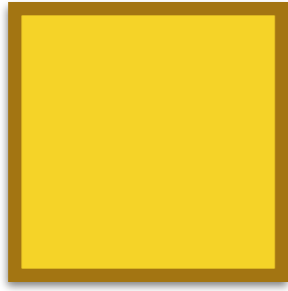


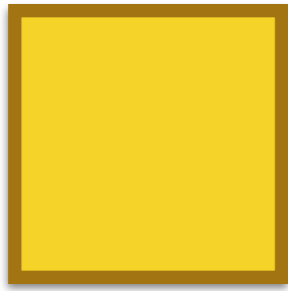
# Introduction To Recursive Functions

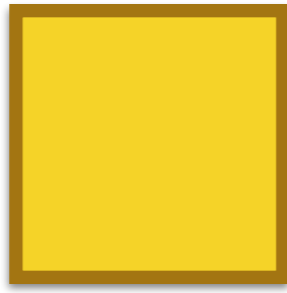
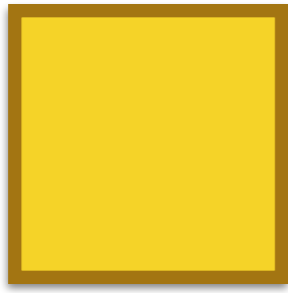
D. Thiebaut — CSC212  
Fall 2014

**LARGE TASK**



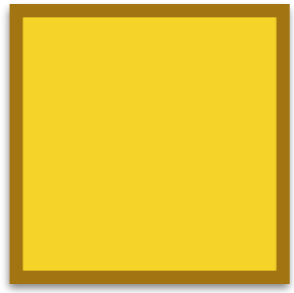
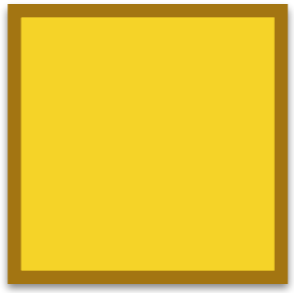
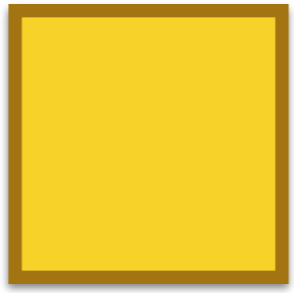
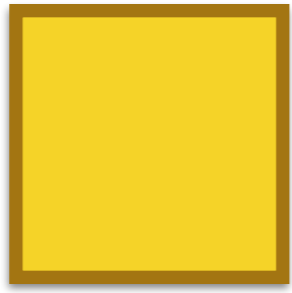
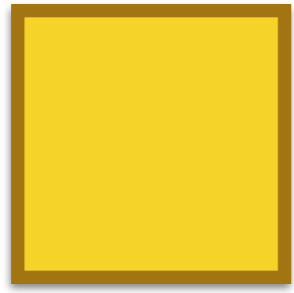


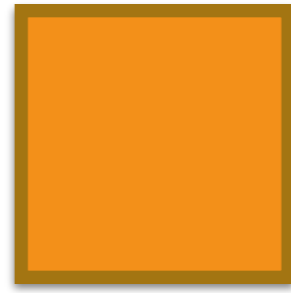
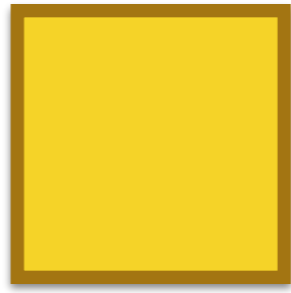
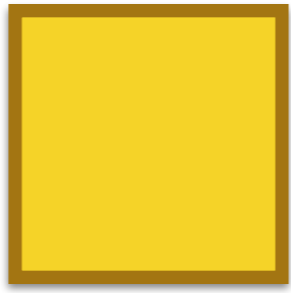
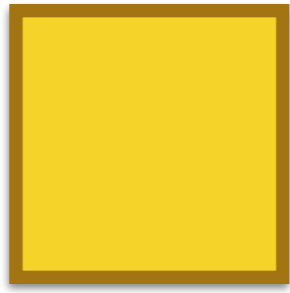
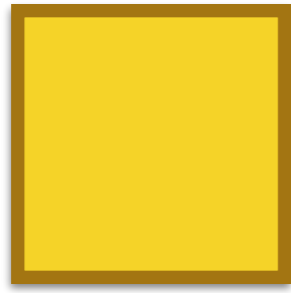




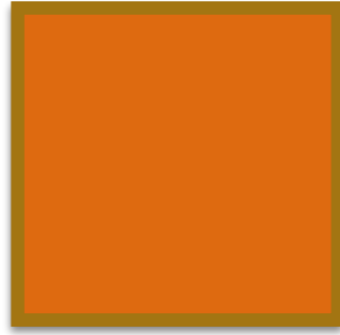
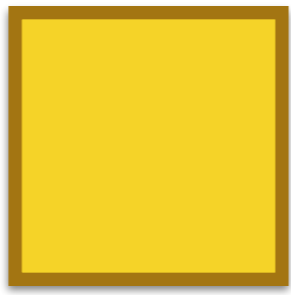
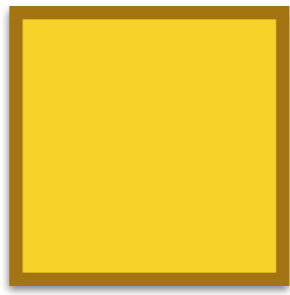
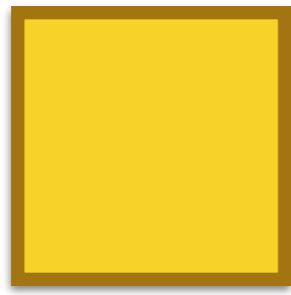


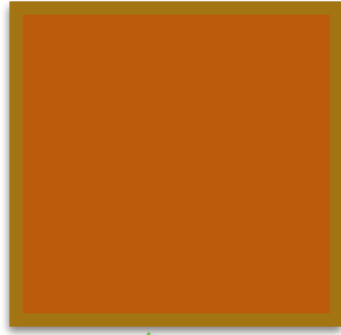
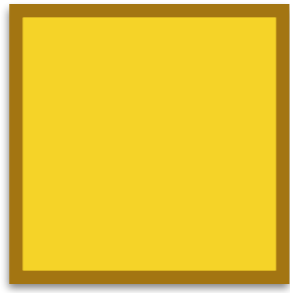
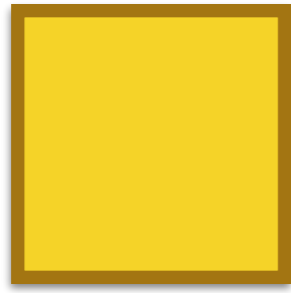
**AFTER A WHILE...**





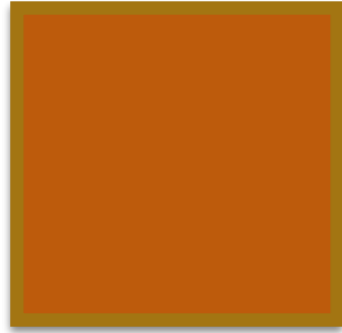
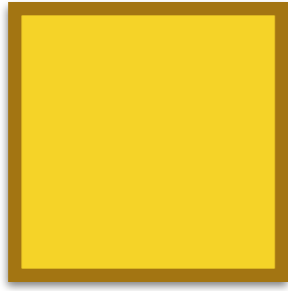




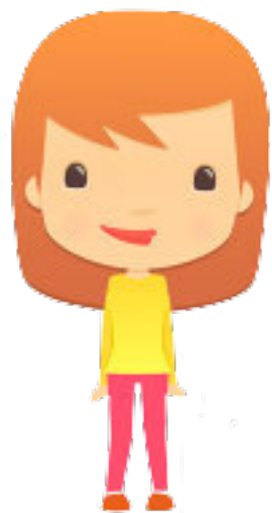




**AFTER A WHILE...**



THE  
ANSWER!



# Important Concept: 1

- Recursive step reduces the problem in a small, but significant way, getting closer to a solution
- Work done during a recursive call builds up on the partial solution found so far.

# Important Concept: 2

- Recursion requires
  1. **Stopping** Condition
  2. Recursive Step **Reducing Size of Problem** and leading closer to solution.

Examples



# Draft Algorithm, then Code

- Factorial
- Sum up an array
- Find the largest element of an array
- Find a key in an unsorted array
- Find a key in a sorted array
- Evaluate an RPN expression

# Evaluating Time Complexity

- Factorial

```
private static int factorial(int n) {  
    if ( n <= 1 )  
        return 1;  
  
    return n * factorial( n - 1 );  
}
```

# Evaluating Time Complexity

- Binary Search

```
private static int binSearch( ArrayList A, int low, int high, int key ) {  
    if ( low > high )  
        return -1;  
  
    int mid = ( low+high )/2;  
    if ( (int) A.get( mid ) == key )  
        return mid;  
  
    if ( (int) A.get( mid ) < key )  
        return binSearch( A, mid+1, high, key );  
    else  
        return binSearch( A, low, mid-1, key );  
}
```

# Tail Recursion



# Let's Revisit Fibonacci

```
private static long computeFibRecursively( int n ) {  
    if ( n <= 1 )  
        return 1;  
  
    return computeFibRecursively( n-1 ) + computeFibRecursively( n-2 );  
}
```

# Fib's Call Tree

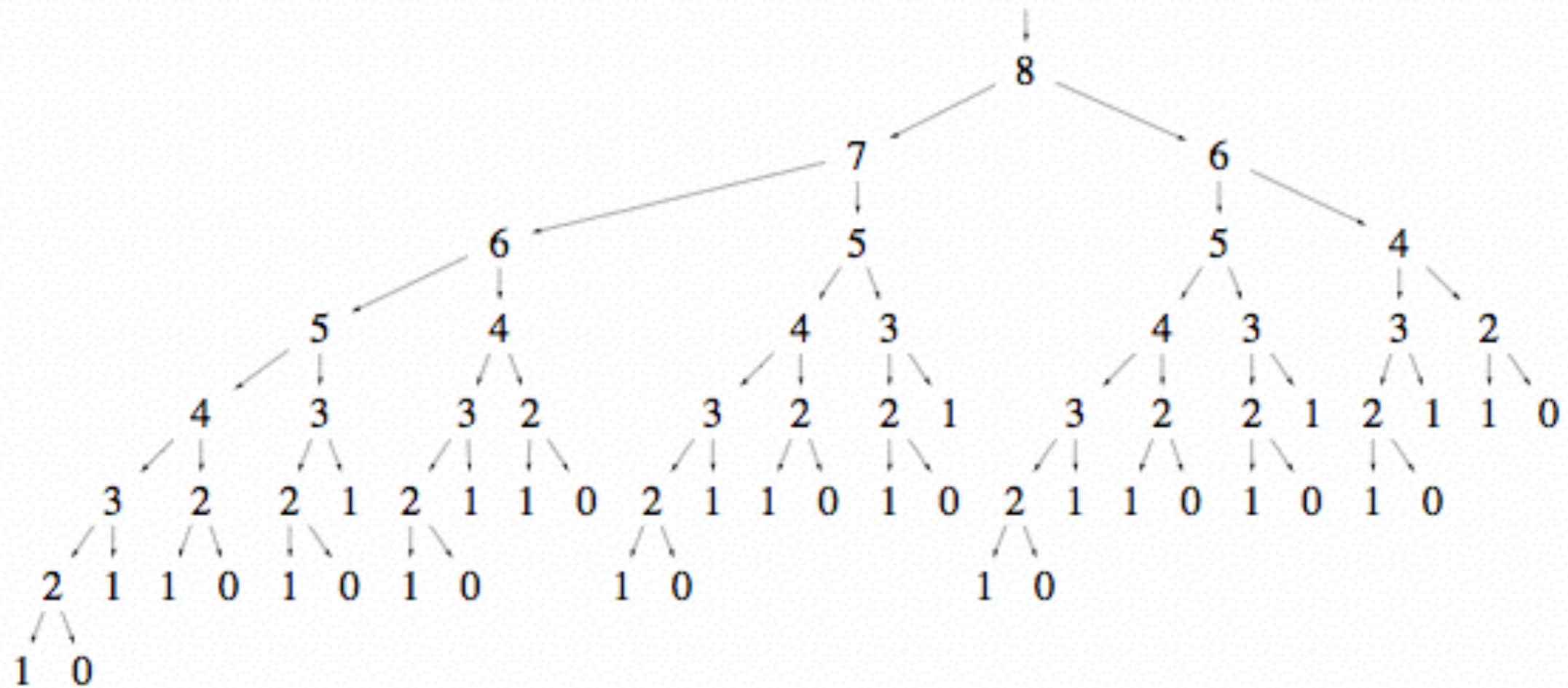
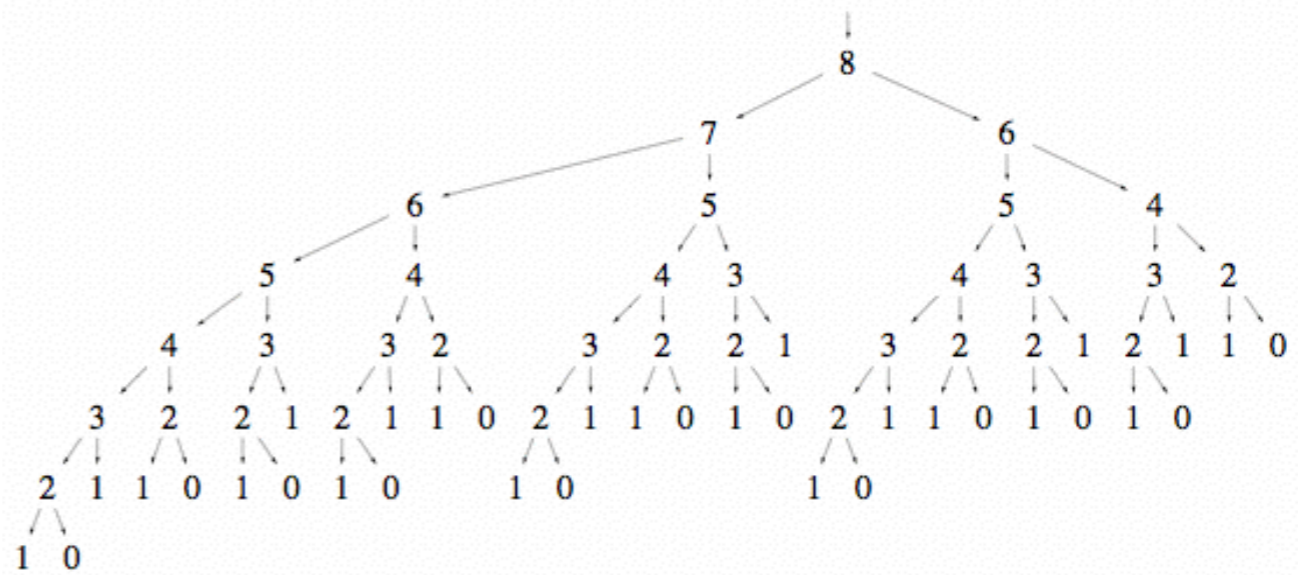


Image taken from [http://www.cs.ucr.edu/~neal/2005/cs141/wikidb/uploads/fib\\_call\\_graph\\_8.gif](http://www.cs.ucr.edu/~neal/2005/cs141/wikidb/uploads/fib_call_graph_8.gif)



# Observations:



- **So Many leaves!**
- Most of the work in the lower part of the tree, where the leaves are...
- If we could "prune" the tree, we could reduce the amount of work done...

# Solution? Cut the Tail-End Recursion!

```
private static long computeFibRecursively( int n ) {  
    long[] f10 = new long[] {1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89};  
  
    if ( n <= 10 )  
        return f10[n];  
  
    return computeFibRecursively( n-1 )  
        + computeFibRecursively( n-2 );  
}
```

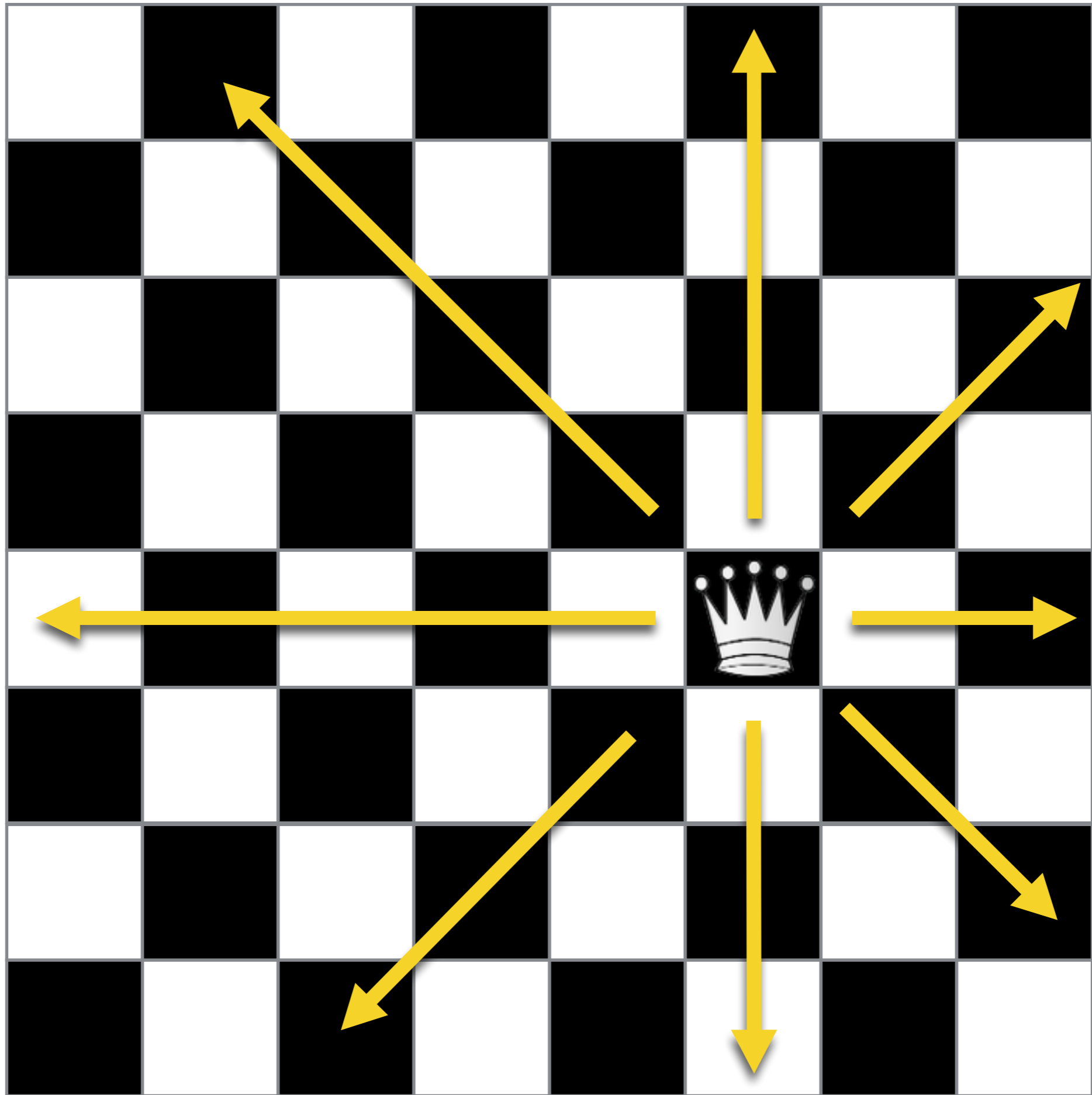


```
Desktop — Beowulf2 — ssh — 71x24
Beowulf2
[beowulf2]
[07:07:35] ~/public_html/classes/212$:
[beowulf2]
[07:07:36] ~/public_html/classes/212$:
[beowulf2]
[07:07:36] ~/public_html/classes/212$:
[beowulf2]
[07:07:36] ~/public_html/classes/212$: figlet DEMO TIME

  _   _   _   _   _   _   _   _   _   _
 | | | | | | | | | | | | | | | | | |
 | |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
 | |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|
 |_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|_|

[beowulf2]
[07:07:38] ~/public_html/classes/212$:
```

# The N-Queens Problem



**Question:** Can one  
put 8 queens  
on a chess board,  
such that no two queens  
can take each other?



TRY IT!



# Questions Before Coding

- What data structure can we use?
- How do we represent a placed queen?
- How do we represent a cell "covered" by a queen?
- How do we represent an empty cell?

# Important Concept Of the Day

- **Back-Tracking**: the action of returning from recursive exploration of a sub-problem, undoing some computation, selecting a new unexplored path, and starting exploring it recursively.

