Informal Analysis of Merge Sort

- suppose the running time (the number of operations) of merge sort is a function of the number of elements to sort
  - let the function be $T(n)$
- merge sort works by splitting the list into two sub-lists (each about half the size of the original list) and sorting the sub-lists
  - this takes $2T(n/2)$ running time
- then the sub-lists are merged
  - this takes $O(n)$ running time
- total running time $T(n) = 2T(n/2) + O(n)$

Solving the Recurrence Relation

$$
\begin{aligned}
T(n) &\to 2T(n/2) + O(n) \\
&\approx 2T(n/2) + n \\
&= 2[\ 2T(n/4) + n/2\ ] + n \\
&= 4T(n/4) + 2n \\
&= 4[\ 2T(n/8) + n/4\ ] + 2n \\
&= 8T(n/8) + 3n \\
&= 8[\ 2T(n/16) + n/8\ ] + 3n \\
&= 16T(n/16) + 4n \\
&= 2^k T(n/2^k) + kn
\end{aligned}
$$

$T(n)$ approaches...

Solving the Recurrence Relation

$$T(n) \quad = \quad 2^k T(\underline{n/2^k}) + kn$$

- for a list of length 1 we know $T(1) = 1$
  - if we can substitute $T(1)$ into the right-hand side of $T(n)$ we might be able to solve the recurrence
  - we have $T(n/2^k)$ on the right-hand side, so we need to find some value of k such that

$$\underline{n/2^k} = 1 \;\Rightarrow\; 2^k = n \Rightarrow k = \log_2(n)$$

Solving the Recurrence Relation

$$\begin{aligned} T(n) &= 2^{\log_2 n}\, T\left(n/2^{\log_2 n}\right) + n \log_2 n \\ &= n\, T(1) + n \log_2 n \\ &= n + n \log_2 n \\ &\in n \log_2 n \end{aligned}$$

Proof for $O(n \log_2 n)$

$n < n \log_2 n$ for $n > 2$

$\therefore \quad n + n \log_2 n < 2n \log_2 n$ for $n > 2$

$\therefore \quad n + n \log_2 n < Mn \log_2 n$ for $n > m$

is true for $M = 2$ and $m = 2$

$\therefore \quad n + n \log_2 n \in O(n \log_2 n)$

Recursion

notes Chapter 8

Is Merge Sort Efficient?

- consider a simpler (non-recursive) sorting algorithm called insertion sort

```
// to sort an array a[0]..a[n-1]                    not Java
for i = 0 to (n-1) {
   k = index of smallest element in sub-array a[i]..a[n-1]
   swap a[i] and a[k]
}
```

```
for i = 0 to (n-1) {                                not Java
   for j = (i+1) to (n-1) {
      if (a[j] < a[i]) {
         k = j;
      }
   }
   tmp = a[i];   a[i] = a[k];   a[k] = tmp;
}
```

find smallest element

1 comparison + 1 assignment

3 assignments

$$T(n) = \sum_{i=0}^{n-1}\left(\left(\sum_{j=i+1}^{n-1} 2\right) + 3\right)$$

$$= \left(\sum_{i=0}^{n-1}\big(2(n-1-(i+1)+1)\big)\right) + \sum_{i=0}^{n-1} 3$$

$$= \left(\sum_{i=0}^{n-1} 2(n-i-1)\right) + 3n$$

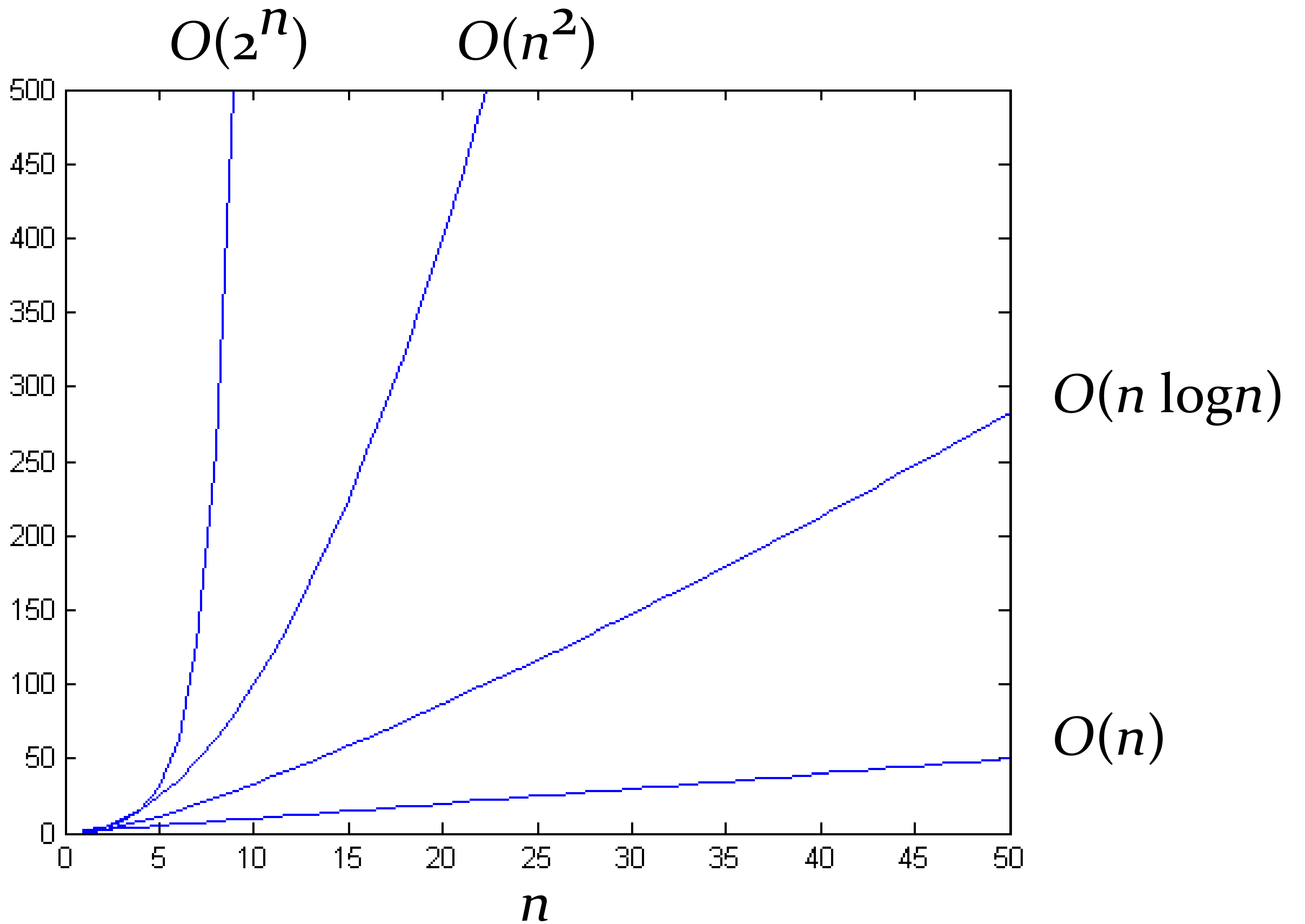
$$= \left(2\sum_{i=0}^{n-1} n\right) - \left(2\sum_{i=0}^{n-1} i\right) - \left(2\sum_{i=0}^{n-1} 1\right) + 3n$$

$$= 2n^2 - 2\frac{n(n-1)}{2} - 2n + 3n$$

$$= 2n^2 - n^2 + n - 2n + 3n$$

$$= n^2 + 2n \qquad \in O(n^2)$$

Comparing Rates of Growth



Comments

- big O complexity tells you something about the running time of an algorithm as the size of the input, n, approaches infinity
  - we say that it describes the limiting, or asymptotic, running time of an algorithm
- for small values of n it is often the case that a less efficient algorithm (in terms of big O) will run faster than a more efficient one
  - insertion sort is typically faster than merge sort for short lists of numbers

Revisiting the Fibonacci Numbers

- the recursive implementation based on the definition of the Fibonacci numbers is inefficient

```
public static int fibonacci(int n) {
  if (n == 0) {
   return 0;
  }
  else if (n == 1) {
   return 1;
  }
  int f = fibonacci(n - 1) + fibonacci(n - 2);
  return f;
}
```

- how inefficient is it?
- let $T(n)$ be the running time to compute the nth Fibonacci number
  - $T(0) = T(1) = 1$
  - $T(n)$ is a recurrence relation

$$
\begin{aligned}
T(n) &= T(n-1) + T(n-2) \\
&= \big(T(n-2) + T(n-3)\big) + T(n-2) \\
&= 2T(n-2) + T(n-3) \\
&> 2T(n-2) \\
&> 2\big(2T(n-4)\big) = 4T(n-4) \\
&> 4\big(2T(n-6)\big) = 8T(n-6) \\
&> 8\big(2T(n-8)\big) = 16T(n-8) \\
&> 2^k T(n-2k)
\end{aligned}
$$

Solving the Recurrence Relation

$$T(n) \quad > \qquad 2^k T(\underline{n - 2k})$$

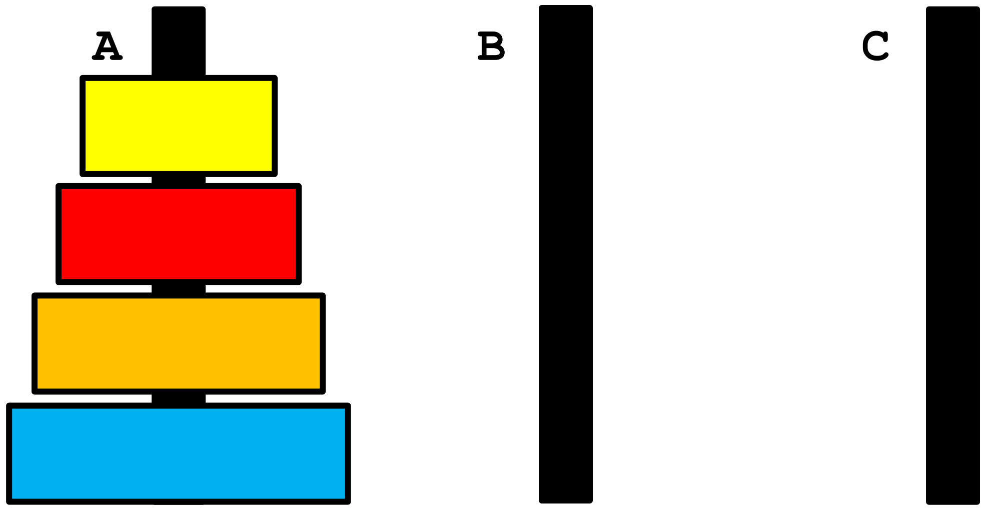
- we know $T(1) = 1$
  - if we can substitute $T(1)$ into the right-hand side of $T(n)$ we might be able to solve the recurrence
  - we have $T(n - 2k)$ so we need to find a value for k such that:

$$\underline{n - 2k} = 1 \;\Rightarrow\; 1 + 2k = n \Rightarrow k = (\mathrm{n} - 1)/2$$

$$T(n) > 2^k T(n-2k) \quad = \quad 2^{(n-1)/2} T(1) \quad = \quad 2^{(n-1)/2} \quad \in \quad O(2^n)$$

Towers of Hanoi

- a problem easily solved using recursion



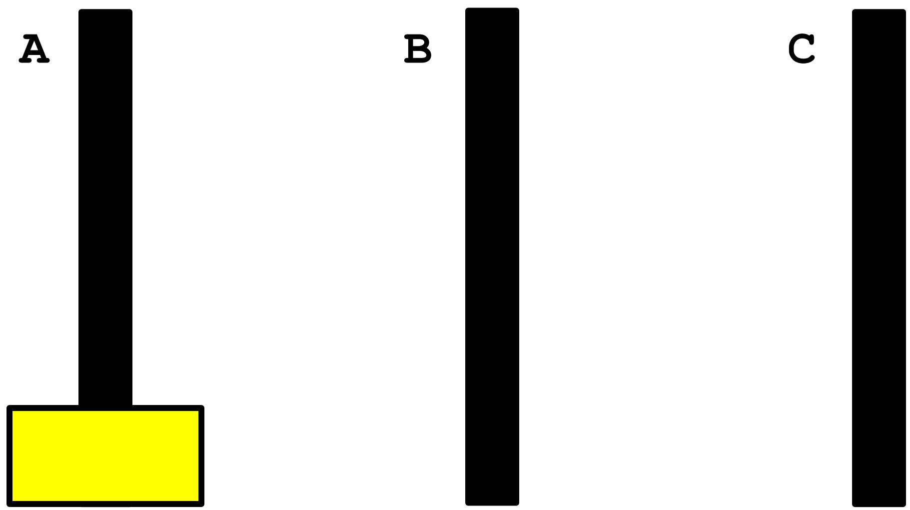
  - move the stack of *n* disks from A to C
    - can move one disk at a time from the top of one stack onto another stack
    - cannot move a larger disk onto a smaller disk

Towers of Hanoi

- legend says that the world will end when a 64 disk version of the puzzle is solved
- several appearances in pop culture
  - Doctor Who
  - Rise of the Planet of the Apes
  - Survior: South Pacific

Towers of Hanoi

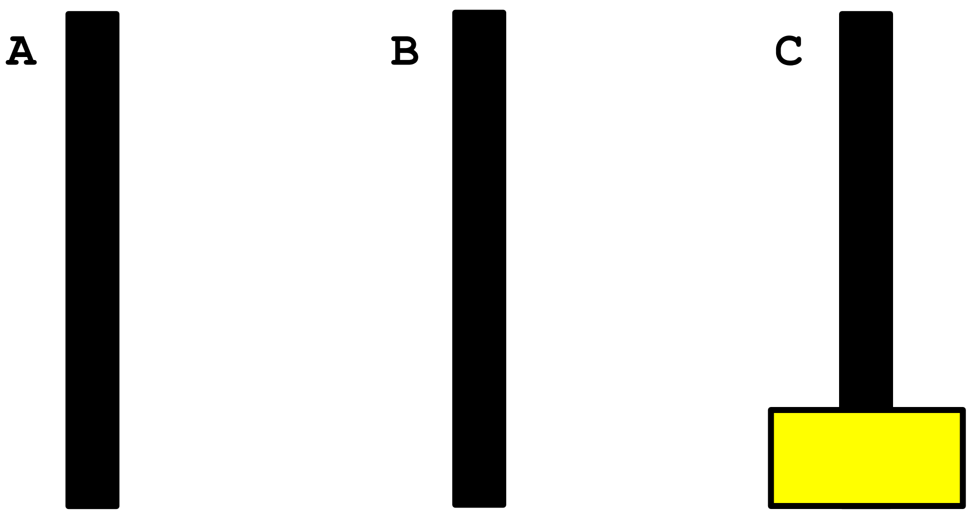
- n = 1



- move disk from A to C

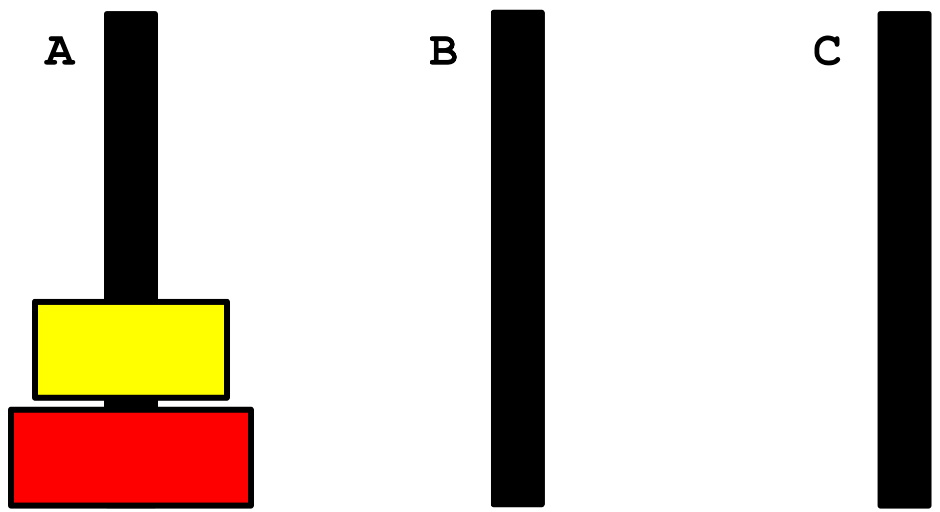
Towers of Hanoi

- n = 1



Towers of Hanoi

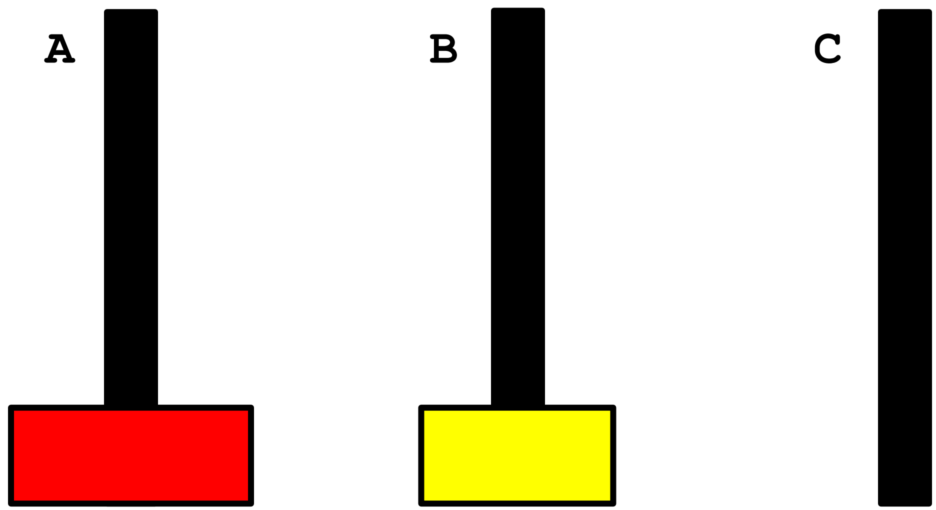
- n = 2



- move disk from A to B

Towers of Hanoi

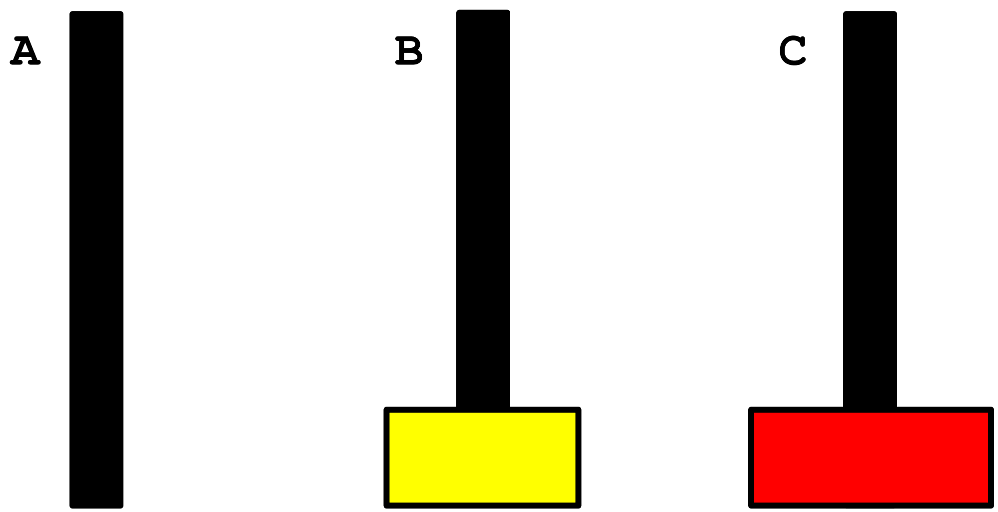
- n = 2



- move disk from A to C

Towers of Hanoi

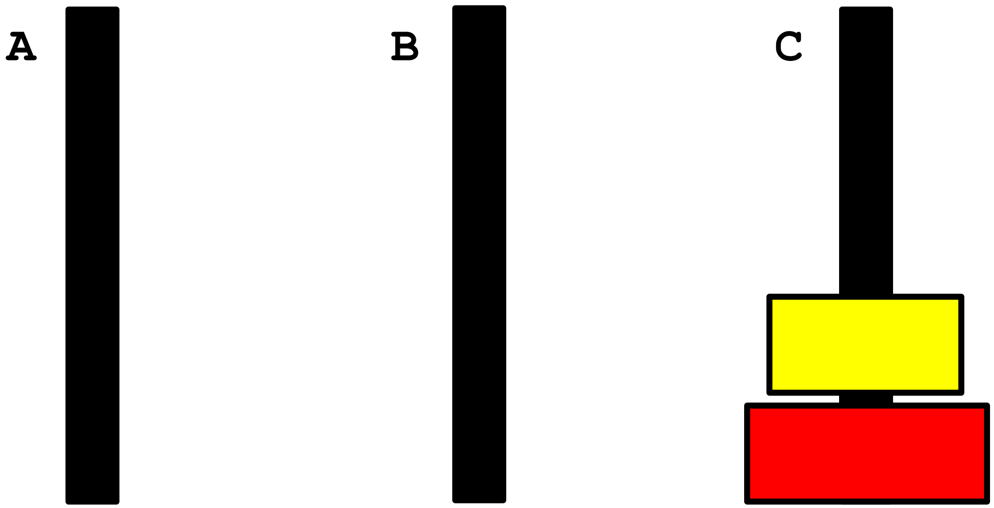
- n = 2



- move disk from B to C

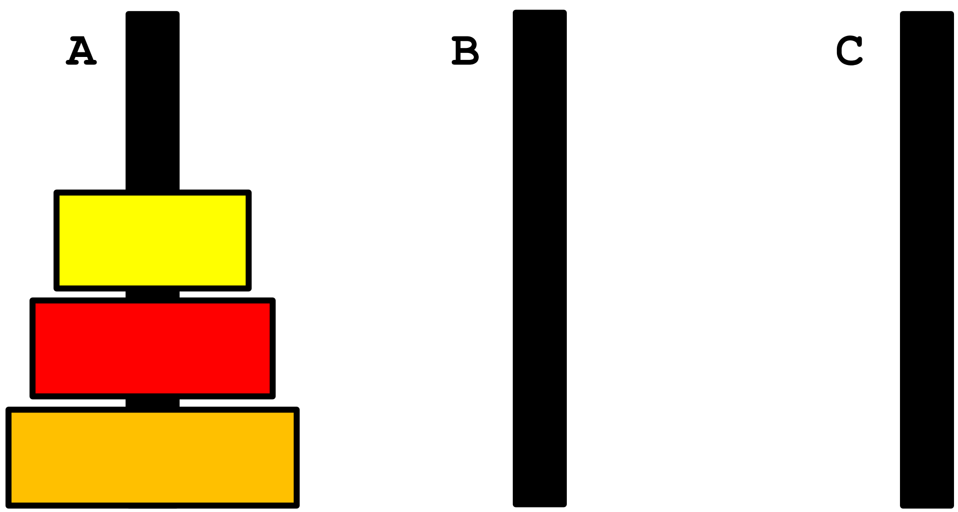
Towers of Hanoi

- $n = 2$



Towers of Hanoi

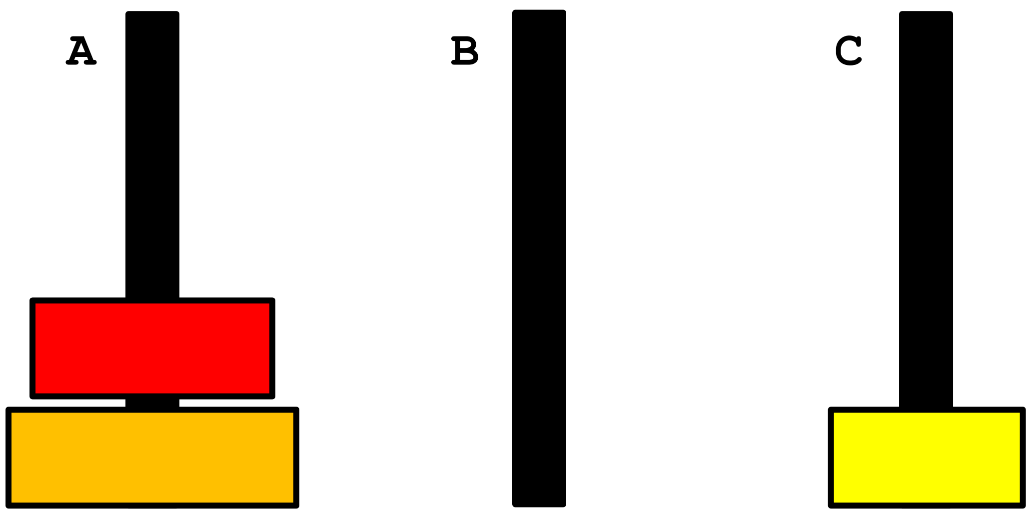
- n = 3



- move disk from A to C

Towers of Hanoi

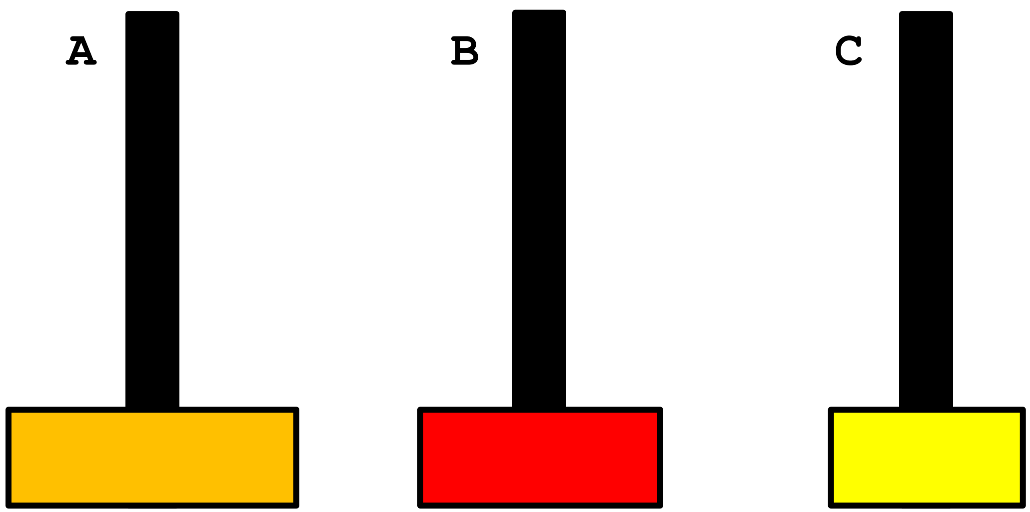
- n = 3



- move disk from A to B

Towers of Hanoi

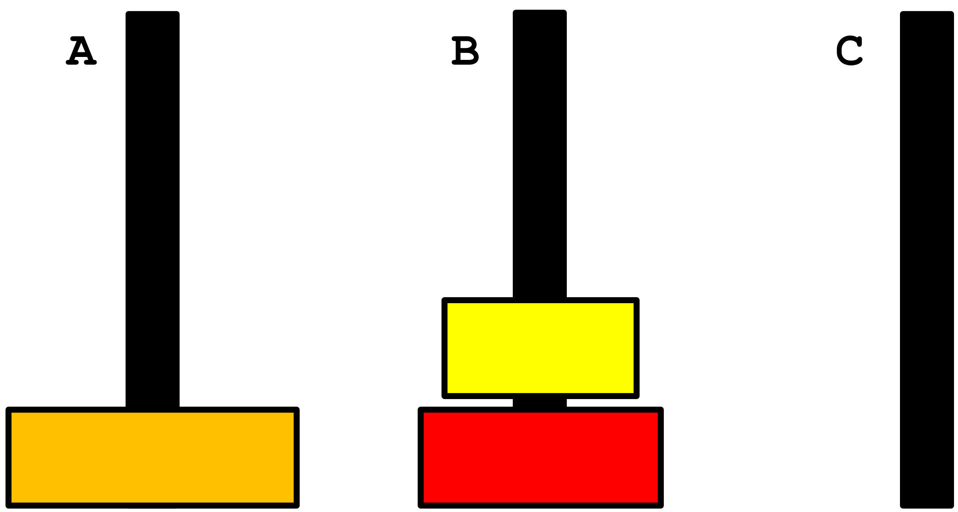
- n = 3



- move disk from C to B

Towers of Hanoi

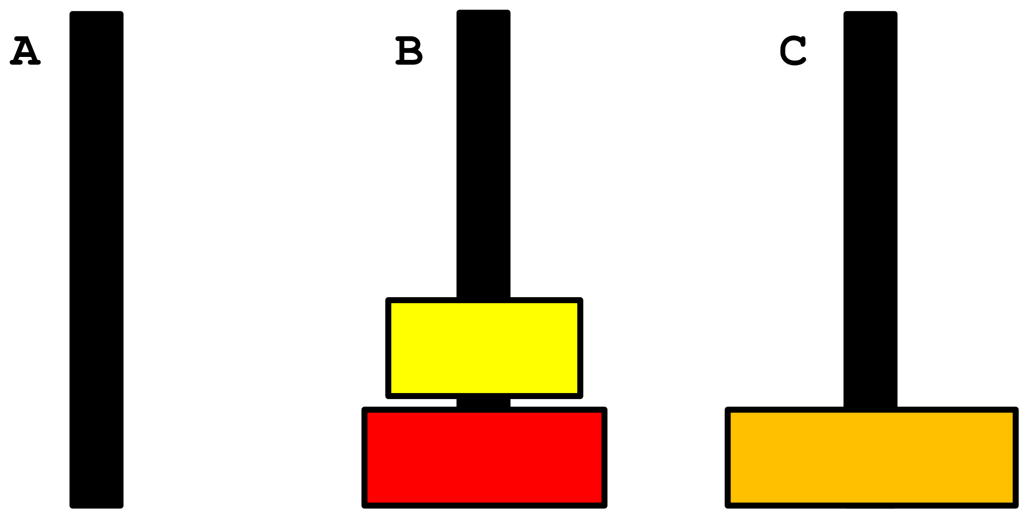
- n = 3



- move disk from A to C

Towers of Hanoi

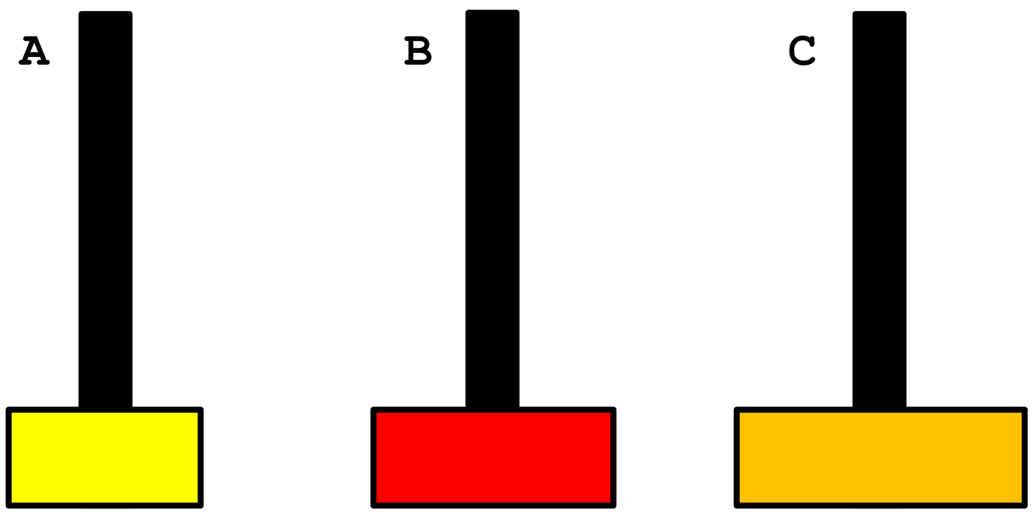
- n = 3



- move disk from B to A

Towers of Hanoi

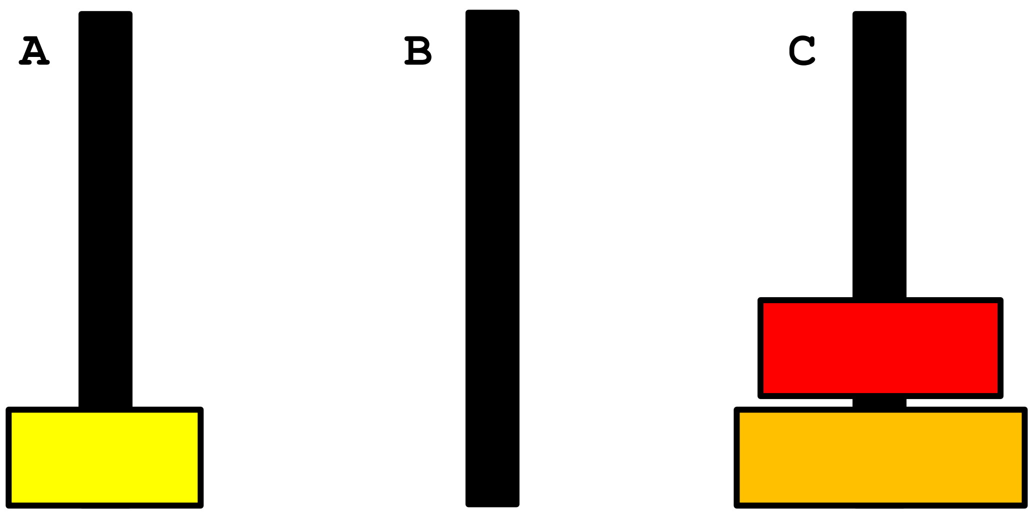
- n = 3



- move disk from B to C

Towers of Hanoi

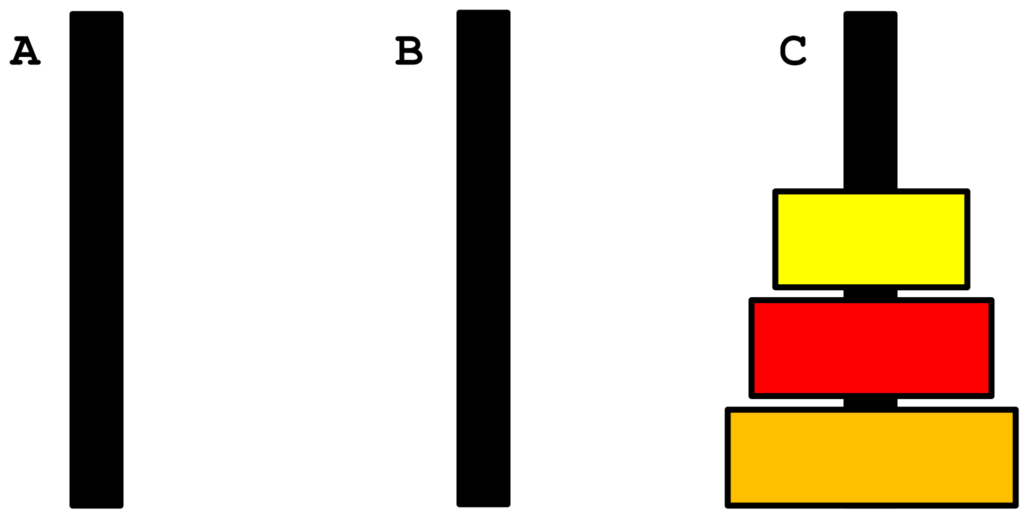
- n = 3



- move disk from A to C

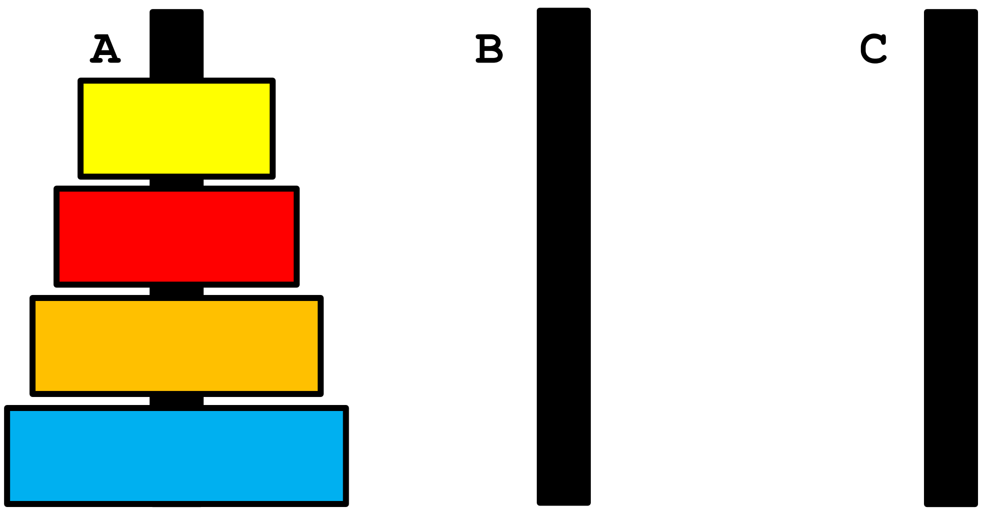
Towers of Hanoi

- $n = 3$



Towers of Hanoi

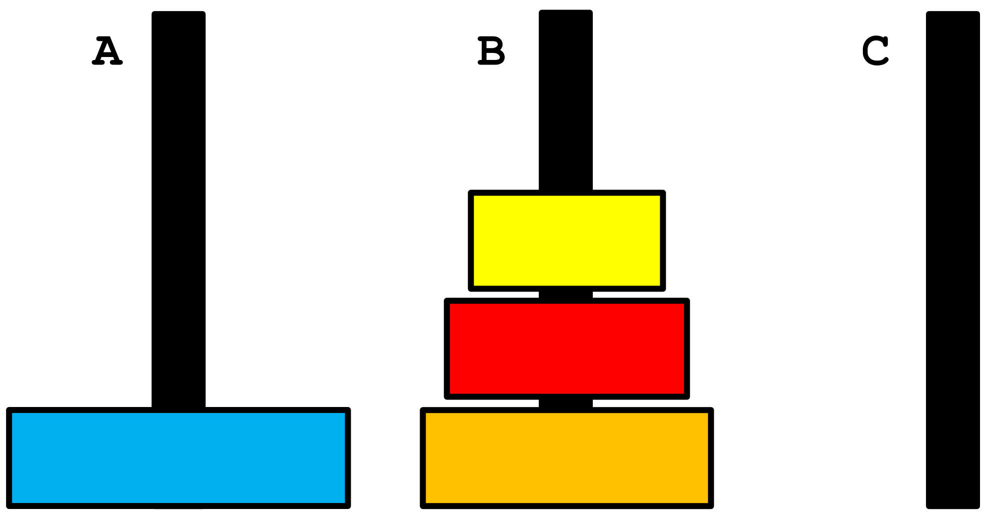
- n = 4



- move (n – 1) disks from A to B using C

Towers of Hanoi

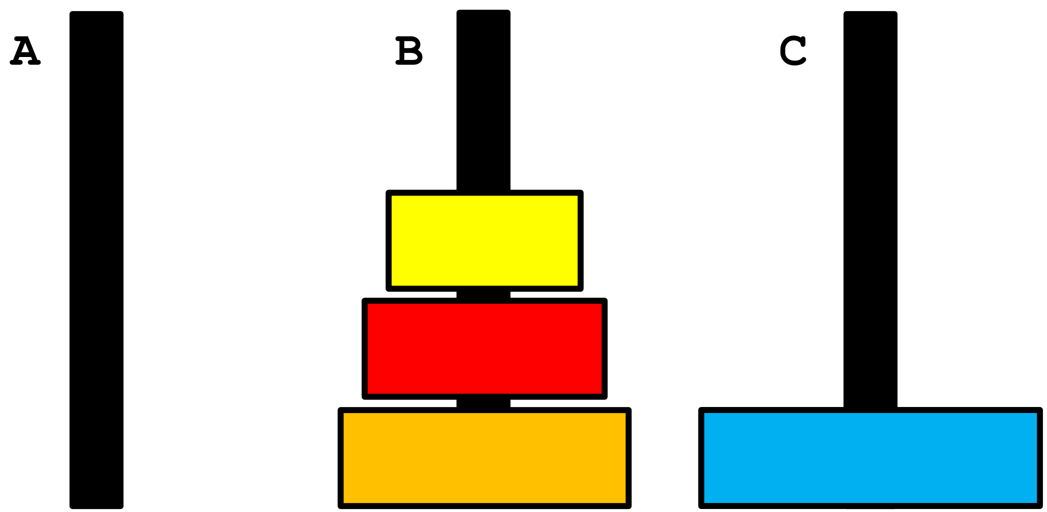
- n = 4



- move disk from A to C

Towers of Hanoi

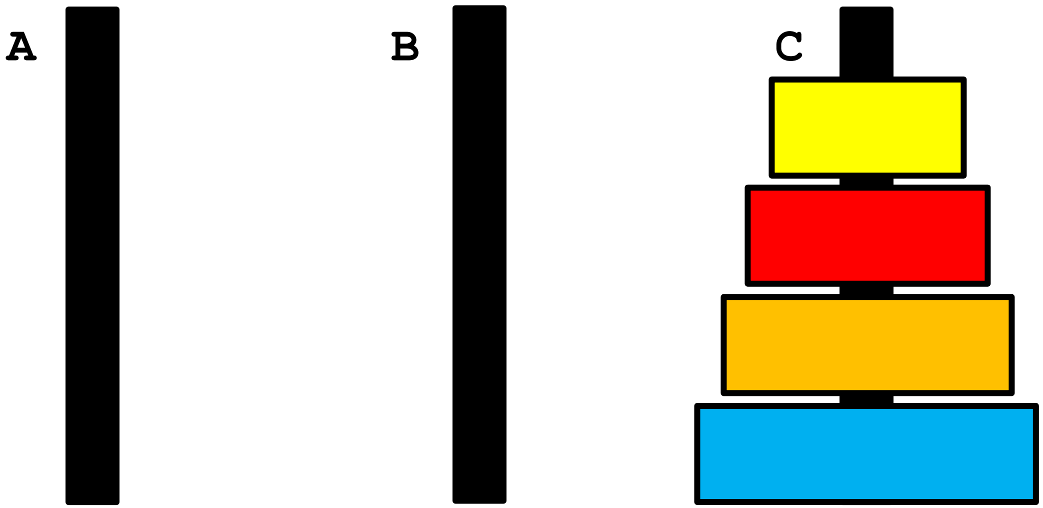
- n = 4



- move (n – 1) disks from B to C using A

Towers of Hanoi

- n = 4



- base case $n = 1$
  1. move disk from A to C
- recursive case
  1. move $(n - 1)$ disks from A to B
  2. move 1 disk from A to C
  3. move $(n - 1)$ disks from B to C

Towers of Hanoi

```
public static void move(int n,
                        String from,
                        String to,
                        String using) {
  if(n == 1) {
    System.out.println("move disk from " + from + " to " + to);
  }
  else {
    move(n - 1, from, using, to);
    move(1, from, to, using);
    move(n - 1, using, to, from);
  }
}
```