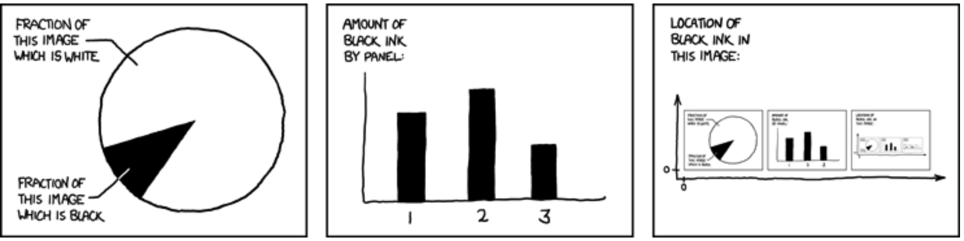
More Recursion!



http://xkcd.com/688/

http://xkcd.com/981/

Recursion - examples

- Problem: given a string as input, return the string with characters reversed.
- Base case?
- Recursion

Tail recursion

- Tail recursion is a recursive call that occurs as the last action in a method.
- This is not tail recursion:

```
public int factorial(int n){
   if (n==0)
      return 1;
   return n * factorial(n-1);
}
```

- How can we make the call to factorial the last thing?

Tail recursion

- Tail recursion is a recursive call that occurs as the last action in a method.
- This is not tail recursion:

```
public int factorial(int n){
   if (n==0)
      return 1;
   return n * factorial(n-1);
}
```

- How can we make the call to factorial the last thing?
- Yep! Must use * in a new argument.

Tail recursion

- Non tail-recursive:

```
public int factorial(int n){
   if (n == 0)
      return 1;
   return n * factorial(n-1);
}
```

- Tail-recursive:

```
public int factorial(int n, int product) {
   if (n == 0)
      return product;
   return factorial(n-1, n * product);
}
```

Tail recursion

- Let's hide this additional argument:

```
public int factorial(int n) {
   return factorialTail(n, 1);
}
private int factorialTail(int n, int product) {
    if(n == 0)
        return product;
    return factorialTail(n-1, n * product);
}
```

- But why would you care? Compilers can optimize memory usage when they detect tail recursion. When making a recursive call, you no longer need to save the information about the local variables within the calling method.

Dictionary lookup

- Suppose you're looking up a word in the dictionary (paper one, not online!)
- You probably won't scan linearly through the pages – inefficient.
- What would be your strategy?

Binary search

```
binarySearch(dictionary,  word){

  // base case
      ????

  else {// recursive case

      open the dictionary to a point near the middle
      determine which half of the dictionary contains word

      if (word is in first half of the dictionary) {
       binarySearch(first half of dictionary, word)
      }
      else {
       binarySearch(second half of dictionary, word)
      }
  }
```

Binary search

```
binarySearch(dictionary,  word){

   if (dictionary has one page) {// base case
         scan the page for word
   }

   else {// recursive case

         open the dictionary to a point near the middle
         determine which half of the dictionary contains word

         if (word is in first half of the dictionary) {
          binarySearch(first half of dictionary, word)
         }
         else {
          binarySearch(second half of dictionary, word)
         }
   }
```

Binary search

- Let's write a method called **`binarySearch`** that accepts a sorted array of integers and a target integer and returns the index of an occurrence of that value in the array.
  - If the target value is not found, return -1

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92

```
int index  = binarySearch(data, 42);   // 10
int index2 = binarySearch(data, 66);   // -1
```

Binary search

index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
value	-4	2	7	10	15	20	22	25	30	36	42	50	56	68	85	92

- How can we implement this?
  - Create two smaller arrays?
  - Pass start and end indicies?

Binary search

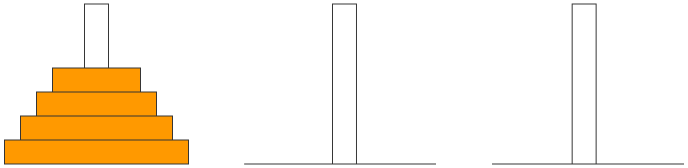
```
// Precondition: a is sorted
// Postcondition: Returns the index of an occurrence of the given value, or -1.
public int binarySearch(int[] a, int target) {
    return binarySearch(a, target, 0, a.length - 1);
}
// Recursive helper to implement search.
private int binarySearch(int[] a, int target, int first, int last) {
    if (first > last) {
        return -1;    // not found
    } else {
        int mid = (first + last) / 2;
        if (a[mid] == target) {
            return mid;

        } else if (a[mid] < target) {
            return binarySearch(a, target, mid+1, last);

        } else {
            return binarySearch(a, target, first, mid-1);
        }
    }
}
```

Towers of Hanoi

Example: Towers of Hanoi, move all disks to third peg without ever placing a larger disk on a smaller one.

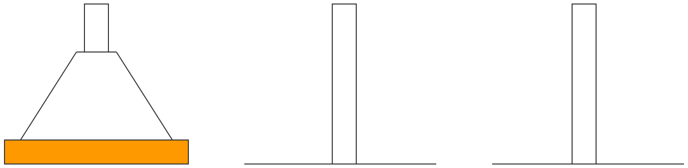


Try to find the pattern by cases

- One disk is easy
- Two disks...
- Three disks...
- Four disk...

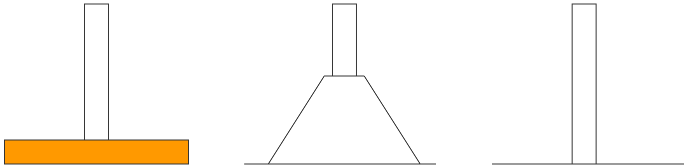
Towers of Hanoi

Example: Towers of Hanoi, move all disks to third peg without ever placing a larger disk on a smaller one.



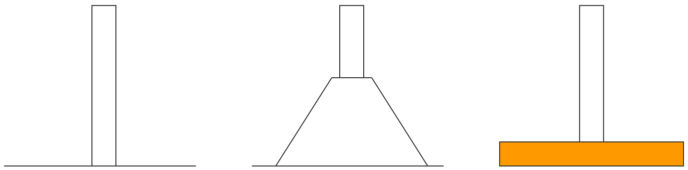
Towers of Hanoi

Example: Towers of Hanoi, move all disks to third peg without ever placing a larger disk on a smaller one.



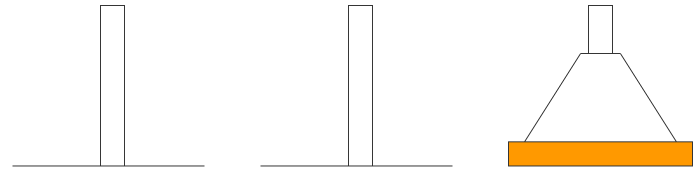
Towers of Hanoi

Example: Towers of Hanoi, move all disks to third peg without ever placing a larger disk on a smaller one.



Towers of Hanoi

Example: Towers of Hanoi, move all disks to third peg without ever placing a larger disk on a smaller one.



Let's go play with it at:
https://www.mathsisfun.com/games/towerofhanoi.html
https://www.youtube.com/watch?v=4_KtPENqCb0

Fibonacci's Rabbits

- Suppose a newly-born pair of rabbits, one male, one female, are put on an island.
  - A pair of rabbits doesn't breed until 2 months old.
  - Thereafter each pair produces another pair each month
  - Rabbits never die.
- How many pairs will there be after n months?

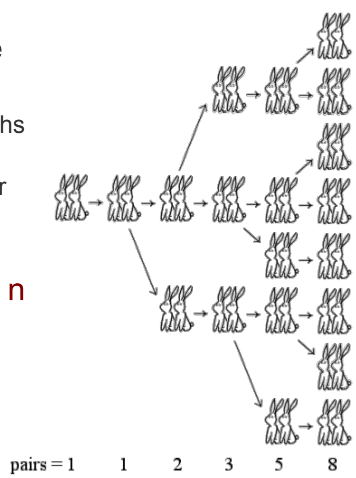


image from: http://www.jimloy.com/algebra/fibo.htm

Fibonacci numbers

- The *Fibonacci numbers* are a sequence of numbers F_0, F_1, ... F_n defined by:

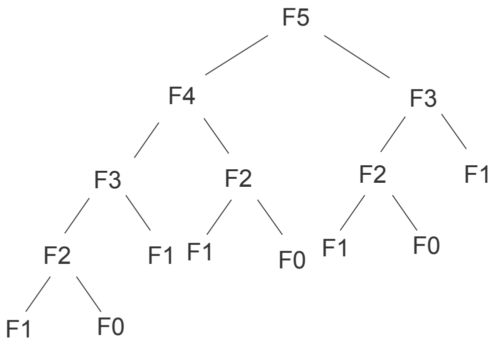
$$F_0 = F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2} \text{ for any } i > 1$$

- Write a method that, when given an integer *i*, computes the *nth* Fibonacci number.

Fibonacci numbers

- Let's run it for n = 1,2,3,... 10, ... , 20,...
- If n is large the computation takes a long time! Why?

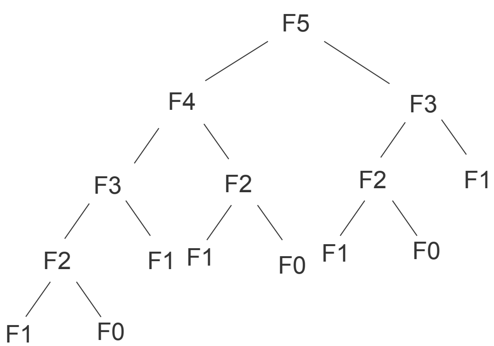


Fibonacci numbers

- recursive Fibonacci was expensive because it made many, recursive calls
  - fibonacci(n) recomputed fibonacci(n-1),…,fibonacci(1) many times in finding its answer!
  - This is a case where the sub-tasks handled by the recursion are redundant with each other and get recomputed

Fibonacci numbers

- Every time n is incremented by 2, the call tree more than doubles.



Growth of rabbit population

1 1 2 3 5 8 13 21 34 ...

The fibonacci numbers themselves also grow rapidly: every 2 months the population at least **DOUBLES**

Fractals – the Koch curve and Sierpinski Triangle