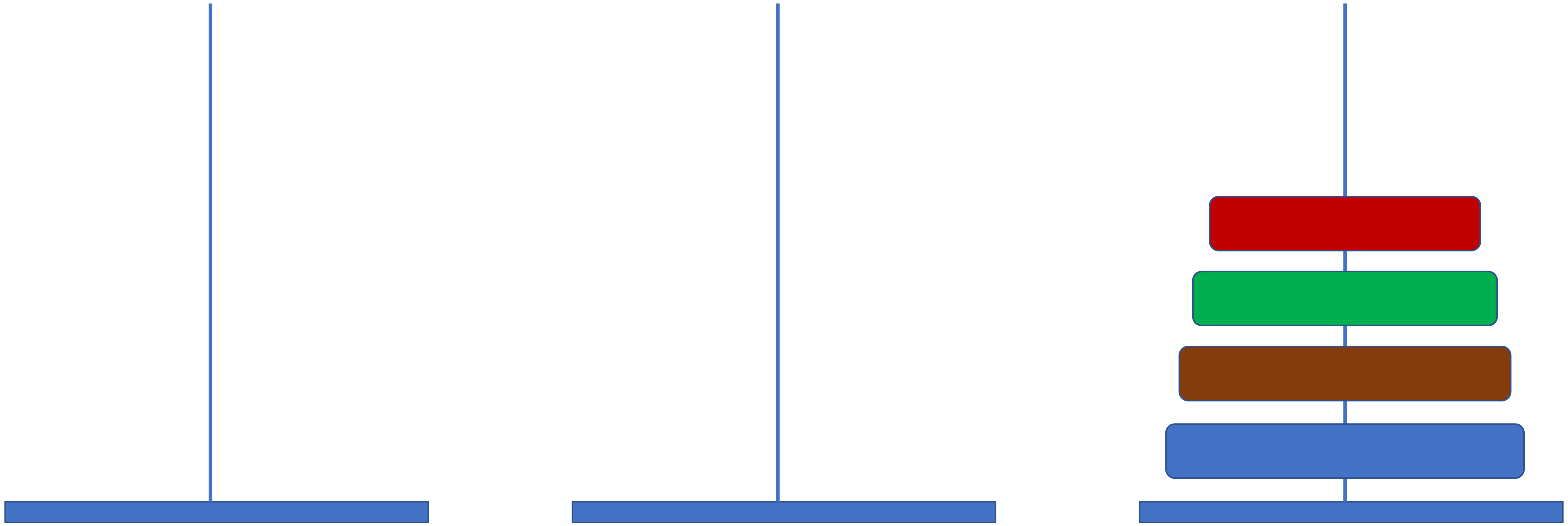
# Towers of Hanoi

The case of four discs…?



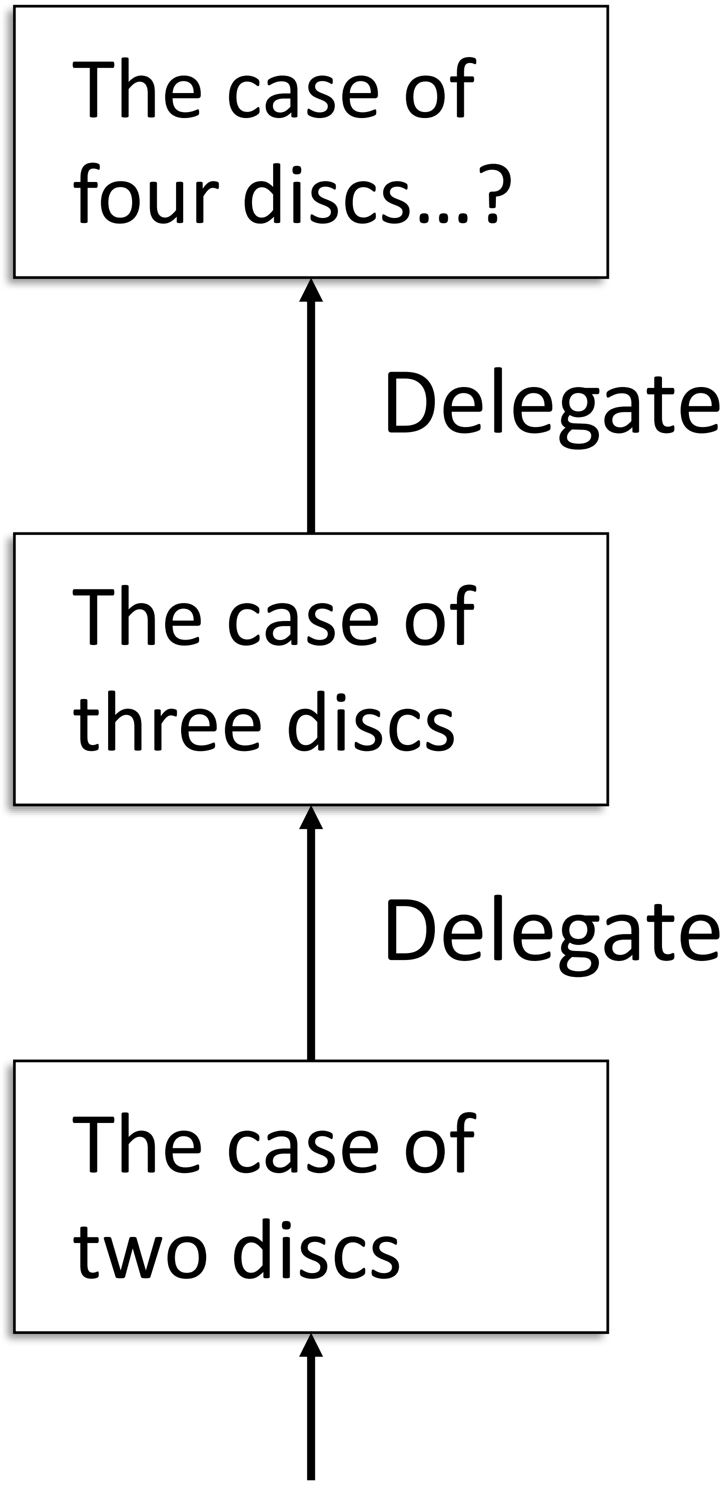
# Towers of Hanoi

The case of four discs…?

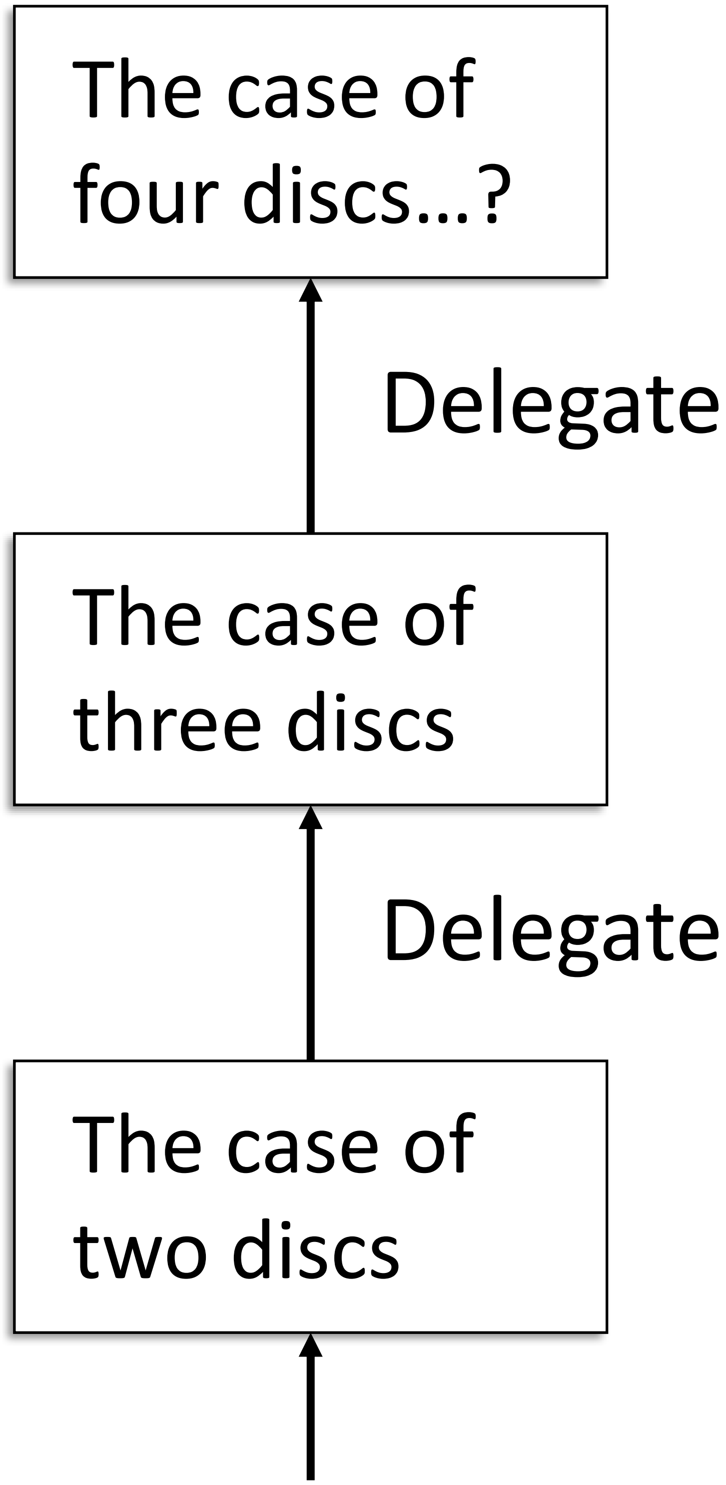


Move three to the right

# Towers of Hanoi – the Pattern



# Towers of Hanoi – the Pattern




**Hanoi(4):**
**1. Hanoi(3, middle)**
**2. Move(left, right)**
**3. Hanoi(3, right)**

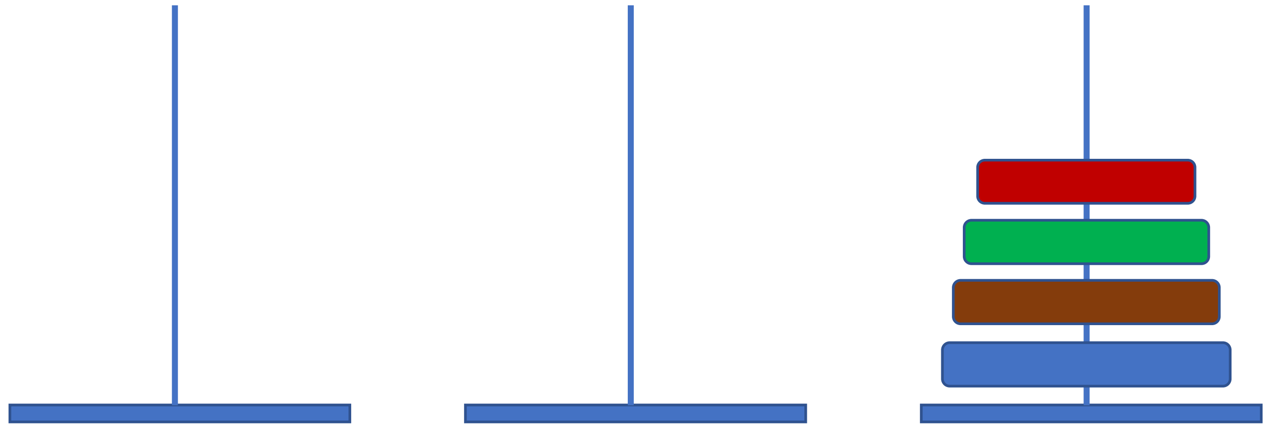
**Hanoi(3):**
**1. Hanoi(2, middle)**
**2. Move(left, right)**
**3. Hanoi(2, right)**

# Towers of Hanoi – the Pattern

**Towers(n, src, dest, tmp)**
If($n = 1$)
    Move(src, dest)
Else
    Towers($n-1$, src, tmp, dest)
    Move(src, dest)
    Towers($n-1$, tmp, dest, src)

Move all but largest out of the way

Move the rest



# Towers of Hanoi – the Pattern

**Towers(n, src, dest, tmp)**
If($n\ = \ 1$)
    Move(src, dest)
Else
    Towers($n - 1$, src, tmp, dest)
    Move(src, dest)
    Towers($n - 1$, tmp, dest, src)

**Correctness:**
Clearly, the algorithm works correctly if $n = 1$.

In the other cases, it works correctly if the case of $n - 1$ is correct.

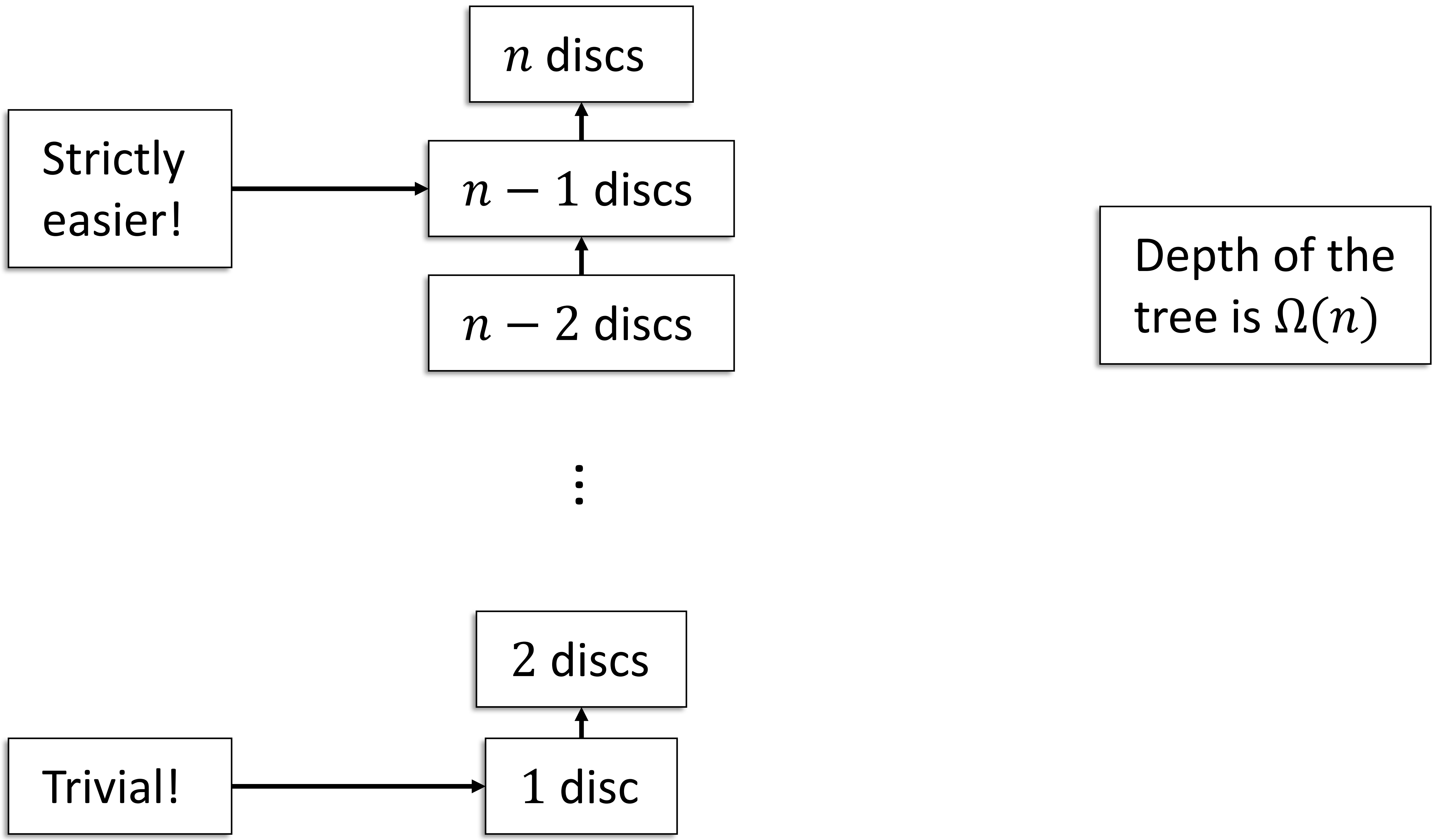
Induction!

# Towers of Hanoi – the Pattern

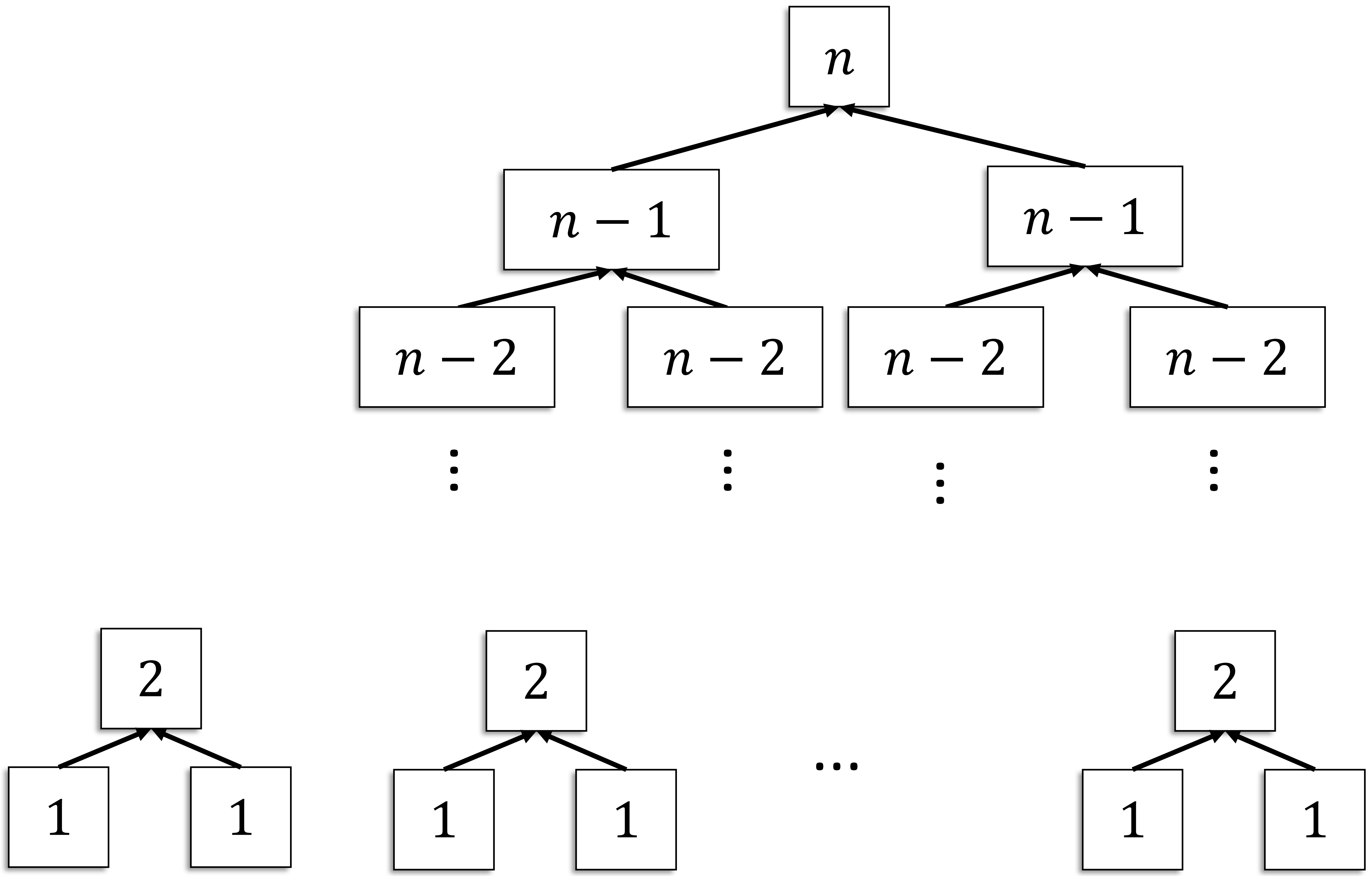
**Runtime:**
Two recursive calls in each recursion ”level” and $n - 1$ recursion levels.

Think of the recursion tree.

# Towers of Hanoi – Recursion Tree



# Towers of Hanoi – Recursion Tree




Two recursive calls to the lower level

# Towers of Hanoi – the Pattern

**Runtime:**
Denote cost for $k$ discs by $T(k)$.
We get the recurrence

$$T(k) = 2 \cdot T(k-1) + 1$$

Call black box twice

# Towers of Hanoi – the Pattern

**Runtime:**
Denote cost for $k$ discs by $T(k)$.
We get the recurrence
$$T(k) = 2 \cdot T(k-1) + 1$$

Solve recurrence:
$$T(n) = 2^{n-1} \cdot T(1) - 1$$
$$= O(2^n)$$

Call black box twice

# Towers of Hanoi – Recurrence

**Base case:**

$$T(1) = 1$$

# Towers of Hanoi – Recurrence

**Base case:**
$$T(1) = 1$$

**Inductive hypothesis:**
$$T(i) \leq 2^i - 1$$
**Recurrence:**
$$T(k) = 2 \cdot T(k-1) + 1$$

# Towers of Hanoi – Recurrence

**Base case:**

$$T(1) = 1$$

**Inductive hypothesis:**

$$T(i) \leq 2^i - 1$$

**Recurrence:**

$$T(k) = 2 \cdot T(k-1) + 1$$

**Inductive step:**

$T(i+1) = 2 \cdot T(i) + 1$

$\leq 2 \cdot \left(2^i - 1\right) + 1 = 2^{i+1} - 2 + 1$

$= 2^{i+1} - 1$

# Towers of Hanoi – Recurrence

**Base case:**

$$T(1) = 1$$

**Inductive hypothesis:**

$$T(i) \leq 2^i - 1$$

**Recurrence:**

$$T(k) = 2 \cdot T(k-1) + 1$$

**Inductive step:**

$T(i+1) = 2 \cdot T(i) + 1$

$\leq 2 \cdot \left(2^i - 1\right) + 1 = 2^{i+1} - 2 + 1$

$= 2^{i+1} - 1$

# Towers of Hanoi – Recurrence

**Base case:**

$$T(1) = 1$$

**Inductive hypothesis:**

$$T(i) \leq 2^i - 1$$

**Recurrence:**

$$T(k) = 2 \cdot T(k-1) + 1$$

**Inductive step:**

$T(i+1) = 2 \cdot T(i) + 1$

$\leq 2 \cdot \left(2^i - 1\right) + 1 = 2^{i+1} - 2 + 1$

$= 2^{i+1} - 1$

# Towers of Hanoi – Recurrence

**Base case:**

$$T(1) = 1$$

**Inductive hypothesis:**

$$T(i) \leq 2^i - 1$$

**Recurrence:**

$$T(k) = 2 \cdot T(k-1) + 1$$

**Inductive step:**

$$T(i+1) = 2 \cdot T(i) + 1$$
$$\leq 2 \cdot \left(2^i - 1\right) + 1 = 2^{i+1} - 2 + 1$$
$$= 2^{i+1} - 1$$

**By induction:**

$$T(n) = O(2^n)$$

# Outline

- Recursion trees
- Towers of Hanoi
  - Delegate!
- Mergesort
  - Divide and conquer

# Sorting

**Input:**
An array $A[n]$ of $n$ integers.

**Output:**
An array $B[n]$ such that $B$ contains the entries of $A[n]$ in an ascending order.

# Sorting

**Input:**
An array $A[n]$ of $n$ integers.

| Index | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| Value | 10 | 2 | 5 | 0 | 11 | 65 | 4 | 2 | 9 |

**Output:**
An array $B[n]$ such that $B$ contains the entries of $A[n]$ in an ascending order.

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2 | 4 | 5 | 9 | 10 | 11 | 65 |

# Sorting

**Input:**
An array $A[n]$ of $n$ integers.

| Index | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|---|
| Value | 10 | 2 | 5 | 0 | 11 | 65 | 4 | 2 | 9 |

**Output:**
An array $B[n]$ such that $B$ contains the entries of $A[n]$ in an ascending order.

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2 | 4 | 5 | 9 | 10 | 11 | 65 |

For any $0 \leq i < n$ and $i < j < n$, it holds that $B[i] \leq B[j]$

# Mergesort

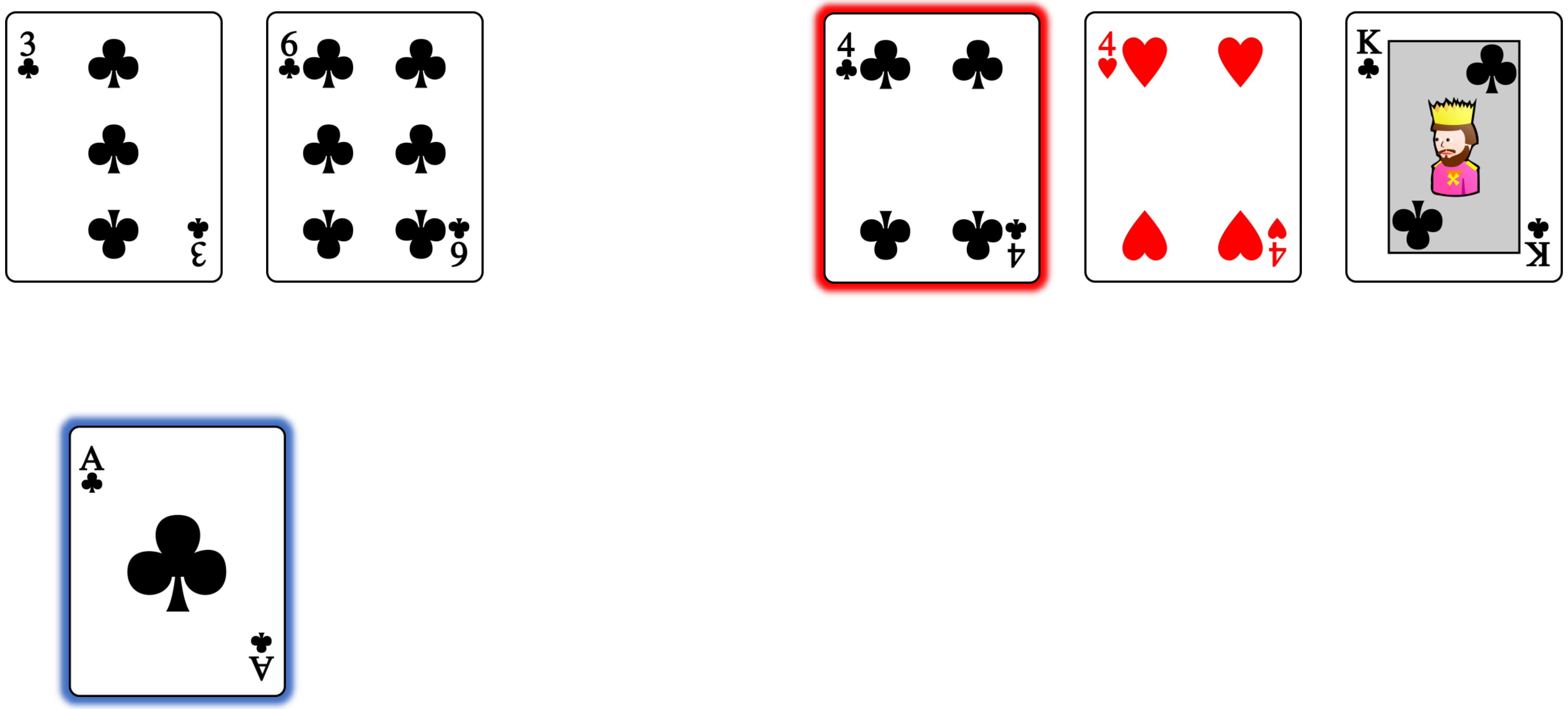
**Idea:**
Combining two sorted arrays is easy.

# Mergesort

**Idea:**
Combining two sorted arrays is easy.

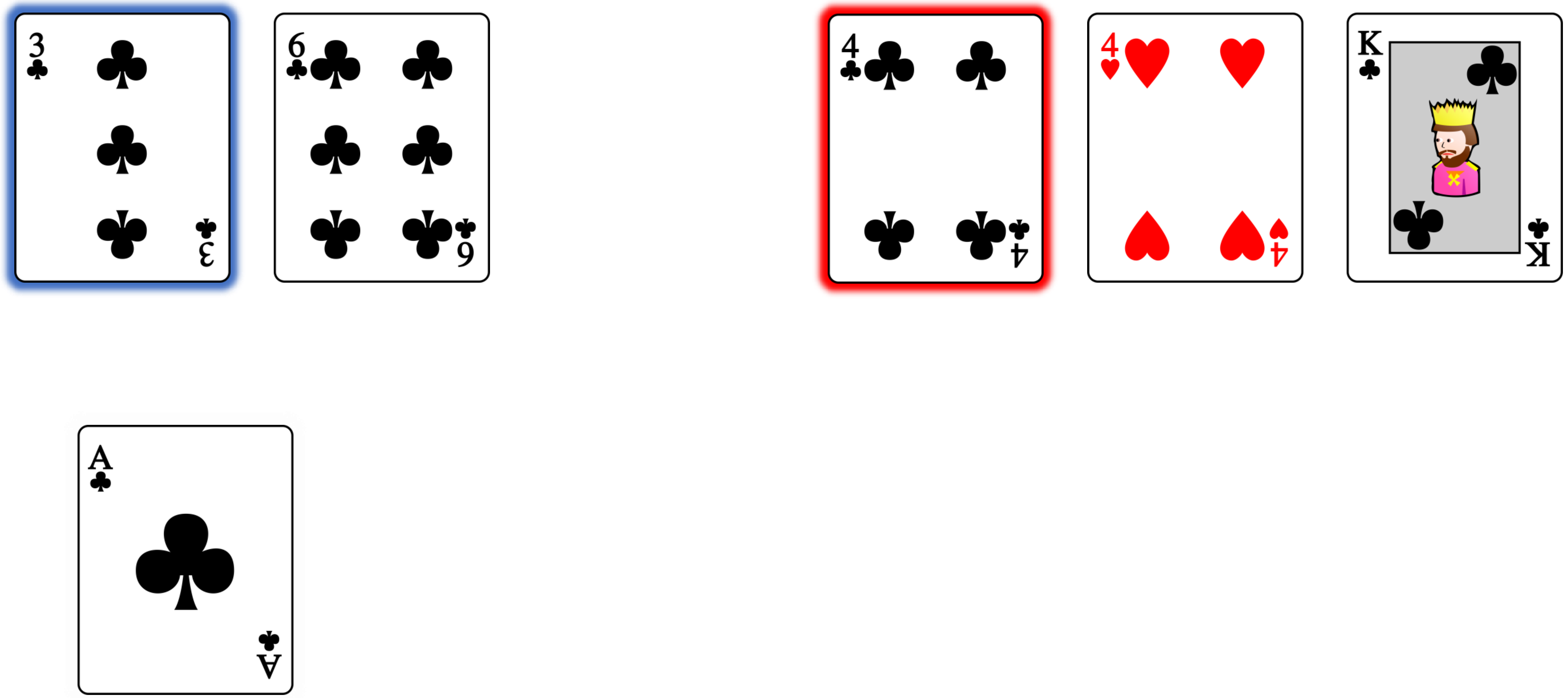
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



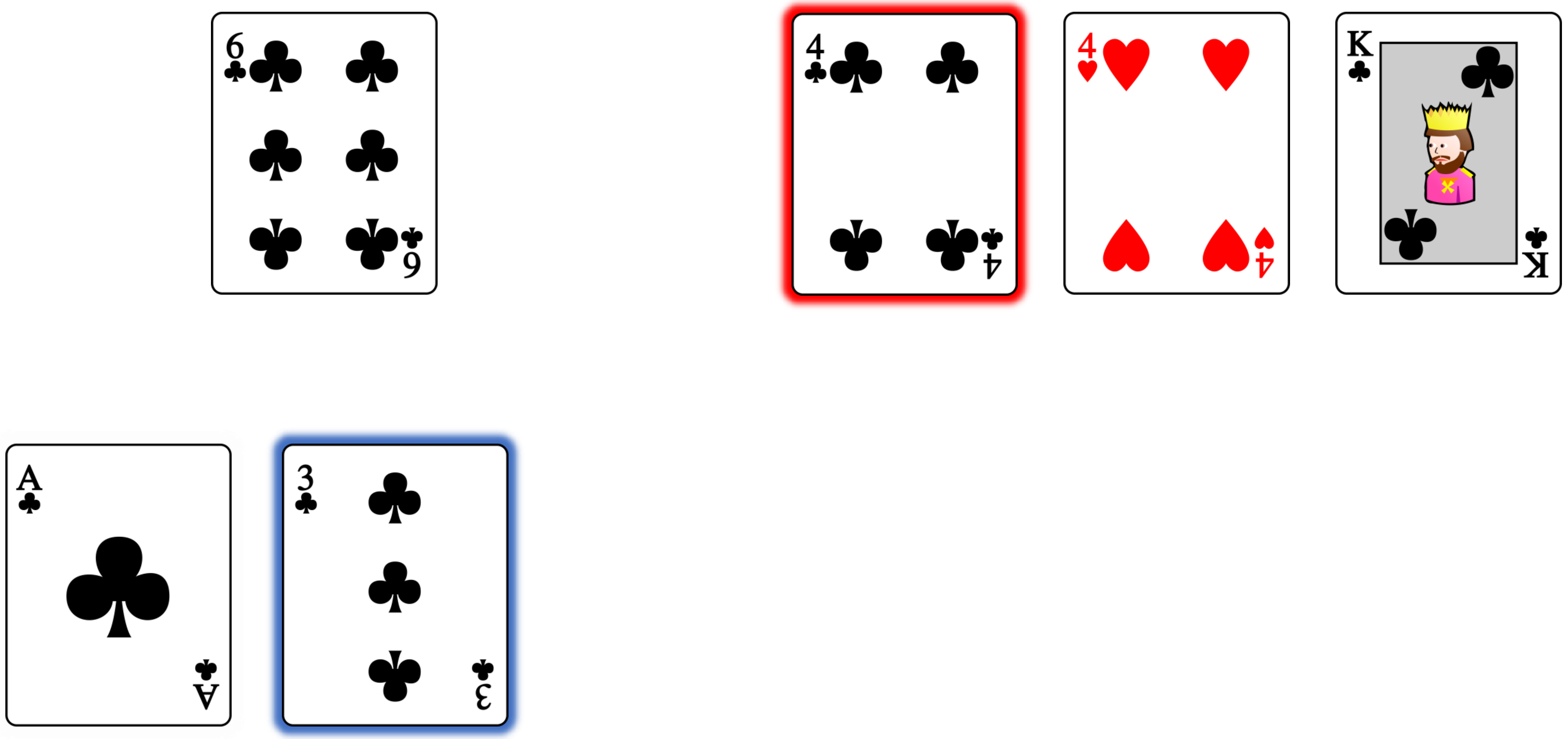
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



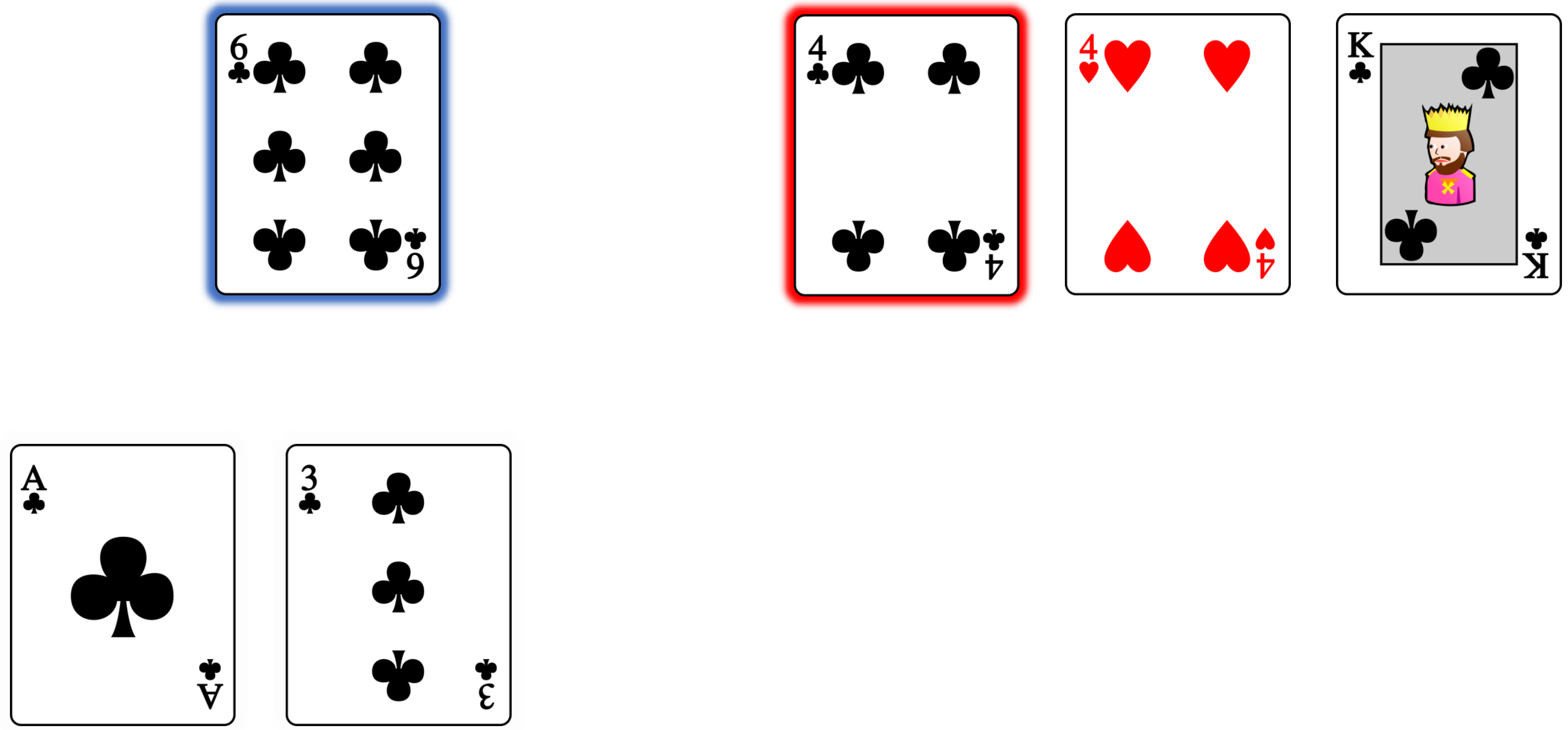
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



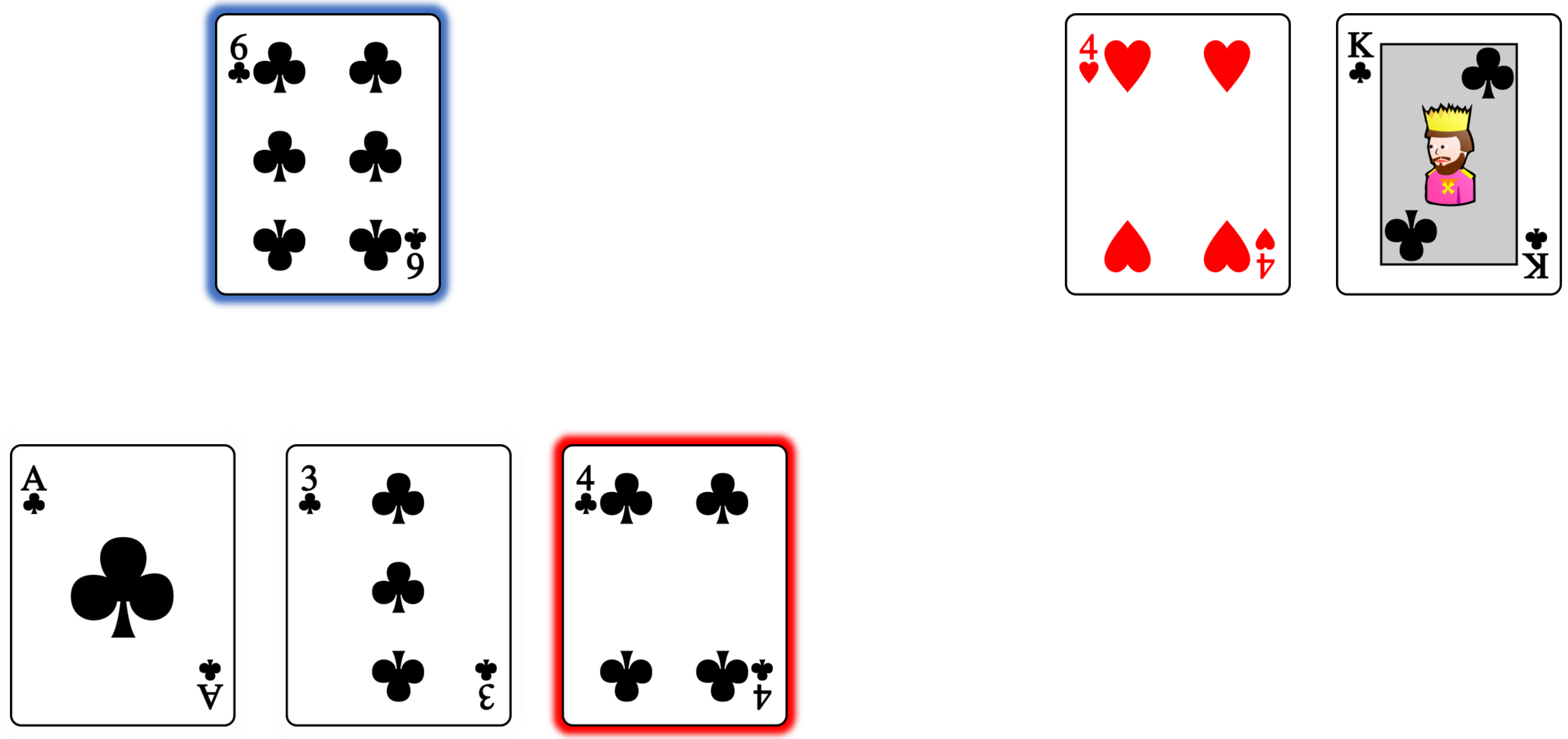
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



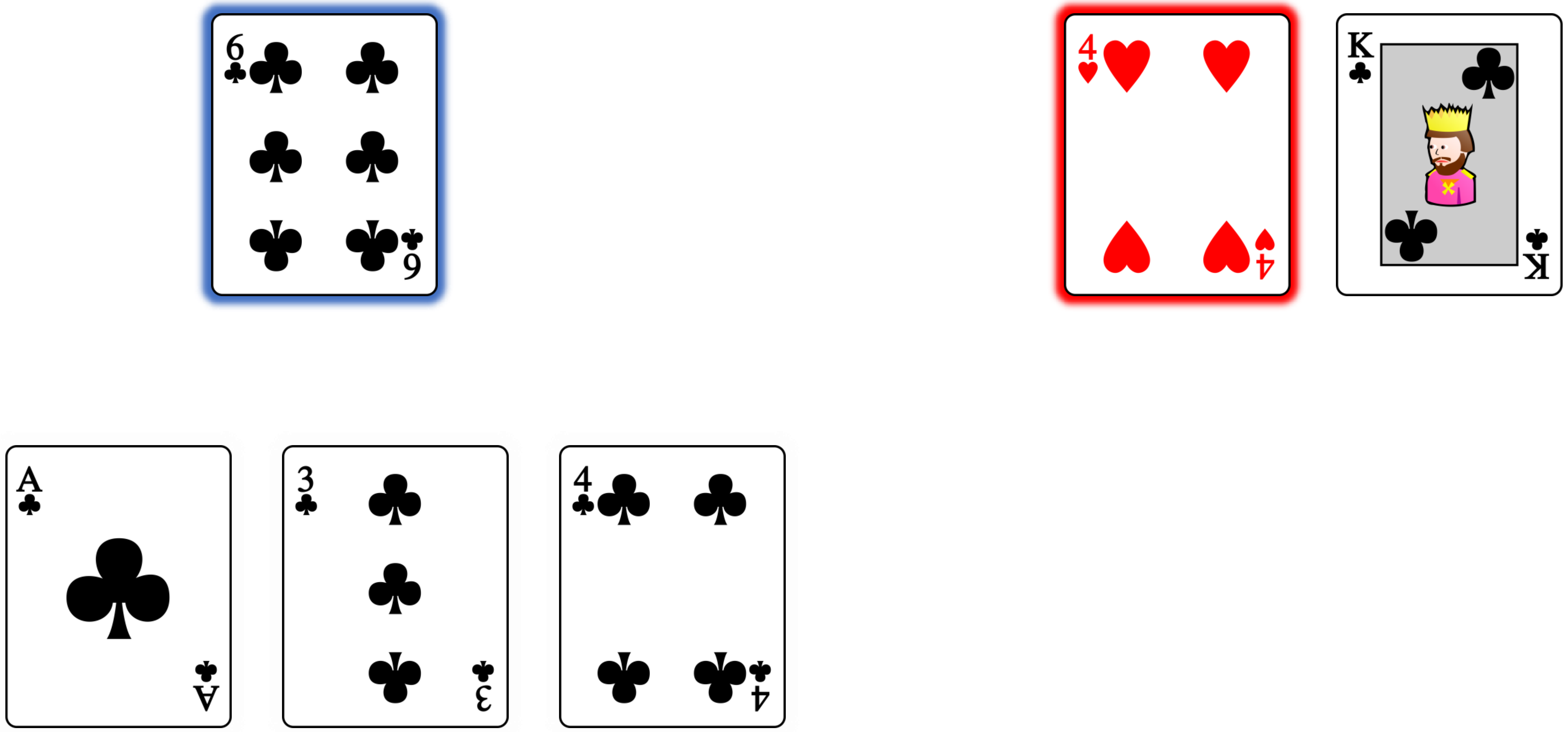
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



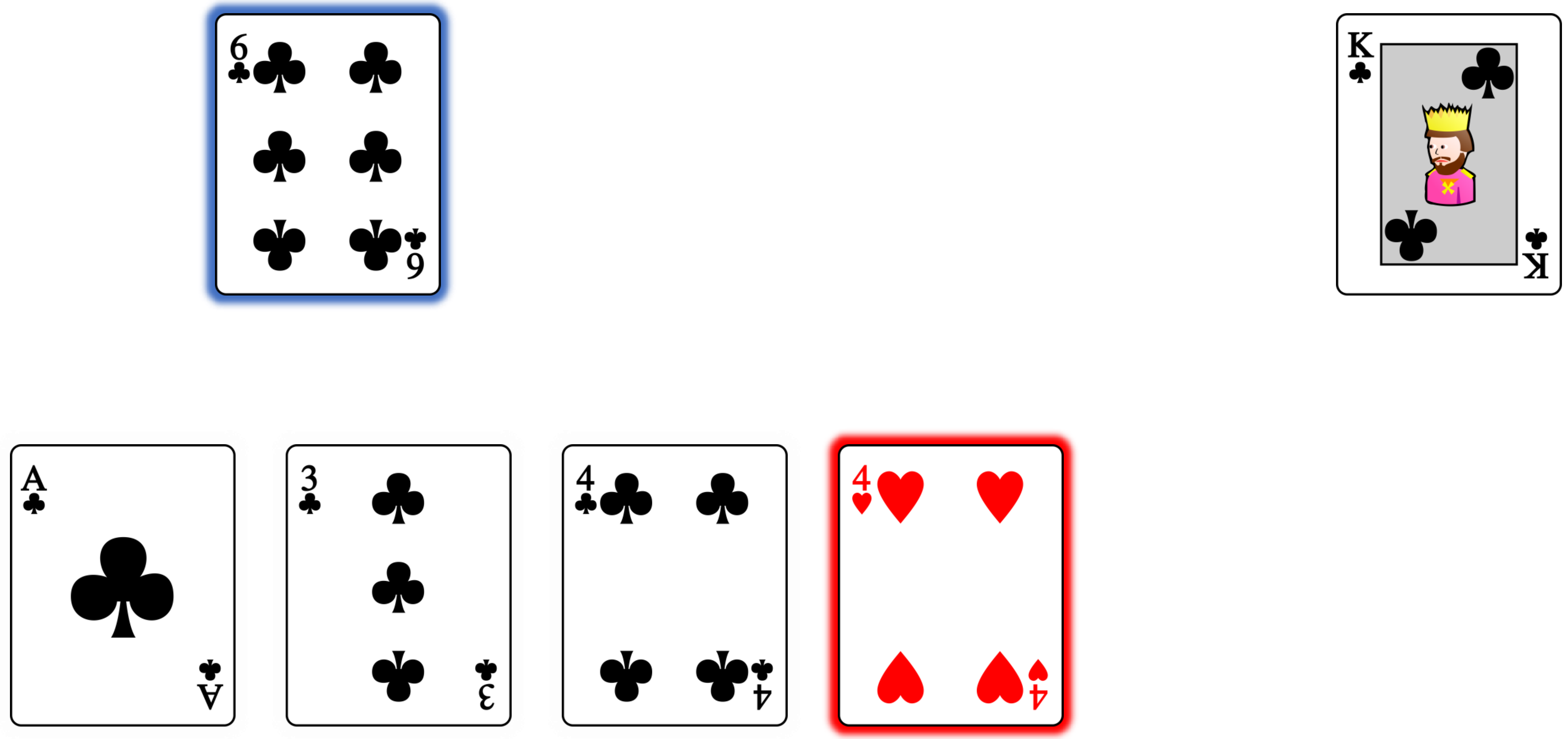
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



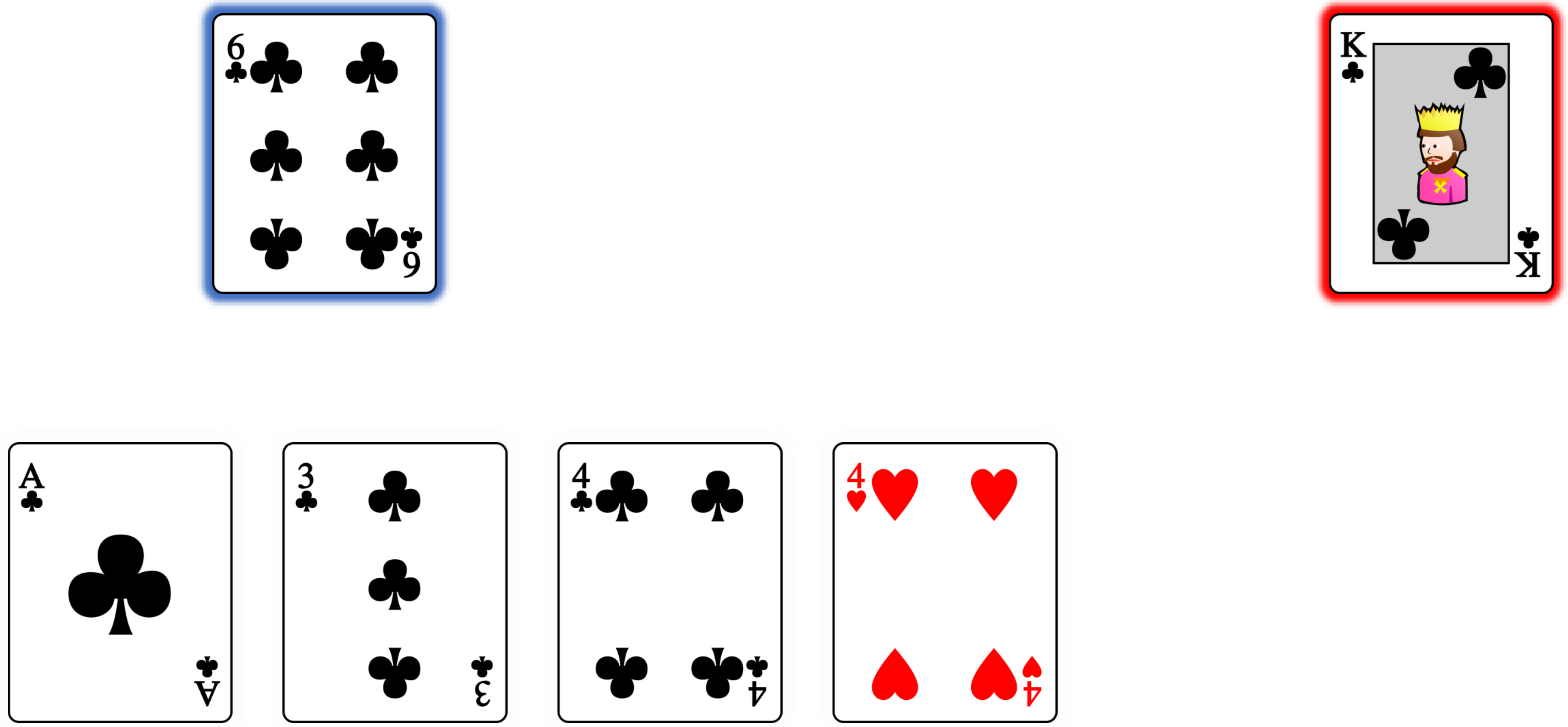
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



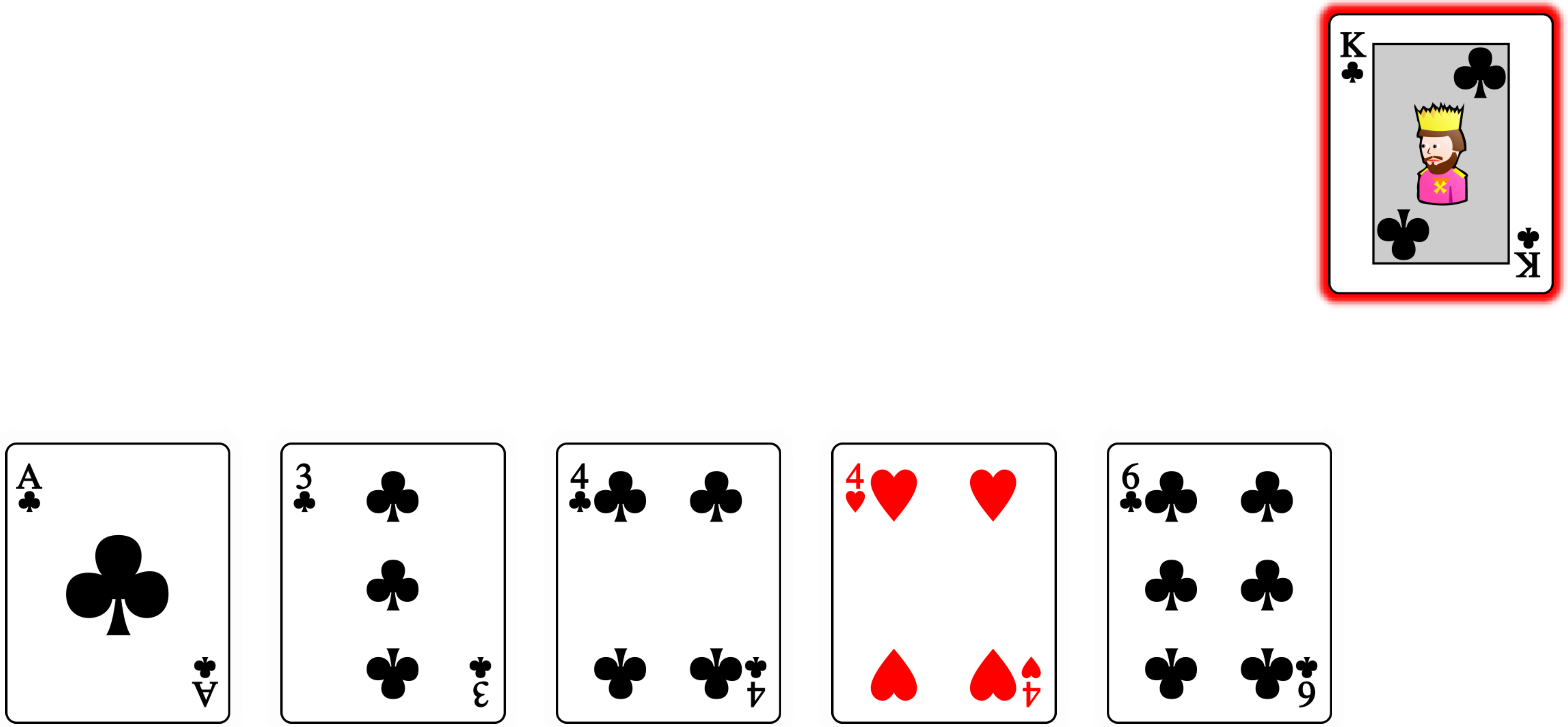
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



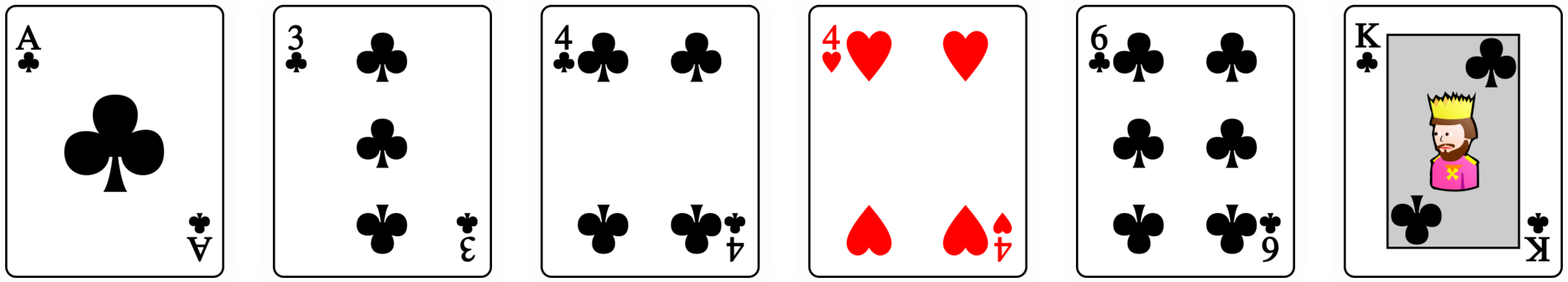
# Mergesort

**Idea:**
Combining two sorted arrays is easy.



# Mergesort

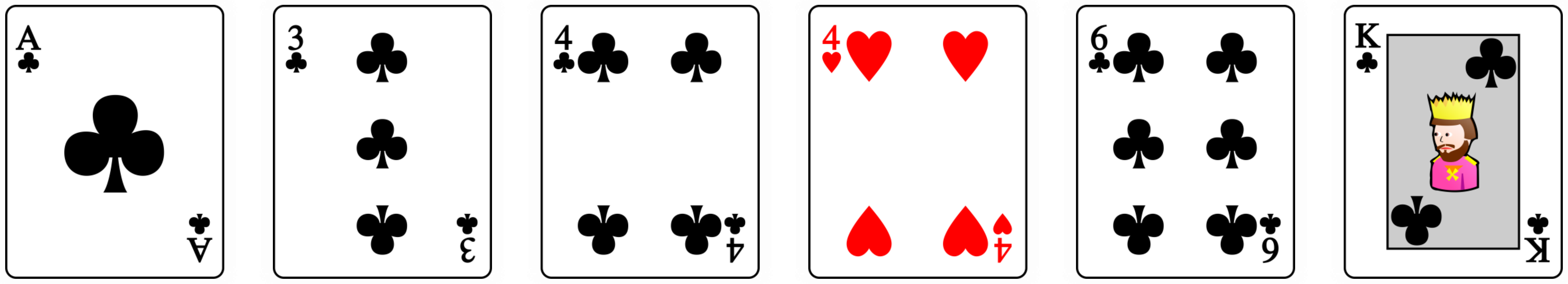
**Idea:**
Combining two sorted arrays is easy.



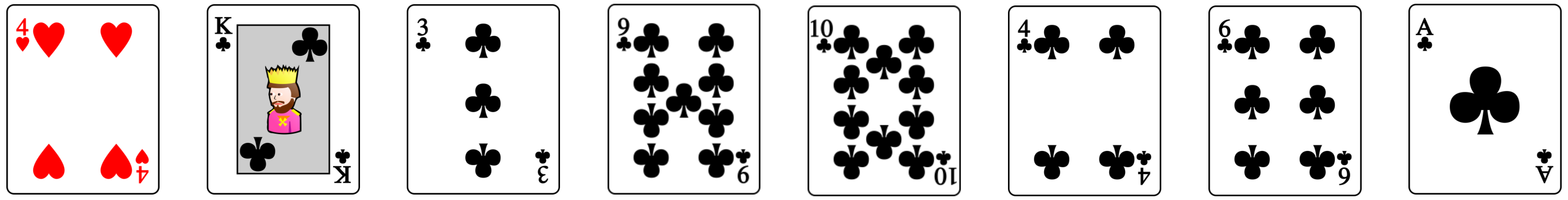
# Mergesort

**Idea:**
Combining two sorted arrays is easy.

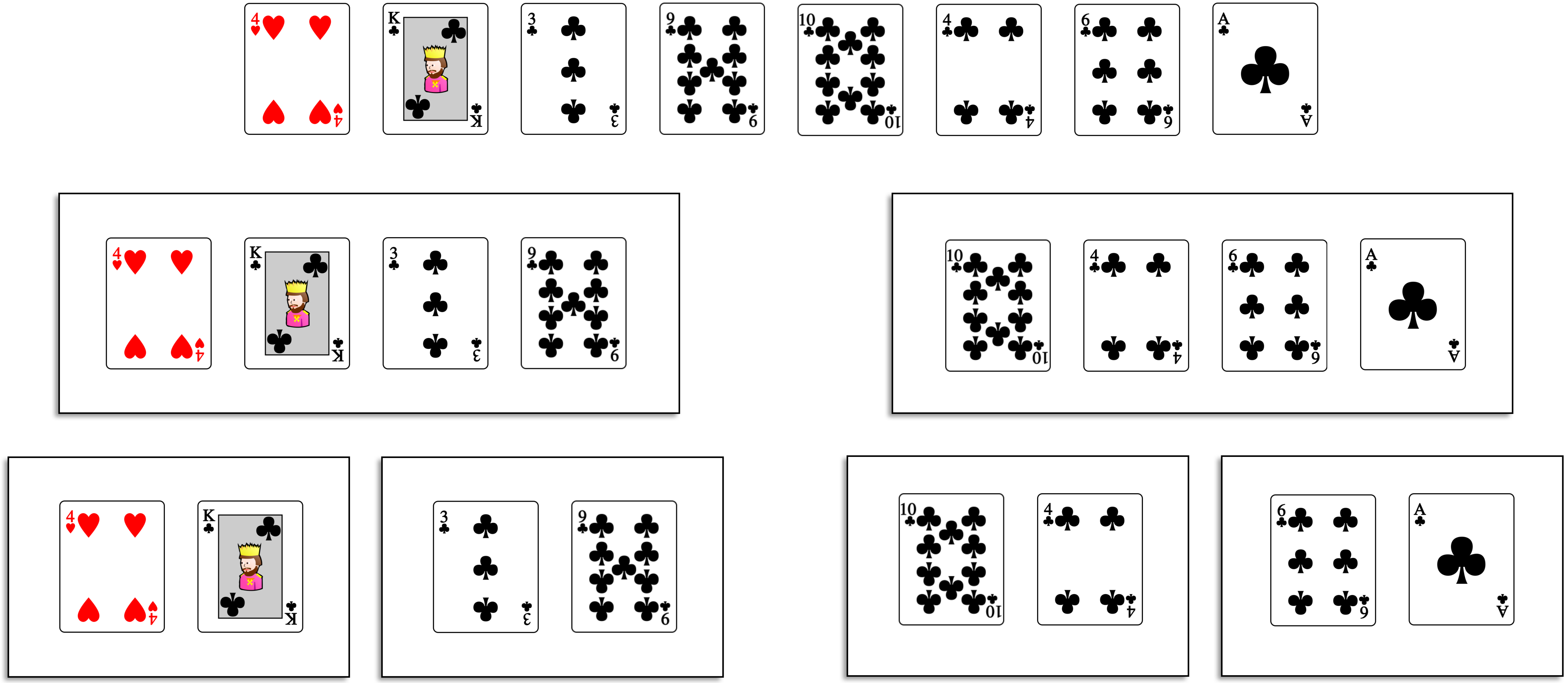
**Runtime:**
Linear in the length of the longer array



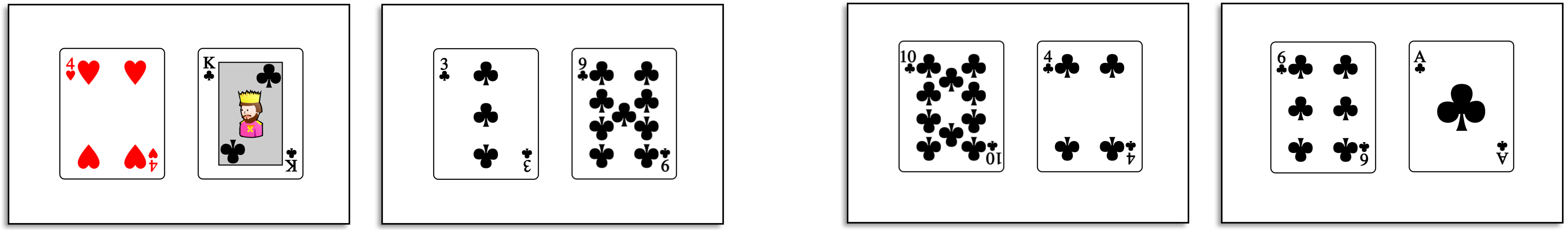
# Mergesort



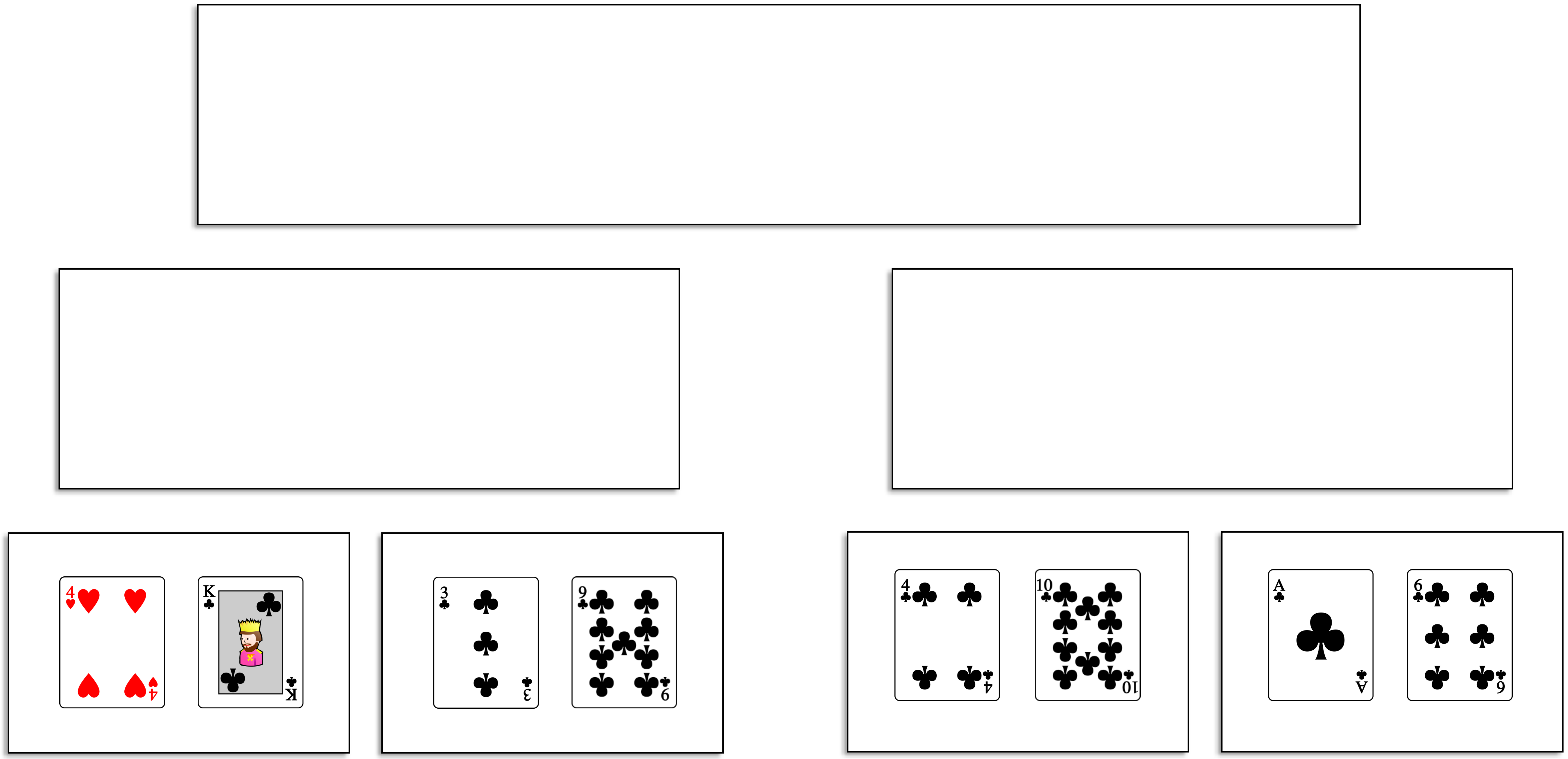
# Mergesort



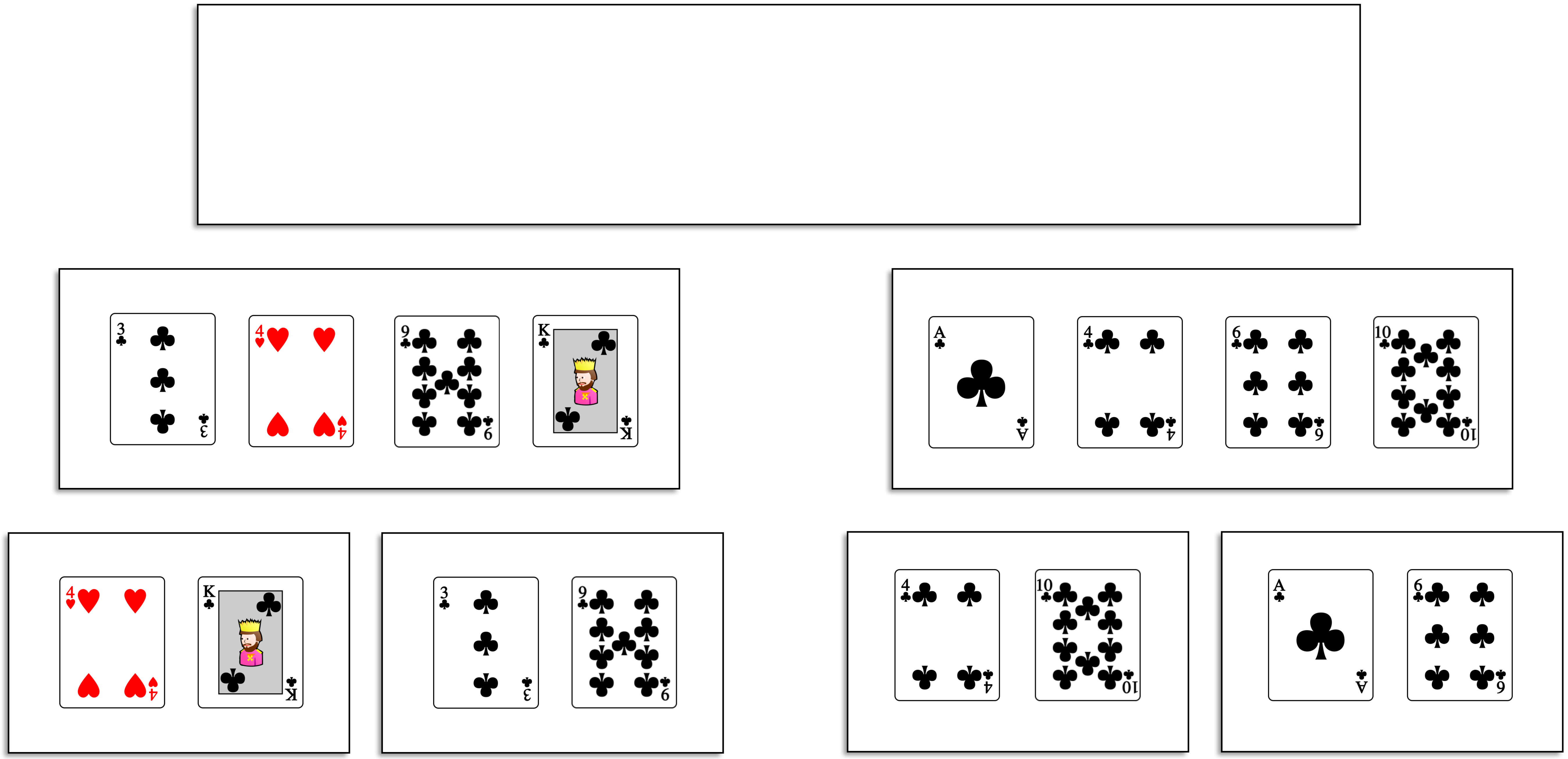
# Mergesort



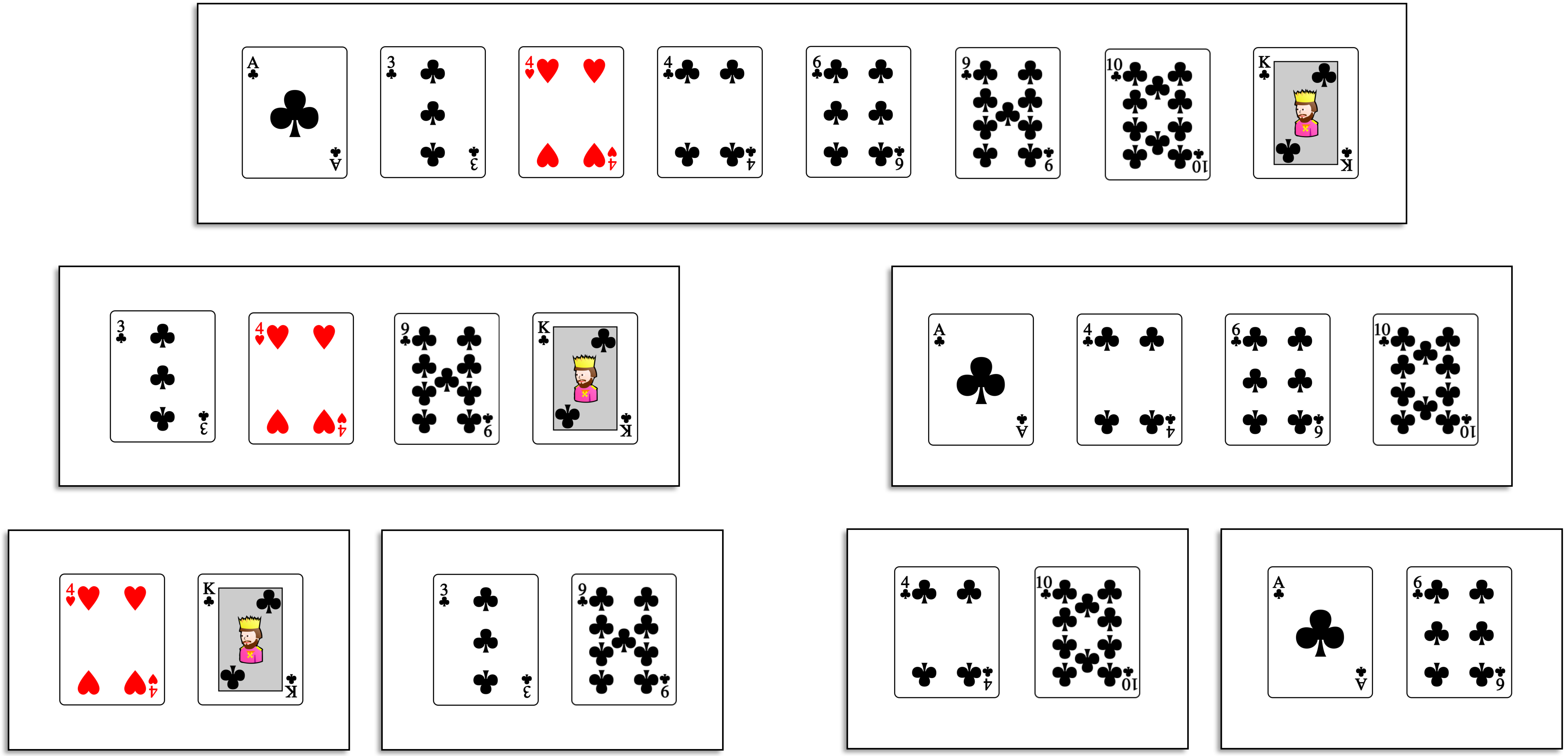
# Mergesort



# Mergesort



# Mergesort



# Mergesort: Correctness

**Merge:**
Need to show that two sorted arrays $A$ and $A'$ of length $k$ are merged into a sorted array $B$.

**Induction:**
In the first iteration, we add the smaller of $A[0]$ and $A'[0]$ to $B[0]$. This settles the base case of the induction.

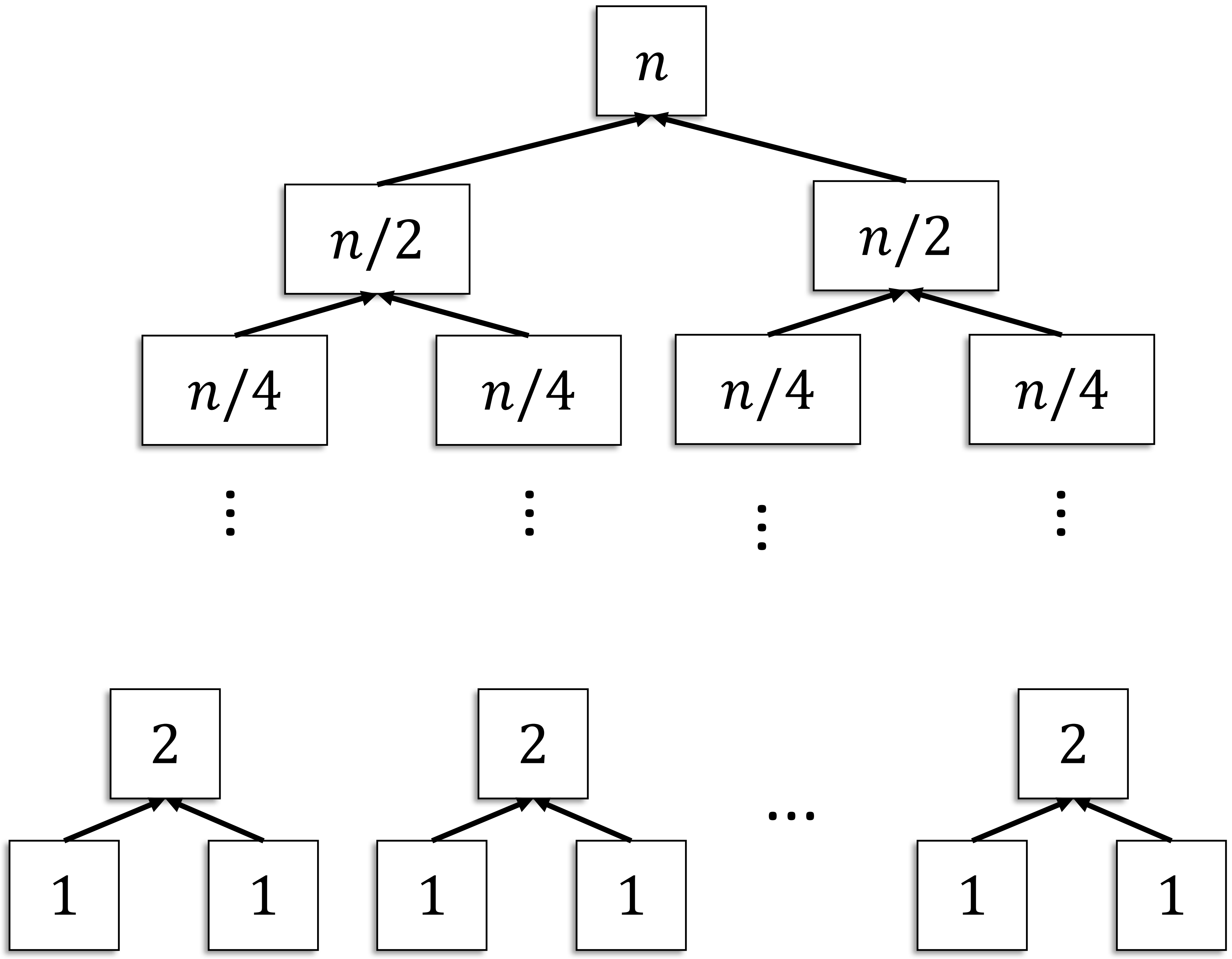
In iteration $i$, we compare the first remaining elements of $A$ and $A'$. Since $A$ and $A'$ are ordered, these are the smallest in the respective arrays. Since $B[i-1]$ gets assigned the smaller one, array $B$ now contains the $i$ smallest elements of arrays $A$ and $A'$.

# Mergesort: Runtime

**Observation:**
There are $O(\log_2 n)$ levels of recursion

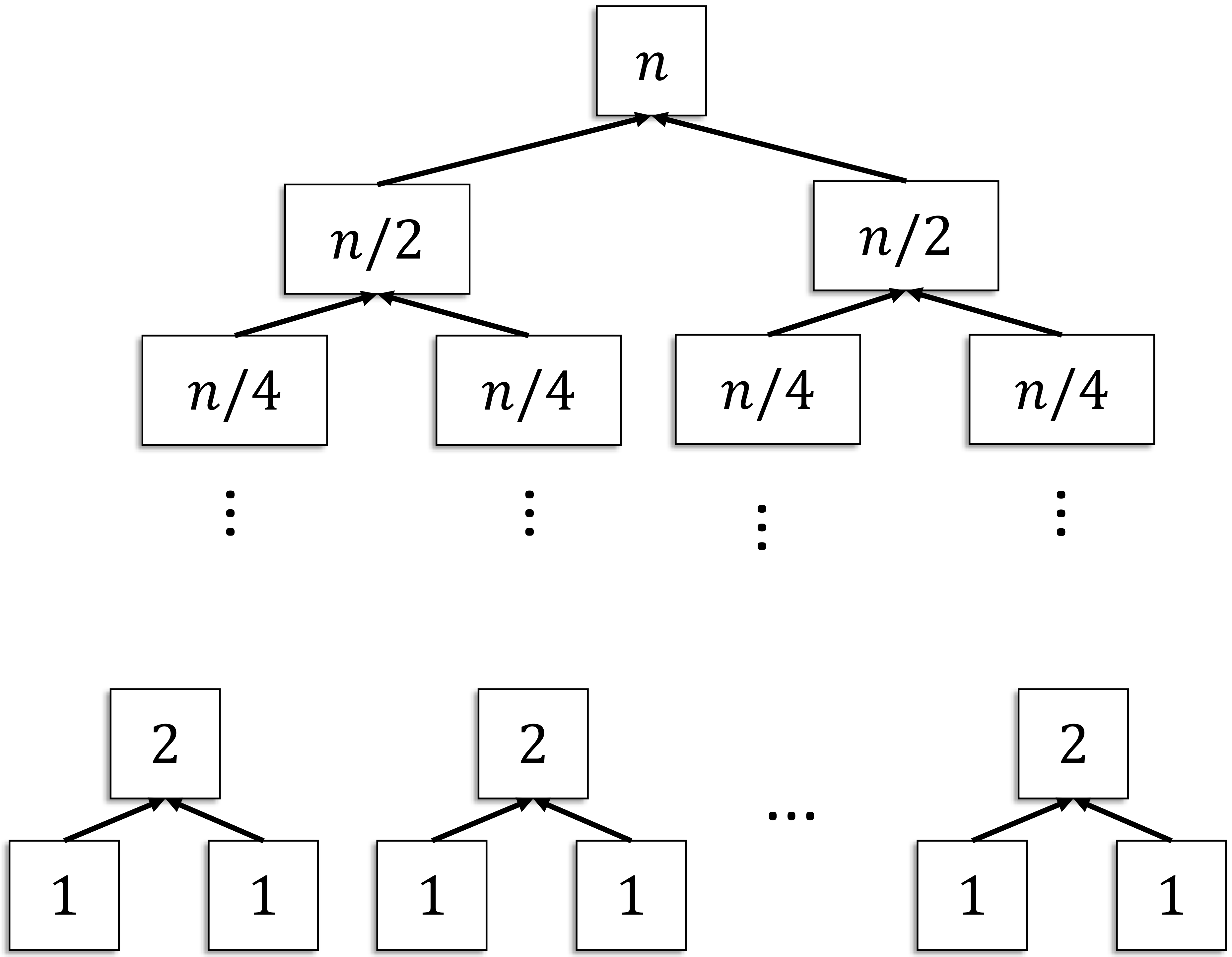
**Observation:**
Sorting an array with $O(1)$ elements takes $O(1)$ time

**Length:**



# Mergesort: Runtime

**Length:**




**Divide and conquer:**

1) Divide into easy subproblems
2) Combine into a solution to the original problem

# Mergesort: Runtime

**Observation:**
There are $\mathrm{O}(\log_2 n)$ levels of recursion

**Lemma:**
At most $O(n)$ comparisons per level

**Observation:**
Sorting the lowest level takes $O(1) \cdot n = O(n)$ time

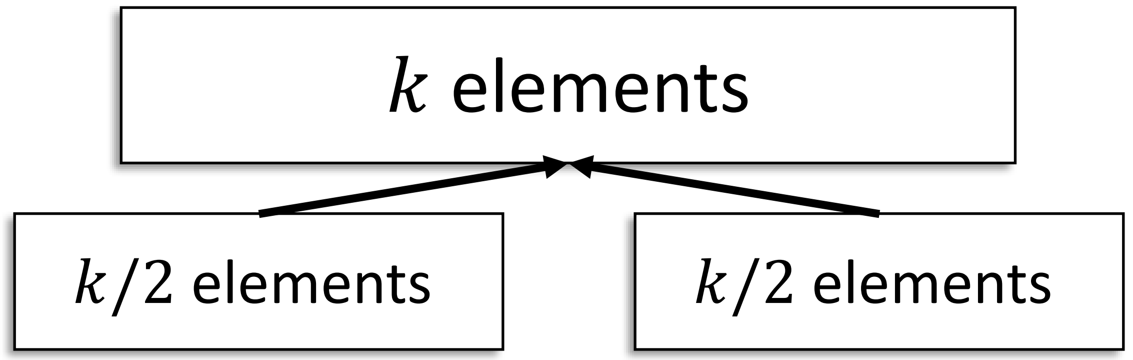
**Argument:**
After each comparison, one element is added to the array (of the corresponding layer)

**Runtime:**

$$O(n \log n)$$

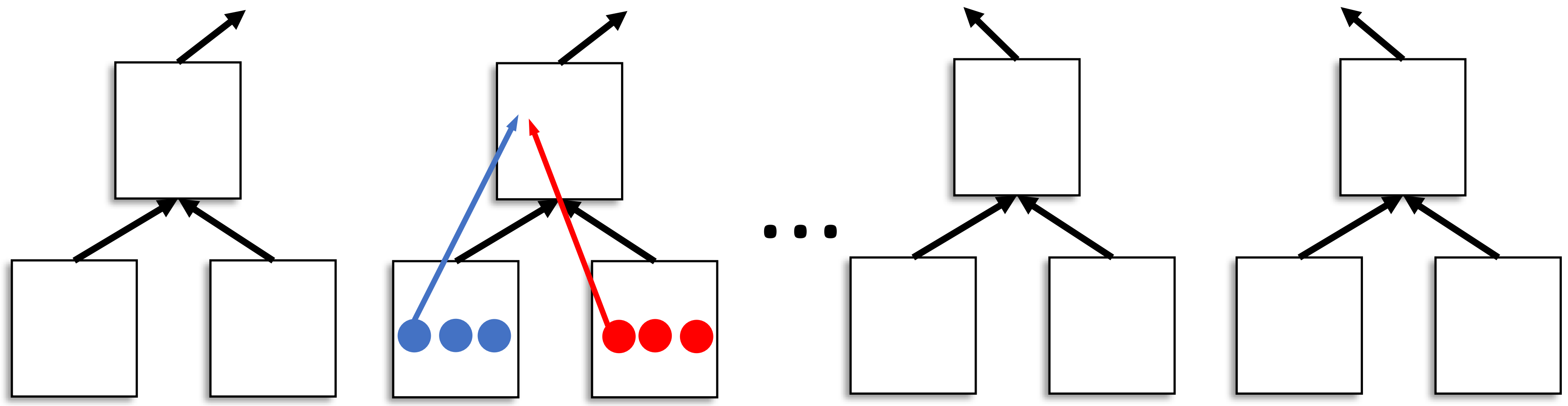
# Mergesort: Runtime

$O(k)$
comparisons



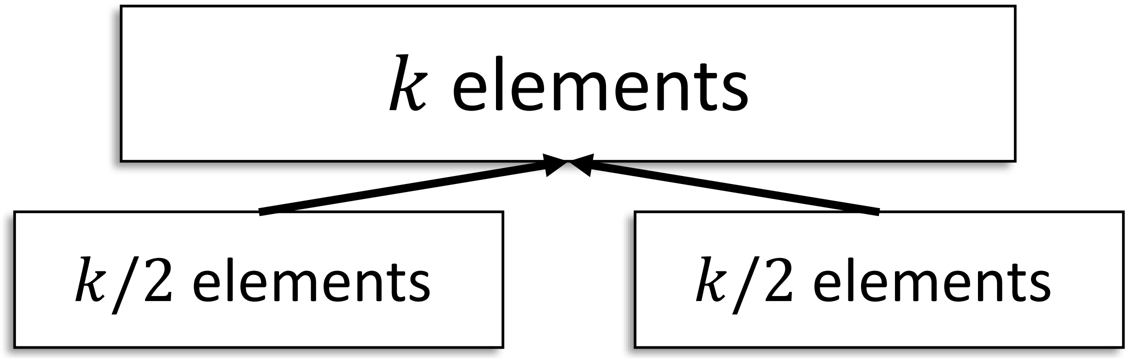

**Lemma :**
At most $O(n)$
comparisons per level

**Argument:**
Sum of length of lists in each level is $n$.
In each comparison, one element is added to some list in the higher level



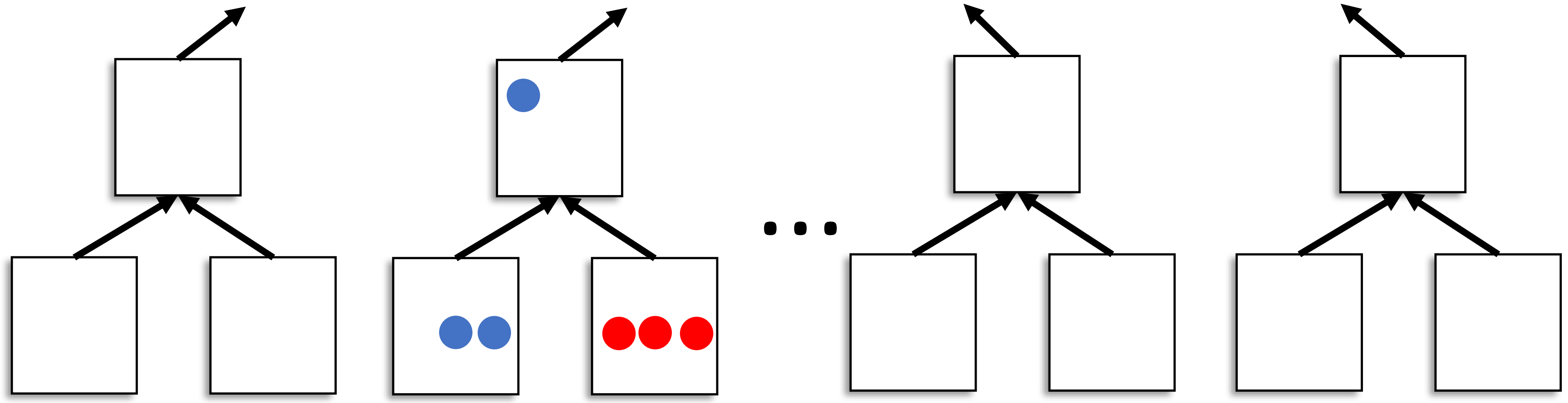
# Mergesort: Runtime

$O(k)$
comparisons



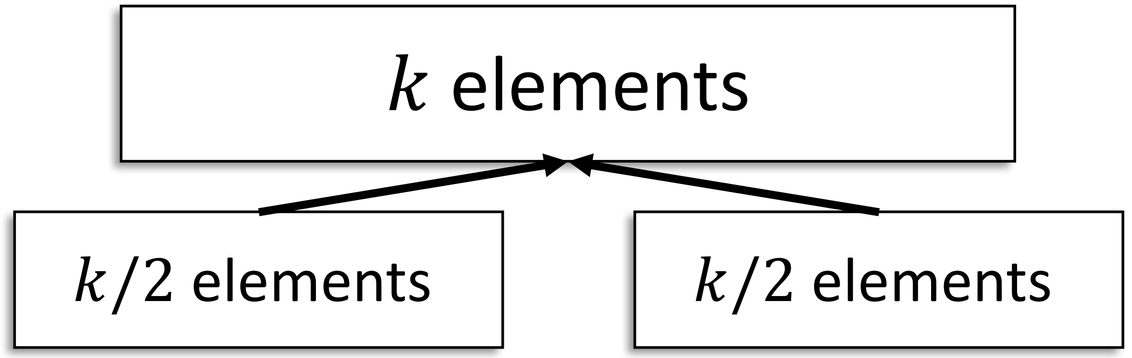

**Lemma :**
At most $O(n)$
comparisons per level

**Argument:**
Sum of length of lists in each level is $n$.
In each comparison, one element is
added to some list in the higher level



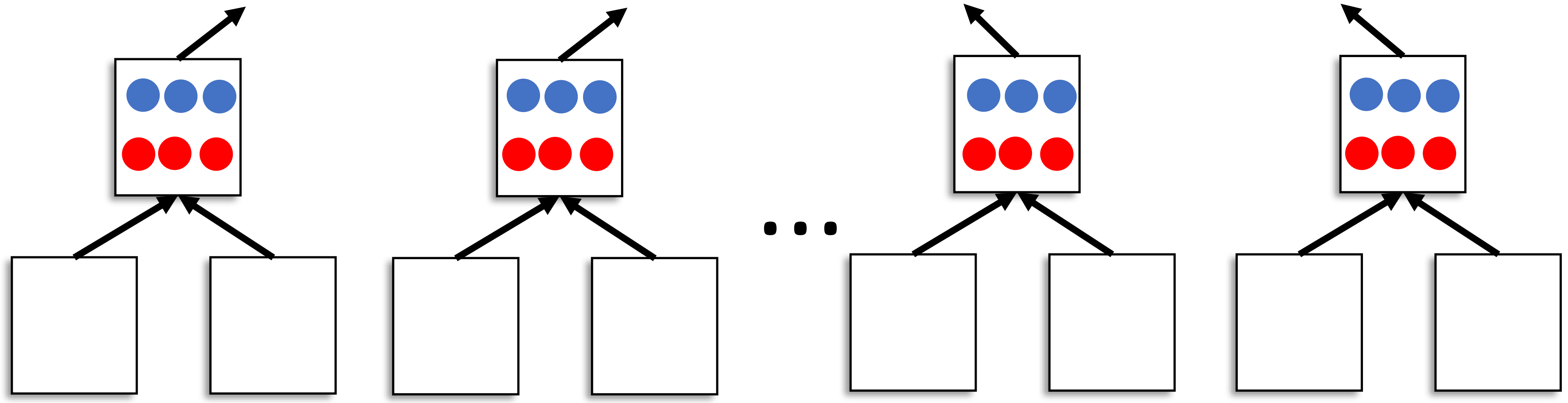
# Mergesort: Runtime

$O(k)$
comparisons




**Lemma :**
At most $O(n)$
comparisons per level

**Argument:**
Sum of length of lists in each level is $n$.
In each comparison, one element is
added to some list in the higher level



# Mergesort: Runtime

**Observation:**
There are $\mathrm{O}(\log_2 n)$ levels of recursion

**Observation:**
Sorting the lowest level takes $O(1) \cdot n = O(n)$ time

**Lemma:**
At most $O(n)$ comparisons per level

# Mergesort: Runtime

**Observation:**
There are $\mathrm{O}(\log_2 n)$ levels of recursion

**Observation:**
Sorting the lowest level takes $O(1) \cdot n = O(n)$ time

**Lemma:**
At most $O(n)$ comparisons per level

**Runtime:**
$O(n \log n)$

# Mergesort: Runtime

**Observation:**
There are $\mathrm{O}(\log_2 n)$ levels of recursion

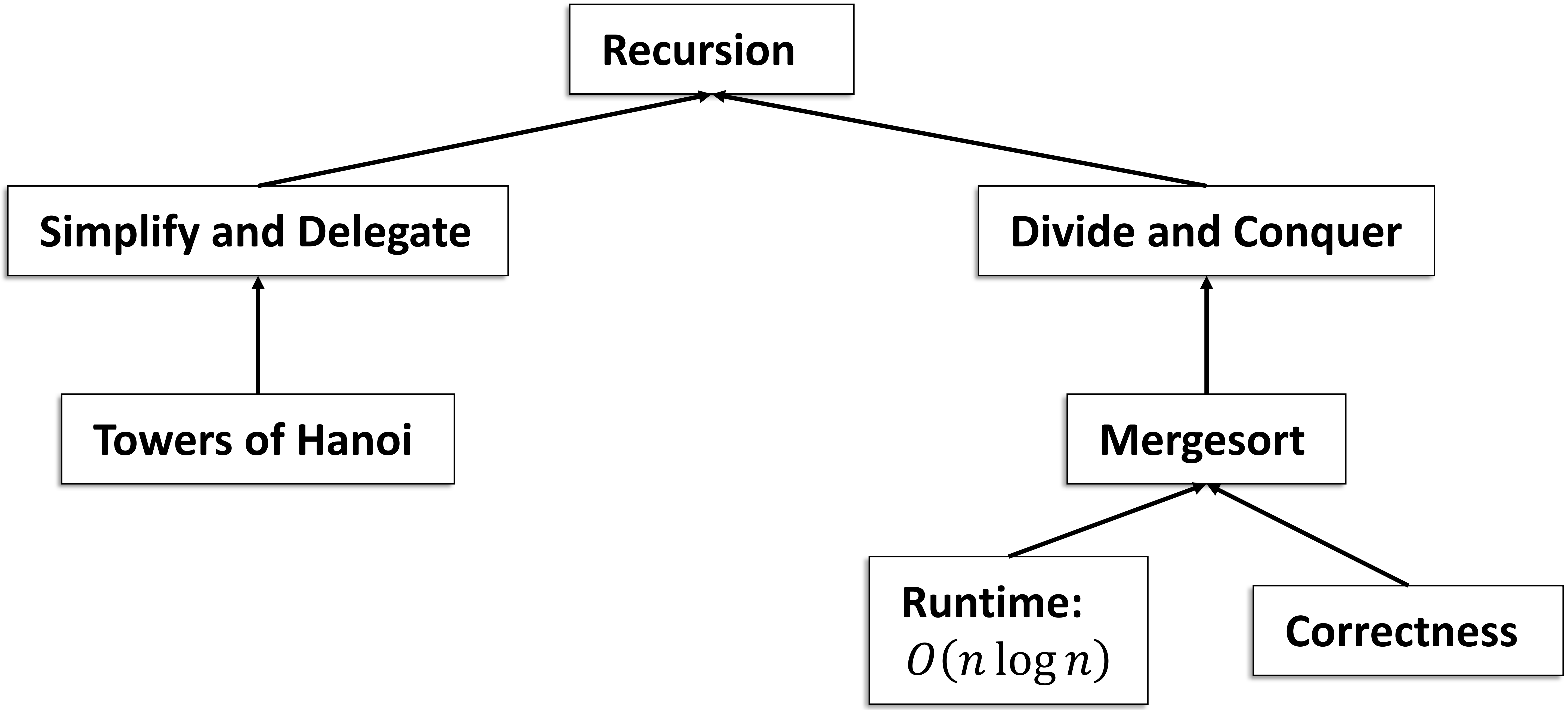
**Observation:**
Sorting the lowest level takes $O(1) \cdot n = O(n)$ time

**Lemma:**
At most $O(n)$ comparisons per level

**Runtime:**
$O(n \log n)$

**Master theorem:**
$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) = O(n \log n)$
Plug $a \coloneqq 2, b \coloneqq 2, k \coloneqq 0$

# Wrap-up


Recursion
Simplify and Delegate
Divide and Conquer
Towers of Hanoi
Mergesort
Runtime: $O(n \log n)$
Correctness