

Two Examples

CS105 : Saelee

Blackjack, anyone?



Blackjack Hand Value

- Compute the value of a Blackjack hand
 - Hand expressed as an array of strings
 - Cards: 'A', '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K'
- Gotcha: 'A' is either 1 or 11; try to get value as close to (but not over) 21 as possible

Blackjack Hand Value

```
def hand_value(hand)
  value = 0
  # First calculate values ignoring aces
  hand.each do |card|
    if card == 'J' || card == 'Q' || card == 'K'
      value += 10
    elsif card != 'A'
      value += card.to_i
    end
  end
  # Then add aces as 11 unless they would bust the hand
  hand.each do |card|
    if card == 'A'
      if value > 10
        value += 1
      else
        value += 11
      end
    end
  end
  value
end
```

```
hand_value ['A', 'K']      => 21
hand_value [2, 2, 5, 'A'] => 20
hand_value [2, 2, 5, 5, 'A'] => 15
```

The Dealer's Game

- Next, using `hand_value`, simulate a dealer's game, specifying an "upcard"
- Dealer game: keep "hitting" until at or over 17
 - Need to randomize cards "dealt" to the dealer

Dealer Game

```
CARDS = %w(A 2 3 4 5 6 7 8 9 10 J Q K)

def play_blackjack(upcard)
  hand = [upcard]
  while hand_value(hand) < 17
    hand << CARDS[rand(13)]
  end
  hand_value(hand)
end
```

```
>> 5.times { puts play_blackjack(5) }
17
21
17
17
20
```

Counting Busts

- Next, **simulate** multiple runs of a dealer game holding the upcard constant
 - Keep track of the number of dealer busts

Simulation

```
def simulate(upcard, num_games)
  busts = 0
  num_games.times do
    hand_val = play_blackjack(upcard)
    if hand_val > 21
      busts = busts + 1
    end
  end
  busts
end
```

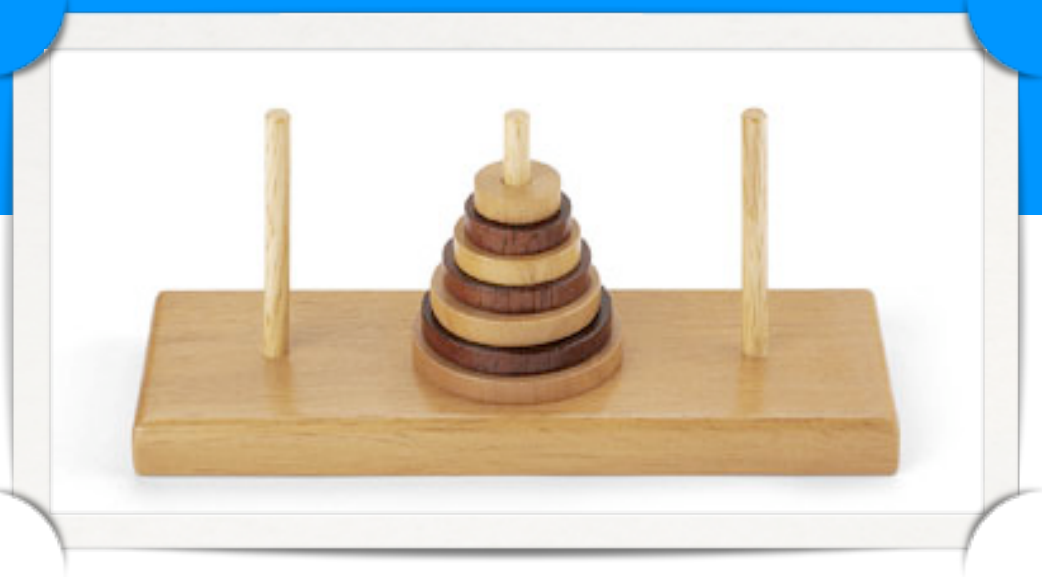
```
>> simulate(2, 1000)
=> 366
>> simulate('A', 1000)
=> 131
```

Obtaining Statistics

```
CARDS.each do |card|  
  puts "Busts for #{card} = #{simulate(card, 1000)}"  
end
```

```
Busts for A = 132  
Busts for 2 = 348  
Busts for 3 = 362  
Busts for 4 = 396  
Busts for 5 = 389  
Busts for 6 = 432  
Busts for 7 = 260  
Busts for 8 = 219  
Busts for 9 = 230  
Busts for 10 = 215  
Busts for J = 223  
Busts for Q = 224  
Busts for K = 185
```

Towers of Hanoi



Recursive hanoi

```
def hanoi(n, from, to, using)
  if (n == 1)
    puts "move from #{from} to #{to}"
    return
  end
  hanoi(n - 1, from, using, to)
  hanoi(1, from, to, using)
  hanoi(n - 1, using, to, from)
end
```

```
>> hanoi(3, 'A', 'C', 'B')
move from A to C
move from A to B
move from C to B
move from A to C
move from B to A
move from B to C
move from A to C
```

hanoi(4) & hanoi(5)

```
>> hanoi(4, 'A', 'C', 'B')  
move from A to B  
move from A to C  
move from B to C  
move from A to B  
move from C to A  
move from C to B  
move from A to B  
move from A to C  
move from B to C  
move from B to A  
move from C to A  
move from B to C  
move from A to B  
move from A to C  
move from B to C
```

```
>> hanoi(5, 'A', 'C', 'B')  
move from A to C  
move from A to B  
move from C to B  
move from A to C  
move from B to A  
move from B to C  
move from A to C  
move from A to B  
move from C to B  
move from C to A  
move from B to A  
move from C to B  
move from A to C  
move from A to B  
move from C to B  
move from A to C  
move from B to A  
move from B to C  
move from A to C  
move from B to A  
move from C to B  
move from C to A  
move from B to A  
move from B to C  
move from A to C  
move from A to B  
move from C to B  
move from A to C  
move from B to A  
move from B to C  
move from A to C
```

The Legend of the Towers of Hanoi

In a temple outside Hanoi, Vietnam, there are 3 stakes and 64 golden discs that were placed there at the beginning of time.

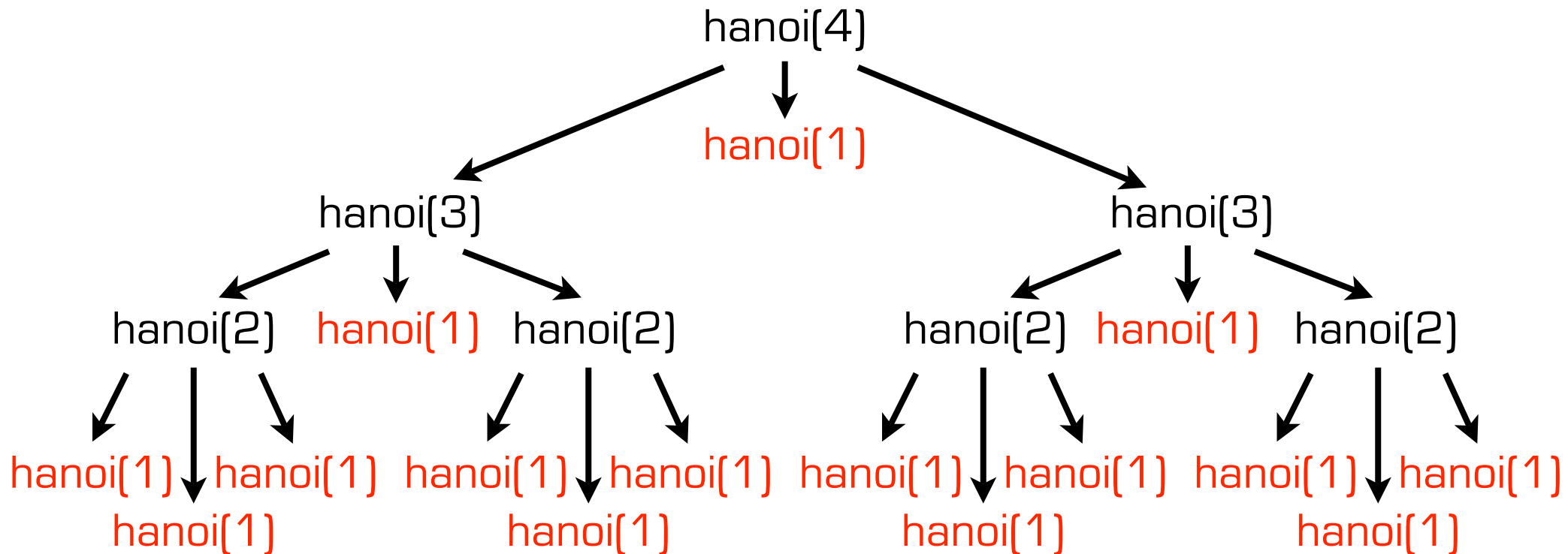
Monks move these discs in accordance with the rules of the puzzle at the rate of one a day. On the day of the last move, upon which the original stack shall be transferred to its final destination, the world will come to an end.

How long do we have?

Runtime of Hanoi?

- Runtime with respect to “size” of input: number of blocks
- Consider call to **hanoi(4, ...)**
 - Results in call to **hanoi(3, ...)**, **hanoi(1, ...)**, and **hanoi(3, ...)**
 - **hanoi(1, ...)** is the base case, with constant runtime C

Runtime Characterization



`hanoi(4)` results in **15** actual moves; runtime = $15 \cdot C$

Runtime for n blocks: $T(n)$

$$T(n) = \begin{cases} C & : n == 1 \\ 2 \cdot T(n-1) + T(1) & : n > 1 \end{cases}$$

$$T(n) = C \cdot (2^n - 1)$$

$$T(n) = \Theta(2^n)$$

- We say that `hanoi(n)` has a runtime that is an **exponential function** of the input n

Exponential Runtimes

- 3 blocks $\Rightarrow 2^3 - 1 = 7$ steps
- 10 blocks $\Rightarrow 2^{10} - 1 = 1,023$ steps
- 64 blocks $\Rightarrow 2^{64} - 1 = 18,446,744,073,709,551,615$ steps
 - At the rate of 1 block a day, this would take us 584,942,417,355 years! (We're quite safe)

Classifying Runtimes

- Classifying runtimes allows for gross algorithm comparisons without considering implementation specifics
- Typical runtime growth behaviors:
 - Constant
 - Linear
 - Logarithmic
 - Polynomial
 - Exponential