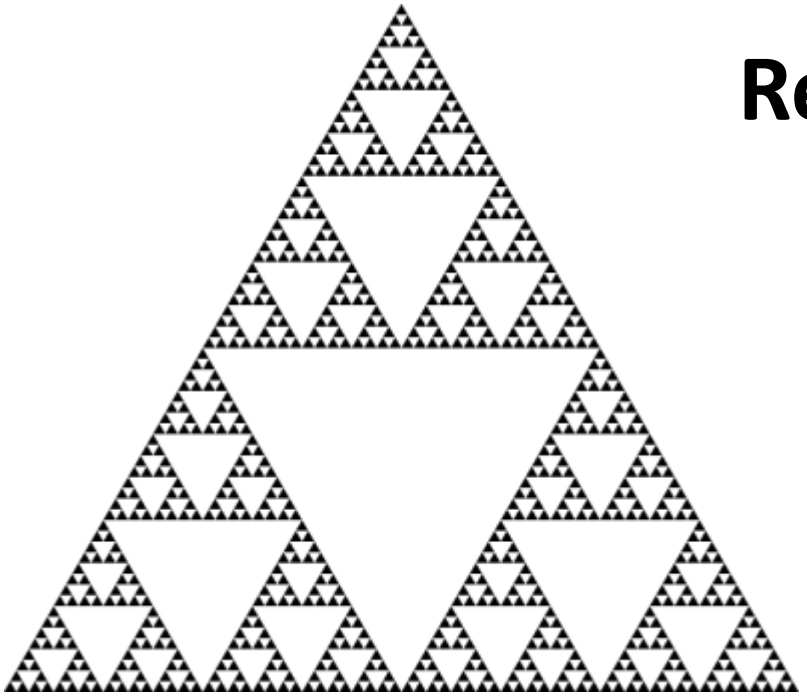
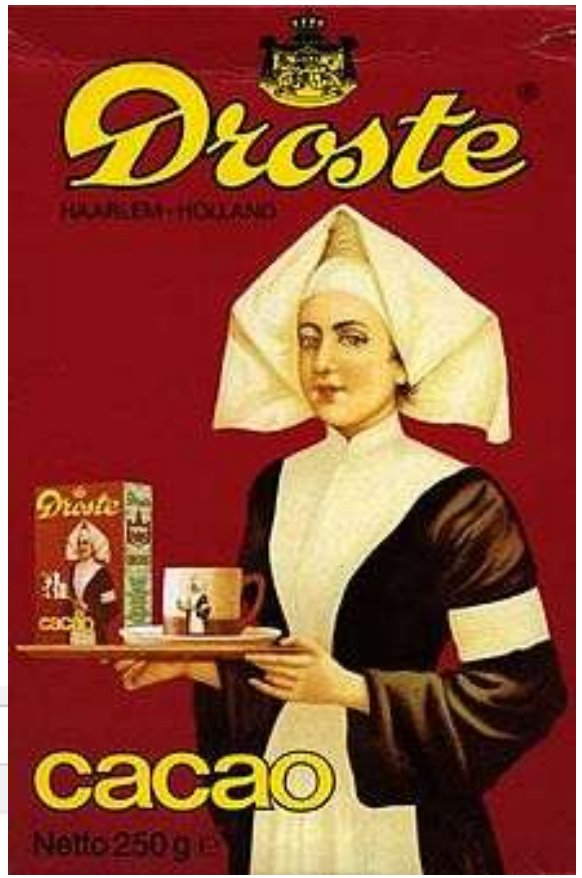
Recursion





Google recursion

Search About 9,160,000 results (0.10 seconds)

Everything
Images
Maps
Videos
News

Did you mean: ***recursion***

Recursion - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Recursion**
Recursion is the process of repeating items in a self-similar way. For instance, when the surfaces of two mirrors are exactly parallel with each other the nested ...
↳ Formal definitions of recursion - Recursion in language

Overview

- Recursion
  - A method calling itself
  - A new way of thinking about a problem
    - Divide and conquer
  - A powerful programming paradigm
  - Related to mathematical induction
- Example applications
  - Factorial
  - Binary search
  - Pretty graphics
  - Sorting things

Mathematical induction

- Prove a statement involving an integer N
  - Base case: Prove it for small N (usually 0 or 1)
  - Induction step:
    - Assume true for size N-1
    - Prove it is true for size N

- Example:
  - Prove T(N) = 1 + 2 + 3 + ... + N = N(N + 1) / 2 for all N
  - Base case: T(1) = 1 = 1(1 + 1) / 2
  - Induction step:
    - Assume true for size N - 1, T(N - 1) = (N - 1)(N) / 2
    - T(N) = 1 + 2 + 3 + ... + N-1 + N

      = (N – 1)(N) / 2 + N

      = (N – 1)(N) / 2 + 2N / 2

      = (N + 1)(N) / 2

Hello recursion

- Goal: Compute factorial N! = 1 * 2 * 3 ... * N
  - Base case: 0! = 1
  - Induction step:
    - Assume we know (N – 1)!
    - Use (N – 1)! to find N!

e.g. 4! = 4 * 3 * 2 * 1 = 24

```
public static long fact(long N)
{
   if (N == 0)
      return 1;
   return fact(N - 1) * N;
}

public static void main(String [] args)
{
   int N = Integer.parseInt(args[0]);
   System.out.println(N + "! = " + fact(N));
}
```

base case

induction step

Instrumented Factorial

```
public static long fact(long N)
{
   System.out.println("start, fact " + N);
   if (N == 0)
   {
      System.out.println("end base, fact " + N);
      return 1;
   }
   long step = fact(N - 1);
   System.out.println("end, fact " + N );
   return step * N;
}
```

```
start, fact 4
start, fact 3
start, fact 2
start, fact 1
start, fact 0
end base, fact 0
end, fact 1
end, fact 2
end, fact 3
end, fact 4
4! = 24
```

Recursion vs. Iterative

- Recursive algorithms also have an iterative version

```
public static long fact(long N)
{
   if (N == 0)
      return 1;
   return fact(N - 1) * N;
}
```

Recursive algorithm

```
public static long fact(long N)
{
   long result = 1;
   for (int i = 1; i <= N; i++)
      result *= i;
   return result;
}
```

Iterative algorithm

- Reasons to use recursion:
  - Code is more compact and easier to understand
  - Easer to reason about correctness
- Reasons not to use recursion:
  - If you end up recalculating things repeatedly (stay tuned)
  - Processor with very little memory (e.g. 8051 = 128 bytes)

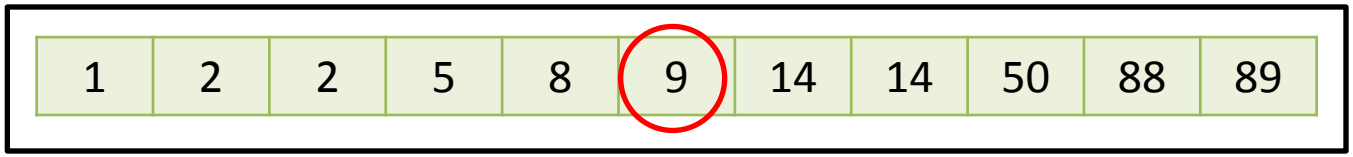
A useful example of recursion

- Binary search
  - Given an array of N sorted numbers
  - Find out if some number t is in the list
  - Do it faster than going linearly through the list, i.e. O(N)
- Basic idea:
  - Like playing higher/lower number guessing

Me	You
I'm thinking of a number between 1 and 100.	50
Higher	75
Lower	63
Higher	69
Higher	72
You got it	Wow I'm super smart!

Binary search

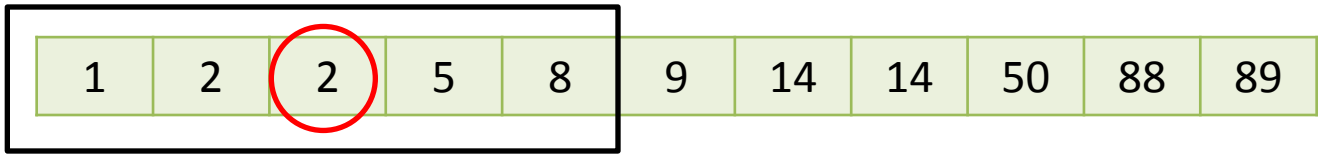
- Binary search algorithm
  - Find midpoint of sorted array
  - Is that element the one you're looking for?
    - If yes, you found it!
  - If target is < midpoint, search lower half
  - If target is > midpoint, search upper half
- Example: Is 5 in the sorted array?




```
target (value)    = 5
low (index)       = 0
high (index)      = 10
midpoint (index)  = (0 + 10) / 2 = 5
```

Binary search

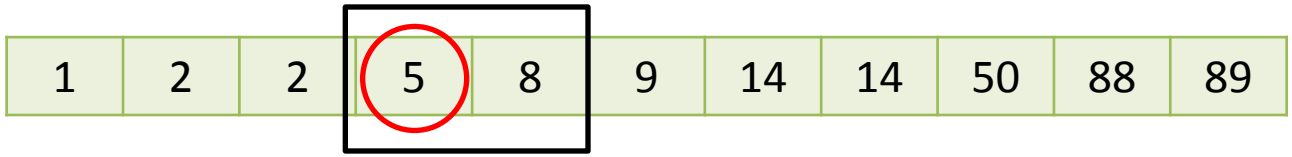
- Binary search algorithm
  - Find midpoint of sorted array
  - Is that element the one you're looking for?
    - If yes, you found it!
  - If target is < midpoint, search lower half
  - If target is > midpoint, search upper half
- Example: Is 5 in the sorted array?




```
target (value)    = 5
low (index)       = 0
high (index)      = 5
midpoint (index)  = (0 + 5) / 2 = 2
```

Binary search

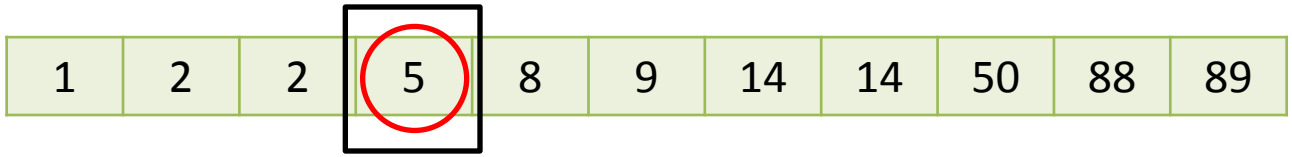
- Binary search algorithm
  - Find midpoint of sorted array
  - Is that element the one you're looking for?
    - If yes, you found it!
  - If target is < midpoint, search lower half
  - If target is > midpoint, search upper half
- Example: Is 5 in the sorted array?




```
target (value)    = 5
low (index)       = 3
high (index)      = 4
midpoint (index) = (3 + 4) / 2 = 3
```

Binary search

- Binary search algorithm
  - Find midpoint of sorted array
  - Is that element the one you're looking for?
    - If yes, you found it!
  - If target is < midpoint, search lower half
  - If target is > midpoint, search upper half
- Example: Is 5 in the sorted array?




```
YES.  Element at new midpoint is target!
```

Binary search, recursive algorithm

```
public static boolean binarySearch(int target, int low, int high, int[] d)
{
   int mid = (low + high) / 2;
   System.out.printf("low %d, high %d, mid %d\n", low, high, mid);
   if (d[mid] == target)
      return true;
   if (high < low)
      return false;
   if (d[mid] < target)
      return binarySearch(target, mid + 1, high, d);
   else
      return binarySearch(target, low, mid - 1, d);
}

public static void main(String [] args)
{
   int [] d   = {1, 2, 2, 5, 8, 9, 14, 14, 50, 88, 89};
   int target = Integer.parseInt(args[0]);
   System.out.println("found " + target + "? " +
                      binarySearch(target, 0, d.length - 1, d));
}
```

Things to avoid

- Missing base case

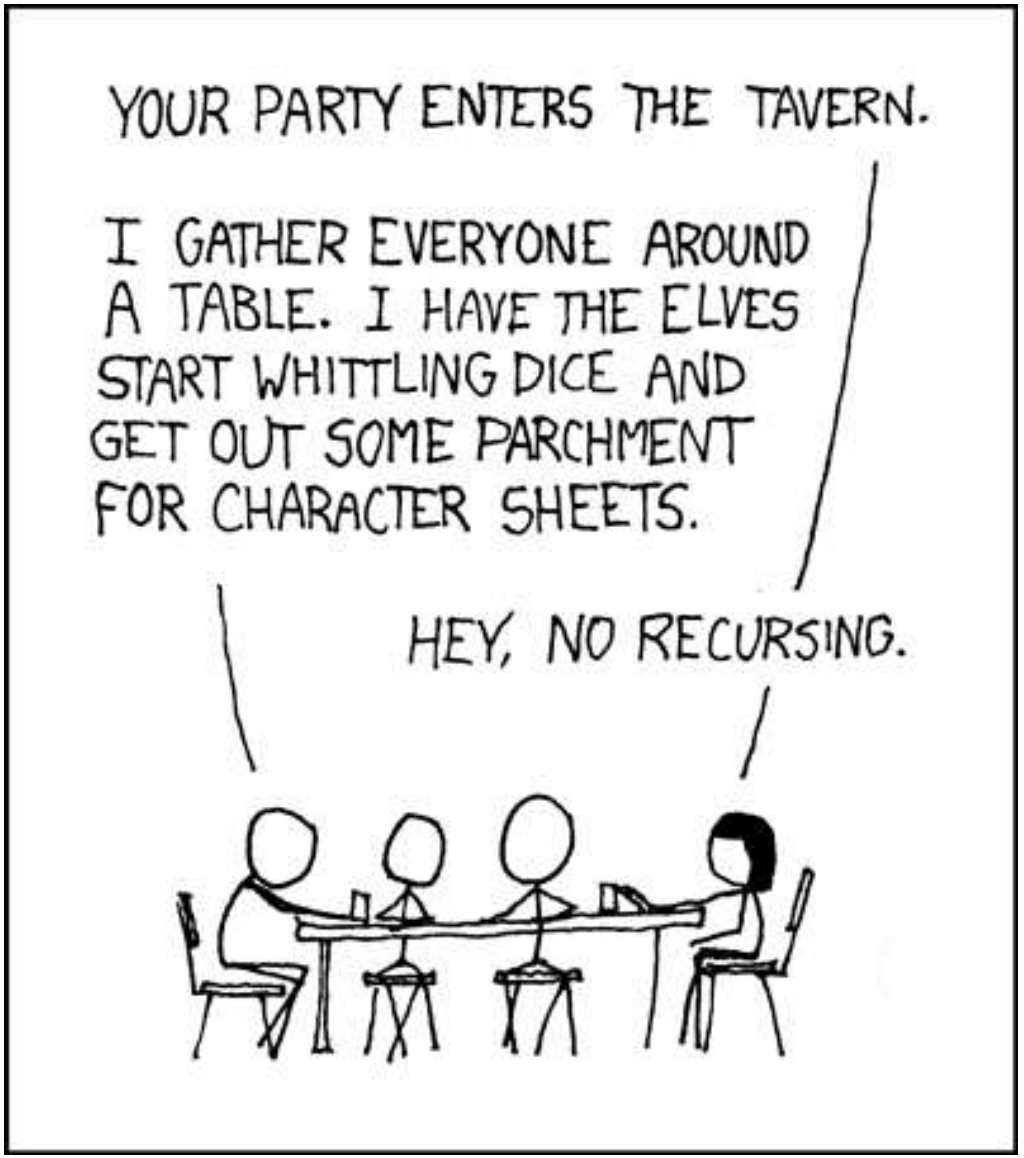
```
public static long fact(long N)
{
   return fact(N - 1) * N;
}
```

```
% java Factorial 5
Exception in thread "main"
java.lang.StackOverflowError
        at Factorial.fact(Factorial.java:8)
        at Factorial.fact(Factorial.java:8)
        at Factorial.fact(Factorial.java:8)
        at Factorial.fact(Factorial.java:8)
        at Factorial.fact(Factorial.java:8)
        at Factorial.fact(Factorial.java:8)
        at Factorial.fact(Factorial.java:8)
        at Factorial.fact(Factorial.java:8)
        at Factorial.fact(Factorial.java:8)
        ...
```

- No convergence guarantee

```
public static double badIdea(int N)
{
   if (N == 1)
      return 1.0;
   return badIdea(1 + N/2) + 1.0/N;
}
```

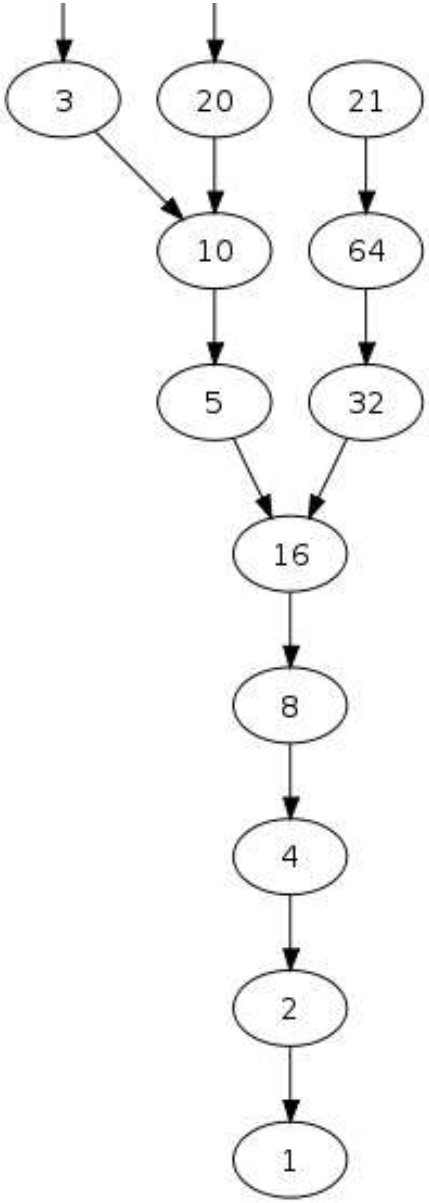
- Both result in infinite recursion = stack overflow



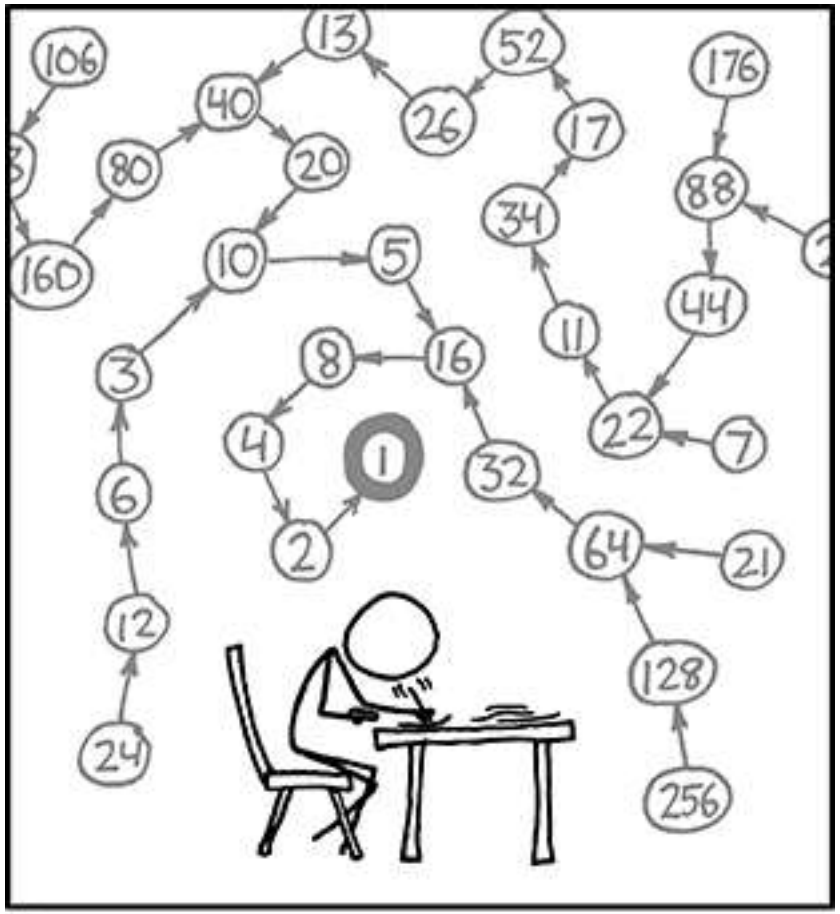
http://xkcd.com/244/

Sometimes we don't know...

- Collatz sequence
  - If N = 1, stop
  - If N is even, divide by 2
  - If N is odd, multiply by 3 and add 1
  - e.g. 24 12 6 3 10 5 16 8 4 2 1
  - No one knows if this terminates for all N!

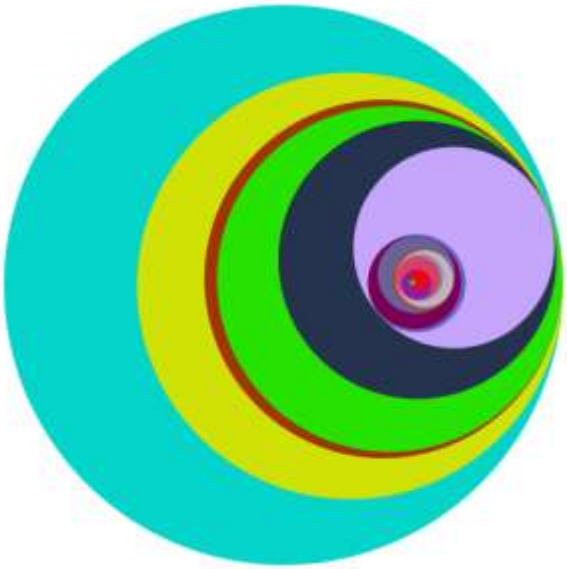
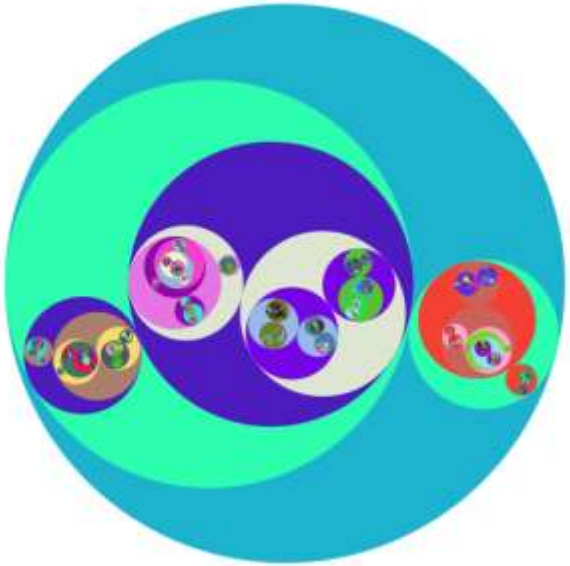


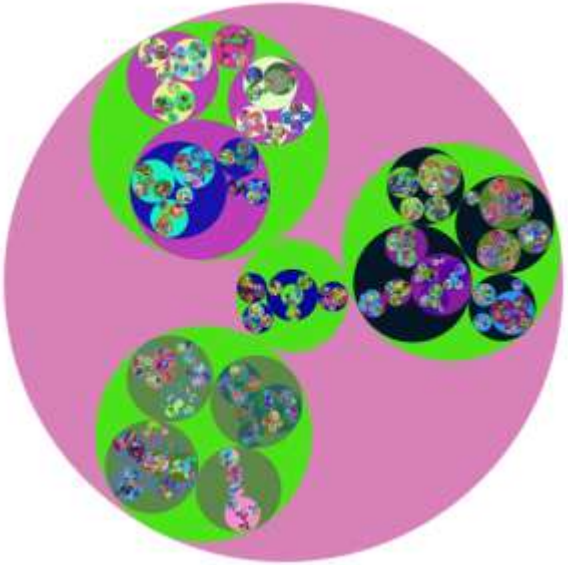
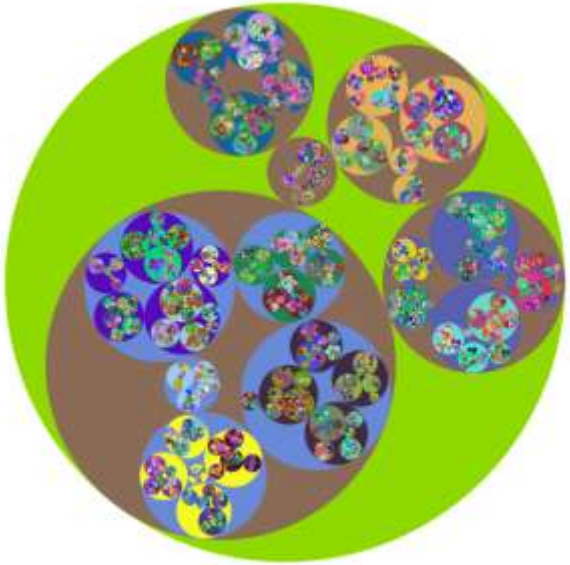
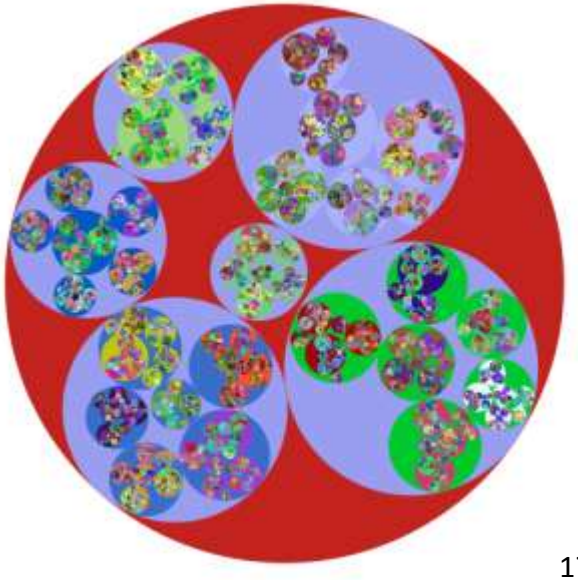

```
public static void collatz(long N)
{
   System.out.print(N + " ");
   if (N == 1)
      return;
   if (N % 2 == 0)
      collatz(N / 2);
   else
      collatz(3 * N + 1);
}
```

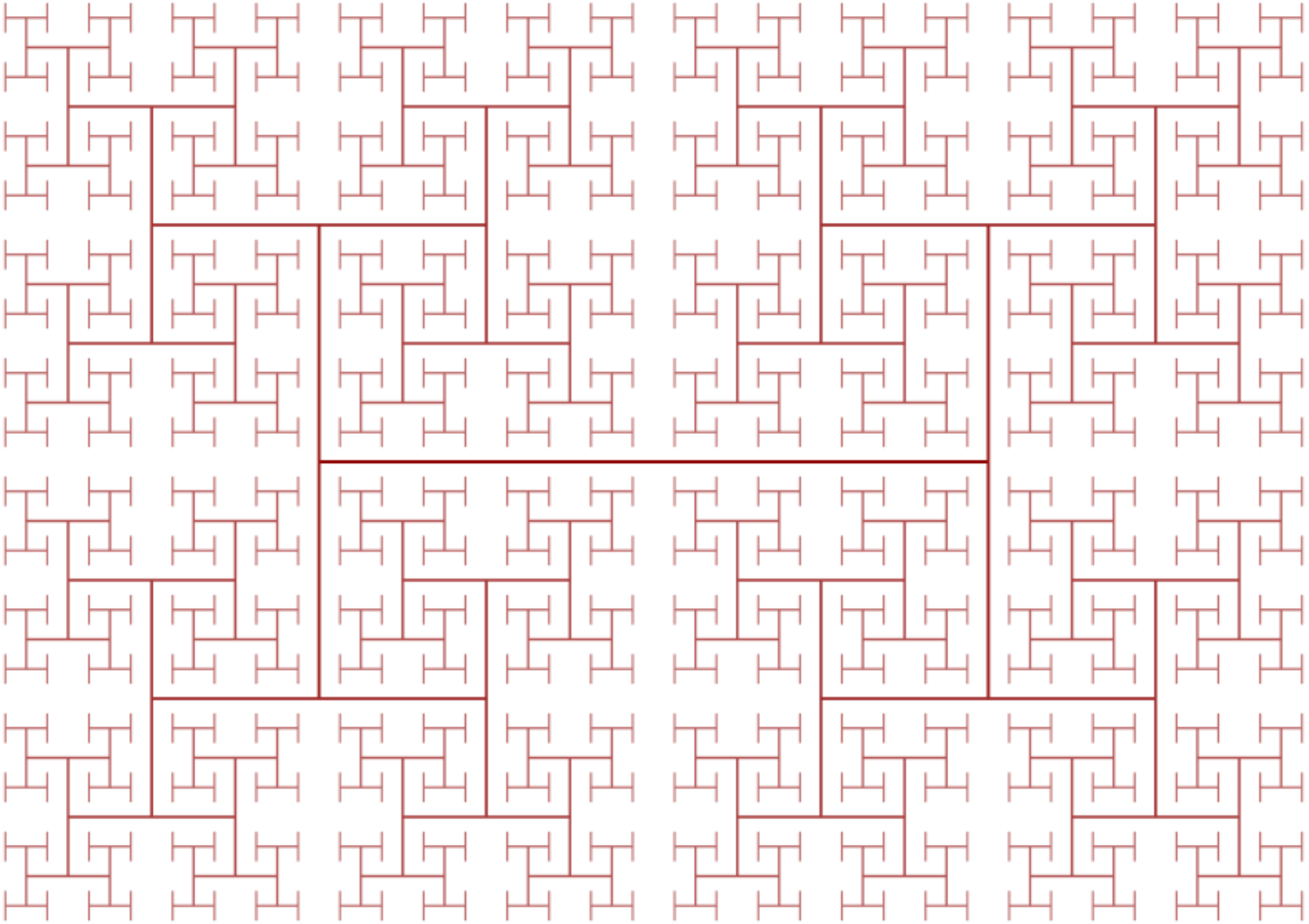


THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.

Recursive graphics

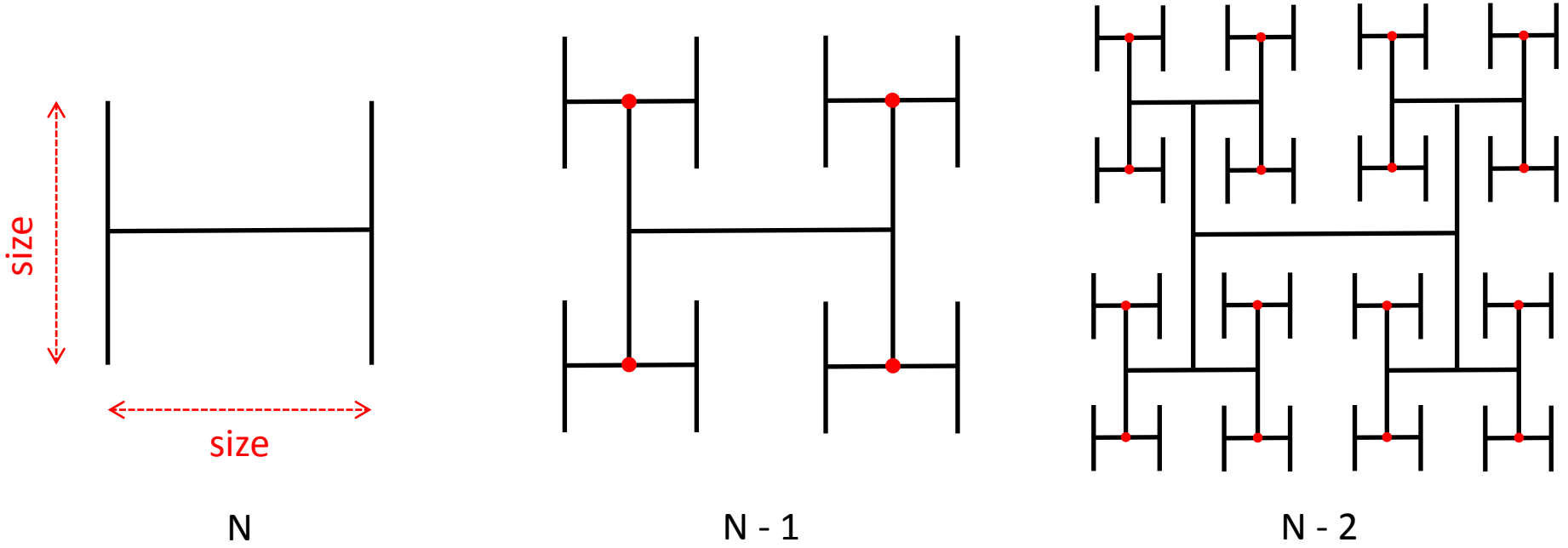





H-tree

- H-tree of order N
  - Draw an H
  - Recursively draw 4 H-trees
    - One at each "tip" of the original H, half the size
    - Stop once N = 0

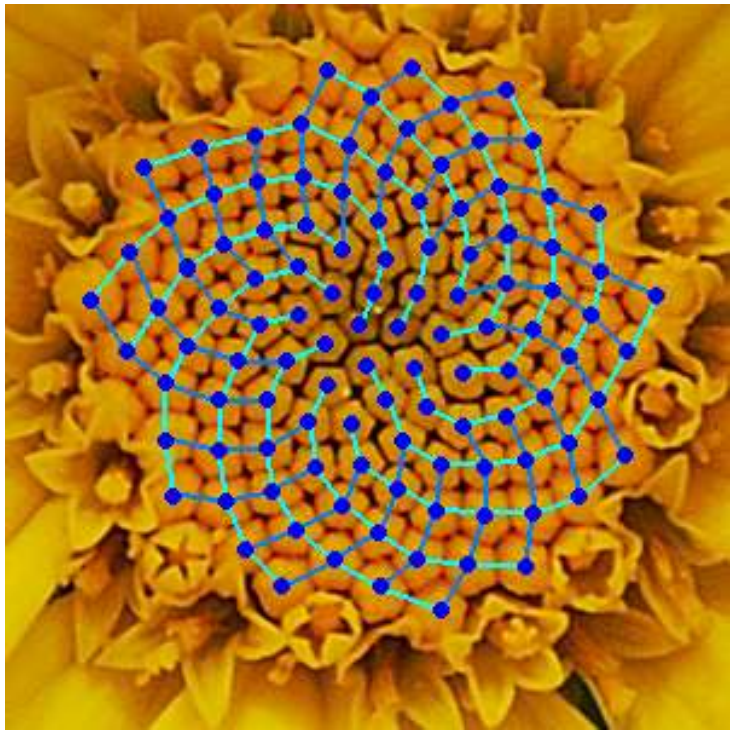


Fibonacci Numbers

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, …

$$F_0 = 0$$
$$F_1 = 1$$
$$F_n = F_{n-1} + F_{n-2}$$

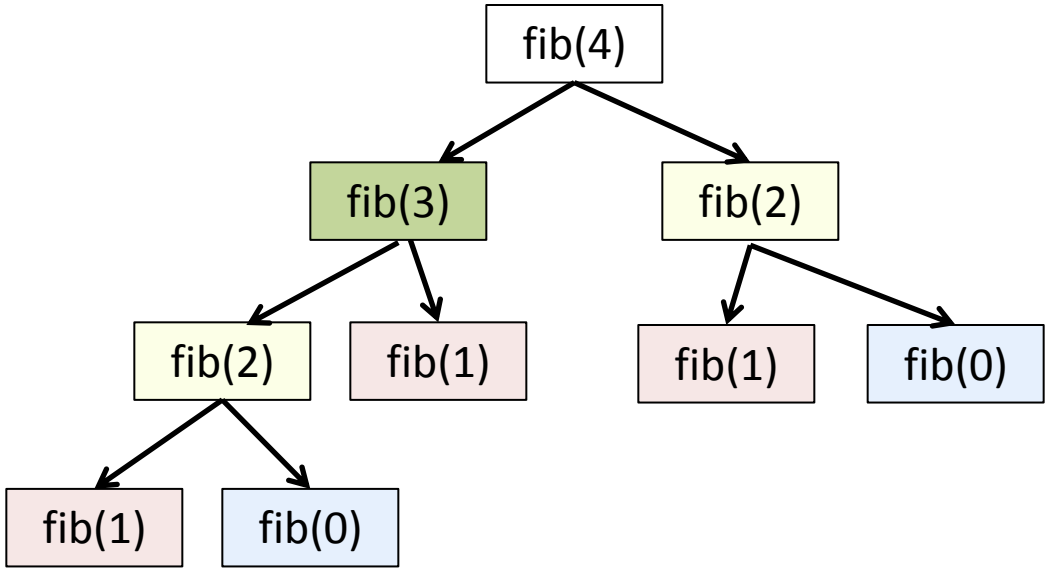
Natural fit for recursion?



```
public static long fib(long n)
{
    if (n == 0)
        return 0;
    if (n == 1)
        return 1;
    return fib(n - 1) + fib(n -2);
}
```

Trouble in recursion city…

N	time, fib(N)
10	0.000
20	0.002
30	0.011
40	0.661
41	1.080
42	1.748
43	2.814
44	4.531
45	7.371
46	11.860
47	19.295
48	31.319
49	50.668
50	81.542



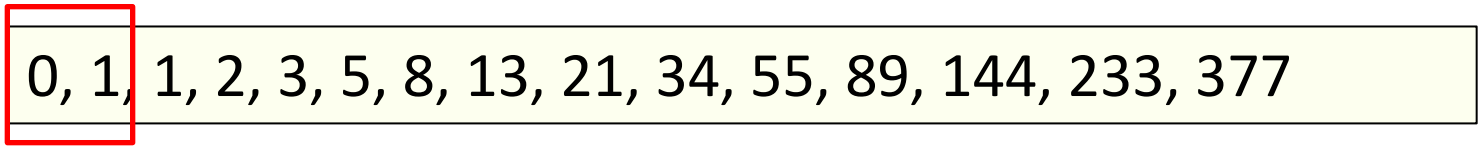
Bad news bears.

Order of growth:

Exponential!

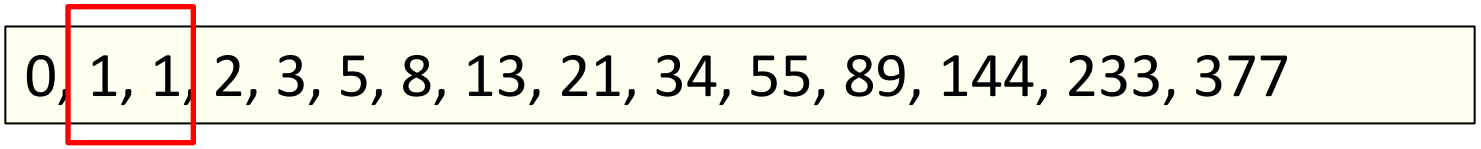
Efficient Fibonacci version

- Remember last two numbers
  - Trade a (very) little amount of memory for a lot of time



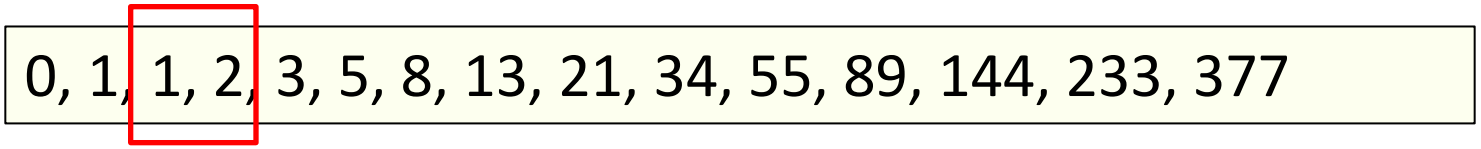
Efficient Fibonacci version

- Remember last two numbers
  - Use F_{n-2} and F_{n-1} to get F_n



Efficient Fibonacci version

- Remember last two numbers
  - Use F_{n-2} and F_{n-1} to get F_n



Efficient Fibonacci version

- Remember last two numbers
  - Use F_{n-2} and F_{n-1} to get F_n

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377

```
public static long fibFast(int n)
{
    long n2 = 0;
    long n1 = 1;
    if (n == 0)
        return 0;
    for (int i = 1; i < n; i++)
    {
        long n0 = n1 + n2;
        n2 = n1;
        n1 = n0;
    }
    return n1;
}
```

N	time, fib(N)
50	0.001
100	0.001
200	0.001
400	0.001
10,000,000	0.010
20,000,000	0.016
40,000,000	0.028
80,000,000	0.051
160,000,000	0.096

Summary

- Recursion
  - A method calling itself
    - Sometimes just once, e.g. binary search
    - Sometimes multiple times, e.g. H-tree
  - All good recursion must come to an end
    - Sooner or later method must NOT call itself recursively
  - A powerful tool in computer science
    - Allows writing elegant and easy to understand algorithms
    - (Once you get your head around it)

http://www.flickr.com/photos/zeusandhera/2496831822/